## Contents:

1 Installation  
   1.1 Installation in Anaconda .................................................. 3  
   1.2 Installation in pip ............................................................. 3  
      1.2.1 Install Requirements .................................................. 3  
      1.2.2 Install Library ......................................................... 4  
   1.3 Installation from Source ................................................... 4  
   1.4 Uninstall ................................................................. 4  

2 Examples: Basics  
   2.1 A standard use case ......................................................... 5  
   2.2 A simple and common augmentation sequence .............................. 5  
   2.3 Heavy Augmentations ....................................................... 6  

3 Examples: Keypoints  
   3.1 Notebook ................................................................. 11  
   3.2 A simple example .......................................................... 11  

4 Examples: Bounding Boxes  
   4.1 Notebook ................................................................. 13  
   4.2 A simple example .......................................................... 13  
   4.3 Dealing with bounding boxes outside of the image ...................... 14  
   4.4 Shifting/Moving Bounding Boxes ......................................... 17  
   4.5 Projection of BBs Onto Rescaled Images .................................. 17  
   4.6 Computing Intersections, Unions and IoUs .............................. 19  

5 Examples: Heatmaps  
   5.1 Notebook ................................................................. 23  
   5.2 A simple example .......................................................... 24  
   5.3 Multiple sub-heatmaps per heatmaps object ............................. 25  
   5.4 Accessing the heatmap array .............................................. 27  
   5.5 Resizing heatmaps .......................................................... 28  
   5.6 Padding heatmaps .......................................................... 31  

6 Examples: Segmentation Maps and Masks  
   6.1 Notebook ................................................................. 33  
   6.2 Upcoming Changes ......................................................... 34  
   6.3 A simple example .......................................................... 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stochastic Parameters</td>
<td>39</td>
</tr>
<tr>
<td>7.1 Introduction</td>
<td>39</td>
</tr>
<tr>
<td>7.2 Continuous Probability Distributions</td>
<td>40</td>
</tr>
<tr>
<td>7.3 Discrete Probability Distributions</td>
<td>48</td>
</tr>
<tr>
<td>7.4 Arithmetic</td>
<td>51</td>
</tr>
<tr>
<td>7.5 Special Parameters</td>
<td>54</td>
</tr>
<tr>
<td>7.6 Noise Parameters</td>
<td>60</td>
</tr>
<tr>
<td>Blending/Overlaying images</td>
<td>61</td>
</tr>
<tr>
<td>8.1 Introduction</td>
<td>61</td>
</tr>
<tr>
<td>8.2 Constant Alpha</td>
<td>62</td>
</tr>
<tr>
<td>8.3 SimplexNoiseAlpha</td>
<td>64</td>
</tr>
<tr>
<td>8.4 FrequencyNoiseAlpha</td>
<td>68</td>
</tr>
<tr>
<td>8.5 IterativeNoiseAggregator</td>
<td>70</td>
</tr>
<tr>
<td>8.6 Sigmoid</td>
<td>71</td>
</tr>
<tr>
<td>Overview of Augmenters</td>
<td>75</td>
</tr>
<tr>
<td>9.1 Sequential</td>
<td>75</td>
</tr>
<tr>
<td>9.2 SomeOf</td>
<td>76</td>
</tr>
<tr>
<td>9.3 OneOf</td>
<td>78</td>
</tr>
<tr>
<td>9.4 Sometimes</td>
<td>78</td>
</tr>
<tr>
<td>9.5 WithColorspace</td>
<td>80</td>
</tr>
<tr>
<td>9.6 WithChannels</td>
<td>81</td>
</tr>
<tr>
<td>9.7 Noop</td>
<td>82</td>
</tr>
<tr>
<td>9.8 Lambda</td>
<td>82</td>
</tr>
<tr>
<td>9.9 AssertLambda</td>
<td>82</td>
</tr>
<tr>
<td>9.10 AssertShape</td>
<td>83</td>
</tr>
<tr>
<td>9.11 Resize</td>
<td>83</td>
</tr>
<tr>
<td>9.12 CropAndPad</td>
<td>84</td>
</tr>
<tr>
<td>9.13 Pad</td>
<td>87</td>
</tr>
<tr>
<td>9.14 Crop</td>
<td>87</td>
</tr>
<tr>
<td>9.15 Fliplr</td>
<td>87</td>
</tr>
<tr>
<td>9.16 Flipud</td>
<td>87</td>
</tr>
<tr>
<td>9.17 Superpixels</td>
<td>88</td>
</tr>
<tr>
<td>9.18 ChangeColorspace</td>
<td>89</td>
</tr>
<tr>
<td>9.19 Grayscale</td>
<td>90</td>
</tr>
<tr>
<td>9.20 GaussianBlur</td>
<td>91</td>
</tr>
<tr>
<td>9.21 AverageBlur</td>
<td>91</td>
</tr>
<tr>
<td>9.22 MedianBlur</td>
<td>91</td>
</tr>
<tr>
<td>9.23 Convolve</td>
<td>91</td>
</tr>
<tr>
<td>9.24 Sharpen</td>
<td>97</td>
</tr>
<tr>
<td>9.25 Emboss</td>
<td>97</td>
</tr>
<tr>
<td>9.26 EdgeDetect</td>
<td>97</td>
</tr>
<tr>
<td>9.27 DirectedEdgeDetect</td>
<td>98</td>
</tr>
<tr>
<td>9.28 Add</td>
<td>98</td>
</tr>
<tr>
<td>9.29 AddElementwise</td>
<td>99</td>
</tr>
<tr>
<td>9.30 AdditiveGaussianNoise</td>
<td>99</td>
</tr>
<tr>
<td>9.31 Multiply</td>
<td>103</td>
</tr>
<tr>
<td>9.32 MultiplyElementwise</td>
<td>103</td>
</tr>
<tr>
<td>9.33 Dropout</td>
<td>107</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>13.24 imgaug.augmenters.size</td>
<td>407</td>
</tr>
<tr>
<td>13.25 imgaug.augmenters.weather</td>
<td>427</td>
</tr>
<tr>
<td>14 Indices and tables</td>
<td>443</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>445</td>
</tr>
<tr>
<td>Index</td>
<td>447</td>
</tr>
</tbody>
</table>
imgaug is a library for image augmentation in machine learning experiments. It supports a wide range of augmentation techniques, allows to easily combine these and to execute them in random order or on multiple CPU cores, has a simple yet powerful stochastic interface and can not only augment images, but also keypoints/landmarks, bounding boxes, heatmaps and segmentation maps.

![Example augmentations of a single input image.](image)

Fig. 1: Example augmentations of a single input image.
The library uses python, which must be installed. Python 2.7, 3.4, 3.5, 3.6 and 3.7 are supported.
The below sections explain how to install the library in anaconda or via pip. If you don’t know what anaconda (or conda) are, simply use pip instead as that should always work.

1.1 Installation in Anaconda

To install in anaconda simply perform the following commands

```
conda config --add channels conda-forge
conda install imgaug
```

Note that you may also use the pip-based installation commands described below. They work with and without anaconda.

1.2 Installation in pip

1.2.1 Install Requirements

To install all requirements, use

```
pip install six numpy scipy Pillow matplotlib scikit-image opencv-python imageio

#Shapely
```

Note that if you already have OpenCV, you might not need opencv-python. If you get any “permission denied” errors, try adding sudo in front of the command. If your encounter issues installing Shapely you can skip that library. It is only imported when actually needed. At least polygon and line string augmentation will likely crash without it.
1.2.2 Install Library

Once the required packages are available, imgaug can be installed using the following command:

```bash
pip install imgaug
```

This installs the latest version from pypi, which often lags behind the latest version on github by a few months. To instead get the very latest version use

```bash
pip install git+https://github.com/aleju/imgaug
```

1.3 Installation from Source

In rare cases, one might prefer to install directly from source. This is possible using

```bash
git clone https://github.com/aleju/imgaug.git && cd imgaug && python setup.py install
```

Note that this is effectively identical to using `pip install <github link>`.

1.4 Uninstall

To deinstall the library use

```bash
conda remove imgaug
```

on anaconda and

```bash
pip uninstall imgaug
```

otherwise.
CHAPTER 2

Examples: Basics

2.1 A standard use case

The following example shows a standard use case. An augmentation sequence (crop + horizontal flips + gaussian blur) is defined once at the start of the script. Then many batches are loaded and augmented before being used for training.

```python
from imgaug import augmenters as iaa

seq = iaa.Sequential(
    
    iaa.Crop(px=(0, 16)),  
    iaa.Fliplr(0.5),       
    iaa.GaussianBlur(sigma=(0, 3.0))
)

for batch_idx in range(1000):
    images = load_batch(batch_idx)
    images_aug = seq(images=images)
    train_on_images(images_aug)
```

2.2 A simple and common augmentation sequence

The following example shows an augmentation sequence that might be useful for many common experiments. It applies crops and affine transformations to images, flips some of the images horizontally, adds a bit of noise and blur and also changes the contrast as well as brightness.

```python
from imgaug import augmenters as iaa

seq = iaa.Sequential(
    
    iaa.Crop(px=(0, 16)),  
    iaa.Fliplr(0.5),       
    iaa.GaussianBlur(sigma=(0, 3.0))
)

for batch_idx in range(1000):
    images = load_batch(batch_idx)
    images_aug = seq(images=images)
    train_on_images(images_aug)
```
import numpy as np
import imgaug as ia
import imgaug.augmenters as iaa

ia.seed(1)

# Example batch of images.
# The array has shape (32, 64, 64, 3) and dtype uint8.
images = np.array(
    [ia.quokka(size=(64, 64)) for _ in range(32)],
    dtype=np.uint8
)

seq = iaa.Sequential(
    
    iaa.Fliplr(0.5),  # horizontal flips
    iaa.Crop(percent=(0, 0.1)),  # random crops
    # Small gaussian blur with random sigma between 0 and 0.5.
    # But we only blur about 50% of all images.
    iaa.Sometimes(0.5,
        iaa.GaussianBlur(sigma=(0, 0.5))
    ),
    # Strengthen or weaken the contrast in each image.
    iaa.ContrastNormalization((0.75, 1.5)),
    # Add gaussian noise.
    # For 50% of all images, we sample the noise once per pixel.
    # For the other 50% of all images, we sample the noise per pixel AND
    # channel. This can change the color (not only brightness) of the
    # pixels.
    iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255), per_channel=0.5),
    # Make some images brighter and some darker.
    # In 20% of all cases, we sample the multiplier once per channel,
    # which can end up changing the color of the images.
    iaa.Multiply((0.8, 1.2), per_channel=0.2),
    # Apply affine transformations to each image.
    # Scale/zoom them, translate/move them, rotate them and shear them.
    iaa.Affine(
        scale={"x": (0.8, 1.2), "y": (0.8, 1.2)},
        translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)},
        rotate=(-25, 25),
        shear=(-8, 8)
    ),
    random_order=True)  # apply augmenters in random order

images_aug = seq(images=images)

2.3 Heavy Augmentations

The following example shows a large augmentation sequence containing many different augmenters, leading to significant changes in the augmented images. Depending on the use case, the sequence might be too strong. Occasionally it can also break images by changing them too much. To weaken the effects you can lower the value of iaa.SomeOf((0, 5), ...) to e.g. (0, 3) or decrease the probability of some augmenters to be applied by decreasing in sometimes = lambda aug: iaa.Sometimes(0.5, aug) the value 0.5 to e.g. 0.3.
import numpy as np
import imgaug as ia
import imgaug.augmenters as iaa

ia.seed(1)

# Example batch of images.
# The array has shape (32, 64, 64, 3) and dtype uint8.
images = np.array([ia.quokka(size=(64, 64)) for _ in range(32)], dtype=np.uint8)

# Sometimes(0.5, ...) applies the given augmenter in 50% of all cases,
# e.g. Sometimes(0.5, GaussianBlur(0.3)) would blur roughly every second
# image.
sometimes = lambda aug: iaa.Sometimes(0.5, aug)

# Define our sequence of augmentation steps that will be applied to every image.
seq = iaa.Sequential(
    [
        # Apply the following augmenters to most images.
        # iaa.Fliplr(0.5), # horizontally flip 50% of all images
        # iaa.Flipud(0.2), # vertically flip 20% of all images

        # crop some of the images by 0-10% of their height/width
        sometimes(iaa.Crop(percent=(0, 0.1))),

        # Apply affine transformations to some of the images
    ]
)

Fig. 1: Example results of the above simple augmentation sequence.

(continues on next page)

2.3. Heavy Augmentations
# - scale to 80-120% of image height/width (each axis independently)
# - translate by -20 to +20 relative to height/width (per axis)
# - rotate by -45 to +45 degrees
# - shear by -16 to +16 degrees
# - order: use nearest neighbour or bilinear interpolation (fast)
# - mode: use any available mode to fill newly created pixels
#     see API or scikit-image for which modes are available
# - cval: if the mode is constant, then use a random brightness
#     for the newly created pixels (e.g. sometimes black,  
#     sometimes white)

sometimes(iaa.Affine(
    scale={"x": (0.8, 1.2), "y": (0.8, 1.2)},
    translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2)},
    rotate=(-45, 45),
    shear=(-16, 16),
    order=[0, 1],
    cval=(0, 255),
    mode=ia.ALL
)),

# Execute 0 to 5 of the following (less important) augmenters per
# image. Don't execute all of them, as that would often be way too
# strong.
# iaa.SomeOf((0, 5),
#     [
#         # Convert some images into their superpixel representation,
#         # sample between 20 and 200 superpixels per image, but do
#         # not replace all superpixels with their average, only
#         # some of them (p_replace).
#         sometimes(
#             iaa.Superpixels(
#                 p_replace=(0, 1.0),
#                 n_segments=(20, 200)
#             ),
#         ),
#         # Blur each image with varying strength using
#         # gaussian blur (sigma between 0 and 3.0),
#         # average/uniform blur (kernel size between 2x2 and 7x7)
#         # median blur (kernel size between 3x3 and 11x11).
#         iaa.OneOf([
#             iaa.GaussianBlur((0, 3.0)),
#             iaa.AverageBlur(k=(2, 7)),
#             iaa.MedianBlur(k=(3, 11)),
#         ]),
#         # Sharpen each image, overlay the result with the original
#         # image using an alpha between 0 (no sharpening) and 1
#         # (full sharpening effect).
#         iaa.Sharpen(alpha=(0, 1.0), lightness=(0.75, 1.5)),
#         # Same as sharpen, but for an embossing effect.
#         iaa.Emboss(alpha=(0, 1.0), strength=(0, 2.0)),
#         # Search in some images either for all edges or for
# directed edges. These edges are then marked in a black
# and white image and overlayed with the original image
# using an alpha of 0 to 0.7.
sometimes(iaa.OneOf([  
    iaa.EdgeDetect(alpha=(0, 0.7)),
    iaa.DirectedEdgeDetect(
        alpha=(0, 0.7), direction=(0.0, 1.0)
    ),
]),
]),

# Add gaussian noise to some images.
# In 50% of these cases, the noise is randomly sampled per
# channel and pixel.
# In the other 50% of all cases it is sampled once per
# pixel (i.e. brightness change).
iaa.AdditiveGaussianNoise(
    loc=0, scale=(0.0, 0.05*255), per_channel=0.5
),

# Either drop randomly 1 to 10% of all pixels (i.e. set
# them to black) or drop them on an image with 2-5% percent
# of the original size, leading to large dropped
# rectangles.
iaa.OneOf([  
    iaa.Dropout((0.01, 0.1), per_channel=0.5),
    iaa.CoarseDropout(
        (0.03, 0.15), size_percent=(0.02, 0.05),
        per_channel=0.2
    ),
]),

# Invert each image’s channel with 5% probability.
# This sets each pixel value v to 255-v.
iaa.Invert(0.05, per_channel=True),  # invert color channels

# Add a value of -10 to 10 to each pixel.
iaa.Add((-10, 10), per_channel=0.5),

# Change brightness of images (50-150% of original value).
iaa.Multiply((0.5, 1.5), per_channel=0.5),

# Improve or worsen the contrast of images.
iaa.ContrastNormalization((0.5, 2.0), per_channel=0.5),

# Convert each image to grayscale and then overlay the
# result with the original with random alpha. I.e. remove
# colors with varying strengths.
iaa.Grayscale(alpha=(0.0, 1.0)),

# In some images move pixels locally around (with random
# strengths).
sometimes(
    iaa.ElasticTransformation(alpha=(0.5, 3.5), sigma=0.25)
),

# In some images distort local areas with varying strength.
sometimes(iaa.PiecewiseAffine(scale=(0.01, 0.05)))
(continues on next page)
```python
images_aug = seq(images=images)
```

Fig. 2: Example results of the above heavy augmentation sequence.
Examples: Keypoints

`imgaug` can handle not only images, but also keypoints/landmarks on these. E.g. if an image is rotated during augmentation, the library can also rotate all landmarks correspondingly.

### 3.1 Notebook

A jupyter notebook for keypoint augmentation is available at Jupyter Notebooks. The notebooks are usually more up to date and contain more examples than the ReadTheDocs documentation.

### 3.2 A simple example

The following example loads an image and places four keypoints on it. The image is then augmented to be brighter, slightly rotated and scaled. These augmentations are also applied to the keypoints. The image is then shown before and after augmentation (with keypoints drawn on it).

```python
import imgaug as ia
import imgaug.augmenters as iaa
from imgaug.augmentables import Keypoint, KeypointsOnImage

ia.seed(1)

image = ia.quokka(size=(256, 256))
kps = KeypointsOnImage(
    [keypoint(x=65, y=100),
     keypoint(x=75, y=200),
     keypoint(x=100, y=100),
     keypoint(x=200, y=80)
    ], shape=image.shape
)

seq = iaa.Sequential({
```

(continues on next page)
```python
iaa.Multiply((1.2, 1.5)), # change brightness, doesn't affect keypoints
iaa.Affine(
    rotate=10,
    scale=(0.5, 0.7)
) # rotate by exactly 10deg and scale to 50-70%, affects keypoints

# Augment keypoints and images.
image_aug, kps_aug = seq(image=image, keypoints=kps)

# print coordinates before/after augmentation (see below)
# use after.x_int and after.y_int to get rounded integer coordinates
for i in range(len(kps.keypoints)):
    before = kps.keypoints[i]
    after = kps_aug.keypoints[i]
    print("Keypoint %d: (%.8f, %.8f) -> (%.8f, %.8f)" % (i, before.x, before.y, after.x, after.y))

# image with keypoints before/after augmentation (shown below)
image_before = kps.draw_on_image(image, size=7)
image_after = kps_aug.draw_on_image(image_aug, size=7)
```

Console output of the example:

| Keypoint 0: (65.00000000, 100.00000000) -> (97.86113503, 107.69632182) |
| Keypoint 1: (75.00000000, 200.00000000) -> (93.93710117, 160.01366917) |
| Keypoint 2: (100.00000000, 100.00000000) -> (115.85492750, 110.86911292) |
| Keypoint 3: (200.00000000, 80.00000000) -> (169.07878659, 109.65206321) |

Fig. 1: Image with keypoints, before (left) and after (right) augmentation. Keypoints are shown in green and drawn in after the augmentation process.
Examples: Bounding Boxes

*imgaug* offers support for bounding boxes (aka rectangles, regions of interest). E.g. if an image is rotated during augmentation, the library can also rotate all bounding boxes on it correspondingly.

Features of the library’s bounding box support:

- Represent bounding boxes as objects (*imgaug.augmentables.bbs.BoundingBox*).
- Augment bounding boxes.
- Draw bounding boxes on images.
- Move/shift bounding boxes on images, project them onto other images (e.g. onto the same image after resizing), compute their intersections/intersections and IoU values.

### 4.1 Notebook

A jupyter notebook for bounding box augmentation is available at [Jupyter Notebooks](#). The notebooks are usually more up to date and contain more examples than the ReadTheDocs documentation.

### 4.2 A simple example

The following example loads an image and places two bounding boxes on it. The image is then augmented to be brighter, slightly rotated and scaled. These augmentations are also applied to the bounding boxes. The image is then shown before and after augmentation (with bounding boxes drawn on it).

```python
import imgaug as ia
import imgaug.augmenters as iaa
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage

ia.seed(1)
```
image = ia.quokka(size=(256, 256))
bbs = BoundingBoxesOnImage([
    BoundingBox(x1=65, y1=100, x2=200, y2=150),
    BoundingBox(x1=150, y1=80, x2=200, y2=130)
], shape=image.shape)

seq = iaa.Sequential([
    iaa.Multiply((1.2, 1.5)),  # change brightness, doesn't affect BBs
    iaa.Affine(
        translate_px={"x": 40, "y": 60},
        scale=(0.5, 0.7)
    )  # translate by 40/60px on x/y axis, and scale to 50-70%, affects BBs
])

# Augment BBs and images.
image_aug, bbs_aug = seq(image=image, bounding_boxes=bbs)

# print coordinates before/after augmentation (see below)
# use .x_int, .y_int, ... to get integer coordinates
for i in range(len(bbs.bounding_boxes)):
    before = bbs.bounding_boxes[i]
    after = bbs_aug.bounding_boxes[i]
    print("BB %d: (%.4f, %.4f, %.4f, %.4f) -> (%.4f, %.4f, %.4f, %.4f)" % (
        i,
        before.x1, before.y1, before.x2, before.y2,
        after.x1, after.y1, after.x2, after.y2)
    )

image_before = bbs.draw_on_image(image, size=2)
image_after = bbs_aug.draw_on_image(image_aug, size=2, color=[0, 0, 255])

Console output of the example:

BB 0: (65.0000, 100.0000, 200.0000, 150.0000) -> (130.7524, 171.3311, 210.1272, 200.7524)
BB 1: (150.0000, 80.0000, 200.0000, 130.0000) -> (180.7291, 159.5718, 210.1272, 188.9699)

Note that the bounding box augmentation works by augmenting each box’s edge coordinates and then drawing a bounding box around these augmented coordinates. Each of these new bounding boxes is therefore axis-aligned. This can sometimes lead to oversized new bounding boxes, especially in the case of rotation. The following image shows the result of the same code as in the example above, but Affine was replaced by Affine(rotate=45):

### 4.3 Dealing with bounding boxes outside of the image

When augmenting images and their respective bounding boxes, the boxes can end up fully or partially outside of the image plane. By default, the library still returns these boxes, even though that may not be desired. The following example shows how to (a) remove bounding boxes that are fully/partially outside of the image and (b) how to clip bounding boxes that are partially outside of the image so that they are fully inside.

```python
import numpy as np
import imgaug as ia
```
Fig. 1: Image with bounding boxes, before (left) and after (right) augmentation. Bounding boxes are shown in green (before augmentation) and blue (after augmentation).

Fig. 2: Image with bounding boxes, before (left) and after (right) augmentation. The image was augmentated by rotating it by 45 degrees. The axis-aligned bounding box around the augmented keypoints ends up being oversized.

4.3. Dealing with bounding boxes outside of the image
import imgaug.augmenters as iaa
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage

ia.seed(1)
GREEN = [0, 255, 0]
ORANGE = [255, 140, 0]
RED = [255, 0, 0]

# Pad image with a 1px white and (BY-1)px black border
def pad(image, by):
    image_border1 = ia.pad(image, top=1, right=1, bottom=1, left=1,
                           mode="constant", cval=255)
    image_border2 = ia.pad(image_border1, top=by-1, right=by-1,
                           bottom=by-1, left=by-1,
                           mode="constant", cval=0)
    return image_border2

# Draw BBs on an image
# and before doing that, extend the image plane by BORDER pixels.
# Mark BBs inside the image plane with green color, those partially inside
# with orange and those fully outside with red.
def draw_bbs(image, bbs, border):
    image_border = pad(image, border)
    for bb in bbs.bounding_boxes:
        if bb.is_fully_within_image(image.shape):
            color = GREEN
        elif bb.is_partly_within_image(image.shape):
            color = ORANGE
        else:
            color = RED
        image_border = bb.shift(left=border, top=border)
        .draw_on_image(image_border, size=2, color=color)
    return image_border

# Define example image with three small square BBs next to each other.
# Augment these BBs by shifting them to the right.
image = ia.quokka(size=(256, 256))
bbs = BoundingBoxesOnImage(
    [BoundingBox(x1=25, x2=75, y1=25, y2=75),
     BoundingBox(x1=100, x2=150, y1=25, y2=75),
     BoundingBox(x1=175, x2=225, y1=25, y2=75)], shape=image.shape)

seq = iaa.Affine(translate_px={"x": 120})
image_aug, bbs_aug = seq(image=image, bounding_boxes=bbs)

# Draw the BBs (a) in their original form, (b) after augmentation,
# (c) after augmentation and removing those fully outside the image,
# (d) after augmentation and removing those fully outside the image and
# clipping those partially inside the image so that they are fully inside.
image_before = draw_bbs(image, bbs, 100)
image_after1 = draw_bbs(image_aug, bbs_aug, 100)
image_after2 = draw_bbs(image_aug, bbs_aug.remove_out_of_image(), 100)
image_after3 = draw_bbs(image_aug, bbs_aug.remove_out_of_image().clip_out_of_image(), 100)
4.4 Shifting/Moving Bounding Boxes

The function `shift(top=<int>, right=<int>, bottom=<int>, left=<int>)` can be used to change the x/y position of all or specific bounding boxes.

```python
import imgaug as ia
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage

ia.seed(1)

# Define image and two bounding boxes
image = ia.quokka(size=(256, 256))
bbs = BoundingBoxesOnImage([
    BoundingBox(x1=25, x2=75, y1=25, y2=75),
    BoundingBox(x1=100, x2=150, y1=25, y2=75)
], shape=image.shape)

# Move both BBs 25px to the right and the second BB 25px down
bbs_shifted = bbs.shift(left=25)

# Draw images before/after moving BBs
image = bbs.draw_on_image(image, color=[0, 255, 0], size=2, alpha=0.75)
image = bbs_shifted.draw_on_image(image, color=[0, 0, 255], size=2, alpha=0.75)
```

4.5 Projection of BBs Onto Rescaled Images

Bounding boxes can easily be projected onto rescaled versions of the same image using the function `.on(image)`. This changes the coordinates of the bounding boxes. E.g. if the top left coordinate of the bounding box was before at x=10% and y=15%, it will still be at x/y 10%/15% on the new image, though the absolute pixel values will change depending on the height/width of the new image.

```python
import imgaug as ia
from imgaug.augmentables.bbs import BoundingBox, BoundingBoxesOnImage

ia.seed(1)

# Define image with two bounding boxes
image = ia.quokka(size=(256, 256))
bbs = BoundingBoxesOnImage([
    BoundingBox(x1=25, x2=75, y1=25, y2=75),
    BoundingBox(x1=100, x2=150, y1=25, y2=75)
], shape=image.shape)

# Rescale image and bounding boxes
image_rescaled = ia.imresize_single_image(image, (512, 512))
bbs_rescaled = bbs.on(image_rescaled)
```
Fig. 3: Results of the above example code. Top left: Original/unaugmented image with bounding boxes (here visualized with an additional black border around the image). Right, top: Image after augmentation (translation 120px to the right). One bounding box is now fully outside of the image area (red), one is partially outside of it (orange). Right, middle: After using `remove_out_of_image()` the BB that was fully outside of the image area was removed. Right, center: After using `remove_out_of_image()` and `clip_out_of_image()`, one BB was removed and the one partially outside of the image area was clipped to be fully inside it.
4.6 Computing Intersections, Unions and IoUs

Computing intersections, unions and especially IoU values (intersection over union) is common for many machine learning experiments. The library offers easy functions for that.

```python
import numpy as np
import imgaug as ia
from imgaug.augmentables.bbs import BoundingBox

ia.seed(1)

# Define image with two bounding boxes.
image = ia.quokka(size=(256, 256))
bb1 = BoundingBox(x1=50, x2=100, y1=25, y2=75)
bb2 = BoundingBox(x1=75, x2=125, y1=50, y2=100)

# Compute intersection, union and IoU value
# Intersection and union are both bounding boxes. They are here decreased/increased in size purely for better visualization.
bb_inters = bb1.intersection(bb2).extend(all_sides=-1)
```

(continues on next page)
Fig. 5: Using `on()` to project bounding boxes from one image to the other, here onto an image of 2x the original size. New coordinates are determined based on their relative positions on the old image.

```python
bb_union = bb1.union(bb2).extend(all_sides=2)
iou = bb1.iou(bb2)

# Draw bounding boxes, intersection, union and IoU value on image.
image_bbs = np.copy(image)
image_bbs = bb1.draw_on_image(image_bbs, size=2, color=[0, 255, 0])
image_bbs = bb2.draw_on_image(image_bbs, size=2, color=[0, 255, 0])
image_bbs = bb_inters.draw_on_image(image_bbs, size=2, color=[255, 0, 0])
image_bbs = bb_union.draw_on_image(image_bbs, size=2, color=[0, 0, 255])
image_bbs = ia.draw_text(
    image_bbs, text="IoU=%.2f" % (iou,),
    x=bb_union.x2+10, y=bb_union.y1+bb_union.height//2,
    color=[255, 255, 255], size=13
)```

(continued from previous page)
Fig. 6: Two bounding boxes on an image (green), their intersection (red, slightly shrunk), their union (blue, slightly extended) and their IoU value (white).
Examples: Heatmaps

\textit{imgaug} offers support for heatmap-like data. This can be used e.g. for depth map or keypoint/landmark localization maps. Heatmaps can be augmented correspondingly to images, e.g. if an image is rotated by 45\degree, the corresponding heatmap for that image will also be rotated by 45\degree.

Note:

- Heatmap support is currently in a \textbf{Beta} phase. Unittests are not yet finished.
- Heatmaps have to be bounded within value ranges, e.g. 0.0 to 1.0 for keypoint localization maps or something like 0.0 to 200.0 (meters) for depth maps. Choosing arbitrarily low/high min/max values for unbounded heatmaps is not recommended as it could lead to numerical inaccuracies.
- All augmentation functions for heatmaps are implemented under the assumption of augmenting \textbf{ground truth} data. As such, heatmaps will be affected by augmentations that change the geometry of images (e.g. affine transformations, cropping, resizing), but not by other augmentations (e.g. gaussian noise, saturation changes, grayscaling, dropout, \ldots).

Features of the library’s heatmap support:

- Represent heatmaps as objects (\texttt{imgaug.augmentables.heatmaps.HeatmapsOnImage}).
- Augment heatmaps (only geometry-affecting augmentations, e.g. affine transformations, cropping, \ldots).
- Use different resolutions for heatmaps than for images (e.g. 32x32 heatmaps for 256x256 images).
- Draw heatmaps – on their own or on images (\texttt{HeatmapsOnImage.draw()}, \texttt{HeatmapsOnImage.draw_on_image()}).
- Resize, average pool or max pool heatmaps (\texttt{HeatmapsOnImage.scale()}, \texttt{HeatmapsOnImage.avg_pool()}, \texttt{HeatmapsOnImage.max_pool()}).
- Pad heatmaps by pixel amounts or to desired aspect ratios (\texttt{HeatmapsOnImage.pad()}).
5.1 Notebook

A jupyter notebook for heatmap augmentation is available at Jupyter Notebooks. The notebooks are usually more up to date and contain more examples than the ReadTheDocs documentation.

5.2 A simple example

The following example loads a standard image and a generates a corresponding heatmap. The heatmap is supposed to be a depth map, i.e. is supposed to resemble the depth of objects in the image, where higher values indicate that objects are further away. (For simplicity we just use a simple gradient as a depth map with a cross in the center, so there is no real correspondence between the image and the depth values.)

This example shows:

- Creating heatmaps via HeatmapsOnImage(heatmap_array, shape=image_shape).
- Using value ranges outside of simple 0.0 to 1.0 (here 0.0 to 50.0) by setting min_value and max_value in the HeatmapsOnImage constructor.
- Resizing heatmaps, here via HeatmapsOnImage.avg_pool(kernel_size) (i.e. average pooling).
- Augmenting heatmaps via Augmenter.__call__(), which is equivalent to Augmenter.augment().
- Drawing heatmaps as overlays over images HeatmapsOnImage.draw_on_image(image).
- Drawing heatmaps on their own via HeatmapsOnImage.draw() in jet color map or via HeatmapsOnImage.draw(cmap=None) as intensity maps.

```python
import imageio
import numpy as np
import imgaug as ia
import imgaug.augmenters as iaa
from imgaug.augmentables.heatmaps import HeatmapsOnImage

ia.seed(1)

# Load an example image (uint8, 128x128x3).
image = ia.quokka(size=(128, 128), extract="square")

# Create an example depth map (float32, 128x128).
# Here, we use a simple gradient that has low values (around 0.0)
# towards the left of the image and high values (around 50.0)
# towards the right. This is obviously a very unrealistic depth
# map, but makes the example easier.
depth = np.linspace(0, 50, 128).astype(np.float32)  # 128 values from 0.0 to 50.0
depth = np.tile(depth.reshape(1, 128), (128, 1))  # change to a horizontal gradient

depth[64-2:64+2, 16:128-16] = 0.75 * 50.0  # line from left to right
depth[16:128-16, 64-2:64+2] = 1.0 * 50.0  # line from top to bottom

# We add a cross to the center of the depth map, so that we can more easily see the
# effects of augmentations.
depth[64-2:64+2, 16:128-16] = 0.75 * 50.0  # line from left to right
depth[16:128-16, 64-2:64+2] = 1.0 * 50.0  # line from top to bottom

depth = HeatmapsOnImage(depth, shape=image.shape, min_value=0.0, max_value=50.0)
```

(continues on next page)
# To save some computation time, we want our models to perform downscaling and hence need the ground truth depth maps to be at a resolution of 64x64 instead of the 128x128 of the input image. Here, we use simple average pooling to perform the downscaling.

depth = depth.avg_pool(2)

# Define our augmentation pipeline.
seq = iaa.Sequential([iaa.Dropout([0.05, 0.2]),  # drop 5% or 20% of all pixels
                      iaa.Sharpen((0.0, 1.0)),  # sharpen the image
                      iaa.Affine(rotate=(-45, 45)),  # rotate by -45 to 45 degrees (affects heatmaps)
                      iaa.ElasticTransformation(alpha=50, sigma=5)  # apply water effect (affects heatmaps)
], random_order=True)

# Augment images and heatmaps.
images_aug = []
heatmaps_aug = []
for _ in range(5):
    images_aug_i, heatmaps_aug_i = seq(image=image, heatmaps=depth)
    images_aug.append(images_aug_i)
    heatmaps_aug.append(heatmaps_aug_i)

# We want to generate an image of original input images and heatmaps before/after augmentation. It is supposed to have five columns: (1) original image, (2) augmented image, (3) augmented heatmap on top of augmented image, (4) augmented heatmap on its own in jet color map, (5) augmented heatmap on its own in intensity colormap. We now generate the cells of these columns.
# Note that we add a [0] after each heatmap draw command. That’s because the heatmaps object can contain many sub-heatmaps and hence we draw command returns a list of drawn sub-heatmaps. We only used one sub-heatmap, so our lists always have one entry.
cells = []
for image_aug, heatmap_aug in zip(images_aug, heatmaps_aug):
    cells.append(image)  # column 1
    cells.append(image_aug)  # column 2
    cells.append(heatmap_aug.draw_on_image(image_aug)[0])  # column 3
    cells.append(heatmap_aug.draw(size=image_aug.shape[:2])[0])  # column 4
    cells.append(heatmap_aug.draw(size=image_aug.shape[:2], cmap=None)[0])  # column 5

# Convert cells to grid image and save.
grid_image = ia.draw_grid(cells, cols=5)
imageio.imwrite("example_heatmaps.jpg", grid_image)

5.3 Multiple sub-heatmaps per heatmaps object

The above example augmented a single heatmap with shape \((H, W)\) for the example image. If you want to augment more heatmaps per image, you can simply extend the heatmap array’s shape to \((H, W, C)\), where \(C\) is the number of heatmaps. The following example instantiates one heatmap object containing three sub-heatmaps and draws them onto the image. Heatmap augmentation would be done in the exactly same way as in the previous example.
Fig. 1: Results of the above example code. Columns show: (1) Original image, (2) augmented image, (3) augmented heatmap overlayed with augmented image, (4) augmented heatmap alone in jet color map, (5) augmented heatmap alone as intensity map.
import imageio
import numpy as np
import imgaug as ia
from imgaug.augmentables.heatmaps import HeatmapsOnImage

# Load an image and generate a heatmap array with three sub-heatmaps.
# Each sub-heatmap contains just three horizontal lines, with one of them
# having a higher value (1.0) than the other two (0.2).
image = ia.quokka(size=(128, 128), extract="square")
heatmap = np.zeros((128, 128, 3), dtype=np.float32)
for i in range(3):
    heatmap[1*30-5:1*30+5, 10:-10, i] = 1.0 if i == 0 else 0.5
    heatmap[2*30-5:2*30+5, 10:-10, i] = 1.0 if i == 1 else 0.5
    heatmap[3*30-5:3*30+5, 10:-10, i] = 1.0 if i == 2 else 0.5
heatmap = HeatmapsOnImage(heatmap, shape=image.shape)

# Draw image and the three sub-heatmaps on it.
# We draw four columns: (1) image, (2-4) heatmaps one to three drawn on
# top of the image.
subheatmaps_drawn = heatmap.draw_on_image(image)
cells = [image, subheatmaps_drawn[0], subheatmaps_drawn[1],
            subheatmaps_drawn[2]]
grid_image = np.hstack(cells)  # Horizontally stack the images
imageio.imwrite("example_multiple_heatmaps.jpg", grid_image)

Fig. 2: Results of the above example code. It shows the original image with three heatmaps. The three heatmaps were combined in one HeatmapsOnImage object.

5.4 Accessing the heatmap array

After augmentation you probably want to access the heatmap’s numpy array. This is done using the function HeatmapsOnImage.get_arr(). That functions output shape will match your original heatmap array’s shape, i.e. either \((H, W)\) or \((H, W, C)\). The below code shows an example, where that function’s result is changed and then used to instantiate a new HeatmapsOnImage object.

Alternatively you could also change the heatmap object’s internal array, saved as HeatmapsOnImage.arr_0to1. As the name indicates, it is always normalized to the range 0.0 to 1.0, while get_arr() reverses that normalization. It has also always shape \((H, W, C)\), with \(C\geq 1\).
import imgaug as ia
from imgaug.augmentables.heatmaps import HeatmapsOnImage

# Load an image and generate a heatmap array containing one horizontal line.
image = ia.quokka(size=(128, 128), extract="square")
heatmap = np.zeros((128, 128, 1), dtype=np.float32)
heatmap[64-4:64+4, 10:-10, 0] = 1.0
heatmap1 = HeatmapsOnImage(heatmap, shape=image.shape)

# Extract the heatmap array from the heatmap object, change it and create a second heatmap.
arr = heatmap1.get_arr()
arr[10:-10, 64-4:64+4] = 0.5
heatmap2 = HeatmapsOnImage(arr, shape=image.shape)

# Draw image and heatmaps before/after changing the array.
# We draw three columns: (1) original image, (2) heatmap drawn on image, (3) heatmap drawn on image with some changes made to the heatmap array.
cells = [image, heatmap1.draw_on_image(image)[0], heatmap2.draw_on_image(image)[0]]
grid_image = np.hstack(cells)  # Horizontally stack the images
imageio.imwrite("example_heatmaps_arr.jpg", grid_image)

Fig. 3: Results of the above example code. It shows the original image, a corresponding heatmap and again the same heatmap after its array was read out and changed.

5.5 Resizing heatmaps

When working with heatmaps it is common that the size of the input images and the heatmap sizes don’t match or are supposed to not match (e.g. because predicted network output are of low resolution). HeatmapsOnImage offers several functions to deal with such situations: HeatmapsOnImage.avg_pool(kernel_size) applies average pooling to images, HeatmapsOnImage.max_pool(kernel_size) analogously max pooling and HeatmapsOnImage.resize(size, [interpolation]) performs resizing. For the pooling functions the kernel size is expected to be a single integer or a tuple of two/three entries (size along each dimension). For resize, the size is expected to be a (height, width) tuple and interpolation can be one of the strings nearest (nearest neighbour interpolation), linear, cubic (default) or area.

The below code shows an example. It instantiates a simple 128x128 heatmap with two horizontal lines (one of which is blurred) and a small square in the center. It then applies average pooling, max pooling and resizing to heatmap sizes 64x64, 32x32 and 16x16. Then, an output image is generated with six rows: The first three show the results
of average/max pooling and resizing, while the rows three to six show the same results after again resizing them to 128x128 using nearest neighbour upscaling.

```python
import imageio
import numpy as np
import imgaug as ia
from imgaug.augmentables.heatmaps import HeatmapsOnImage

def pad_by(image, amount):
    return ia.pad(image,
    top=amount, right=amount, bottom=amount, left=amount)

def draw_heatmaps(heatmaps, upscale=False):
    drawn = []
    for heatmap in heatmaps:
        if upscale:
            drawn.append(
                heatmap.resize((128, 128), interpolation="nearest"
                .draw())[0]
            )
        else:
            size = heatmap.get_arr().shape[0]
            pad_amount = (128-size)//2
            drawn.append(pad_by(heatmap.draw()[0], pad_amount))

    return drawn

# Generate an example heatmap with two horizontal lines (first one blurry, # second not) and a small square.
heatmap = np.zeros((128, 128, 1), dtype=np.float32)
heatmap[32-4:32+4, 10:-10, 0] = 1.0
heatmap = iaa.GaussianBlur(3.0).augment_image(heatmap)
heatmap[96-4:96+4, 10:-10, 0] = 1.0
heatmap[64-2:64+2, 64-2:64+2, 0] = 1.0
heatmap = HeatmapsOnImage(heatmap, shape=(128, 128, 1))

# Scale the heatmaps using average pooling, max pooling and resizing with # default interpolation (cubic).
avg_pooled = [heatmap, heatmap.avg_pool(2), heatmap.avg_pool(4), heatmap.avg_pool(8)]
max_pooled = [heatmap, heatmap.max_pool(2), heatmap.max_pool(4), heatmap.max_pool(8)]
resized = [heatmap, heatmap.resize((64, 64)), heatmap.resize((32, 32)), heatmap.resize((16, 16))]

# Draw an image of all scaled heatmaps.
cells = draw_heatmaps(avg_pooled)\
    + draw_heatmaps(max_pooled)\
    + draw_heatmaps(resized)\
    + draw_heatmaps(avg_pooled, upscale=True)\
    + draw_heatmaps(max_pooled, upscale=True)\
    + draw_heatmaps(resized, upscale=True)
grid_image = ia.draw_grid(cells, cols=4)
imageio.imwrite("example_heatmaps_scaling.jpg", grid_image)
```

5.5. Resizing heatmaps 29
Fig. 4: Results of the above example code. It shows six rows: (Rows 1-3) scaling via average pooling, max pooling and cubic resizing to 64x64 (column 2), 32x32 (column 3) and 16x16 (column 4) and then zero-padding to 128x128. (Rows 4-6) Doing the same again, but not padding to 128x128 but instead resizing using nearest neighbour upscaling.
5.6 Padding heatmaps

Another common operation is padding of images and heatmaps, especially to squared sizes. This is done for images using `imgaug.pad(image, [top], [right], [bottom], [left], [mode], [cval])` and `imgaug.pad_to_aspect_ratio(image, aspect_ratio, [mode], [cval], [return_pad_amounts])`. For heatmaps it is done using `HeatmapsOnImage.pad([top], [right], [bottom], [left], [mode], [cval])` and `HeatmapsOnImage.pad_to_aspect_ratio(aspect_ratio, [mode], [cval], [return_pad_amounts])`. In both cases, `pad()` expects pixel amounts (i.e. integers) and `pad_to_aspect_ratio()` the target aspect ratio, given as a float denoting `width/height` (i.e. a value of `1.0` would lead to a squared image/heatmap, while `2.0` would lead to a fairly wide image/heatmap).

The below code shows an example for padding. It starts with a squared sized image and heatmap, cuts both so that they are more wide than high and then zero-pads both back to squared size.

```python
import imageio
import numpy as np
import imgaug as ia
from imgaug.augmentables.heatmaps import HeatmapsOnImage

# Load example image and generate example heatmap with one horizontal line
image = ia.quokka((128, 128), extract="square")
heatmap = np.zeros((128, 128, 1), dtype=np.float32)
heatmap[64-4:64+4, 10:-10, 0] = 1.0

# Cut image and heatmap so that they are no longer squared
image = image[32:-32, :, :]
heatmap = heatmap[32:-32, :, :]

# Pad images and heatmaps by pixel amounts or to aspect ratios
# We pad both back to squared size of 128x128
images_padded = [
    ia.pad(image, top=32, bottom=32),
    ia.pad_to_aspect_ratio(image, 1.0)
]
heatmaps_padded = [
    heatmap.pad(top=32, bottom=32),
    heatmap.pad_to_aspect_ratio(1.0)
]

# Draw an image of all padded images and heatmaps
cells = [
    images_padded[0],
    heatmaps_padded[0].draw_on_image(images_padded[0])[0],
    images_padded[1],
    heatmaps_padded[1].draw_on_image(images_padded[1])[0]
]

grid_image = ia.draw_grid(cells, cols=2)
imageio.imwrite("example_heatmaps_padding.jpg", grid_image)
```
Fig. 5: Results of the above example code. It shows an input image and a heatmap that were both first cut to 64x128 and then padded back to squared size of 128x128. First row uses \texttt{pad()}, second uses \texttt{pad_to_aspect_ratio()}. 

Examples: Segmentation Maps and Masks

*imgaug* offers support for segmentation map data, such as semantic segmentation maps, instance segmentation maps or ordinary masks. Segmentation maps can be augmented correspondingly to images. E.g. if an image is rotated by 45°, the corresponding segmentation map for that image will also be rotated by 45°.

**Note:**

- Segmentation map support is currently in a Beta phase. Unit tests are not yet finished.
- All augmentation functions for segmentation map are implemented under the assumption of augmenting **ground truth** data. As such, heatmaps will be affected by augmentations that change the geometry of images (e.g. affine transformations, cropping, resizing), but not by other augmentations (e.g. gaussian noise, saturation changes, grayscaling, dropout, ...).

Features of the library’s segmentation map support:

- Represent segmentation maps as objects (*imgaug.augmentables.segmaps.SegmentationMapOnImage*).
- Support integer maps (integer dtypes, usually int32), boolean masks (*np.bool_* and float32 maps (these are essentially identical to heatmaps).
- Augment segmentation maps (only geometry-affecting augmentations, e.g. affine transformations, cropping, ...).
- Use different resolutions for segmentation maps than for images (e.g. 32x32 segmentation maps for 256x256 images).
- Draw segmentation maps – on their own or on images (*SegmentationMapOnImage.draw()*, *SegmentationMapOnImage.draw_on_image()*).
- Resize segmentation maps (*SegmentationMapOnImage.scale()*).
- Pad segmentation maps by pixel amounts or to desired aspect ratios (*SegmentationMapOnImage.pad()*, *SegmentationMapOnImage.pad_to_aspect_ratio()*).
6.1 Notebook

A jupyter notebook for segmentation map augmentation is available at Jupyter Notebooks. The notebooks are usually more up to date and contain more examples than the ReadTheDocs documentation.

6.2 Upcoming Changes

Segmentation maps are currently implemented internally as a wrapper around heatmaps and as such use float-based arrays for their internal representation. This will be changed to integer-based arrays in 0.3.0. The arguments nb_classes, background_class_id and background_threshold will be removed or deprecated. The method get_arr_int() will be removed or deprecated. Resizing via cubic interpolation will no longer be recommended and the default resizing method will change to nearest neighbour interpolation. The class will be renamed from SegmentationMapOnImage to SegmentationMapsOnImage and get an optional channel-axis to represent multiple segmentation maps on the same image (e.g. multiple object classes with each having one instance segmentation map).

6.3 A simple example

The following example loads a standard image and generates a corresponding int32 segmentation map. The image and segmentation map are augmented in the same way and the results are visualized.

```python
import imageio
import numpy as np
import imgaug as ia
import imgaug.augmenters as iaa
from imgaug.augmentables.segmaps import SegmentationMapOnImage

ia.seed(1)

# Load an example image (uint8, 128x128x3).
image = ia.quokka(size=(128, 128), extract="square")

# Create an example segmentation map (int32, 128x128).
# Here, we just randomly place some squares on the image.
# Class 0 is the background class.
segmap = np.zeros((128, 128), dtype=np.int32)
segmap[28:71, 35:85] = 1
segmap[10:25, 30:45] = 2
segmap[10:25, 70:85] = 3
segmap[10:110, 5:10] = 4
segmap[118:123, 10:110] = 5
segmap = SegmentationMapOnImage(segmap, shape=image.shape, nb_classes=1+5)

# Define our augmentation pipeline.
seq = iaa.Sequential([
    iaa.Dropout([0.05, 0.2]),  # drop 5% or 20% of all pixels
    iaa.Sharpen((0.0, 1.0)),  # sharpen the image
    iaa.Affine(rotate=(-45, 45)),  # rotate by -45 to 45 degrees (affects segmaps)
    iaa.ElasticTransformation(alpha=50, sigma=5)  # apply water effect (affects segmaps)
], random_order=True)
```

(continues on next page)
# Augment images and segmaps.
images_aug = []
segmaps_aug = []
for _ in range(5):
    images_aug_i, segmaps_aug_i = seq(image=image, segmentation_maps=segmap)
    images_aug.append(images_aug_i)
    segmaps_aug.append(segmaps_aug_i)

# We want to generate an image of original input images and segmaps
# before/after augmentation.
# It is supposed to have five columns: (1) original image, (2) original
# image with segmap, (3) augmented image, (4) augmented
# segmap on augmented image, (5) augmented segmap on its own in.
# We now generate the cells of these columns.
cells = []
for image_aug, segmap_aug in zip(images_aug, segmaps_aug):
    cells.append(image) # column 1
    cells.append(segmap.draw_on_image(image)) # column 2
    cells.append(image_aug) # column 3
    cells.append(segmap_aug.draw_on_image(image_aug)) # column 4
    cells.append(segmap_aug.draw(size=image_aug.shape[:2])) # column 5

# Convert cells to grid image and save.
grid_image = ia.draw_grid(cells, cols=5)
imageio.imwrite("example_segmaps.jpg", grid_image)

6.4 Using boolean masks

In order to augment masks, you can simply use boolean arrays. Everything else is identical to int32 maps. The below code shows an example and is very similar to the previous code for int32 maps. It notably changes `np.zeros((128, 128), dtype=np.int32)` to `np.zeros((128, 128), dtype=bool).

```python
# Load an example image (uint8, 128x128x3).
image = ia.quokka(size=(128, 128), extract="square")

# Create an example mask (bool, 128x128).
# Here, we just randomly place a square on the image.
segmap = np.zeros((128, 128), dtype=bool)
segmap[28:71, 35:85] = True
segmap = SegmentationMapOnImage(segmap, shape=image.shape)

# Draw three columns: (1) original image,
# (2) original image with mask on top, (3) only mask
cells = [
    image,
    segmap.draw_on_image(image),
    segmap.draw(size=image.shape[:2])
]
```

6.4. Using boolean masks
Fig. 1: Results of the above example code. Columns show: (1) Original image, (2) original segmentation map drawn on original image, (3) augmented image, (4) augmented segmentation map drawn on augmented image, (5) augmented segmentation map on its own.
# Convert cells to grid image and save.
grid_image = ia.draw_grid(cells, cols=3)
imageio.imwrite("example_segmaps_bool.jpg", grid_image)

Fig. 2: Results of the above example code. Columns show: (1) Original image, (2) boolean segmentation map (i.e. mask) drawn on image, (3) boolean segmentation map drawn on its own.

## 6.5 Accessing the segmentation map array

After augmentation it is often desired to re-access the segmentation map array. This can be done using `SegmentationMapOnImage.get_arr_int([background_threshold], [background_class_id])`, which returns an int32 array. The two background parameters are related to converting float32 heatmaps to int32 arrays (the segmentation map is handled by the library as a set of heatmaps). The parameters can usually be ignored.

The below code shows an example that accesses and changes the array.

```python
import imageio
import numpy as np
import imgaug as ia
from imgaug.augmentables.segmaps import SegmentationMapOnImage

# Load an example image (uint8, 128x128x3).
image = ia.quokka(size=(128, 128), extract="square")

# Create an example segmentation map (int32, 128x128).
# Here, we just randomly place some squares on the image.
# Class 0 is the background class.
segmap = np.zeros((128, 128), dtype=np.int32)
segmap[28:71, 35:85] = 1
segmap[10:25, 30:45] = 2
segmap[10:25, 70:85] = 3
segmap[10:110, 5:10] = 4
segmap[118:123, 10:110] = 5
segmap1 = SegmentationMapOnImage(segmap, shape=image.shape, nb_classes=1+5)

# Read out the segmentation map's array, change it and create a new segmentation map
arr = segmap1.get_arr_int()
arr[10:110, 5:10] = 5
```

(continues on next page)
```python
segmap2 = ia.SegmentationMapOnImage(arr, shape=image.shape, nb_classes=1+5)

# Draw three columns: (1) original image, (2) original image with
# unaltered segmentation map on top, (3) original image with altered
# segmentation map on top
cells = [
    image,
    segmap1.draw_on_image(image),
    segmap2.draw_on_image(image)
]

# Convert cells to grid image and save.
grid_image = ia.draw_grid(cells, cols=3)
imageio.imwrite("example_segmaps_array.jpg", grid_image)
```

Fig. 3: Results of the above example code. Columns show: (1) Original image, (2) original segmentation map drawn on original image, (3) segmentation map with modified array drawn on image.

### 6.6 Resizing and padding

Segmentation maps can be easily resized and padded. The methods are identical to the ones used for heatmaps (see Examples: Heatmaps), though segmentation maps are not offering resizing via average or max pooling. Note that segmentation maps are handled internally as heatmaps (one per class) and as such can be resized using cubic interpolation. The functions for resizing and padding are:

- **SegmentationMapOnImage.scale(sizes, interpolation="cubic")**: Resizes to `sizes` given as a tuple `(height, width)`. Interpolation can be nearest, linear, cubic, area.

- **SegmentationMapOnImage.pad(top=0, right=0, bottom=0, left=0, mode="constant", cval=0.0)**: Pads the segmentation map by given pixel amounts. Uses by default constant value padding with value 0.0, i.e. zero-padding. Padding is applied to segmentation maps as heatmaps (can be imagined as probability maps), making values between 0.0 and 1.0 possible. However, only 0.0 is recommended for constant value padding. Possible padding modes are the same as for `numpy.pad()`, i.e. constant, edge, linear_ramp, maximum, mean, median, minimum, reflect symmetric wrap.

- **SegmentationMapOnImage.pad_to_aspect_ratio(aspect_ratio, mode="constant", cval=0.0, return_pad_amounts=False)**: Same as `pad()`, but pads an image towards a desired aspect ratio `(width/height)`. E.g. use 1.0 for squared images or 2.0 for images that are twice as wide as they are high.
7.1 Introduction

When augmenting images during experiments, usually one wants to augment each image in different ways. E.g. when rotating images, not every image is supposed to be rotated by 10 degrees. Instead, only some are supposed to be rotated by 10 degrees, while others should be rotated by 17 degrees or 5 degrees or -12 degrees - and so on. This can be achieved using random functions, but reimplementing these, making sure that they generate the expected values and getting them to work with determinism is cumbersome. To avoid all of this work, the library uses Stochastic Parameters. These are usually abstract representations of probability distributions, e.g. the normal distribution $N(0, 1.0)$ or the uniform range $[0.0, 10.0]$. Basically all augmenters accept these stochastic parameters, making it easy to control value ranges. They are all adapted to work with determinism out of the box.

The below code shows their usage:

```python
from imgaug import augmenters as iaa
from imgaug import parameters as iap

seq = iaa.Sequential([
    iaa.GaussianBlur(
        sigma=iap.Uniform(0.0, 1.0)
    ),
    iaa.ContrastNormalization(
        iap.Choice([1.0, 1.5, 3.0], p=[0.5, 0.3, 0.2])
    ),
    iaa.Affine(
        rotate=iap.Normal(0.0, 30),
        translate_px=iap.RandomSign(iap.Poisson(3))
    ),
    iaa.AddElementwise(
        iap.Discretize(
            (iap.Beta(0.5, 0.5) * 2 - 1.0) * 64
        )
    )
])
```

(continues on next page)
The example does the following:

- Blur each image by $\sigma$, where $\sigma$ is sampled from the uniform range $[0.0, 1.0)$. Example values: 0.053, 0.414, 0.389, 0.277, 0.981.
- Increase the contrast either to 100% (50% chance of being chosen) or by 150% (30% chance of being chosen) or 300% (20% chance of being chosen).
- Rotate each image by a random amount of degrees, where the degree is sampled from the normal distribution $N(0, 30)$. Most of the values will be in the range -60 to 60.
- Translate each image by $n$ pixels, where $n$ is sampled from a poisson distribution with alpha=3 (pick should be around $x=3$). As we can’t translate by a fraction of a pixel, we pick a discrete distribution here, which poisson is. However, we do not just want to translate towards the right/top (only positive values). So we randomly flip the sign sometimes to get negative pixel amounts too.
- Add to each pixel a random value, sampled from the beta distribution $\text{Beta}(0.5, 0.5)$. This distribution has its peaks around 0.0 and 1.0. We multiply this with 2 and subtract 1 to get it into the range $[-1, 1]$. Then we multiply by 64 to get the range $[-64, 64]$. As we beta distribution is continuous, we convert it to a discrete distribution. The result is that a lot of pixel intensities are shifted by -64 or 64 (or a value very close to these two). Some other pixel intensities are kept (mostly) at their old values.
- We use Multiply to make each image brighter. The brightness increase is sampled from a normal distribution, converted to have only positive values. So most values are expected to be in the range 0.0 to 0.2. We add 1.0 to set the brightness to 1.0 (100%) to 1.2 (120%).

### 7.2 Continuous Probability Distributions

The following continuous probability distributions are available:

- **Normal(loc, scale):** The popular normal distribution with mean $\text{loc}$ and standard deviation $\text{scale}$. Example:

```python
from imgaug import parameters as iap
params = [
    iap.Normal(0, 1),
    iap.Normal(5, 3),
    iap.Normal(iap.Choice([-3, 3]), 1),
    iap.Normal(iap.Uniform(-3, 3), 1)
]
iap.show_distributions_grid(params)
```

- **Laplace(loc, scale):** Similarly shaped to a normal distribution. Has its peak at $\text{loc}$ and width $\text{scale}$. Example:

```python
from imgaug import parameters as iap
params = [
    iap.Laplace(0, 1),
    iap.Laplace(5, 3),
    iap.Laplace(iap.Choice([-3, 3]), 1),
    iap.Laplace(iap.Uniform(-3, 3), 1)
]
```
• **ChiSquare(df):** The chi-square ("X^2") distribution with df degrees of freedom. Roughly similar to a continuous version of the poisson distribution. Has its peak at df and no negative values, only positive ones. Example:

```python
from imgaug import parameters as iap
params = [iap.ChiSquare(1),
         iap.ChiSquare(3),
         iap.ChiSquare(iap.Choice([1, 5])),
         ]
```

(continues on next page)
iap.RandomSign(iap.ChiSquare(3))
]
iap.show_distributions_grid(params)

- **Weibull(a)**: Weibull distribution with shape \( a \). Example:

```python
from imgaug import parameters as iap
params = [
    iap.Weibull(0.5),
    iap.Weibull(1),
    iap.Weibull(1.5),
    iap.Weibull((0.5, 1.5))
]
```
• *Uniform*(\(a, b\)): Uniform distribution in the range \([a, b)\). Example:

```python
from imgaug import parameters as iap
params = [
    iap.Uniform(0, 1),
    iap.Uniform(iap.Normal(-3, 1), iap.Normal(3, 1)),
    iap.Uniform([-1, 0], 1),
    iap.Uniform((-1, 0), 1)
]
iap.show_distributions_grid(params)
```
- Beta(alpha, beta): Beta distribution with parameters alpha and beta. Example:

```python
from imgaug import parameters as iap
params = [
    iap.Beta(0.5, 0.5),
    iap.Beta(2.0, 2.0),
    iap.Beta(1.0, 0.5),
    iap.Beta(0.5, 1.0)
]
iap.show_distributions_grid(params)
```
7.3 Discrete Probability Distributions

The following discrete probability distributions are available:

- **Binomial**(p): The common binomial distribution with probability p. Useful to simulate coinflips. Example:

```python
from imgaug import parameters as iap
params = [
    iap.Binomial(0.5),
    iap.Binomial(0.9)
]
iap.show_distributions_grid(params)
```

- **DiscreteUniform**(a, b): The discrete uniform distribution in the range \([a..b]\). Example:

```python
from imgaug import parameters as iap
params = [
    iap.DiscreteUniform(0, 10),
    iap.DiscreteUniform(-10, 10),
    iap.DiscreteUniform([-10, -9, -8, -7], 10),
    iap.DiscreteUniform((-10, -7), 10)
]
iap.show_distributions_grid(params)
```

- **Poisson**(lam): Poisson distribution with shape lam. Generates no negative values. Example:

```python
from imgaug import parameters as iap
params = [
    iap.Poisson(1),
    iap.Poisson(2.5),
    iap.Poisson((1, 2.5)),
    iap.RandomSign(iap.Poisson(2.5))
]
iap.show_distributions_grid(params)
```
7.4 Arithmetic

The library supports arithmetic operations on stochastic parameters. This allows to modify values sampled from distributions or combine several distributions with each other.

- **Add(param, val, elementwise)**: Add `val` to the values sampled from `param`. The shortcut is `+`, e.g. `Uniform(...) + 1`. `val` can be a stochastic parameter itself. Usually, only one value is sampled from `val` per sampling run and added to all samples generated by `param`. Alternatively, `elementwise` can be set to `True` in order to generate as many samples from `val` as from `param` and add them elementwise. Note that `Add` merely adds to the results of `param` and does not combine probability density functions (see e.g. example image 3 and 4). Example:

```python
from imgaug import parameters as iap
params = [
    iap.Uniform(0, 1) + 1,  # identical to: Add(Uniform(0, 1), 1)
    iap.Add(iap.Uniform(0, 1), iap.Choice([0, 1], p=[0.7, 0.3])),
    iap.Normal(0, 1) + iap.Uniform(-5.5, -5) + iap.Uniform(5, 5.5),
    iap.Normal(0, 1) + iap.Uniform(-7, 5) + iap.Poisson(3),
    iap.Add(iap.Normal(-3, 1), iap.Normal(3, 1)),
    iap.Add(iap.Normal(-3, 1), iap.Normal(3, 1), elementwise=True)
]
iap.show_distributions_grid(
    params,
    rows=2,
    sample_sizes=[  # (iterations, samples per iteration)
        (1000, 1000), (1000, 1000), (1000, 1000),
        (1000, 1000), (1, 100000), (1, 100000)
    ]
)
```
Subtract(param, val, elementwise): Same as Add, but subtracts val from the results of param. The shortcut is -, e.g. Uniform(...) - 1.

Multiply(param, val, elementwise): Same as Add, but multiplies val with the results of param. The shortcut is *, e.g. Uniform(...) * 2. Example:

```python
from imgaug import parameters as iap
params = [
    iap.Uniform(0, 1) * 2,  # identical to: Multiply(Uniform(0, 1), 2)
    iap.Multiply(iap.Uniform(0, 1), iap.Choice(([0, 1], p=[0.7, 0.3])),
    (iap.Normal(0, 1) * iap.Uniform(-5.5, -5)) * iap.Uniform(5, 5.5),
    (iap.Normal(0, 1) * iap.Uniform(-7, 5)) * iap.Poisson(3),
    iap.Multiply(iap.Normal(-3, 1), iap.Normal(3, 1)),
    iap.Multiply(iap.Normal(-3, 1), iap.Normal(3, 1), elementwise=True)
]
iap.show_distributions_grid(
    params,
    rows=2,
    sample_sizes=[  # (iterations, samples per iteration)
        (1000, 1000), (1000, 1000), (1000, 1000),
        (1000, 1000), (1, 100000), (1, 100000)
    ]
)
```

Divide(param, val, elementwise): Same as Multiply, but divides by val. The shortcut is /, e.g. Uniform(...) / 2. Division by zero is automatically prevented (zeros are replaced by ones). Example:

```python
from imgaug import parameters as iap
```

(continues on next page)
params = [
    iap.Uniform(0, 1) / 2,  # identical to: Divide(Uniform(0, 1), 2)
    iap.Divide(iap.Uniform(0, 1), iap.Choice([0, 2], p=[0.7, 0.3])),
    (iap.Normal(0, 1) / iap.Uniform(-5.5, -5)) / iap.Uniform(5, 5.5),
    (iap.Normal(0, 1) * iap.Uniform(-7, 5)) / iap.Poisson(3),
    iap.Divide(iap.Normal(-3, 1), iap.Normal(3, 1)),
    iap.Divide(iap.Normal(-3, 1), iap.Normal(3, 1), elementwise=True)
]

iap.show_distributions_grid(params, rows=2, sample_sizes=[
    # (iterations, samples per iteration)
    (1000, 1000), (1000, 1000), (1000, 1000),
    (1000, 1000), (1, 100000), (1, 100000)
])

• **Power(param, val, elementwise):** Same as Add, but raises sampled values to the exponent val. The shortcut is `**`. Example:

```python
from imgaug import parameters as iap
params = [
    iap.Uniform(0, 1) ** 2,  # identical to: Power(Uniform(0, 1), 2)
    iap.Clip(iap.Uniform(-1, 1) ** iap.Normal(0, 1), -4, 4)
]

iap.show_distributions_grid(params, rows=1)
```
7.5 Special Parameters

- **Deterministic(v)**: A constant. Upon sampling, this always returns v.

- **Choice(values, replace=True, p=None)**: Upon sampling, this parameter picks randomly elements from a list `values`. If `replace` is set to `True` (default), the picking happens with replacement. By default, all elements have the same probability of being picked. This can be modified using `p`. Note that `values` may also contain strings and other stochastic parameters. In the latter case, each picked parameter will be replaced by a sample from that parameter. This allows merging of probability mass functions, but is a rather slow process. All elements in `values` should have the same datatype (except for stochastic parameters). Example:

```python
from imgaug import parameters as iap
params = [
    iap.Choice([0, 1, 2]),
    iap.Choice([0, 1, 2], p=[0.15, 0.5, 0.35]),
    iap.Choice([iap.Normal(-3, 1), iap.Normal(3, 1)]),
    iap.Choice([iap.Normal(-3, 1), iap.Poisson(3)])
]
iap.show_distributions_grid(params)
```

- **Clip(param, minval=None, maxval=None)**: Clips the values sampled from `param` to the range `[minval, maxval]`. `minval` and `maxval` may be `None`. In that case, only minimum or maximum clipping is applied (depending on what is `None`). Example:

```python
from imgaug import parameters as iap
params = [
    iap.Clip(iap.Normal(0, 1), -2, 2),
    iap.Clip(iap.Normal(0, 1), -2, None)
]
iap.show_distributions_grid(params, rows=1)
```

- **Discretize(param)**: Converts a continuous parameter `param` into a discrete one (using rounding). Discrete parameters are not changed. Example:
from imgaug import parameters as iap
params = [
    iap.Discretize(iap.Normal(0, 1)),
    iap.Discretize(iap.ChiSquare(3))
]  
iap.show_distributions_grid(params, rows=1)

- **Absolute(param)**: Applies an absolute function to each value sampled from `param`, turning them to positive ones. Example:

```python
from imgaug import parameters as iap
```
params = [
    iap.Absolute(iap.Normal(0, 1)),
    iap.Absolute(iap.Laplace(0, 1))
]  
iap.show_distributions_grid(params, rows=1)

• RandomSign(param, p_positive=0.5): Randomly flips the signs of values sampled from param. Optionally, the probability of flipping a value’s sign towards positive can be set. Example:

```python
from imgaug import parameters as iap
params = [
    iap.ChiSquare(3),
    iap.RandomSign(iap.ChiSquare(3)),
    iap.RandomSign(iap.ChiSquare(3), p_positive=0.75),
    iap.RandomSign(iap.ChiSquare(3), p_positive=0.9)
]  
iap.show_distributions_grid(params)
```

• ForceSign(param, positive, mode="invert", reroll_count_max=2): Converts all values sampled from param to positive or negative ones. Signs of positive/negative values may simply be flipped (mode="invert") or resampled from param (mode="reroll"). When rerolling, the number of iterations is limited to reroll_count_max (afterwards mode="invert" is used). Example:

```python
from imgaug import parameters as iap
params = [
    iap.ForceSign(iap.Normal(0, 1), positive=True),
    iap.ChiSquare(3) - 3.0,
    iap.ForceSign(iap.ChiSquare(3) - 3.0, positive=True, mode="invert"),
    iap.ForceSign(iap.ChiSquare(3) - 3.0, positive=True, mode="reroll")
]  
iap.show_distributions_grid(params)
```

• Positive(other_param, mode="invert", reroll_count_max=2): Shortcut for ForceSign with positive=True. E.g. Positive(Normal(0, 1)) restricts a normal distribution to only positive values.

7.5. Special Parameters
• \texttt{Negative(other\_param, mode="invert", reroll\_count\_max=2)}: Shortcut for \texttt{ForceSign} with \texttt{positive=False}. E.g. \texttt{Negative(Normal(0, 1))} restricts a normal distribution to only negative values.

• \texttt{FromLowerResolution(other\_param, size\_percent=None, size\_px=None, method="nearest", min\_size=1)}: Intended for 2d-sampling processes, e.g. for masks. Samples these in a lower resolution space. E.g. instead of sampling a mask at 100x100, this allows to sample it at 10x10 and then upsample to 100x100. One advantage is, that this can be faster. Another possible use is, that the upsampling may result in large, correlated blobs (linear interpolation) or rectangles (nearest neighbour interpolation).

### 7.6 Noise Parameters

TODO
8.1 Introduction

Most augmenters in the library affect images in uniform ways per image. Sometimes one might not want that and instead desires more localized effects (e.g. change the color of some image regions, while keeping the others unchanged) or wants to keep a fraction of the old image (e.g. blur the image and mix in a bit of the unblurred image). Alpha-based augmenters are intended for these use cases. They either mix two images using a constant alpha factor or using a pixel-wise mask. Below image shows examples.

```python
# First row
iaa.Alpha(
    (0.0, 1.0),
    first=iaa.MedianBlur(11),
    per_channel=True
)

# Second row
iaa.SimplexNoiseAlpha(
    first=iaa.EdgeDetect(1.0),
    per_channel=False
)

# Third row
iaa.SimplexNoiseAlpha(
    first=iaa.EdgeDetect(1.0),
    second=iaa.ContrastNormalization((0.5, 2.0)),
    per_channel=0.5
)

# Forth row
iaa.FrequencyNoiseAlpha(
    first=iaa.Affine(
        rotate=(-10, 10),
        translate_px={"x": (-4, 4), "y": (-4, 4)}
    )
)
```
8.2 Constant Alpha

The augmenter *Alpha* allows to mix the results of two image sources using an alpha factor that is constant throughout the whole image, i.e. it follows roughly $I_{\text{blend}} = \alpha I_a + (1 - \alpha) I_b$ per image, where $I_a$ is the image from the first image source and $I_b$ is the image from the second image source. Often, the first source will be an augmented version of the image and the second source will be the original image, leading to a blend of augmented and unaugmented image. The second image source can also be an augmented version of the image, leading to a blend of two distinct augmentation effects. Alpha is already built into some augmenters as a parameter, e.g. into *EdgeDetect*.

The below example code generates images that are a blend between *Sharpen* and *CoarseDropout*. Notice how the sharpening does not affect the black rectangles from dropout, as the two augmenters are both applied to the original images and merely blended.

```python
import imgaug as ia
from imgaug import augmenters as iaa
ia.seed(1)

# Example batch of images.
# The array has shape (8, 128, 128, 3) and dtype uint8.
images = np.array([ia.quokka(size=(128, 128)) for _ in range(8)], dtype=np.uint8)

seq = iaa.Alpha(
    factor=(0.2, 0.8),
    first=iaa.Sharpen(1.0, lightness=2),
    second=iaa.CoarseDropout(p=0.2)
)
```
Fig. 1: Various effects of combining alpha-augmenters with other augmenters. First row shows Alpha with Median-Blur, second SimplexNoiseAlpha with EdgeDetect, third SimplexNoiseAlpha with EdgeDetect and ContrastNormalization, third shows FrequencyNoiseAlpha with Affine and AddToHueAndSaturation and forth row shows a mixture SimplexNoiseAlpha and FrequencyNoiseAlpha.
Similar to other augmenters, Alpha supports a per_channel mode, in which it samples overlay strengths for each channel independently. As a result, some channels may show more from the first (or second) image source than other channels. This can lead to visible color effects. The following example is the same as the one above, only per_channel was activated.

```python
iaa.Alpha(..., per_channel=True)
```

Alpha can also be used with augmenters that change the position of pixels, leading to “ghost” images. (This should not be done when also augmenting keypoints, as their position becomes unclear.)

```python
seq = iaa.Alpha(
    factor=(0.2, 0.8),
    first=iaa.Affine(rotate=(-20, 20)),
    per_channel=True
)
```

### 8.3 SimplexNoiseAlpha

Alpha uses a constant blending factor per image (or per channel). This limits the possibilities. Often a more localized factor is desired to create unusual patterns. SimplexNoiseAlpha is an augmenter that does that. It generates continuous masks following simplex noise and uses them to perform local blending. The following example shows a combination of SimplexNoiseAlpha and Multiply (with per_channel=True) that creates blobs of various colors in the image.
Fig. 3: Mixing Sharpen and CoarseDropout via Alpha and per_channel set to True.

Fig. 4: Mixing original images with their rotated version. Some channels are more visibly rotated than others.
import imgaug as ia
from imgaug import augmenters as iaa

ia.seed(1)

# Example batch of images.
# The array has shape (8, 128, 128, 3) and dtype uint8.
images = np.array(
    [ia.quokka(size=(128, 128)) for _ in range(8)],
    dtype=np.uint8
)

seq = iaa.SimplexNoiseAlpha(
    first=iaa.Multiply(iap.Choice([0.5, 1.5]), per_channel=True)
)

images_aug = seq.augment_images(images)

Fig. 5: Mixing original images with their versions modified by Multiply (with per_channel set to True). Simplex noise masks are used for the blending process, leading to blobby patterns.

SimplexNoiseAlpha also supports per_channel=True, leading to unique noise masks sampled per channel. The following example shows the combination of SimplexNoiseAlpha (with per_channel=True) and EdgeDetect. Even though EdgeDetect usually generates black and white images (white=edges, black=everything else), here the combination leads to strong color effects as the channel-wise noise masks only blend EdgeDetect’s result for some channels.

seq = iaa.SimplexNoiseAlpha(
    first=iaa.EdgeDetect(1.0),
    per_channel=True
)

SimplexNoiseAlpha uses continuous noise masks (2d arrays with values in the range [0.0, 1.0]) to blend images. The below image shows examples of 64x64 noise masks generated by SimplexNoiseAlpha with default settings. Values close to 1.0 (white) indicate that pixel colors will be taken from the first image source, while 0.0 (black) values indicate
Fig. 6: Blending images via simplex noise can lead to unexpected but diverse patterns when per_channel is set to True. Here, a mixture of original images with `EdgeDetect(1.0)` is used.

that pixel colors will be taken from the second image source. (Often only one image source will be given in the form of augmenters and the second will fall back to the original images fed into `SimplexNoiseAlpha`.)

Fig. 7: Examples of noise masks generated by `SimplexNoiseAlpha` using default settings.

`SimplexNoiseAlpha` generates its noise masks in low resolution images and then upscaling the masks to the size of the input images. During upscaling it usually uses nearest neighbour interpolation (nearest), linear interpolation (linear) or cubic interpolation (cubic). Nearest neighbour interpolation leads to noise maps with rectangular blobs. The below example shows noise maps generated when only using nearest neighbour interpolation.

```
seq = iaa.SimplexNoiseAlpha(
    ...,
    upscale_method="nearest"
)
```

Fig. 8: Examples of noise masks generated by `SimplexNoiseAlpha` when restricting the upscaling method to nearest.

Similarly, the following example shows noise maps generated when only using linear interpolation.

```
seq = iaa.SimplexNoiseAlpha(
    ...,
    upscale_method="linear"
)
```
Fig. 9: Examples of noise masks generated by SimplexNoiseAlpha when restricting the upscaling method to linear.

### 8.4 FrequencyNoiseAlpha

*FrequencyNoiseAlpha* is mostly identical to *SimplexNoiseAlpha*. In contrast to *SimplexNoiseAlpha* it uses a different sampling process to generate the noise maps. The process is based on starting with random frequencies, weighting them with a random exponent and then transforming from frequency domain to spatial domain. When using a low exponent value this leads to large, smooth blobs. Slightly higher exponents lead to cloudy patterns. High exponent values lead to recurring, small patterns. The below example shows the usage of *FrequencyNoiseAlpha*.

```python
import imgaug as ia
from imgaug import augmenters as iaa
from imgaug import parameters as iap

ia.seed(1)

# Example batch of images.
# The array has shape (8, 64, 64, 3) and dtype uint8.
images = np.array(
    [ia.quokka(size=(128, 128)) for _ in range(8)],
    dtype=np.uint8
)

seq = iaa.FrequencyNoiseAlpha(
    first=iaa.Multiply(iap.Choice([0.5, 1.5]), per_channel=True),
)

images_aug = seq.augment_images(images)
```

Similarly to simplex noise, *FrequencyNoiseAlpha* also supports *per_channel=True*, leading to different noise maps per image channel.

```python
seq = iaa.FrequencyNoiseAlpha(
    first=iaa.EdgeDetect(1.0),
    per_channel=True
)
```

The below image shows random example noise masks generated by *FrequencyNoiseAlpha* with default settings.

The following image shows the effects of varying *exponent* between -4.0 and 4.0. To show these effects more clearly, a few features of *FrequencyNoiseAlpha* were deactivated (e.g. multiple iterations). In the code, *E* is the value of the exponent (e.g. *E*=-2.0).

```python
seq = iaa.FrequencyNoiseAlpha(
    exponent=E,
    first=iaa.Multiply(iap.Choice([0.5, 1.5]), per_channel=True),
    size_px_max=32,
    upscale_method="linear",
    iterations=1,
    sigmoid=False
)
```

Similarly to *SimplexNoiseAlpha*, *FrequencyNoiseAlpha* also generates the noise masks as low resolution versions and
Fig. 10: Mixing original images with their versions modified by Multiply (with per_channel set to True). Simplex noise masks are used for the blending process, leading to blobby patterns.

Fig. 11: Blending images via frequency noise can lead to unexpected but diverse patterns when per_channel is set to True. Here, a mixture of original images with EdgeDetect(1.0) is used.
Fig. 12: Examples of noise masks generated by FrequencyNoiseAlpha using default settings.

Fig. 13: Examples of noise masks generated by FrequencyNoiseAlpha using default settings with varying exponents.

then upscales them to the full image size. The following images show the usage of nearest neighbour interpolation
(upscale_method="nearest") and linear interpolation (upscale_method="linear").

Fig. 14: Examples of noise masks generated by FrequencyNoiseAlpha when restricting the upscaling method to nearest.

8.5 IterativeNoiseAggregator

Both SimplexNoiseAlpha and FrequencyNoiseAlpha wrap around IterativeNoiseAggregator, a component to generate
noise masks in multiple iterations. It has parameters for the number of iterations (1 to N) and for the aggregation
methods, which controls how the noise masks from the different iterations are to be combined. Valid aggregation
methods are “min”, “avg” and “max”, where min takes the minimum over all iteration’s masks, max the maximum
and avg the average. As a result, masks generated with method min tend to be close to 0.0 (mostly black values),
those generated with max close to 1.0 and avg converges towards 0.5. (0.0 means that the results of the second image
Fig. 15: Examples of noise masks generated by FrequencyNoiseAlpha when restricting the upscaling method to linear.

dominate the final image, so in many cases the original images before the augmenter). The following image shows the effects of changing the number of iterations when combining FrequencyNoise with IterativeNoiseAggregator.

```python
# This is how the iterations would be changed for FrequencyNoiseAlpha.
# (Same for `SimplexNoiseAlpha`.)
seq = iaa.FrequencyNoiseAlpha(
    ...,  
    iterations=N
)
```

8.6 Sigmoid

Generated noise masks can often end up having many values around 0.5, especially when running IterativeNoiseAggregator with many iterations and aggregation method avg. This can be undesired. Sigmoid is a method to compensate that. It applies a sigmoid function to the noise masks, forcing the values to mostly lie close to 0.0 or 1.0 and only rarely in between. This can lead to blobs of values close to 1.0 ("use only colors from images coming from source images")
Fig. 17: Examples of varying the methods and iterations in IterativeNoiseAggregator (here in combination with FrequencyNoise).

A”), surrounded by blobs with values close to 0.0 (“use only colors from images coming from source B’’). This is similar to taking either from one image source (per pixel) or the other, but usually not both. Sigmoid is integrated into both SimplexNoiseAlpha and FrequencyNoiseAlpha. It can be dynamically activated/deactivated and has a threshold parameter that controls how aggressive and pushes the noise values towards 1.0.

```python
# This is how the Sigmoid would be activated/deactivated for FrequencyNoiseAlpha (same for SimplexNoiseAlpha).
# P is the probability of the Sigmoid being activated (can be True/False), T is the threshold (sane values are usually around -10 to +10, can be a tuple, e.g. sigmoid_thresh=(-10, 10), to indicate a uniform range).
seq = iaa.FrequencyNoiseAlpha(
    ..., sigmoid=P, sigmoid_thresh=T
)
```

The below image shows the effects of applying Sigmoid to noise masks generated by FrequencyNoise.

Fig. 18: Examples of noise maps without and with activated Sigmoid (noise maps here from FrequencyNoise).

The below image shows the effects of varying the sigmoid’s threshold. Lower values place the threshold further to the “left” (lower x values), leading to more x-values being above the threshold values, leading to more 1.0s in the noise masks.
Fig. 19: Examples of varying the Sigmoid threshold from -10.0 to 10.0.
CHAPTER 9

Overview of Augmenters

9.1 Sequential

List augmenter that may contain other augmenters to apply in sequence or random order.

Apply in predefined order:

```python
aug = iaa.Sequential(%
    iaa.Affine(translate_px={"x": -40}),
    iaa.AdditiveGaussianNoise(scale=0.1*255)
%)
```

![Sequential example images](image-url)
Apply in random order (note that the order is sampled once per batch and then the same for all images within the batch):

```python
aug = iaa.Sequential(
    [iaa.Affine(translate_px={"x": -40}),
     iaa.AdditiveGaussianNoise(scale=0.1*255)],
    random_order=True)
```

---

### 9.2 SomeOf

List augmenter that applies only some of its children to images.

Apply two of four given augmenters:

```python
aug = iaa.SomeOf(2, [
    iaa.Affine(rotate=45),
    iaa.AdditiveGaussianNoise(scale=0.2*255),
    iaa.Add(50, per_channel=True),
    iaa.Sharpen(alpha=0.5)
])
```

Apply 0 to `<max>` given augmenters (where `<max>` is automatically replaced with the number of children):

```python
aug = iaa.SomeOf((0, None), [
    iaa.Affine(rotate=45),
    iaa.AdditiveGaussianNoise(scale=0.2*255),
    iaa.Add(50, per_channel=True),
    iaa.Sharpen(alpha=0.5)
])
```

Pick two of four given augmenters and apply them in random order:
 augment = iaa.SomeOf(2, [
    iaa.Affine(rotate=45),
    iaa.AdditiveGaussianNoise(scale=0.2*255),
    iaa.Add(50, per_channel=True),
    iaa.Sharpen(alpha=0.5)
], random_order=True)

9.3 OneOf

Augmenter that always executes exactly one of its children.

Apply one of four augmenters to each image:

 augment = iaa.OneOf({
    iaa.Affine(rotate=45),
    iaa.AdditiveGaussianNoise(scale=0.2*255),
    iaa.Add(50, per_channel=True),
    iaa.Sharpen(alpha=0.5)
})

9.4 Sometimes

Augment only p percent of all images with one or more augmenters.

Apply gaussian blur to about 50% of all images:

 augment = iaa.Sometimes(0.5, iaa.GaussianBlur(sigma=2.0))

Apply gaussian blur to about 50% of all images. Apply a mixture of affine rotations and sharpening to the other 50%.
aug = iaa.Sometimes(0.5,
    iaa.GaussianBlur(sigma=2.0),
    iaa.Sequential([iaa.Affine(rotate=45), iaa.Sharpen(alpha=1.0)]))

### 9.5 WithColorspace

Apply child augmenters within a specific colorspace.

Convert images to HSV, then increase each pixels H-value by 10 to 50:

```python
aug = iaa.WithColorspace(
    to_colorspace="HSV",
    from_colorspace="RGB",
    children=iaa.WithChannels(0, iaa.Add((10, 50)))
)
```
9.6 WithChannels

Apply child augmenters to specific channels.

Increase each pixel’s R-value (redness) by 10 to 100:

```python
aug = iaa.WithChannels(0, iaa.Add((10, 100)))
```

Rotate each image’s red channel by 0 to 45 degrees:

```python
aug = iaa.WithChannels(0, iaa.Affine(rotate=(0, 45)))
```
9.7 Noop

Augmenter that never changes input images (“no operation”).

```python
aug = iaa.Noop()
```

9.8 Lambda

Augmenter that calls a lambda function for each batch of input image. Replace in every image each fourth row with black pixels:

```python
def img_func(images, random_state, parents, hooks):
    for img in images:
        img[::4] = 0
    return images

def keypoint_func(keypoints_on_images, random_state, parents, hooks):
    return keypoints_on_images

aug = iaa.Lambda(img_func, keypoint_func)
```

9.9 AssertLambda

Augmenter that runs an assert on each batch of input images using a lambda function as condition.

TODO examples
9.10 AssertShape

Augmenter to make assumptions about the shape of input image(s) and keypoints.

Check if each image in a batch has shape 32x32x3, otherwise raise an exception:

```python
seq = iaa.Sequential([    iaa.AssertShape((None, 32, 32, 3)),    iaa.Fliplr(0.5)  # only executed if shape matches ])
```

Check if each image in a batch has a height in the range 32<=x<64, a width of exactly 64 and either 1 or 3 channels:

```python
seq = iaa.Sequential([    iaa.AssertShape((None, (32, 64), 32, [1, 3])),    iaa.Fliplr(0.5)  ])
```

9.11 Resize

Augmenter that resizes images to specified heights and widths.

Resize each image to height=32 and width=64:

```python
aug = iaa.Resize("height": 32, "width": 64)
```
Resize each image to height=32 and keep the aspect ratio for width the same:

```
aug = iaa.Resize({"height": 32, "width": "keep-aspect-ratio"})
```

Resize each image to something between 50 and 100% of its original size:

```
aug = iaa.Resize((0.5, 1.0))
```

Resize each image's height to 50-75% of its original size and width to either 16px or 32px or 64px:

```
aug = iaa.Resize({"height": (0.5, 0.75), "width": [16, 32, 64]})
```

9.12 CropAndPad

Augmenter that crops/pads images by defined amounts in pixels or percent (relative to input image size).

NOTE: This augmenter automatically resizes images back to their original size after it has augmented them. To deactivate this, add the parameter `keep_size=False`.

Crop or pad each side by up to 10 percent relative to its original size (negative values result in cropping, positive in padding):

```
aug = iaa.CropAndPad(percentage=(-0.25, 0.25))
```

Pad each side by 0 to 20 percent. This adds new pixels to the sides. These pixels will either be filled with a constant value (mode=constant) or filled with the value on the closest edge (mode=edge). If a constant value is used, it will be a random value between 0 and 128 (sampled per image).
Augmentation using the `CropAndPad` class in `imgaug`.

```python
aug = iaa.CropAndPad(
    percent=(0, 0.2),
    pad_mode=['constant', 'edge'],
    pad_cval=(0, 128)
)
```

Pad the top side of each image by 0 to 30 pixels, the right side by 0-10px, bottom side by 0-30px and left side by 0-10px. Use any of the available modes to fill new pixels and if the mode is `constant` then use a constant value between 0 and 128.

```python
aug = iaa.CropAndPad(
    px=((0, 30), (0, 10), (0, 30), (0, 10)),
    pad_mode=ia.ALL,
    pad_cval=(0, 128)
)
```

Crop/pad each side by up to 10px. The value will be sampled once per image and used for all sides (i.e. all sides gain/lose the same number of rows/columns).

```python
aug = iaa.CropAndPad(
    px=(-10, 10),
    sample_independently=False
)
```
9.13 Pad

Augmenter that pads images, i.e. adds columns/rows to them.
This is a proxy for CropAndPad. It only accepts positive pixel/percent values.

9.14 Crop

Augmenter that crops/cuts away pixels at the sides of the image.
This is a proxy for CropAndPad. It only accepts positive pixel/percent values and transfers them as negative values to CropAndPad.

9.15 Fliplr

Flip/mirror input images horizontally.
Flip 50% of all images horizontally:

```
aug = iaa.Fliplr(0.5)
```

NOTE: the default probability is 0, so to flip all images, do:

```
aug = iaa.Fliplr(1)
```

9.16 Flipud

Flip/mirror input images vertically.
Flip 50% of all images vertically:

```python
aug = iaa.Flipud(0.5)
```

NOTE: the default probability is 0, so to flip all images, do:

```python
aug = iaa.Flipud(1)
```

---

### 9.17 Superpixels

Completely or partially transform images to their superpixel representation.

Generate about 64 superpixels per image. Replace each one with a probability of 50% by its average pixel color.

```python
aug = iaa.Superpixels(p_replace=0.5, n_segments=64)
```

Generate 16 to 128 superpixels per image. Replace each superpixel with a probability between 10 and 100% (sampled once per image) by its average pixel color.

```python
aug = iaa.Superpixels(p_replace=(0.1, 1.0), n_segments=(16, 128))
```
Effect of setting \texttt{n\_segments} to a fixed value of 64 and then increasing \texttt{p\_replace} from 0.0 and 1.0:

Effect of setting \texttt{p\_replace} to a fixed value of 1.0 and then increasing \texttt{n\_segments} from 1*16 to 9*16=144:

### 9.18 \texttt{ChangeColorspace}

Augmenter to change the colorspace of images.

The following example shows how to change the colorspace from RGB to HSV, then add 50-100 to the first channel, then convert back to RGB. This increases the hue value of each image.

```python
aug = iaa.Sequential([
    iaa.ChangeColorspace(from_colorspace="RGB", to_colorspace="HSV"),
    iaa.WithChannels(0, iaa.Add((50, 100))),
    iaa.ChangeColorspace(from_colorspace="HSV", to_colorspace="RGB")
])
```
9.19 Grayscale

Augmenter to convert images to their grayscale versions.

Change images to grayscale and overlay them with the original image by varying strengths, effectively removing 0 to 100% of the color:

```python
aug = iaa.Grayscale(alpha=(0.0, 1.0))
```

Visualization of increasing `alpha` from 0.0 to 1.0 in 8 steps:
9.20 GaussianBlur

Augmenter to blur images using gaussian kernels.
Blur each image with a gaussian kernel with a sigma of 3.0:

```python
aug = iaa.GaussianBlur(sigma=(0.0, 3.0))
```

9.21 AverageBlur

Blur an image by computing simple means over neighbourhoods.
Blur each image using a mean over neighbourhoods that have a random size between 2x2 and 11x11:

```python
aug = iaa.AverageBlur(k=(2, 11))
```
Blur each image using a mean over neighbourhoods that have random sizes, which can vary between 5 and 11 in height and 1 and 3 in width:

```python
aug = iaa.AverageBlur(k=((5, 11), (1, 3)))
```

9.22 MedianBlur

Blur an image by computing median values over neighbourhoods.
Blur each image using a median over neighbourhoods that have a random size between 3x3 and 11x11:

```python
aug = iaa.MedianBlur(k=(3, 11))
```

9.23 Convolve

Apply a Convolution to input images.
Convolve each image with a 3x3 kernel:

```python
matrix = np.array([[0, -1, 0],
                   [-1, 4, -1],
                   [0, -1, 0]])
aug = iaa.Convolve(matrix=matrix)
```
Convolve each image with a 3x3 kernel, which is chosen dynamically per image:
def gen_matrix(image, nb_channels, random_state):
    matrix_A = np.array([[0, -1, 0],
                         [-1, 4, -1],
                         [0, -1, 0]])
    matrix_B = np.array([[0, 0, 0],
                         [0, -4, 1],
                         [0, 2, 1]])
    if random_state.rand() < 0.5:
        return [matrix_A] * nb_channels
    else:
        return [matrix_B] * nb_channels

aug = iaa.Convolve(matrix=gen_matrix)
9.24 Sharpen

Augmenter that sharpens images and overlays the result with the original image.

Sharpen an image, then overlay the results with the original using an alpha between 0.0 and 1.0:

```python
aug = iaa.Sharpen(alpha=(0.0, 1.0), lightness=(0.75, 2.0))
```

Effects of keeping lightness fixed at 1.0 and then varying alpha between 0.0 and 1.0 in 8 steps:

Effects of keeping alpha fixed at 1.0 and then varying lightness between 0.75 and 1.5 in 8 steps:

9.25 Emboss

Augmenter that embosses images and overlays the result with the original image.

Emboss an image, then overlay the results with the original using an alpha between 0.0 and 1.0:

```python
aug = iaa.Emboss(alpha=(0.0, 1.0), strength=(0.5, 1.5))
```

Effects of keeping strength fixed at 1.0 and then varying alpha between 0.0 and 1.0 in 8 steps:

Effects of keeping alpha fixed at 1.0 and then varying strength between 0.5 and 1.5 in 8 steps:

9.26 EdgeDetect

Augmenter that detects all edges in images, marks them in a black and white image and then overlays the result with the original image.

9.24 Sharpen
Detect edges in images, turning them into black and white images and then overlay these with the original images using random alphas between 0.0 and 1.0:

```
aug = iaa.EdgeDetect(alpha=(0.0, 1.0))
```

Effect of increasing alpha from 0.0 to 1.0 in 8 steps:

### 9.27 DirectedEdgeDetect

Augmenter that detects edges that have certain directions and marks them in a black and white image and then overlays the result with the original image.

Detect edges having random directions (0 to 360 degrees) in images, turning the images into black and white versions and then overlay these with the original images using random alphas between 0.0 and 1.0:

```
aug = iaa.DirectedEdgeDetect(alpha=(0.0, 1.0), direction=(0.0, 1.0))
```

Effect of fixing direction to 0.0 and then increasing alpha from 0.0 to 1.0 in 8 steps:

Effect of fixing alpha to 1.0 and then increasing direction from 0.0 to 1.0 (0 to 360 degrees) in 8 steps:

### 9.28 Add

Add a value to all pixels in an image.

Add random values between -40 and 40 to images, with each value being sampled once per image and then being the same for all pixels:
Add random values between -40 and 40 to images. In 50% of all images the values differ per channel (3 sampled value). In the other 50% of all images the value is the same for all channels:

```
aug = iaa.Add((-40, 40), per_channel=0.5)
```

### 9.29 AddElementwise

Add values to the pixels of images with possibly different values for neighbouring pixels.

Add random values between -40 and 40 to images, with each value being sampled per pixel:

```
aug = iaa.AddElementwise((-40, 40))
```

Add random values between -40 and 40 to images. In 50% of all images the values differ per channel (3 sampled values per pixel). In the other 50% of all images the value is the same for all channels per pixel:

```
aug = iaa.AddElementwise((-40, 40), per_channel=0.5)
```

### 9.30 AdditiveGaussianNoise

Add gaussian noise (aka white noise) to images.

Add gaussian noise to an image, sampled once per pixel from a normal distribution $N(0, s)$, where $s$ is sampled per image and varies between 0 and 0.05*255:

```
aug = iaa.AdditiveGaussianNoise(scale=(0, 0.05*255))
```

Add gaussian noise to an image, sampled once per pixel from a normal distribution $N(0, 0.05*255)$:
Additive Gaussian Noise

Add Gaussian noise to an image, sampled once per pixel from a normal distribution $N(0, 0.05 \times 255)$ for 50% of all images and sampled three times (channel-wise) for the other 50% from the same normal distribution:

```
aug = iaa.AdditiveGaussianNoise(scale=0.05*255, per_channel=0.5)
```

### 9.31 Multiply

Multiply all pixels in an image with a specific value, thereby making the image darker or brighter.

Multiply each image with a random value between 0.5 and 1.5:

```
aug = iaa.Multiply((0.5, 1.5))
```

Multiply 50% of all images with a random value between 0.5 and 1.5 and multiply the remaining 50% channel-wise, i.e. sample one multiplier independently per channel:

```
aug = iaa.Multiply((0.5, 1.5), per_channel=0.5)
```

### 9.32 MultiplyElementwise

Multiply values of pixels with possibly different values for neighbouring pixels, making each pixel darker or brighter.

Multiply each pixel with a random value between 0.5 and 1.5:

```
aug = iaa.MultiplyElementwise((0.5, 1.5))
```

Multiply in 50% of all images each pixel with random values between 0.5 and 1.5 and multiply in the remaining 50% of all images the pixels channel-wise, i.e. sample one multiplier independently per channel and pixel:
Augmenter that sets a certain fraction of pixels in images to zero.

Sample per image a value \( p \) from the range \( 0 \leq p \leq 0.2 \) and then drop \( p \) percent of all pixels in the image (i.e. convert them to black pixels):

\[
aug = \text{iaa.Dropout}(p=(0, 0.2))
\]
Sample per image a value p from the range 0<=p<=0.2 and then drop p percent of all pixels in the image (i.e. convert them to black pixels), but do this independently per channel in 50% of all images:

\[
\text{aug = iaa.Dropout(p=(0, 0.2), per_channel=0.5)}
\]

### 9.34 CoarseDropout

Augmenter that sets rectangular areas within images to zero.

Drop 2% of all pixels by converting them to black pixels, but do that on a lower-resolution version of the image that has 50% of the original size, leading to 2x2 squares being dropped.
Drop 0 to 5% of all pixels by converting them to black pixels, but do that on a lower-resolution version of the image that has 5% to 50% of the original size, leading to large rectangular areas being dropped:

```
aug = iaa.CoarseDropout((0.0, 0.05), size_percent=(0.02, 0.25))
```

Drop 2% of all pixels by converting them to black pixels, but do that on a lower-resolution version of the image that has 50% of the original size, leading to 2x2 squares being dropped. Also do this in 50% of all images channel-wise, so that only the information of some channels in set to 0 while others remain untouched:

```
aug = iaa.CoarseDropout(0.02, size_percent=0.5)
```
9.35 Invert

Augmenter that inverts all values in images, i.e. sets a pixel from value $v$ to $255 - v$.

Invert in 50% of all images all pixels:

```python
aug = iaa.Invert(0.5)
```

For 50% of all images, invert all pixels in these images with 25% probability (per image). In the remaining 50% of all images, invert 25% of all channels:

```python
aug = iaa.Invert(0.25, per_channel=0.5)
```

9.36 ContrastNormalization

Augmenter that changes the contrast of images.

Normalize contrast by a factor of 0.5 to 1.5, sampled randomly per image:
Normalize contrast by a factor of 0.5 to 1.5, sampled randomly per image and for 50% of all images also independently per channel:

```python
aug = iaa.ContrastNormalization((0.5, 1.5), per_channel=0.5)
```

9.37 **Affine**

Augmenter to apply affine transformations to images.

Scale images to a value of 50 to 150% of their original size:

```python
aug = iaa.Affine(scale=(0.5, 1.5))
```

Scale images to a value of 50 to 150% of their original size, but do this independently per axis (i.e. sample two values per image):
Translate images by -20 to +20% on x- and y-axis independently:

```python
aug = iaa.Affine(translate_percent={"x": (-0.2, 0.2), "y": (-0.2, 0.2))
```

Translate images by -20 to 20 pixels on x- and y-axis independently:

```python
aug = iaa.Affine(translate_px={"x": (-20, 20), "y": (-20, 20))
```

Rotate images by -45 to 45 degrees:

```python
aug = iaa.Affine(rotate=(-45, 45))
```

Shear images by -16 to 16 degrees:

```python
aug = iaa.Affine(shear=(-16, 16))
```

When applying affine transformations, new pixels are often generated, e.g. when translating to the left, pixels are generated on the right. Various modes exist to set how these pixels are ought to be filled. Below code shows an
example that uses all modes, sampled randomly per image. If the mode is constant (fill all with one constant value),
then a random brightness between 0 and 255 is used:

```
aug = iaa.Affine(translate_percent={"x": -0.20}, mode=ia.ALL, cval=(0, 255))
```

![Image showing the effect of the Affine transformation with different modes]

### 9.38 PiecewiseAffine

Augmenter that places a regular grid of points on an image and randomly moves the neighbourhood of these point
around via affine transformations. This leads to local distortions.

Distort images locally by moving points around, each with a distance v (percent relative to image size), where v is
sampled per point from $N(0, z)$ $z$ is sampled per image from the range 0.01 to 0.05:

```
aug = iaa.PiecewiseAffine(scale=(0.01, 0.05))
```

![Image showing the effect of the PiecewiseAffine transformation with different scales]

Effect of increasing `scale` from 0.01 to 0.3 in 8 steps:

PiecewiseAffine works by placing a regular grid of points on the image and moving them around. By default this grid
consists of 4x4 points. The below image shows the effect of increasing that value from 2x2 to 16x16 in 8 steps:
9.39 ElasticTransformation

Augmenter to transform images by moving pixels locally around using displacement fields.

Distort images locally by moving individual pixels around following a distortions field with strength 0.25. The strength of the movement is sampled per pixel from the range 0 to 5.0:

```
aug = iaa.ElasticTransformation(alpha=(0, 5.0), sigma=0.25)
```

Effect of keeping sigma fixed at 0.25 and increasing alpha from 0 to 5.0 in 8 steps:

Effect of keeping alpha fixed at 2.5 and increasing sigma from 0.01 to 1.0 in 8 steps:
Below are performance measurements of each augmenter for image augmentation (`augment_images()`), heatmap augmentation (`augment_heatmaps()`), and keypoint/landmark augmentation (`augment_keypoints()`). (Last updated for 0.2.8.)

**System**: The numbers were computed based on a haswell-generation i7 3.2Ghz CPU with DDR3 memory. That is a rather dated system by today’s standards. A modern, high-end system should achieve higher bandwidths.

**Experiments Settings**: All augmenters were run with reasonable parameter choices that should reflect expected real-world usage, while avoiding too simple parameter values that would lead to inflated scores. Some parameter choices are listed below, the remaining ones can be looked up in `measure_performance.py`. Kernel sizes were all set to $3 \times 3$, unless otherwise mentioned. The inputs focused on a small and large image-size setting, using $64 \times 64 \times 3$ and $224 \times 224 \times 3$ as the respective sizes. The base image was taken from `skimage.data.astronaut`, which should be a representative real-world image. Batch sizes of 1 and 128 were tested. Each augmenter was run **100** times on the generated input and the average of the measured runtimes was computed to derive bandwidth in mbit per second and the raw number of augmented items (e.g. images) per second.

### 10.1 Results Overview

From the results, the following points can be derived.

**Inputs:**

- Use large batch sizes whenever possible. Many augmenters are significantly faster with these.
- Large image sizes lead to higher throughput based on mbit/sec. Smaller images lead to lower throughput, but significantly more items/sec (roughly 4-10x more). Use small images whenever possible.
- For keypoint-based and heatmap-based augmentation, try to increase the number of items per augmented instance. E.g. `augment_keypoints()` accepts a list of `KeypointsOnImage` instances, with each such instance representing the keypoints on an image. Try to place for each image all keypoints in the respective `KeypointsOnImage` instance instead of splitting them into multiple such instances (which would be more work anyways). The same is true for bounding boxes, heatmaps and segmentation maps.
• Keypoint- and heatmap-based inputs are only affected by augmenters that change the geometry of the image (e.g. Crop or Affine). Other augmenters are essentially free to execute as they do not perform any changes.

• Keypoint-based augmentation is very fast for almost all augmenters, reaching several 100k keypoints per second. Slower augmenters are ElasticTransformation and PiecewiseAffine, as these currently have to fall back to image-based algorithms.

Parameter choices:

• When possible, nearest neighbour interpolation or linear interpolation should be used as these are significantly faster than other options. Most augmenters that use interpolation offer either an order parameter (0=nearest neighbour, 1=linear) or an interpolation parameter (“nearest”, “linear”).

• Using keep_size=True is the default setting in all augmenters that change image sizes. It is convenient, as it ensures that image sizes are not altered by the augmentation. It does however incur a significant performance penalty, often more than halving the bandwidth. Try keep_size=False when possible. You can still resize images manually after augmentation or by using KeepSizeByResize(Sequential(<augmenters>)).

• When augmenters offer modes to fill newly created pixels in user-defined ways (e.g. pad_mode=constant in Pad to fill up all padded pixels with a specified constant color), using edge instead of constant will usually not incur a significant performance penalty.

Specific Augmenter suggestions:

• For augmenters where an elementwise sibling exists (e.g. Multiply and MultiplyElementwise), the elementwise augmenter is usually significantly slower than the non-elementwise one.

• If blurring is required, AverageBlur is the fastest choice, followed by GaussianBlur.

• Augmenters that operate on coarser images (e.g. CoarseDropout vs Dropout) can be significantly faster than their non-coarse siblings.

• Contrast normalizing augmenters are all comparable in performance, except for histogram-based ones, which are significantly slower.

• PiecewiseAffine is a very slow augmenter and should usually be replaced by ElasticTransformation, which achieves similar outputs and is quite a bit faster.

• Superpixels is a fairly slow augmenter and should usually be wrapped in e.g. Sometimes to not apply it very often and reduce its performance impact.

• Weather augmenters other than FastSnowyLandscape are rather slow and should only be used when sensible.

10.2 Images

Numbers below are for small images (64x64x3) and large images (224x224x3). B=1 denotes a batch size of 1, B=128 one of 128.

In mbit/sec:

<table>
<thead>
<tr>
<th>Augmenter</th>
<th>64x64x3, uint8</th>
<th>224x224x3, uint8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential (2xNoop)</td>
<td>2052.0</td>
<td>20772.2</td>
</tr>
<tr>
<td>Sequential (2xNoop, random_order)</td>
<td>1278.3</td>
<td>37809.2</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop)</td>
<td>471.6</td>
<td>5460.4</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop, random_order)</td>
<td>457.2</td>
<td>5414.6</td>
</tr>
<tr>
<td>OneOf (3xNoop)</td>
<td>799.7</td>
<td>6351.8</td>
</tr>
<tr>
<td>Sometimes (Noop)</td>
<td>746.9</td>
<td>7580.3</td>
</tr>
<tr>
<td>Function</td>
<td>Time 1</td>
<td>Time 2</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>WithChannels ([1,2], Noop)</td>
<td>867.2</td>
<td>8079.6</td>
</tr>
<tr>
<td>Noop</td>
<td>4593.6</td>
<td>22467.1</td>
</tr>
<tr>
<td>Lambda (return input)</td>
<td>4571.2</td>
<td>43614.9</td>
</tr>
<tr>
<td>AssertLambda (return True)</td>
<td>4459.2</td>
<td>22597.5</td>
</tr>
<tr>
<td>AssertShape (None, H, W, None)</td>
<td>2710.3</td>
<td>41203.8</td>
</tr>
<tr>
<td>ChannelShuffle (0.5)</td>
<td>829.3</td>
<td>2473.4</td>
</tr>
<tr>
<td>Add</td>
<td>626.6</td>
<td>2643.9</td>
</tr>
<tr>
<td>AddElementwise</td>
<td>412.5</td>
<td>633.5</td>
</tr>
<tr>
<td>AdditiveGaussianNoise</td>
<td>259.4</td>
<td>336.9</td>
</tr>
<tr>
<td>AdditiveLaplaceNoise</td>
<td>244.3</td>
<td>308.6</td>
</tr>
<tr>
<td>AdditivePoissonNoise</td>
<td>160.9</td>
<td>184.1</td>
</tr>
<tr>
<td>Multiply</td>
<td>651.1</td>
<td>2682.8</td>
</tr>
<tr>
<td>MultiplyElementwise</td>
<td>392.8</td>
<td>596.7</td>
</tr>
<tr>
<td>Dropout (1-5%)</td>
<td>4459.2</td>
<td>22597.5</td>
</tr>
<tr>
<td>CoarseDropout (1-5%, size=1-10%)</td>
<td>258.9</td>
<td>354.5</td>
</tr>
<tr>
<td>ReplaceElementwise</td>
<td>227.8</td>
<td>293.6</td>
</tr>
<tr>
<td>ImpulseNoise</td>
<td>194.9</td>
<td>231.9</td>
</tr>
<tr>
<td>SaltAndPepper</td>
<td>200.4</td>
<td>245.3</td>
</tr>
<tr>
<td>CoarseSaltAndPepper</td>
<td>141.5</td>
<td>185.4</td>
</tr>
<tr>
<td>Salt</td>
<td>168.5</td>
<td>197.7</td>
</tr>
<tr>
<td>CoarseSalt</td>
<td>138.9</td>
<td>156.8</td>
</tr>
<tr>
<td>Pepper</td>
<td>169.4</td>
<td>195.9</td>
</tr>
<tr>
<td>CoarsePepper</td>
<td>140.1</td>
<td>140.6</td>
</tr>
<tr>
<td>Invert (10%)</td>
<td>809.4</td>
<td>7439.3</td>
</tr>
<tr>
<td>JpegCompression (50-99%)</td>
<td>85.7</td>
<td>92.1</td>
</tr>
<tr>
<td>GaussianBlur (sigma=(1,5))</td>
<td>536.6</td>
<td>1252.2</td>
</tr>
<tr>
<td>AverageBlur</td>
<td>925.1</td>
<td>4149.9</td>
</tr>
<tr>
<td>MedianBlur</td>
<td>216.9</td>
<td>272.6</td>
</tr>
<tr>
<td>BilateralBlur</td>
<td>199.1</td>
<td>284.2</td>
</tr>
<tr>
<td>MotionBlur</td>
<td>131.0</td>
<td>141.3</td>
</tr>
<tr>
<td>WithColorspace (HSV, Noop)</td>
<td>188.0</td>
<td>523.7</td>
</tr>
<tr>
<td>AddToHueAndSaturation</td>
<td>192.0</td>
<td>338.1</td>
</tr>
<tr>
<td>ChangeColorspace (HSV)</td>
<td>511.0</td>
<td>982.9</td>
</tr>
<tr>
<td>Grayscale</td>
<td>271.6</td>
<td>430.3</td>
</tr>
<tr>
<td>GammaContrast</td>
<td>449.5</td>
<td>719.6</td>
</tr>
<tr>
<td>SigmoidContrast</td>
<td>402.5</td>
<td>627.4</td>
</tr>
<tr>
<td>LogContrast</td>
<td>450.9</td>
<td>719.2</td>
</tr>
<tr>
<td>LinearContrast</td>
<td>538.9</td>
<td>1014.3</td>
</tr>
<tr>
<td>AllChannelsHistogramEqualization</td>
<td>930.7</td>
<td>1752.6</td>
</tr>
<tr>
<td>HistogramEqualization</td>
<td>209.8</td>
<td>476.5</td>
</tr>
<tr>
<td>AllChannelsCLAHE</td>
<td>209.9</td>
<td>373.5</td>
</tr>
<tr>
<td>CLAHE</td>
<td>115.0</td>
<td>369.6</td>
</tr>
<tr>
<td>Convolve (3x3)</td>
<td>1028.2</td>
<td>2539.3</td>
</tr>
<tr>
<td>Sharpen</td>
<td>427.6</td>
<td>608.1</td>
</tr>
<tr>
<td>Emboss</td>
<td>462.0</td>
<td>637.4</td>
</tr>
<tr>
<td>EdgeDetect</td>
<td>524.9</td>
<td>770.3</td>
</tr>
<tr>
<td>DirectedEdgeDetect</td>
<td>197.2</td>
<td>229.4</td>
</tr>
<tr>
<td>Flplr (p=100%)</td>
<td>1116.0</td>
<td>2646.0</td>
</tr>
<tr>
<td>Flipud (p=100%)</td>
<td>1505.5</td>
<td>7514.2</td>
</tr>
<tr>
<td>Affine (order=0, constant)</td>
<td>228.0</td>
<td>551.4</td>
</tr>
<tr>
<td>Transformation</td>
<td>64x64x3, uint8</td>
<td>224x224x3, uint8</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Augmenter B=1</td>
<td>21887.5</td>
<td>221570.1</td>
</tr>
<tr>
<td>Augmenter B=128</td>
<td>221570.1</td>
<td>12510.2</td>
</tr>
<tr>
<td>Sequential (2xNoop)</td>
<td>21887.5</td>
<td>221570.1</td>
</tr>
<tr>
<td>Sequential (2xNoop, random_order)</td>
<td>13635.6</td>
<td>10127.5</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop)</td>
<td>5030.5</td>
<td>58244.5</td>
</tr>
<tr>
<td>Function Description</td>
<td>Mean (Run 1)</td>
<td>Mean (Run 2)</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop, random_order)</td>
<td>4876.8</td>
<td>57755.8</td>
</tr>
<tr>
<td>OneOf (3xNoop)</td>
<td>8530.6</td>
<td>67752.3</td>
</tr>
<tr>
<td>Sometimes (Noop)</td>
<td>7966.7</td>
<td>80856.6</td>
</tr>
<tr>
<td>WithChannels ([1,2], Noop)</td>
<td>9250.2</td>
<td>86182.7</td>
</tr>
<tr>
<td>Noop</td>
<td>48998.9</td>
<td>239648.8</td>
</tr>
<tr>
<td>Lambda (return input)</td>
<td>48759.6</td>
<td>465226.1</td>
</tr>
<tr>
<td>AssertLambda (return True)</td>
<td>47565.3</td>
<td>241040.0</td>
</tr>
<tr>
<td>AssertShape (None, H, W, None)</td>
<td>439506.9</td>
<td>16511.7</td>
</tr>
<tr>
<td>ChannelShuffle (0.5)</td>
<td>28910.3</td>
<td>439506.9</td>
</tr>
<tr>
<td>Noop</td>
<td>48998.9</td>
<td>239648.8</td>
</tr>
<tr>
<td>Lambda (return input)</td>
<td>48759.6</td>
<td>465226.1</td>
</tr>
<tr>
<td>AssertLambda (return True)</td>
<td>47565.3</td>
<td>241040.0</td>
</tr>
<tr>
<td>AssertShape (None, H, W, None)</td>
<td>439506.9</td>
<td>16511.7</td>
</tr>
<tr>
<td>ChannelShuffle (0.5)</td>
<td>28910.3</td>
<td>439506.9</td>
</tr>
</tbody>
</table>
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Time (s)</th>
<th>Memory (MB)</th>
<th>Time (s)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fliplr (p=100%)</td>
<td>11903.8</td>
<td>28223.5</td>
<td>2203.1</td>
<td>1094.5</td>
</tr>
<tr>
<td>Flipud (p=100%)</td>
<td>16059.1</td>
<td>80152.0</td>
<td>5343.9</td>
<td>1260.4</td>
</tr>
<tr>
<td>Affine (order=0, constant)</td>
<td>2431.8</td>
<td>5881.5</td>
<td>1366.7</td>
<td>536.3</td>
</tr>
<tr>
<td>Affine (order=1, constant)</td>
<td>2253.9</td>
<td>5002.8</td>
<td>1094.5</td>
<td>1260.4</td>
</tr>
<tr>
<td>Affine (order=3, constant)</td>
<td>1922.5</td>
<td>3595.1</td>
<td>536.3</td>
<td>1260.4</td>
</tr>
<tr>
<td>Affine (order=1, edge)</td>
<td>2228.8</td>
<td>5081.3</td>
<td>1260.4</td>
<td>28.4</td>
</tr>
<tr>
<td>Affine (order=1, constant, skimage)</td>
<td>985.9</td>
<td>1309.5</td>
<td>1260.4</td>
<td>8.0</td>
</tr>
<tr>
<td>PiecewiseAffine (4x4, order=1, constant)</td>
<td>16059.1</td>
<td>80152.0</td>
<td>5343.9</td>
<td>1260.4</td>
</tr>
<tr>
<td>PiecewiseAffine (4x4, order=0, constant)</td>
<td>1580.2</td>
<td>2251.8</td>
<td>410.7</td>
<td>378.5</td>
</tr>
<tr>
<td>PiecewiseAffine (4x4, order=1, edge)</td>
<td>1610.8</td>
<td>2250.4</td>
<td>296.4</td>
<td>378.5</td>
</tr>
<tr>
<td>PiecewiseAffine (8x8, order=1, constant)</td>
<td>12.0</td>
<td>12.5</td>
<td>8.0</td>
<td>8.0</td>
</tr>
<tr>
<td>PerspectiveTransform</td>
<td>2856.9</td>
<td>4290.0</td>
<td>1313.5</td>
<td>378.5</td>
</tr>
<tr>
<td>PerspectiveTransform (keep_size)</td>
<td>2124.2</td>
<td>2925.3</td>
<td>759.2</td>
<td>759.2</td>
</tr>
<tr>
<td>ElasticTransformation (order=0, constant)</td>
<td>1718.8</td>
<td>2431.7</td>
<td>432.8</td>
<td>432.8</td>
</tr>
<tr>
<td>ElasticTransformation (order=1, constant)</td>
<td>1580.2</td>
<td>2251.8</td>
<td>410.7</td>
<td>410.7</td>
</tr>
<tr>
<td>ElasticTransformation (order=1, nearest)</td>
<td>1600.0</td>
<td>2250.4</td>
<td>378.5</td>
<td>378.5</td>
</tr>
<tr>
<td>ElasticTransformation (order=1, reflect)</td>
<td>1610.8</td>
<td>2250.4</td>
<td>296.4</td>
<td>296.4</td>
</tr>
<tr>
<td>Rot90</td>
<td>10653.0</td>
<td>66578.0</td>
<td>8201.5</td>
<td>8201.5</td>
</tr>
<tr>
<td>Rot90 (keep_size)</td>
<td>7738.6</td>
<td>23425.3</td>
<td>2183.6</td>
<td>2183.6</td>
</tr>
<tr>
<td>Superpixels (max_size=64, cubic)</td>
<td>110.6</td>
<td>116.2</td>
<td>104.3</td>
<td>104.3</td>
</tr>
<tr>
<td>Superpixels (max_size=64, linear)</td>
<td>110.5</td>
<td>116.5</td>
<td>105.0</td>
<td>105.0</td>
</tr>
<tr>
<td>Superpixels (max_size=128, linear)</td>
<td>111.6</td>
<td>116.8</td>
<td>42.1</td>
<td>42.1</td>
</tr>
<tr>
<td>Superpixels (max_size=224, linear)</td>
<td>109.8</td>
<td>116.7</td>
<td>15.2</td>
<td>15.2</td>
</tr>
<tr>
<td>Resize (nearest)</td>
<td>4956.4</td>
<td>11816.3</td>
<td>2683.0</td>
<td>2683.0</td>
</tr>
<tr>
<td>Resize (linear)</td>
<td>4576.0</td>
<td>9708.7</td>
<td>1859.8</td>
<td>1859.8</td>
</tr>
<tr>
<td>Resize (cubic)</td>
<td>4154.6</td>
<td>8181.4</td>
<td>1320.8</td>
<td>1320.8</td>
</tr>
<tr>
<td>CropAndPad</td>
<td>5523.3</td>
<td>8320.0</td>
<td>3843.3</td>
<td>3843.3</td>
</tr>
<tr>
<td>CropAndPad (edge)</td>
<td>5413.8</td>
<td>8410.5</td>
<td>3559.6</td>
<td>3559.6</td>
</tr>
<tr>
<td>CropAndPad (keep_size)</td>
<td>3559.7</td>
<td>4399.1</td>
<td>1607.6</td>
<td>1607.6</td>
</tr>
<tr>
<td>Crop</td>
<td>4956.4</td>
<td>11251.4</td>
<td>4928.9</td>
<td>4928.9</td>
</tr>
<tr>
<td>Crop (keep_size)</td>
<td>4100.6</td>
<td>5196.4</td>
<td>1867.2</td>
<td>1867.2</td>
</tr>
<tr>
<td>Pad</td>
<td>5437.1</td>
<td>8276.8</td>
<td>3366.2</td>
<td>3366.2</td>
</tr>
<tr>
<td>Pad (edge)</td>
<td>5413.8</td>
<td>8180.1</td>
<td>3488.0</td>
<td>3488.0</td>
</tr>
<tr>
<td>Pad (keep_size)</td>
<td>3462.2</td>
<td>4311.5</td>
<td>1542.7</td>
<td>1542.7</td>
</tr>
<tr>
<td>PadToFixedSize</td>
<td>6059.2</td>
<td>52076.2</td>
<td>4615.9</td>
<td>4615.9</td>
</tr>
<tr>
<td>CropToFixedSize</td>
<td>10562.9</td>
<td>208269.5</td>
<td>8347.9</td>
<td>8347.9</td>
</tr>
<tr>
<td>KeepSizeByResize (CropToFixedSize(nearest))</td>
<td>3765.4</td>
<td>14464.6</td>
<td>2455.6</td>
<td>2455.6</td>
</tr>
<tr>
<td>KeepSizeByResize (CropToFixedSize(linear))</td>
<td>3444.2</td>
<td>11868.3</td>
<td>1773.4</td>
<td>1773.4</td>
</tr>
<tr>
<td>KeepSizeByResize (CropToFixedSize(cubic))</td>
<td>3326.9</td>
<td>10122.9</td>
<td>1293.0</td>
<td>1293.0</td>
</tr>
<tr>
<td>FastSnowyLandscape</td>
<td>3061.6</td>
<td>5770.6</td>
<td>579.1</td>
<td>579.1</td>
</tr>
<tr>
<td>Clouds</td>
<td>237.0</td>
<td>280.9</td>
<td>58.9</td>
<td>58.9</td>
</tr>
<tr>
<td>Fog</td>
<td>462.8</td>
<td>491.5</td>
<td>100.8</td>
<td>100.8</td>
</tr>
<tr>
<td>CloudLayer</td>
<td>454.6</td>
<td>488.0</td>
<td>99.3</td>
<td>99.3</td>
</tr>
<tr>
<td>Snowflakes</td>
<td>235.4</td>
<td>287.5</td>
<td>96.5</td>
<td>96.5</td>
</tr>
<tr>
<td>SnowflakesLayer</td>
<td>550.9</td>
<td>574.2</td>
<td>201.5</td>
<td>201.5</td>
</tr>
</tbody>
</table>

### 10.3 Heatmaps and Segmentation Maps

Numbers below are for heatmaps on large images, i.e. 224x224x3. Smaller images were skipped for brevity. The heatmaps themselves can be small (64x64xN) or large (224x224xN), with N denoting the number of heatmaps per...
HeatmapsOnImage instance (i.e. the number of channels in the heatmaps array), for which below 1 and 5 are used. \(B=1\) denotes a batch size of 1, \(B=128\) one of 128.

Segmentation maps have roughly the same values as below, calculate \(N=C\), where \(C\) is the number of classes that are visible in an image (i.e. at least one pixel having that class).

mbit/sec for 64x64x5 or 224x224x5 heatmaps on 224x224x3 images:

<table>
<thead>
<tr>
<th>Augmenter</th>
<th>64x64x5 on 224x224x3</th>
<th>224x224x5 on 224x224x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential (2xNoop)</td>
<td>1479.4</td>
<td>6363.3</td>
</tr>
<tr>
<td>Sequential (2xNoop, random_order)</td>
<td>1304.8</td>
<td>6150.1</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop)</td>
<td>1053.3</td>
<td>6014.7</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop, random_order)</td>
<td>1051.6</td>
<td>6039.6</td>
</tr>
<tr>
<td>OneOf (3xNoop)</td>
<td>1275.8</td>
<td>6027.4</td>
</tr>
<tr>
<td>Sometimes (Noop)</td>
<td>1243.2</td>
<td>6640.8</td>
</tr>
<tr>
<td>WithChannels ([1,2], Noop)</td>
<td>1429.6</td>
<td>6428.3</td>
</tr>
<tr>
<td>Noop</td>
<td>1667.9</td>
<td>6551.2</td>
</tr>
<tr>
<td>Lambda (return input)</td>
<td>1652.2</td>
<td>6437.7</td>
</tr>
<tr>
<td>AssertLambda (return True)</td>
<td>1659.5</td>
<td>6341.9</td>
</tr>
<tr>
<td>AssertShape (None, H, W, None)</td>
<td>1572.9</td>
<td>5953.0</td>
</tr>
<tr>
<td>ChannelShuffle (0.5)</td>
<td>1781.4</td>
<td>5914.0</td>
</tr>
<tr>
<td>Add</td>
<td>1742.4</td>
<td>5666.5</td>
</tr>
<tr>
<td>AddElementwise</td>
<td>1726.9</td>
<td>6293.7</td>
</tr>
<tr>
<td>AdditiveGaussianNoise</td>
<td>1777.7</td>
<td>6244.0</td>
</tr>
<tr>
<td>AdditiveLaplaceNoise</td>
<td>1788.6</td>
<td>6219.2</td>
</tr>
<tr>
<td>AdditivePoissonNoise</td>
<td>1779.1</td>
<td>6246.8</td>
</tr>
<tr>
<td>Multiply</td>
<td>1726.8</td>
<td>6357.2</td>
</tr>
<tr>
<td>MultiplyElementwise</td>
<td>1748.7</td>
<td>6005.0</td>
</tr>
<tr>
<td>Dropout (1-5%)</td>
<td>1722.1</td>
<td>5828.1</td>
</tr>
<tr>
<td>CoarseDropout (1-5%, size=1-10%)</td>
<td>1769.1</td>
<td>6427.0</td>
</tr>
<tr>
<td>ReplaceElementwise</td>
<td>1738.2</td>
<td>6506.4</td>
</tr>
<tr>
<td>ImpulseNoise</td>
<td>1769.9</td>
<td>6457.6</td>
</tr>
<tr>
<td>SaltAndPepper</td>
<td>1750.8</td>
<td>6214.2</td>
</tr>
<tr>
<td>CoarseSaltAndPepper</td>
<td>1750.7</td>
<td>5498.7</td>
</tr>
<tr>
<td>Salt</td>
<td>1710.4</td>
<td>6434.7</td>
</tr>
<tr>
<td>CoarseSalt</td>
<td>1742.9</td>
<td>6470.4</td>
</tr>
<tr>
<td>Pepper</td>
<td>1705.4</td>
<td>6557.1</td>
</tr>
<tr>
<td>CoarsePepper</td>
<td>1778.3</td>
<td>6301.4</td>
</tr>
<tr>
<td>Invert (10%)</td>
<td>1750.6</td>
<td>6428.1</td>
</tr>
<tr>
<td>JpegCompression (50-99%)</td>
<td>1746.6</td>
<td>6544.5</td>
</tr>
<tr>
<td>GaussianBlur (sigma=(1,5))</td>
<td>1698.3</td>
<td>6466.6</td>
</tr>
<tr>
<td>AverageBlur</td>
<td>1717.4</td>
<td>6439.6</td>
</tr>
<tr>
<td>MedianBlur</td>
<td>1777.6</td>
<td>6478.5</td>
</tr>
<tr>
<td>BilateralBlur</td>
<td>1756.9</td>
<td>6536.3</td>
</tr>
<tr>
<td>MotionBlur</td>
<td>1694.3</td>
<td>6475.4</td>
</tr>
<tr>
<td>WithColorspace (HSV, Noop)</td>
<td>1457.7</td>
<td>6274.9</td>
</tr>
<tr>
<td>AddToHueAndSaturation</td>
<td>1733.9</td>
<td>6532.5</td>
</tr>
<tr>
<td>ChangeColorspace (HSV)</td>
<td>1780.0</td>
<td>6441.3</td>
</tr>
<tr>
<td>Grayscale</td>
<td>1756.9</td>
<td>6551.1</td>
</tr>
<tr>
<td>GammaContrast</td>
<td>1762.6</td>
<td>6265.2</td>
</tr>
<tr>
<td>SigmoidContrast</td>
<td>1790.7</td>
<td>6517.0</td>
</tr>
</tbody>
</table>

10.3. Heatmaps and Segmentation Maps
Table 3 – continued from previous page

| Transformation                      | LogContrast | LinearContrast | AllChannelsHistogramEqualization | HistogramEqualization | AllChannelsCLAHE | CLAHE | Convolve (3x3) | Sharpen | Emboss | EdgeDetect | DirectedEdgeDetect | Fliplr (p=100%) | Flipud (p=100%) | Affine (order=0, constant) | Affine (order=1, constant) | Affine (order=3, constant) | Affine (order=1, edge) | Affine (order=1, constant, skimage) | PiecewiseAffine (4x4, order=1, constant) | PiecewiseAffine (4x4, order=0, constant) | PiecewiseAffine (4x4, order=1, edge) | PiecewiseAffine (8x8, order=1, constant) | PerspectiveTransform | PerspectiveTransform (keep_size) | ElasticTransformation (order=0, constant) | ElasticTransformation (order=1, constant) | ElasticTransformation (order=1, nearest) | ElasticTransformation (order=1, reflect) | Rot90 | Rot90 (keep_size) | Superpixels (max_size=64, cubic) | Superpixels (max_size=64, linear) | Superpixels (max_size=128, linear) | Superpixels (max_size=224, linear) | Resize (nearest) | Resize (linear) | Resize (cubic) | CropAndPad | CropAndPad (edge) | CropAndPad (keep_size) | Crop | Crop (keep_size) | Pad | Pad (edge) | Pad (keep_size) | PadToFixedSize | CropToFixedSize | KeepSizeByResize (CropToFixedSize(nearest)) | KeepSizeByResize (CropToFixedSize(linear)) | KeepSizeByResize (CropToFixedSize(cubic)) | |
|-------------------------------------|------------|---------------|-------------------------------|----------------------|-------------------|--------|----------------|---------|--------|------------|---------------------|----------------|----------------|---------------------------------|---------------------------------|-------------------------------|------------------|----------------|----------------------|------------------|----------------|---------------------|-----------------------------|-----------------------------|------------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|                                    | 1758.6     | 6497.1        | 13411.9                       | 27392.9              | 1828.6            | 6292.2 | 13494.0        | 1811.7  | 1770.0 | 1752.4     | 1769.6              | 1526.7            | 1503.2          | 372.9                          | 328.5                          | 210.0                         | 328.1                         | 322.6                         | 31.2                        | 32.1                        | 30.9                        | 7.3                          | 411.5                       | 409.8                         | 97.8                          | 87.3                        | 89.7                        | 85.6                        | 1268.0                     | 816.8                        | 1802.5                        | 1780.8                        | 1795.7                        | 1830.0                        | 768.7                        | 743.1                        | 725.5                        | 773.2                        | 756.0                        | 538.9                        | 1069.1                       | 683.0                        | 669.5                        | 668.2                        | 453.5                        | 668.0                        | 1208.1                       | 656.9                        | 646.8                        | 564.4                        | |
|                                    |            |               |                               |                      |                   |        |                |         |        |            |                     |                 |                |                                |                                 |                              |                               |                              |                                |                             |                             |                             |                             |                             |                             |                             | 6497.1                      | 6397.2                      | 6322.9                      | 6250.0                      | 6292.2                      | 6296.3                      | 6055.7                      | 6070.6                      | 665.6                        | 512.0                        | 277.3                        | 510.0                        | 512.4                        | 33.2                         | 34.4                         | 32.2                         | 7.5                          | 711.2                        | 648.8                        | 119.3                        | 107.1                        | 106.5                        | 106.5                        | 5594.0                       | 1990.1                       | 6319.1                        | 6292.4                        | 6301.3                        | 6365.5                        | 1743.6                        | 1633.5                        | 1523.8                        | 1507.6                        | 1487.3                        | 874.5                         | 2811.4                       | 1190.7                       | 1190.7                       | 1195.1                        | 781.1                        | 2349.2                        | 5238.8                       | 1849.6                        | 1739.8                        | 1611.0                        | 13411.9                     | 13423.8                     | 13124.1                     | 13257.4                     | 13494.0                     | 13195.6                     | 12113.5                     | 12127.4                     | 908.4                        | 682.7                        | 325.1                        | 692.4                        | 698.6                        | 154.6                        | 194.3                        | 152.1                        | 58.1                         | 1569.0                      | 1249.1                       | 1557.0                       | 1409.2                       | 1397.2                       | 1396.9                       | 11084.6                     | 3393.7                       | 13639.6                     | 13722.2                     | 13637.7                     | 13662.5                     | 4230.3                       | 3841.2                       | 3352.9                       | 4164.0                       | 4158.3                       | 2289.5                       | 10339.3                     | 3502.8                       | 2785.5                       | 2732.9                       | 1735.3                       | 3539.8                       | 11100.1                     | 3976.6                       | 3673.7                       | 3221.1                     |
Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Augmenter</th>
<th>64x64x5 on 224x224x3</th>
<th>224x224x5 on 224x224x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential (2xNoop)</td>
<td>11835.1</td>
<td>49090.6</td>
</tr>
<tr>
<td>Sequential (2xNoop, random_order)</td>
<td>10438.4</td>
<td>49200.7</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop)</td>
<td>8426.2</td>
<td>48117.7</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop, random_order)</td>
<td>8413.1</td>
<td>48316.6</td>
</tr>
<tr>
<td>OneOf (3xNoop)</td>
<td>10206.1</td>
<td>48219.5</td>
</tr>
<tr>
<td>Sometimes (Noop)</td>
<td>9945.8</td>
<td>51254.6</td>
</tr>
<tr>
<td>WithChannels ([1,2], Noop)</td>
<td>11437.0</td>
<td>51426.6</td>
</tr>
<tr>
<td>Noop</td>
<td>13343.3</td>
<td>52409.3</td>
</tr>
<tr>
<td>Lambda (return input)</td>
<td>13217.6</td>
<td>51501.6</td>
</tr>
<tr>
<td>AssertLambda (return True)</td>
<td>13275.9</td>
<td>50735.1</td>
</tr>
<tr>
<td>AssertShape (None, H, W, None)</td>
<td>12583.3</td>
<td>47624.0</td>
</tr>
<tr>
<td>ChannelShuffle (0.5)</td>
<td>14251.3</td>
<td>47311.9</td>
</tr>
<tr>
<td>Add</td>
<td>13939.4</td>
<td>45332.0</td>
</tr>
<tr>
<td>AddElementwise</td>
<td>13815.5</td>
<td>50350.0</td>
</tr>
<tr>
<td>AdditiveGaussianNoise</td>
<td>14221.7</td>
<td>49952.2</td>
</tr>
<tr>
<td>AdditiveLaplaceNoise</td>
<td>14309.1</td>
<td>49753.4</td>
</tr>
<tr>
<td>AdditivePoissonNoise</td>
<td>14232.7</td>
<td>49974.0</td>
</tr>
<tr>
<td>Multiply</td>
<td>13814.0</td>
<td>50857.3</td>
</tr>
<tr>
<td>MultiplyElementwise</td>
<td>13989.2</td>
<td>48039.7</td>
</tr>
<tr>
<td>Dropout (1-5%)</td>
<td>13777.1</td>
<td>46625.1</td>
</tr>
<tr>
<td>CoarseDropout (1-5%, size=1-10%)</td>
<td>14152.9</td>
<td>51416.1</td>
</tr>
<tr>
<td>ReplaceElementwise</td>
<td>13905.8</td>
<td>52051.4</td>
</tr>
<tr>
<td>ImpulseNoise</td>
<td>14159.5</td>
<td>51660.6</td>
</tr>
<tr>
<td>SaltAndPepper</td>
<td>14006.2</td>
<td>49713.6</td>
</tr>
<tr>
<td>CoarseSaltAndPepper</td>
<td>14005.4</td>
<td>43989.7</td>
</tr>
<tr>
<td>Salt</td>
<td>13683.5</td>
<td>51477.4</td>
</tr>
<tr>
<td>CoarseSalt</td>
<td>13943.1</td>
<td>51763.4</td>
</tr>
<tr>
<td>Pepper</td>
<td>13643.4</td>
<td>52457.1</td>
</tr>
<tr>
<td>CoarsePepper</td>
<td>14226.6</td>
<td>50411.0</td>
</tr>
<tr>
<td>Invert (10%)</td>
<td>14004.5</td>
<td>51424.8</td>
</tr>
<tr>
<td>JpegCompression (50-99%)</td>
<td>13973.1</td>
<td>52355.6</td>
</tr>
<tr>
<td>GaussianBlur (sigma=(1,5))</td>
<td>13586.3</td>
<td>51732.4</td>
</tr>
<tr>
<td>AverageBlur</td>
<td>13738.8</td>
<td>51516.9</td>
</tr>
<tr>
<td>MedianBlur</td>
<td>14221.2</td>
<td>51828.0</td>
</tr>
<tr>
<td>BilateralBlur</td>
<td>14055.1</td>
<td>52290.3</td>
</tr>
<tr>
<td>MotionBlur</td>
<td>13554.3</td>
<td>51803.5</td>
</tr>
<tr>
<td>WithColorspace (HSV, Noop)</td>
<td>11661.3</td>
<td>50199.6</td>
</tr>
<tr>
<td>ChangeColorspace (HSV)</td>
<td>14240.1</td>
<td>51530.4</td>
</tr>
</tbody>
</table>

Number of heatmap instances per sec for 64x64x5 or 224x224x5 heatmaps on 224x224x3 images:
<table>
<thead>
<tr>
<th>Method</th>
<th>f :: b</th>
<th>g :: b</th>
<th>h :: b</th>
<th>i :: b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grayscale</td>
<td>14055.0</td>
<td>52408.6</td>
<td>8748.3</td>
<td>17829.0</td>
</tr>
<tr>
<td>GammaContrast</td>
<td>14100.6</td>
<td>52521.2</td>
<td>8767.7</td>
<td>17846.9</td>
</tr>
<tr>
<td>SigmoidContrast</td>
<td>14325.2</td>
<td>52135.7</td>
<td>8661.7</td>
<td>17673.5</td>
</tr>
<tr>
<td>LogContrast</td>
<td>14068.5</td>
<td>51976.5</td>
<td>8758.8</td>
<td>17889.2</td>
</tr>
<tr>
<td>LinearContrast</td>
<td>13415.6</td>
<td>51177.2</td>
<td>8766.6</td>
<td>17957.7</td>
</tr>
<tr>
<td>AllChannelsHistogramEqualization</td>
<td>14080.0</td>
<td>50582.8</td>
<td>8570.9</td>
<td>17915.5</td>
</tr>
<tr>
<td>HistogramEqualization</td>
<td>14516.7</td>
<td>50000.3</td>
<td>8657.9</td>
<td>17842.9</td>
</tr>
<tr>
<td>AllChannelsCLAHE</td>
<td>14628.5</td>
<td>50234.0</td>
<td>8812.4</td>
<td>17914.2</td>
</tr>
<tr>
<td>CLAHE</td>
<td>14325.5</td>
<td>50346.2</td>
<td>8715.4</td>
<td>17911.1</td>
</tr>
<tr>
<td>Convolve (3x3)</td>
<td>14566.4</td>
<td>50370.6</td>
<td>8617.5</td>
<td>17833.2</td>
</tr>
<tr>
<td>Sharpen</td>
<td>14493.9</td>
<td>50616.4</td>
<td>8810.0</td>
<td>16916.2</td>
</tr>
<tr>
<td>Emboss</td>
<td>14159.9</td>
<td>50782.9</td>
<td>8548.6</td>
<td>17861.8</td>
</tr>
<tr>
<td>EdgeDetect</td>
<td>14019.0</td>
<td>50627.1</td>
<td>8615.5</td>
<td>17957.7</td>
</tr>
<tr>
<td>DirectedEdgeDetect</td>
<td>14156.7</td>
<td>50690.8</td>
<td>8585.6</td>
<td>17914.2</td>
</tr>
<tr>
<td>Fliplr (p=100%)</td>
<td>12213.4</td>
<td>48445.9</td>
<td>7910.9</td>
<td>17608.6</td>
</tr>
<tr>
<td>Flipud (p=100%)</td>
<td>12025.6</td>
<td>48564.5</td>
<td>7919.9</td>
<td>16887.9</td>
</tr>
<tr>
<td>Affine (order=0, constant)</td>
<td>2982.9</td>
<td>5324.7</td>
<td>593.3</td>
<td>660.9</td>
</tr>
<tr>
<td>Affine (order=1, constant)</td>
<td>2627.8</td>
<td>4096.2</td>
<td>445.8</td>
<td>501.0</td>
</tr>
<tr>
<td>Affine (order=3, constant)</td>
<td>1679.7</td>
<td>2218.2</td>
<td>212.3</td>
<td>482.5</td>
</tr>
<tr>
<td>Affine (order=1, edge)</td>
<td>2624.5</td>
<td>4079.9</td>
<td>452.2</td>
<td>501.0</td>
</tr>
<tr>
<td>Affine (order=1, constant, skimage)</td>
<td>2580.9</td>
<td>4098.8</td>
<td>456.3</td>
<td>501.0</td>
</tr>
<tr>
<td>PiecewiseAffine (4x4, order=1, constant)</td>
<td>249.3</td>
<td>265.9</td>
<td>100.9</td>
<td>212.3</td>
</tr>
<tr>
<td>PiecewiseAffine (4x4, order=0, constant)</td>
<td>256.9</td>
<td>275.5</td>
<td>126.9</td>
<td>212.3</td>
</tr>
<tr>
<td>PiecewiseAffine (4x4, order=1, edge)</td>
<td>247.1</td>
<td>257.6</td>
<td>99.4</td>
<td>212.3</td>
</tr>
<tr>
<td>PiecewiseAffine (8x8, order=1, constant)</td>
<td>58.2</td>
<td>59.6</td>
<td>37.9</td>
<td>212.3</td>
</tr>
<tr>
<td>PerspectiveTransform</td>
<td>3291.9</td>
<td>5689.2</td>
<td>1024.7</td>
<td>212.3</td>
</tr>
<tr>
<td>PerspectiveTransform (keep_size)</td>
<td>3278.6</td>
<td>5190.1</td>
<td>815.8</td>
<td>212.3</td>
</tr>
<tr>
<td>ElasticTransformation (order=0, constant)</td>
<td>782.8</td>
<td>954.3</td>
<td>1016.8</td>
<td>212.3</td>
</tr>
<tr>
<td>ElasticTransformation (order=1, constant)</td>
<td>698.8</td>
<td>856.4</td>
<td>920.3</td>
<td>212.3</td>
</tr>
<tr>
<td>ElasticTransformation (order=1, nearest)</td>
<td>717.9</td>
<td>852.3</td>
<td>912.5</td>
<td>212.3</td>
</tr>
<tr>
<td>ElasticTransformation (order=1, reflect)</td>
<td>684.4</td>
<td>852.0</td>
<td>912.2</td>
<td>212.3</td>
</tr>
<tr>
<td>Rot90 (keep_size)</td>
<td>10143.7</td>
<td>44751.6</td>
<td>7238.9</td>
<td>212.3</td>
</tr>
<tr>
<td>Superpixels (max_size=64, cubic)</td>
<td>14420.2</td>
<td>50553.0</td>
<td>8907.5</td>
<td>212.3</td>
</tr>
<tr>
<td>Superpixels (max_size=64, linear)</td>
<td>14246.3</td>
<td>50339.3</td>
<td>8961.4</td>
<td>212.3</td>
</tr>
<tr>
<td>Superpixels (max_size=128, linear)</td>
<td>14365.2</td>
<td>50410.5</td>
<td>8906.2</td>
<td>212.3</td>
</tr>
<tr>
<td>Superpixels (max_size=224, linear)</td>
<td>14640.2</td>
<td>50923.9</td>
<td>8922.5</td>
<td>212.3</td>
</tr>
<tr>
<td>Resize (nearest)</td>
<td>6149.7</td>
<td>13948.5</td>
<td>2762.6</td>
<td>212.3</td>
</tr>
<tr>
<td>Resize (linear)</td>
<td>5944.5</td>
<td>13067.6</td>
<td>2508.6</td>
<td>212.3</td>
</tr>
<tr>
<td>Resize (cubic)</td>
<td>5804.2</td>
<td>12190.1</td>
<td>2189.6</td>
<td>212.3</td>
</tr>
<tr>
<td>CropAndPad</td>
<td>6185.8</td>
<td>12060.5</td>
<td>2719.4</td>
<td>212.3</td>
</tr>
<tr>
<td>CropAndPad (edge)</td>
<td>6048.2</td>
<td>11898.6</td>
<td>2715.6</td>
<td>212.3</td>
</tr>
<tr>
<td>CropAndPad (keep_size)</td>
<td>4311.0</td>
<td>6996.1</td>
<td>1495.2</td>
<td>212.3</td>
</tr>
<tr>
<td>Crop</td>
<td>8553.1</td>
<td>22491.2</td>
<td>6752.2</td>
<td>212.3</td>
</tr>
<tr>
<td>Crop (keep_size)</td>
<td>5464.3</td>
<td>9525.2</td>
<td>2287.6</td>
<td>212.3</td>
</tr>
<tr>
<td>Pad</td>
<td>5356.0</td>
<td>9525.6</td>
<td>1819.1</td>
<td>212.3</td>
</tr>
<tr>
<td>Pad (edge)</td>
<td>5345.5</td>
<td>9560.5</td>
<td>1784.7</td>
<td>212.3</td>
</tr>
<tr>
<td>Pad (keep_size)</td>
<td>3627.7</td>
<td>6249.1</td>
<td>1133.3</td>
<td>212.3</td>
</tr>
<tr>
<td>PadToFixedSize</td>
<td>5343.7</td>
<td>18793.3</td>
<td>2311.7</td>
<td>212.3</td>
</tr>
<tr>
<td>CropToFixedSize</td>
<td>9664.7</td>
<td>41910.7</td>
<td>7249.1</td>
<td>212.3</td>
</tr>
</tbody>
</table>
### Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Augmenter</th>
<th>B=1</th>
<th>B=128</th>
<th>B=128</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeepSizeByResize (CropToFixedSize(nearest))</td>
<td>5254.9</td>
<td>14797.0</td>
<td>2597.0</td>
</tr>
<tr>
<td>KeepSizeByResize (CropToFixedSize(linear))</td>
<td>5174.6</td>
<td>13918.2</td>
<td>2399.2</td>
</tr>
<tr>
<td>KeepSizeByResize (CropToFixedSize(cubic))</td>
<td>4515.1</td>
<td>12887.7</td>
<td>2103.6</td>
</tr>
<tr>
<td>FastSnowyLandscape</td>
<td>12297.5</td>
<td>47140.1</td>
<td>6408.3</td>
</tr>
<tr>
<td>Clouds</td>
<td>8969.6</td>
<td>50297.1</td>
<td>8826.7</td>
</tr>
<tr>
<td>Fog</td>
<td>13289.3</td>
<td>51749.8</td>
<td>8684.1</td>
</tr>
<tr>
<td>CloudLayer</td>
<td>13663.1</td>
<td>52331.6</td>
<td>8786.1</td>
</tr>
<tr>
<td>Snowflakes</td>
<td>7469.9</td>
<td>47628.2</td>
<td>6232.2</td>
</tr>
<tr>
<td>SnowflakesLayer</td>
<td>13083.2</td>
<td>50715.4</td>
<td>8729.0</td>
</tr>
</tbody>
</table>

### 10.4 Keypoints and Bounding Boxes

Numbers below are for keypoints on small and large images. Each `KeypointsOnImage` instance contained 10 Keypoint instances. \( B=1 \) denotes a batch size of 1, \( B=128 \) one of 128.

The numbers for bounding boxes can be derived by dividing each value by 4.

**Number of augmented Keypoint instances per sec** (divide by 10 for `KeypointsOnImage` instances):

<table>
<thead>
<tr>
<th>Augmenter</th>
<th>10 KPs on 64x64x3</th>
<th>10 KPs on 224x224x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential (2xNoop)</td>
<td>37410.7</td>
<td>853875.7</td>
</tr>
<tr>
<td>Sequential (2xNoop, random_order)</td>
<td>30112.0</td>
<td>815035.0</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop)</td>
<td>22123.5</td>
<td>527171.0</td>
</tr>
<tr>
<td>SomeOf (1-3, 3xNoop, random_order)</td>
<td>20802.7</td>
<td>501859.7</td>
</tr>
<tr>
<td>OneOf (3xNoop)</td>
<td>29834.9</td>
<td>572531.6</td>
</tr>
<tr>
<td>Sometimes (Noop)</td>
<td>29447.1</td>
<td>809486.6</td>
</tr>
<tr>
<td>WithChannels ([1,2], Noop)</td>
<td>38531.5</td>
<td>838030.6</td>
</tr>
<tr>
<td>Noop</td>
<td>54909.3</td>
<td>1047763.6</td>
</tr>
<tr>
<td>Lambda (return input)</td>
<td>52003.7</td>
<td>996286.5</td>
</tr>
<tr>
<td>AssertLambda (return True)</td>
<td>49527.7</td>
<td>1013667.8</td>
</tr>
<tr>
<td>AssertShape (None, H, W, None)</td>
<td>43618.4</td>
<td>718468.0</td>
</tr>
<tr>
<td>ChannelShuffle (0.5)</td>
<td>57980.4</td>
<td>1071042.8</td>
</tr>
<tr>
<td>Add</td>
<td>54651.8</td>
<td>1018748.9</td>
</tr>
<tr>
<td>AddElementwise</td>
<td>58137.9</td>
<td>1032378.5</td>
</tr>
<tr>
<td>AdditiveGaussianNoise</td>
<td>58192.8</td>
<td>1052636.5</td>
</tr>
<tr>
<td>AdditiveLaplaceNoise</td>
<td>56757.2</td>
<td>1074031.8</td>
</tr>
<tr>
<td>AdditivePoissonNoise</td>
<td>58088.0</td>
<td>1076420.0</td>
</tr>
<tr>
<td>Multiply</td>
<td>56156.2</td>
<td>1067550.0</td>
</tr>
<tr>
<td>MultiplyElementwise</td>
<td>56831.8</td>
<td>1072862.1</td>
</tr>
<tr>
<td>Dropout (1-5%)</td>
<td>57088.7</td>
<td>1034250.0</td>
</tr>
<tr>
<td>CoarseDropout (1-5%, size=1-10%)</td>
<td>56938.3</td>
<td>1049754.9</td>
</tr>
<tr>
<td>ReplaceElementwise</td>
<td>55916.6</td>
<td>1087917.4</td>
</tr>
<tr>
<td>ImpulseNoise</td>
<td>56378.1</td>
<td>1042935.6</td>
</tr>
<tr>
<td>SaltAndPepper</td>
<td>57258.6</td>
<td>1084701.6</td>
</tr>
<tr>
<td>CoarseSaltAndPepper</td>
<td>56112.6</td>
<td>1035994.3</td>
</tr>
<tr>
<td>Salt</td>
<td>55506.6</td>
<td>1034198.2</td>
</tr>
<tr>
<td>CoarseSalt</td>
<td>57707.6</td>
<td>1062281.9</td>
</tr>
<tr>
<td>Pepper</td>
<td>55731.6</td>
<td>1065780.4</td>
</tr>
</tbody>
</table>

10.4. Keypoints and Bounding Boxes 127
<table>
<thead>
<tr>
<th>CoarsePepper</th>
<th>57944.4</th>
<th>1047440.6</th>
<th>57294.5</th>
<th>1066693.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invert (10%)</td>
<td>59806.7</td>
<td>1013199.1</td>
<td>57902.0</td>
<td>1050541.7</td>
</tr>
<tr>
<td>JpegCompression (50-99%)</td>
<td>58660.0</td>
<td>1033913.4</td>
<td>58774.2</td>
<td>1083270.3</td>
</tr>
<tr>
<td>GaussianBlur (sigma=(1,5))</td>
<td>61680.0</td>
<td>1062936.0</td>
<td>60120.5</td>
<td>1076428.6</td>
</tr>
<tr>
<td>AverageBlur</td>
<td>57778.4</td>
<td>1053638.4</td>
<td>57689.3</td>
<td>1079853.8</td>
</tr>
<tr>
<td>MedianBlur</td>
<td>55349.9</td>
<td>108328.9</td>
<td>58755.3</td>
<td>1096220.9</td>
</tr>
<tr>
<td>BilateralBlur</td>
<td>55311.2</td>
<td>1066606.2</td>
<td>58092.9</td>
<td>1098475.1</td>
</tr>
<tr>
<td>MotionBlur</td>
<td>59827.2</td>
<td>1068565.7</td>
<td>54753.1</td>
<td>1044380.1</td>
</tr>
<tr>
<td>WithColorspace (HSV, Noop)</td>
<td>40309.3</td>
<td>827453.6</td>
<td>38397.4</td>
<td>791654.7</td>
</tr>
<tr>
<td>AddToHueAndSaturation</td>
<td>56029.4</td>
<td>1031482.0</td>
<td>58362.0</td>
<td>1055056.8</td>
</tr>
<tr>
<td>ChangeColorspace (HSV)</td>
<td>55347.7</td>
<td>1061282.4</td>
<td>57914.8</td>
<td>1092286.7</td>
</tr>
<tr>
<td>Grayscale</td>
<td>58657.5</td>
<td>1025570.8</td>
<td>58755.3</td>
<td>1045425.4</td>
</tr>
<tr>
<td>GammaContrast</td>
<td>56351.6</td>
<td>1056732.7</td>
<td>60808.2</td>
<td>1081622.5</td>
</tr>
<tr>
<td>SigmoidContrast</td>
<td>56895.8</td>
<td>1028902.3</td>
<td>58092.9</td>
<td>1096220.9</td>
</tr>
<tr>
<td>LogContrast</td>
<td>57724.3</td>
<td>1027712.6</td>
<td>58612.4</td>
<td>1098475.1</td>
</tr>
<tr>
<td>LinearContrast</td>
<td>57850.1</td>
<td>1011155.4</td>
<td>54755.2</td>
<td>1057423.7</td>
</tr>
<tr>
<td>AllChannelsHistogramEqualization</td>
<td>59571.4</td>
<td>1050899.5</td>
<td>57768.0</td>
<td>1058921.2</td>
</tr>
<tr>
<td>HistogramEqualization</td>
<td>55952.4</td>
<td>1060276.3</td>
<td>55578.0</td>
<td>1086884.4</td>
</tr>
<tr>
<td>AllChannelsCLAHE</td>
<td>58632.9</td>
<td>1064765.8</td>
<td>54470.1</td>
<td>1054027.8</td>
</tr>
<tr>
<td>CLAHE</td>
<td>56905.9</td>
<td>1014583.5</td>
<td>56590.3</td>
<td>1011919.6</td>
</tr>
<tr>
<td>Convolve (3x3)</td>
<td>57145.4</td>
<td>1030078.9</td>
<td>55825.8</td>
<td>1112199.5</td>
</tr>
<tr>
<td>Sharpen</td>
<td>59562.1</td>
<td>1003078.9</td>
<td>55825.8</td>
<td>1112199.5</td>
</tr>
<tr>
<td>Emboss</td>
<td>56636.2</td>
<td>1049510.7</td>
<td>57317.2</td>
<td>1086884.4</td>
</tr>
<tr>
<td>EdgeDetect</td>
<td>57702.8</td>
<td>1082004.0</td>
<td>56889.9</td>
<td>107768.0</td>
</tr>
<tr>
<td>DirectedEdgeDetect</td>
<td>59083.9</td>
<td>1086244.5</td>
<td>55578.0</td>
<td>1086884.4</td>
</tr>
<tr>
<td>Flpir (p=100%)</td>
<td>45716.5</td>
<td>840602.4</td>
<td>84006.2</td>
<td>54444.4</td>
</tr>
<tr>
<td>Flipud (p=100%)</td>
<td>47017.1</td>
<td>803220.4</td>
<td>55578.0</td>
<td>54544.4</td>
</tr>
<tr>
<td>Affine (order=0, constant)</td>
<td>12320.8</td>
<td>49152.4</td>
<td>49152.4</td>
<td>13374.9</td>
</tr>
<tr>
<td>Affine (order=1, constant)</td>
<td>12819.3</td>
<td>51745.3</td>
<td>51745.3</td>
<td>12648.8</td>
</tr>
<tr>
<td>Affine (order=3, constant)</td>
<td>12660.6</td>
<td>50627.6</td>
<td>50627.6</td>
<td>13076.4</td>
</tr>
<tr>
<td>Affine (order=1, edge)</td>
<td>13023.4</td>
<td>49492.4</td>
<td>49492.4</td>
<td>13572.6</td>
</tr>
<tr>
<td>Affine (order=1, constant, skimage)</td>
<td>12856.8</td>
<td>49884.6</td>
<td>49884.6</td>
<td>13520.0</td>
</tr>
<tr>
<td>PiecewiseAffine (4x4, order=1, constant)</td>
<td>422.1</td>
<td>439.7</td>
<td>98.8</td>
<td></td>
</tr>
<tr>
<td>PiecewiseAffine (4x4, order=0, constant)</td>
<td>427.3</td>
<td>440.8</td>
<td>103.4</td>
<td></td>
</tr>
<tr>
<td>PiecewiseAffine (4x4, order=1, edge)</td>
<td>428.6</td>
<td>436.3</td>
<td>101.2</td>
<td></td>
</tr>
<tr>
<td>PiecewiseAffine (8x8, order=1, constant)</td>
<td>110.8</td>
<td>111.1</td>
<td>56.5</td>
<td></td>
</tr>
<tr>
<td>PerspectiveTransform</td>
<td>16063.4</td>
<td>43433.2</td>
<td>154983.5</td>
<td>385498.3</td>
</tr>
<tr>
<td>PerspectiveTransform (keep_size)</td>
<td>13844.3</td>
<td>32606.6</td>
<td>13589.3</td>
<td>385498.3</td>
</tr>
<tr>
<td>ElasticTransformation (order=0, constant)</td>
<td>1723.3</td>
<td>1737.5</td>
<td>1172.8</td>
<td></td>
</tr>
<tr>
<td>ElasticTransformation (order=1, constant)</td>
<td>1770.6</td>
<td>1698.3</td>
<td>1094.2</td>
<td></td>
</tr>
<tr>
<td>ElasticTransformation (order=1, nearest)</td>
<td>1743.1</td>
<td>1713.8</td>
<td>1269.7</td>
<td></td>
</tr>
<tr>
<td>ElasticTransformation (order=1, reflect)</td>
<td>1698.1</td>
<td>1765.5</td>
<td>1143.8</td>
<td></td>
</tr>
<tr>
<td>Rot90</td>
<td>33662.4</td>
<td>433458.2</td>
<td>32571.8</td>
<td>32571.8</td>
</tr>
<tr>
<td>Rot90 (keep_size)</td>
<td>33951.8</td>
<td>423581.5</td>
<td>34418.2</td>
<td>34418.2</td>
</tr>
<tr>
<td>Superpixels (max_size=64, cubic)</td>
<td>57351.7</td>
<td>1074717.7</td>
<td>58752.8</td>
<td></td>
</tr>
<tr>
<td>Superpixels (max_size=64, linear)</td>
<td>58975.9</td>
<td>113847.0</td>
<td>58592.8</td>
<td></td>
</tr>
<tr>
<td>Superpixels (max_size=128, linear)</td>
<td>55706.4</td>
<td>1066790.6</td>
<td>59390.1</td>
<td></td>
</tr>
<tr>
<td>Superpixels (max_size=224, linear)</td>
<td>58688.7</td>
<td>1096574.7</td>
<td>59021.5</td>
<td></td>
</tr>
<tr>
<td>Resize (nearest)</td>
<td>28635.7</td>
<td>241052.8</td>
<td>27852.1</td>
<td>26416.3</td>
</tr>
<tr>
<td>Resize (linear)</td>
<td>30130.4</td>
<td>252290.0</td>
<td>26416.3</td>
<td>26416.3</td>
</tr>
<tr>
<td>Function</td>
<td>Time (s) 1</td>
<td>Time (s) 2</td>
<td>Time (s) 3</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>Resize (cubic)</td>
<td>29017.9</td>
<td>257344.5</td>
<td>27124.4</td>
<td></td>
</tr>
<tr>
<td>CropAndPad</td>
<td>23663.2</td>
<td>82977.4</td>
<td>23776.2</td>
<td></td>
</tr>
<tr>
<td>CropAndPad (edge)</td>
<td>23557.0</td>
<td>82546.3</td>
<td>23187.2</td>
<td></td>
</tr>
<tr>
<td>CropAndPad (keep_size)</td>
<td>22563.7</td>
<td>72779.7</td>
<td>22194.6</td>
<td></td>
</tr>
<tr>
<td>Crop</td>
<td>24477.4</td>
<td>93390.4</td>
<td>25635.4</td>
<td></td>
</tr>
<tr>
<td>Crop (keep_size)</td>
<td>23008.8</td>
<td>79362.3</td>
<td>23809.6</td>
<td></td>
</tr>
<tr>
<td>Pad</td>
<td>22378.7</td>
<td>80932.3</td>
<td>23570.4</td>
<td></td>
</tr>
<tr>
<td>Pad (edge)</td>
<td>22584.1</td>
<td>80898.2</td>
<td>23788.3</td>
<td></td>
</tr>
<tr>
<td>Pad (keep_size)</td>
<td>22099.9</td>
<td>70070.6</td>
<td>22519.6</td>
<td></td>
</tr>
<tr>
<td>PadToFixedSize</td>
<td>24045.9</td>
<td>452782.8</td>
<td>24910.9</td>
<td></td>
</tr>
<tr>
<td>CropToFixedSize</td>
<td>28185.3</td>
<td>468207.4</td>
<td>28213.1</td>
<td></td>
</tr>
<tr>
<td>KeepSizeByResize (CropToFixedSize(nearest))</td>
<td>19321.7</td>
<td>270169.7</td>
<td>21448.0</td>
<td></td>
</tr>
<tr>
<td>KeepSizeByResize (CropToFixedSize(linear))</td>
<td>20209.9</td>
<td>267528.0</td>
<td>20008.6</td>
<td></td>
</tr>
<tr>
<td>KeepSizeByResize (CropToFixedSize(cubic))</td>
<td>19883.9</td>
<td>266581.9</td>
<td>20303.9</td>
<td></td>
</tr>
<tr>
<td>FastSnowyLandscape</td>
<td>56019.7</td>
<td>1044994.0</td>
<td>62901.0</td>
<td></td>
</tr>
<tr>
<td>Clouds</td>
<td>22275.3</td>
<td>548719.4</td>
<td>23773.6</td>
<td></td>
</tr>
<tr>
<td>Fog</td>
<td>54518.9</td>
<td>1037778.6</td>
<td>56352.3</td>
<td></td>
</tr>
<tr>
<td>CloudLayer</td>
<td>55555.9</td>
<td>1099284.8</td>
<td>62219.0</td>
<td></td>
</tr>
<tr>
<td>Snowflakes</td>
<td>22950.4</td>
<td>538387.0</td>
<td>21907.2</td>
<td></td>
</tr>
<tr>
<td>SnowflakesLayer</td>
<td>57931.6</td>
<td>1080686.3</td>
<td>56144.9</td>
<td></td>
</tr>
</tbody>
</table>
The function `augment_images()` which all augmenters in `imgaug` offer, works by default with numpy arrays. In most cases, these arrays will have the numpy dtype `uint8`, i.e. the images will have values in the range `[0, 255]`. This is the datatype returned by most image loading functions. Sometimes however you may want to augment other datatypes, e.g. `float64`. While all augmenters support `uint8`, the support for other datatypes varies. The tables further below show which datatype is supported by which augmenter (alongside the dtype support in some helper functions). The API of each augmenter may contain more details.

Note: Whenever possible it is suggested to use `uint8` as that is the most thoroughly tested dtype. In general, the use of large dtypes (i.e. `uint64`, `int64`, `float128`) is discouraged, even when they are marked as supported. That is because writing correct tests for these dtypes can be difficult as no larger dtypes are available to which values can be temporarily cast. Additionally, when using inputs for which precise discrete values are important (e.g. segmentation maps, where an accidental change by 1 would break the map), medium sized dtypes (`uint32`, `int32`) should be used with caution. This is because other libraries may convert temporarily to `float64`, which could lead to inaccuracies for some numbers.

### 11.1 Legend

**Support level** (color of table cells):
- Green: Datatype is considered supported by the augmenter.
- Yellow: Limited support for the datatype, e.g. due to inaccuracies around large values. See the API for the respective augmenter for more details.
- Red: Datatype is not supported by the augmenter.

**Test level** (symbols in table cells):
- `+++`: Datatype support is thoroughly tested (via unittests or integration tests).
- `++`: Datatype support is tested, though not thoroughly.
- `+`: Datatype support is indirectly tested via tests for other augmenters.
- `-`: Datatype support is not tested.
11.2 imgaug helper functions

Fig. 1: Dtype support of helper functions in imgaug, e.g. `import imgaug; imgaug.imresize_single_image(array, size)`.

11.3 imgaug.augmenters.meta

Fig. 2: Dtype support for `augment_images(arrays)`, `augment_image(arr)` and helper functions in `imgaug.augmenters.meta`.
**Fig. 3:** Dtype support for `augment_images(arrays)`, `augment_image(arr)` and helper functions in `imgaug.augmenters.arithmetic`.
11.4 imgaug.augmenters.arithmetic

11.5 imgaug.augmenters.blur

<table>
<thead>
<tr>
<th></th>
<th>uint8</th>
<th>uint16</th>
<th>uint32</th>
<th>uint64</th>
<th>int8</th>
<th>int16</th>
<th>int32</th>
<th>int64</th>
<th>float16</th>
<th>float32</th>
<th>float64</th>
<th>float128</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>GaussianBlur</td>
<td>++++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>AverageBlur</td>
<td>++++</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>MotionBlur</td>
<td>++++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

Fig. 4: Dtype support for `augment_images(arrays)`, `augment_image(arr)` and helper functions in `imgaug.augmenters.blur`.

11.6 imgaug.augmenters.color

<table>
<thead>
<tr>
<th></th>
<th>uint8</th>
<th>uint16</th>
<th>uint32</th>
<th>uint64</th>
<th>int8</th>
<th>int16</th>
<th>int32</th>
<th>int64</th>
<th>float16</th>
<th>float32</th>
<th>float64</th>
<th>float128</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddToHueAndSaturation</td>
<td>++++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
</tr>
</tbody>
</table>

Fig. 5: Dtype support for `augment_images(arrays)`, `augment_image(arr)` and helper functions in `imgaug.augmenters.color`.

11.7 imgaug.augmenters.contrast

<table>
<thead>
<tr>
<th></th>
<th>uint8</th>
<th>uint16</th>
<th>uint32</th>
<th>uint64</th>
<th>int8</th>
<th>int16</th>
<th>int32</th>
<th>int64</th>
<th>float16</th>
<th>float32</th>
<th>float64</th>
<th>float128</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>GammaContrast</td>
<td>++++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>SigmoidContrast</td>
<td>++++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>LogContrast</td>
<td>++++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>LinearContrast</td>
<td>++++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>ARChannelsCLAHE</td>
<td>++++</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>CLAHE</td>
<td>++++</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>ARChannelsHistogramEqualization</td>
<td>++++</td>
<td>+</td>
<td>+++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>HistogramEqualization</td>
<td>++++</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Fig. 6: Dtype support for `augment_images(arrays)`, `augment_image(arr)` and helper functions in `imgaug.augmenters.contrast`.
Fig. 7: Dtype support for `augment_images(arrays)`, `augment_image(arr)` and helper functions in `imgaug.augmenters.convolutional`.

Fig. 8: Dtype support for `augment_images(arrays)`, `augment_image(arr)` and helper functions in `imgaug.augmenters.flip`.

11.8 imgaug.augmenters.convolutional

11.9 imgaug.augmenters.flip

11.10 imgaug.augmenters.geometric

Fig. 9: Dtype support for `augment_images(arrays)`, `augment_image(arr)` and helper functions in `imgaug.augmenters.geometric`.
### Chapter 11. dtype support

#### 11.11 imgaug.augmenters.overlay

#### 11.12 imgaug.augmenters.segmentation

#### 11.13 imgaug.augmenters.size

#### 11.14 imgaug.augmenters.weather

---

#### 11.11 imgaug.augmenters.overlay

#### 11.12 imgaug.augmenters.segmentation

#### 11.13 imgaug.augmenters.size

#### 11.14 imgaug.augmenters.weather

---

#### 11.11 imgaug.augmenters.overlay

#### 11.12 imgaug.augmenters.segmentation

#### 11.13 imgaug.augmenters.size

#### 11.14 imgaug.augmenters.weather
Fig. 13: Dtype support for `augment_images(arrays)`, `augment_image(arr)` and helper functions in `imgaug.augmenters.weather`.

<table>
<thead>
<tr>
<th>Augmenter</th>
<th>uint8</th>
<th>uint16</th>
<th>uint32</th>
<th>uint64</th>
<th>int8</th>
<th>int16</th>
<th>int32</th>
<th>int64</th>
<th>float16</th>
<th>float32</th>
<th>float64</th>
<th>float128</th>
<th>bool</th>
</tr>
</thead>
<tbody>
<tr>
<td>FastSnowyLandscape</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Clouds</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Fog</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CloudLayer</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Snowflakes</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SnowflakesLayer</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Several Jupyter notebooks are available that provide tutorials about *imgaug*’s functionality. They are hosted at `imgaug-doc/notebooks`. The notebooks can be downloaded to interactively modify the examples.

**List of Notebooks:**

- A01 - Load and Augment an Image
- A02 - Stochastic and Deterministic Augmentation
- A03 - Multicore Augmentation
- B01 - Augment Keypoints (aka Landmarks)
- B02 - Augment Bounding Boxes
- B03 - Augment Polygons
- B06 - Augment Line Strings
- B04 - Augment Heatmaps
- B05 - Augment Segmentation Maps
- C01 - Using Probability Distributions as Parameters
- C02 - Using *imgaug* with more Control Flow
- C03 - Copying Random States and Using Multiple Augmentation Sequences
13.1 imgaug

```python
imgaug.imgaug.BackgroundAugmenter(*args, **kwargs)
imgaug.imgaug.Batch(*args, **kwargs)
imgaug.imgaug.BatchLoader(*args, **kwargs)
imgaug.imgaug.BoundingBox(*args, **kwargs)
imgaug.imgaug.BoundingBoxesOnImage(*args, **kwargs)
```

```
exception imgaug.imgaug.DeprecationWarning
    Bases: Warning
    Warning for deprecated calls.
    Since python 2.7 DeprecationWarning is silent by default. So we define our own DeprecationWarning here so that it is not silent by default.
```

```python
imgaug.imgaug.HeatmapsOnImage(*args, **kwargs)
```

```python
class imgaug.imgaug.HooksHeatmaps(activator=None, propagator=None, preprocessor=None, postprocessor=None)
    Bases: imgaug.imgaug.HooksImages
    Class to intervene with heatmap augmentation runs.
    This is e.g. useful to dynamically deactivate some augmenters.
    This class is currently the same as the one for images. This may or may not change in the future.
```

Methods
The `imgaug.HooksImages` class allows for dynamic control over the execution of image augmenters. It provides four methods for intervention:

- `is_activated(images, augmenter, parents, default)` returns whether an augmenter may be executed.
- `is_propagating(images, augmenter, parents, ...)` returns whether an augmenter may call its children to augment an image.
- `postprocess(images, augmenter, parents)` is a function to be called after the augmentation of images was performed.
- `preprocess(images,34 augmenter, parents)` is a function to be called before the augmentation of images starts (per augmenter).

```python
class imgaug.imgaug.HooksImages(activator=None, propagator=None, preprocessor=None, postprocessor=None)
    Bases: object

    Class to intervene with image augmentation runs.

    This is e.g. useful to dynamically deactivate some augmenters.

    Parameters

    • **activator** *(None or callable, optional)* – A function that gives permission to execute an augmenter. The expected interface is `f(images, augmenter, parents, default)`, where `images` are the input images to augment, `augmenter` is the instance of the augmenter to execute, `parents` are previously executed augmenters and `default` is an expected default value to be returned if the activator function does not plan to make a decision for the given inputs.

    • **propagator** *(None or callable, optional)* – A function that gives permission to propagate the augmentation further to the children of an augmenter. This happens after the activator. In theory, an augmenter may augment images itself (if allowed by the activator) and then execute child augmenters afterwards (if allowed by the propagator). If the activator returned False, the propagation step will never be executed. The expected interface is `f(images, augmenter, parents, default)`, with all arguments having identical meaning to the activator.

    • **preprocessor** *(None or callable, optional)* – A function to call before an augmenter performed any augmentations. The interface is `f(images, augmenter, parents)`, with all arguments having identical meaning to the activator. It is expected to return the input images, optionally modified.

    • **postprocessor** *(None or callable, optional)* – A function to call after an augmenter performed augmentations. The interface is the same as for the preprocessor.

Examples

```python
>>> seq = iaa.Sequential([
    iaa.GaussianBlur(3.0, name="blur"),
    iaa.Dropout(0.05, name="dropout"),
    iaa.Affine(translate_px=-5, name="affine")
])
>>> images = [np.zeros((10, 10), dtype=np.uint8)]
>>> def activator(images, augmenter, parents, default):
...    return False if augmenter.name in ["blur", "dropout"] else default
...>>> seq_det = seq.to_deterministic()
>>> images_aug = seq_det.augment_images(images)
```
This augments images and their respective heatmaps in the same way. The heatmaps however are only modified by Affine, not by GaussianBlur or Dropout.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Returns Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>is_activated</strong></td>
<td>Returns whether an augmenter may be executed.</td>
<td>bool</td>
</tr>
<tr>
<td><strong>is_propagating</strong></td>
<td>Returns whether an augmenter may call its children to augment an image.</td>
<td>bool</td>
</tr>
<tr>
<td><strong>postprocess</strong></td>
<td>A function to be called after the augmentation of images was performed.</td>
<td>(N,H,W,C) ndarray or (N,H,W) ndarray or list of (H,W,C) ndarray or list of (H,W) ndarray</td>
</tr>
<tr>
<td><strong>preprocess</strong></td>
<td>A function to be called before the augmentation of images starts (per augmenter).</td>
<td>(N,H,W,C) ndarray or (N,H,W) ndarray or list of (H,W,C) ndarray or list of (H,W) ndarray</td>
</tr>
</tbody>
</table>

#### is_activated

Returns whether an augmenter may be executed.

- **Returns**: If True, the augmenter may be executed. If False, it may not be executed.
- **Return type**: bool

#### is_propagating

Returns whether an augmenter may call its children to augment an image. This is independent of the augmenter itself possible changing the image, without calling its children. (Most (all?) augmenters with children currently dont perform any changes themselves.)

- **Returns**: If True, the augmenter may be propagate to its children. If False, it may not.
- **Return type**: bool

#### postprocess

A function to be called after the augmentation of images was performed.

- **Returns**: The input images, optionally modified.
- **Return type**: (N,H,W,C) ndarray or (N,H,W) ndarray or list of (H,W,C) ndarray or list of (H,W) ndarray

#### preprocess

A function to be called before the augmentation of images starts (per augmenter).

- **Returns**: The input images, optionally modified.
- **Return type**: (N,H,W,C) ndarray or (N,H,W) ndarray or list of (H,W,C) ndarray or list of (H,W) ndarray

### class

**imgaug.imgaug.HooksKeypoints**

Bases: `imgaug.imgaug.HooksImages`

Class to intervene with keypoint augmentation runs.
This is e.g. useful to dynamically deactivate some augmenters.
This class is currently the same as the one for images. This may or may not change in the future.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_activated(images, augmenter, parents, default)</td>
<td>Returns whether an augmenter may be executed.</td>
</tr>
<tr>
<td>is_propagating(images, augmenter, parents, ...)</td>
<td>Returns whether an augmenter may call its children to augment an image.</td>
</tr>
<tr>
<td>postprocess(images, augmenter, parents)</td>
<td>A function to be called after the augmentation of images was performed.</td>
</tr>
<tr>
<td>preprocess(images, augmenter, parents)</td>
<td>A function to be called before the augmentation of images starts (per augmenter).</td>
</tr>
</tbody>
</table>

*imgaug.imgaug.Keypoint(*args, **kwargs)*

*imgaug.imgaug.KeypointsOnImage(*args, **kwargs)*

*imgaug.imgaug.MultiPolygon(*args, **kwargs)*

*imgaug.imgaug.Polygon(*args, **kwargs)*

*imgaug.imgaug.PolygonsOnImage(*args, **kwargs)*

*imgaug.imgaug.SegmentationMapOnImage(*args, **kwargs)*

*imgaug.imgaug.angle_between_vectors(v1, v2)*

Returns the angle in radians between vectors v1 and v2.


**Parameters**

- v1 ((N,) ndarray) – First vector.
- v2 ((N,) ndarray) – Second vector.

**Returns** out – Angle in radians.

**Return type** float

**Examples**

```python
>>> angle_between_vectors(np.float32([1, 0, 0]), np.float32([0, 1, 0]))
1.570796...

>>> angle_between_vectors(np.float32([1, 0, 0]), np.float32([1, 0, 0]))
0.0

>>> angle_between_vectors(np.float32([1, 0, 0]), np.float32([-1, 0, 0]))
3.141592...
```

*imgaug.imgaug.avg_pool(arr, block_size, cval=0, preserve_dtype=True)*

Resize an array using average pooling.

**dtype support:**
Parameters

- **arr** 
  
  \((H, W)\) ndarray or \((H, W, C)\) ndarray – Image-like array to pool. See `imgaug.imgaug.pool()` for details.

- **block_size** 
  
  \(\text{(int or tuple of int or tuple of int)}\) – Size of each block of values to pool. See `imgaug.imgaug.pool()` for details.

- **cval** 
  
  \(\text{(number, optional)}\) – Padding value. See `imgaug.imgaug.pool()` for details.

- **preserve_dtype** 
  
  \(\text{(bool, optional)}\) – Whether to preserve the input array dtype. See `imgaug.imgaug.pool()` for details.

Returns **arr_reduced** – Array after average pooling.

Return type 

\((H', W')\) ndarray or \((H', W', C')\) ndarray

#### imgaug.imgaug.caller_name()

Returns the name of the caller, e.g. a function.

Returns 

The name of the caller as a string

Return type 

str

#### imgaug.imgaug.compute_geometric_median(*args, **kwargs)

Compute the intersection point of two lines.

Taken from https://stackoverflow.com/a/20679579.

Parameters

- **x1**
  
  \(\text{(number)}\) – x coordinate of the first point on line 1. (The lines extends beyond this point.)

- **y1**
  
  \(\text{(number)}\) – y coordinate of the first point on line 1. (The lines extends beyond this point.)

- **x2**
  
  \(\text{(number)}\) – x coordinate of the second point on line 1. (The lines extends beyond this point.)

- **y2**
  
  \(\text{(number)}\) – y coordinate of the second point on line 1. (The lines extends beyond this point.)

- **x3**
  
  \(\text{(number)}\) – x coordinate of the first point on line 2. (The lines extends beyond this point.)

- **y3**
  
  \(\text{(number)}\) – y coordinate of the first point on line 2. (The lines extends beyond this point.)

- **x4**
  
  \(\text{(number)}\) – x coordinate of the second point on line 2. (The lines extends beyond this point.)

- **y4**
  
  \(\text{(number)}\) – y coordinate of the second point on line 2. (The lines extends beyond this point.)

Returns 

The coordinate of the intersection point as a tuple \((x, y)\). If the lines are parallel (no intersection point or an infinite number of them), the result is False.

Return type 

tuple of number or bool

#### imgaug.imgaug.compute_paddings_for_aspect_ratio(arr, aspect_ratio)

Compute the amount of pixels by which an array has to be padded to fulfill an aspect ratio.
The aspect ratio is given as width/height. Depending on which dimension is smaller (height or width), only the corresponding sides (left/right or top/bottom) will be padded. In each case, both of the sides will be padded equally.

**Parameters**

- `aspect_ratio` (float) – Target aspect ratio, given as width/height. E.g. 2.0 denotes the image having twice as much width as height.

**Returns**

- `result` – Required padding amounts to reach the target aspect ratio, given as a tuple of the form (top, right, bottom, left).

**Return type**

tuple of int

---

`imgaug.imgaug.copy_random_state(random_state, force_copy=False)`

Creates a copy of a random state.

**Parameters**

- `random_state` (numpy.random.RandomState) – The random state to copy.
- `force_copy` (bool, optional) – If True, this function will always create a copy of every random state. If False, it will not copy numpy’s default random state, but all other random states.

**Returns**

- `rs_copy` – The copied random state.

**Return type**

numpy.random.RandomState

---

`imgaug.imgaug.current_random_state()`

Returns the current/global random state of the library.

**Returns**

The current/global random state.

**Return type**

numpy.random.RandomState

---

`class imgaug.imgaug.deprecated(alt_func=None, behavior='warn', removed_version=None, comment=None)`

Bases: object

Decorator to mark deprecated functions with warning.

Adapted from <https://github.com/scikit-image/scikit-image/blob/master/skimage/_shared/utils.py>.

**Parameters**

- `alt_func` (None or str, optional) – If given, tell user what function to use instead.
- `behavior` (‘warn’, ‘raise’, optional) – Behavior during call to deprecated function: ‘warn’ = warn user that function is deprecated; ‘raise’ = raise error.
- `removed_version` (None or str, optional) – The package version in which the deprecated function will be removed.
- `comment` (None or str, optional) – An optional comment that will be appended to the warning message.

**Methods**
__call__(func)  Call self as a function.

imgaug.imgaug.derive_random_state(random_state)
Create a new random states based on an existing random state or seed.

Parameters
random_state (numpy.random.RandomState) – Random state or seed from which to
derive the new random state.

Returns
Derived random state.

Return type
numpy.random.RandomState

imgaug.imgaug.derive_random_states(random_state, n=1)
Create N new random states based on an existing random state or seed.

Parameters
• random_state (numpy.random.RandomState) – Random state or seed from which to derive
new random states.
• n (int, optional) – Number of random states to derive.

Returns
Derived random states.

Return type
list of numpy.random.RandomState

imgaug.imgaug.do_assert(condition, message='Assertion failed. ')
Function that behaves equally to an assert statement, but raises an Exception.
This is added because assert statements are removed in optimized code. It replaces assert statements throughout
the library that should be kept even in optimized code.

Parameters
• condition (bool) – If False, an exception is raised.
• message (str, optional) – Error message.

imgaug.imgaug.draw_grid(images, rows=None, cols=None)
Converts multiple input images into a single image showing them in a grid.

dtype support:

* uint8‘: yes; fully tested
* uint16‘: yes; fully tested
* uint32‘: yes; fully tested
* uint64‘: yes; fully tested
* int8‘: yes; fully tested
* int16‘: yes; fully tested
* int32‘: yes; fully tested
* int64‘: yes; fully tested
* float16‘: yes; fully tested
* float32‘: yes; fully tested
* float64‘: yes; fully tested
* float128‘: yes; fully tested
* bool‘: yes; fully tested

Parameters
• images ((N,H,W,3) ndarray or iterable of (H,W,3) array) – The input images to convert to a
grid.
- **rows** *(None or int, optional)* – The number of rows to show in the grid. If None, it will be automatically derived.
- **cols** *(None or int, optional)* – The number of cols to show in the grid. If None, it will be automatically derived.

**Returns** `grid` – Image of the generated grid.

**Return type** `(H', W', 3) ndarray`

```python
imgaug.imgaug.draw_text(img, y, x, text, color=(0, 255, 0), size=25)
```

Draw text on an image.

This uses by default DejaVuSans as its font, which is included in this library.

dtype support:

```text
* `\`uint8\``: yes; fully tested
* `\`uint16\`": no
* `\`uint32\`": no
* `\`uint64\`": no
* `\`int8\`": no
* `\`int16\`": no
* `\`int32\`": no
* `\`int64\`": no
* `\`float16\`": no
* `\`float32\`": yes; not tested
* `\`float64\`": no
* `\`float128\`": no
* `\`bool\`": no
```

TODO check if other dtypes could be enabled

**Parameters**

- **img** `((H,W,3) ndarray)` – The image array to draw text on. Expected to be of dtype uint8 or float32 (value range 0.0 to 255.0).
- **y** *(int)* – x-coordinate of the top left corner of the text.
- **x** *(int)* – y-coordinate of the top left corner of the text.
- **text** *(str)* – The text to draw.
- **color** *(iterable of int, optional)* – Color of the text to draw. For RGB-images this is expected to be an RGB color.
- **size** *(int, optional)* – Font size of the text to draw.

**Returns** `img_np` – Input image with text drawn on it.

**Return type** `(H,W,3) ndarray`

```python
imgaug.imgaug.dummy_random_state()
```

Returns a dummy random state that is always based on a seed of 1.

**Returns** The new random state.

**Return type** `numpy.random.RandomState`

```python
imgaug.imgaug.flatten(nested_iterable)
```

Flattens arbitrarily nested lists/tuples.

**Parameters** `nested_iterable` – A list or tuple of arbitrarily nested values.

**Yields** *any* – Non-list and non-tuple values in `nested_iterable`.

```python
imgaug.imgaug.forward_random_state(random_state)
```
Forward the internal state of a random state.

This makes sure that future calls to the random_state will produce new random values.

**Parameters** `random_state` (*numpy.random.RandomState*) – Random state to forward.

```python
imgaug.imgaug.imresize_many_images(images, sizes=None, interpolation=None)
```
Resize many images to a specified size.

dtype support:

* **`uint8`**: yes; fully tested
* **`uint16`**: yes; tested
* **`uint32`**: no (1)
* **`uint64`**: no (2)
* **`int8`**: yes; tested (3)
* **`int16`**: yes; tested
* **`int32`**: limited; tested (4)
* **`int64`**: no (2)
* **`float16`**: yes; tested (5)
* **`float32`**: yes; tested
* **`float64`**: yes; tested
* **`float128`**: no (1)
* **`bool`**: yes; tested (6)

- (1) rejected by `cv2.imresize`
- (2) results too inaccurate
- (3) mapped internally to `int16` when interpolation!="nearest"
- (4) only supported for interpolation="nearest", other interpolations lead to `cv2` error
- (5) mapped internally to `float32`
- (6) mapped internally to `uint8`

**Parameters**

- **images** (*(N,H,W,[C]) ndarray or list of (H,W,[C]) ndarray*) – Array of the images to resize. Usually recommended to be of dtype uint8.

- **sizes** (*float or iterable of int or iterable of float*) – The new size of the images, given either as a fraction (a single float) or as a `(height, width)` tuple of two integers or as a `(height fraction, width fraction)` tuple of two floats.

- **interpolation** (*None or str or int, optional*) – The interpolation to use during resize. If int, then expected to be one of:
  - `cv2.INTER_NEAREST` (nearest neighbour interpolation)
  - `cv2.INTER_LINEAR` (linear interpolation)
  - `cv2.INTER_AREA` (area interpolation)
  - `cv2.INTER_CUBIC` (cubic interpolation)

  If string, then expected to be one of:
  - `nearest` (identical to `cv2.INTER_NEAREST`
– **linear** (identical to `cv2.INTER_LINEAR`)
– **area** (identical to `cv2.INTER_AREA`)
– **cubic** (identical to `cv2.INTER_CUBIC`)

If None, the interpolation will be chosen automatically. For size increases, area interpolation will be picked and for size decreases, linear interpolation will be picked.

**Returns**
- *result* – Array of the resized images.

**Return type** *(N,H',W',[C]) ndarray*

**Examples**

```python
>>> imresize_many_images(np.zeros((2, 16, 16, 3), dtype=np.uint8), 2.0)
```
Converts 2 RGB images of height and width 16 to images of height and width 16*2 = 32.

```python
>>> imresize_many_images(np.zeros((2, 16, 16, 3), dtype=np.uint8), (16, 32))
```
Converts 2 RGB images of height and width 16 to images of height 16 and width 32.

```python
>>> imresize_many_images(np.zeros((2, 16, 16, 3), dtype=np.uint8), (2.0, 4.0))
```
Converts 2 RGB images of height and width 16 to images of height 32 and width 64.

**imgaug.imgaug.imresize_single_image** *(image, sizes, interpolation=None)*

Resizes a single image.

**dtype support:**

See :func:`imgaug.imgaug.imresize_many_images`.

**Parameters**

- **image** *(H,W,C) ndarray or (H,W) ndarray* – Array of the image to resize. Usually recommended to be of dtype uint8.

- **sizes** *(float or iterable of int or iterable of float)* – See `imgaug.imgaug.imresize_many_images()`.

- **interpolation** *(None or str or int, optional)* – See `imgaug.imgaug.imresize_many_images()`.

**Returns**
- *out* – The resized image.

**Return type** *(H',W',C) ndarray or (H',W') ndarray*

**imgaug.imgaug.imshow** *(image, backend='matplotlib')*

Shows an image in a window.

**dtype support:**

* `uint8`: yes; not tested
* `uint16`: ?
* `uint32`: ?
* `uint64`: ?
* `int8`: ?
* `int16`: ?

(continues on next page)
* ``int32``: ?
* ``int64``: ?
* ``float16``: ?
* ``float32``: ?
* ``float64``: ?
* ``float128``: ?
* ``bool``: ?

Parameters

- **image** ((H,W,3) ndarray) – Image to show.
- **backend** (‘matplotlib’, ‘cv2’, optional) – Library to use to show the image. May be either matplotlib or OpenCV (‘cv2’). OpenCV tends to be faster, but apparently causes more technical issues.

```python
imgaug.imgaug.is_callable(val)
```

Checks whether a variable is a callable, e.g. a function.

**Parameters**

- **val** – The variable to check.

**Returns**

- True if the variable is a callable. Otherwise False.

**Return type**

bool

```python
imgaug.imgaug.is_float_array(val)
```

Checks whether a variable is a numpy float array.

**Parameters**

- **val** – The variable to check.

**Returns**

- True if the variable is a numpy float array. Otherwise False.

**Return type**

bool

```python
imgaug.imgaug.is_generator(val)
```

Checks whether a variable is a generator.

**Parameters**

- **val** – The variable to check.

**Returns**

- True if the variable is a generator. Otherwise False.

**Return type**

bool

```python
imgaug.imgaug.is_integer_array(val)
```

Checks whether a variable is a numpy integer array.

**Parameters**

- **val** – The variable to check.

**Returns**

- True if the variable is a numpy integer array. Otherwise False.

**Return type**

bool

```python
imgaug.imgaug.is_iterable(val)
```

Checks whether a variable is iterable.

**Parameters**

- **val** – The variable to check.

**Returns**

- True if the variable is an iterable. Otherwise False.

**Return type**

bool

```python
imgaug.imgaug.is_np_array(val)
```

Checks whether a variable is a numpy array.
Parameters \texttt{val} – The variable to check.

Returns \texttt{out} – True if the variable is a numpy array. Otherwise False.

Return type \texttt{bool}

\begin{verbatim}
imgaug.imgaug.is_single_bool(val)

Checks whether a variable is a boolean.

Parameters \texttt{val} – The variable to check.

Returns True if the variable is a boolean. Otherwise False.

Return type \texttt{bool}
\end{verbatim}

\begin{verbatim}
imgaug.imgaug.is_single_float(val)

Checks whether a variable is a float.

Parameters \texttt{val} – The variable to check.

Returns True if the variable is a float. Otherwise False.

Return type \texttt{bool}
\end{verbatim}

\begin{verbatim}
imgaug.imgaug.is_single_integer(val)

Checks whether a variable is an integer.

Parameters \texttt{val} – The variable to check.

Returns True if the variable is an integer. Otherwise False.

Return type \texttt{bool}
\end{verbatim}

\begin{verbatim}
imgaug.imgaug.is_single_number(val)

Checks whether a variable is a number, i.e. an integer or float.

Parameters \texttt{val} – The variable to check.

Returns True if the variable is a number. Otherwise False.

Return type \texttt{bool}
\end{verbatim}

\begin{verbatim}
imgaug.imgaug.is_string(val)

Checks whether a variable is a string.

Parameters \texttt{val} – The variable to check.

Returns True if the variable is a string. Otherwise False.

Return type \texttt{bool}
\end{verbatim}

\begin{verbatim}
imgaug.imgaug.max_pool(arr, block_size, cval=0, preserve_dtype=True)

Resize an array using max-pooling.

dtype support:

See :func:`imgaug.imgaug.pool`.
\end{verbatim}

Parameters

- \texttt{arr} ((\(H,W\)) \texttt{ndarray} or \((H,W,C)\) \texttt{ndarray}) – Image-like array to pool. See \texttt{imgaug.pool()} for details.

- \texttt{block_size} (\texttt{int} or tuple of \texttt{int} or tuple of \texttt{int}) – Size of each block of values to pool. See \texttt{imgaug.pool()} for details.

- \texttt{cval} (\texttt{number}, \texttt{optional}) – Padding value. See \texttt{imgaug.pool()} for details.
• **preserve_dtype** *(bool, optional)* – Whether to preserve the input array dtype. See `imgaug.pool()` for details.

**Returns** arr_reduced – Array after max-pooling.

**Return type** *(H',W') ndarray or (H',W',C') ndarray*

`imgaug.imgaug.new_random_state`(seed=None, fully_random=False)

Returns a new random state.

**Parameters**

• **seed** *(None or int, optional)* – Optional seed value to use. The same datatypes are allowed as for `numpy.random.RandomState(seed)`.

• **fully_random** *(bool, optional)* – Whether to use numpy’s random initialization for the RandomState (used if set to True). If False, a seed is sampled from the global random state, which is a bit faster and hence the default.

**Returns** The new random state.

**Return type** `numpy.random.RandomState`

`imgaug.imgaug.pad`(arr, top=0, right=0, bottom=0, left=0, mode='constant', cval=0)

Pad an image-like array on its top/right/bottom/left side.

This function is a wrapper around `numpy.pad()`.

dtype support:

```
• "uint8": yes; fully tested (1)
• "uint16": yes; fully tested (1)
• "uint32": yes; fully tested (2) (3)
• "uint64": yes; fully tested (2) (3)
• "int8": yes; fully tested (1)
• "int16": yes; fully tested (1)
• "int32": yes; fully tested (1)
• "int64": yes; fully tested (2) (3)
• "float16": yes; fully tested (2) (3)
• "float32": yes; fully tested (1)
• "float64": yes; fully tested (1)
• "float128": yes; fully tested (2) (3)
• "bool": yes; tested (2) (3)
```

- (1) Uses `cv2` if `mode` is one of: `"constant"`, `"edge"`, `"reflect"`, `"symmetric``.
  Otherwise uses `numpy`.
- (2) Uses `numpy`
- (3) Rejected by `cv2`

**Parameters**

• **arr** *(H,W) ndarray or (H,W,C) ndarray* – Image-like array to pad.

• **top** *(int, optional)* – Amount of pixels to add at the top side of the image. Must be 0 or greater.

• **right** *(int, optional)* – Amount of pixels to add at the right side of the image. Must be 0 or greater.

• **bottom** *(int, optional)* – Amount of pixels to add at the bottom side of the image. Must be 0 or greater.
• **left** (*int*, *optional*) – Amount of pixels to add at the left side of the image. Must be 0 or greater.

• **mode** (*str*, *optional*) – Padding mode to use. See `numpy.pad()` for details. In case of **mode** `constant`, the parameter **cval** will be used as the **constant_values** parameter to `numpy.pad()`. In case of mode `linear_ramp`, the parameter **cval** will be used as the **end_values** parameter to `numpy.pad()`.

• **cval** (*number*, *optional*) – Value to use for padding if **mode** is `constant`. See `numpy.pad()` for details.

Returns **arr_pad** – Padded array with height $H' = H + \text{top} + \text{bottom}$ and width $W' = W + \text{left} + \text{right}$.

Return type *(H', W') ndarray or (H', W', C) ndarray*

```python
imgaug.imgaug.pad_to_aspect_ratio(arr, aspect_ratio, mode='constant', cval=0, return_pad_amounts=False)
```

Pad an image-like array on its sides so that it matches a target aspect ratio.

Depending on which dimension is smaller (height or width), only the corresponding sides (left/right or top/bottom) will be padded. In each case, both of the sides will be padded equally.

dtype support: See :func:`imgaug.imgaug.pad`.

**Parameters**

• **arr** *(H,W) ndarray or (H,W,C) ndarray* – Image-like array to pad.

• **aspect_ratio** (*float*) – Target aspect ratio, given as width/height. E.g. 2.0 denotes the image having twice as much width as height.

• **mode** (*str*, *optional*) – Padding mode to use. See `numpy.pad()` for details.

• **cval** (*number*, *optional*) – Value to use for padding if **mode** is `constant`. See `numpy.pad()` for details.

• **return_pad_amounts** (*bool*, *optional*) – If False, then only the padded image will be returned. If True, a tuple with two entries will be returned, where the first entry is the padded image and the second entry are the amounts by which each image side was padded. These amounts are again a tuple of the form (top, right, bottom, left), with each value being an integer.

**Returns**

• **arr_padded** *(H',W') ndarray or (H',W',C) ndarray* – Padded image as (H',W') or (H',W',C) ndarray, fulfilling the given aspect_ratio.

• **tuple of int** – Amounts by which the image was padded on each side, given as a tuple (top, right, bottom, left). This tuple is only returned if **return_pad_amounts** was set to True. Otherwise only **arr_padded** is returned.

```python
imgaug.imgaug.pool(arr, block_size, func, cval=0, preserve_dtype=True)
```

Resize an array by pooling values within blocks.

dtype support:
* "uint8": yes; fully tested
* "uint16": yes; tested
* "uint32": yes; tested (2)
* "uint64": no (1)
* "int8": yes; tested
* "int16": yes; tested
* "int32": yes; tested (2)
* "int64": no (1)
* "float16": yes; tested
* "float32": yes; tested
* "float64": yes; tested
* "float128": yes; tested (2)
* "bool": yes; tested

- (1) results too inaccurate (at least when using np.average as func)
- (2) Note that scikit-image documentation says that the wrapped pooling function converts inputs to float64. Actual tests showed no indication of that happening (at least when using preserve_dtype=True).

Parameters

- **arr** 
  
  (H,W) ndarray or (H,W,C) ndarray – Image-like array to pool. Ideally of datatype numpy.float64.

- **block_size** 
  
  (int or tuple of int) – Spatial size of each group of values to pool, aka kernel size. If a single integer, then a symmetric block of that size along height and width will be used. If a tuple of two values, it is assumed to be the block size along height and width of the image-like, with pooling happening per channel. If a tuple of three values, it is assumed to be the block size along height, width and channels.

- **func** 
  
  (callable) – Function to apply to a given block in order to convert it to a single number, e.g. numpy.average(), numpy.min(), numpy.max().

- **cval** 
  
  (number, optional) – Value to use in order to pad the array along its border if the array cannot be divided by block_size without remainder.

- **preserve_dtype** 
  
  (bool, optional) – Whether to convert the array back to the input datatype if it is changed away from that in the pooling process.

Returns **arr_reduced** – Array after pooling.

Return type (H',W') ndarray or (H',W',C') ndarray

```
imgaug.imgaug.quokka(size=None, extract=None)
```

Returns an image of a quokka as a numpy array.

Parameters

- **size** 
  
  (None or float or tuple of int, optional) – Size of the output image. Input into imgaug.imgaug.imresize_single_image(). Usually expected to be a tuple (H, W), where H is the desired height and W is the width. If None, then the image will not be resized.

- **extract** 
  
  (None or ‘square’ or tuple of number or imgaug.BoundingBox or imgaug.BoundingBoxesOnImage) – Subarea of the quokka image to extract:
– If None, then the whole image will be used.
– If string square, then a squared area \((x: 0 \text{ to max 643}, y: 0 \text{ to max 643})\) will be extracted from the image.
– If a tuple, then expected to contain four numbers denoting \(x1, y1, x2\) and \(y2\).
– If a BoundingBox, then that bounding box’s area will be extracted from the image.
– If a BoundingBoxesOnImage, then expected to contain exactly one bounding box and a shape matching the full image dimensions (i.e. \((643, 960, *)\)). Then the one bounding box will be used similar to BoundingBox.

**Returns** `img` – The image array of dtype uint8.

**Return type** (H,W,3) ndarray

### `imgaug.quokka_bounding_boxes` 

```
(size=None, extract=None)
```

Returns example bounding boxes on the standard example quokke image.

Currently only a single bounding box is returned that covers the quokka.

**Parameters**

- `size` *(None or float or tuple of int or tuple of float, optional)* – Size of the output image on which the BBs are placed. If None, then the BBs are not projected to any new size (positions on the original image are used). Floats lead to relative size changes, ints to absolute sizes in pixels.
- `extract` *(None or ‘square’ or tuple of number or imgaug.BoundingBox or imgaug.BoundingBoxesOnImage)* – Subarea to extract from the image. See `imgaug.quokka()`.

**Returns** `bbsoi` – Example BBs on the quokka image.

**Return type** imgaug.BoundingBoxesOnImage

### `imgaug.quokka_heatmap` 

```
(size=None, extract=None)
```

Returns a heatmap (here: depth map) for the standard example quokka image.

**Parameters**

- `size` *(None or float or tuple of int, optional)* – See `imgaug.quokka()`.
- `extract` *(None or ‘square’ or tuple of number or imgaug.BoundingBox or imgaug.BoundingBoxesOnImage)* – See `imgaug.quokka()`.

**Returns** `result` – Depth map as an heatmap object. Values close to 0.0 denote objects that are close to the camera. Values close to 1.0 denote objects that are furthest away (among all shown objects).

**Return type** imgaug.HeatmapsOnImage

### `imgaug.quokka_keypoints` 

```
(size=None, extract=None)
```

Returns example keypoints on the standard example quokka image.

The keypoints cover the eyes, ears, nose and paws.

**Parameters**

- `size` *(None or float or tuple of int or tuple of float, optional)* – Size of the output image on which the keypoints are placed. If None, then the keypoints are not projected to any new size (positions on the original image are used). Floats lead to relative size changes, ints to absolute sizes in pixels.
• `extract` (None or 'square' or tuple of number or imgaug.BoundingBox or imgaug.BoundingBoxesOnImage) – Subarea to extract from the image. See `imgaug.quokka()`.

Returns `kpsoi` – Example keypoints on the quokka image.

Return type `imgaug.KeypointsOnImage`

`imgaug.imgaug.quokka_polygons(size=None, extract=None)`

Returns example polygons on the standard example quokka image.

The result contains one polygon, covering the quokka’s outline.

Parameters

• `size` (None or float or tuple of int or tuple of float, optional) – Size of the output image on which the polygons are placed. If None, then the polygons are not projected to any new size (positions on the original image are used). Floats lead to relative size changes, ints to absolute sizes in pixels.

• `extract` (None or 'square' or tuple of number or imgaug.BoundingBox or imgaug.BoundingBoxesOnImage) – Subarea to extract from the image. See `imgaug.quokka()`.

Returns `psoi` – Example polygons on the quokka image.

Return type `imgaug.PolygonsOnImage`

`imgaug.imgaug.quokka_segmentation_map(size=None, extract=None)`

Returns a segmentation map for the standard example quokka image.

Parameters

• `size` (None or float or tuple of int, optional) – See `imgaug.quokka()`.

• `extract` (None or 'square' or tuple of number or imgaug.BoundingBox or imgaug.BoundingBoxesOnImage) – See `imgaug.quokka()`.

Returns `result` – Segmentation map object.

Return type `imgaug.SegmentationMapOnImage`

`imgaug.imgaug.quokka_square(size=None)`

Returns an (square) image of a quokka as a numpy array.

Parameters `size` (None or float or tuple of int, optional) – Size of the output image. Input into `imgaug.imgaug.imresize_single_image()`. Usually expected to be a tuple (H, W), where H is the desired height and W is the width. If None, then the image will not be resized.

Returns `img` – The image array of dtype uint8.

Return type `(H,W,3) ndarray`

`imgaug.imgaug.seed(seedval)`

Set the seed used by the global random state and thereby all randomness in the library.

This random state is by default by all augmenters. Under special circumstances (e.g. when an augmenter is switched to deterministic mode), the global random state is replaced by another – local – one. The replacement is dependent on the global random state.

Parameters `seedval` (int) – The seed to use.

`imgaug.imgaug.show_grid(images, rows=None, cols=None)`

Converts the input images to a grid image and shows it in a new window.

dtype support:
Parameters

- images ((N,H,W,3) ndarray or iterable of (H,W,3) array) – See imgaug.draw_grid().
- rows (None or int, optional) – See imgaug.draw_grid().
- cols (None or int, optional) – See imgaug.draw_grid().

imgaug.imgaug.warn_deprecated(msg, stacklevel=2)
Generate a non-silent deprecation warning with stacktrace.
The used warning is imgaug.imgaug.DeprecationWarning.

Parameters

- msg (str) – The message of the warning.
- stacklevel (int, optional) – How many steps above this function to “jump” in the stacktrace
  for the displayed file and line number of the error message. Usually 2.

13.2 imgaug.parameters

class imgaug.parameters.Absolute(other_param)
Bases: imgaug.parameters.StochasticParameter
Converts another parameter’s results to absolute values.

Parameters

- other_param (imgaug.parameters.StochasticParameter) – Other parameter which’s
  sampled values are to be modified.

Examples

```python
class param = Absolute(Uniform(-1.0, 1.0))
```

Converts a uniform range \([-1.0, 1.0]\) to \([0.0, 1.0]\).

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td>draw_distribution_graph(title, size, bins)</td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td>draw_sample(random_state)</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

class imgaug.parameters.Add(other_param, val, elementwise=False)
Bases: imgaug.parameters.StochasticParameter
Parameter to add to other parameter’s results.

**Parameters**

- **other_param** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* – Other parameter which’s sampled values are to be modified.

- **val** *(number or tuple of two number or list of number or imgaug.parameters.StochasticParameter)* – Value to add to the other parameter’s results. If this is a StochasticParameter, either a single or multiple values will be sampled and added.

- **elementwise** *(bool, optional)* – Controls the sampling behaviour when val is a StochasticParameter. If set to False, a single value will be sampled from val and added to all values generated by other_param. If set to True and _draw_samples(size=S) is called, S values will be sampled from val and added to the results of other_param.

**Examples**

```python
def add_uniform_and_one(x):
    return Add(Uniform(0.0, 1.0), 1.0).draw_samples([x])
```

Converts a uniform range \([0.0, 1.0)\) to \([1.0, 2.0)\).

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td><code>draw_distribution_graph([title, size, bins])</code></td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td><code>draw_sample([random_state])</code></td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td><code>draw_samples(size[, random_state])</code></td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

**class** `imgaug.parameters.Beta(alpha, beta, epsilon=0.0001)`

**Bases:** `imgaug.parameters.StochasticParameter`

Parameter that resembles a (continuous) beta distribution.

**Parameters**

- **alpha** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* –
  
  alpha parameter of the beta distribution.
  
  - If number, that number will always be used.
  
  - If tuple of two number, a random value will be sampled from the range \([a, b)\) once per call to `imgaug.parameters.Beta._draw_samples()`.
  
  - If list of number, a random element from that list will be sampled per call.
  
  - If a StochasticParameter, a random value will be sampled from that parameter per call.

  alpha has to be a value above 0. If it ends up <=0 it is automatically clipped to \(0+\epsilon\).

- **beta** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* – Beta parameter of the Beta distribution.
Analogous to $alpha$.

- **epsilon (number)** – Clipping parameter. If $alpha$ or $beta$ end up $\leq 0$, they are clipped to $0+\text{epsilon}$.

### Examples

```python
>>> param = Beta(0.5, 0.5)
```

Samples random values from the beta distribution with alpha=beta=0.5.

### Methods

- `copy()` Create a shallow copy of this parameter.
- `deepcopy()` Create a deep copy of this parameter.
- `draw_distribution_graph([title, size, bins])` Generate a plot (image) that shows the parameter’s distribution of values.
- `draw_sample([random_state])` Draws a single sample value from this parameter.
- `draw_samples(size[, random_state])` Draws one or more sample values from the parameter.

```python
class imgaug.parameters.Binomial(p)
Bases: imgaug.parameters.StochasticParameter
```

Binomial distribution.

**Parameters p (number or tuple of number or list of number or imgaug.parameters.StochasticParameter)** – Probability of the binomial distribution. Expected to be in the range $[0, 1]$.

- If this is a number, then that number will always be used as the probability.
- If this is a tuple $(a, b)$, a random value will be sampled from the range $a<=x<b$ per call to `imgaug.parameters.Binomial._draw_samples()`.
- If this is a list of numbers, a random value will be picked from that list per call.
- If this is a StochasticParameter, the value will be sampled once per call.

### Examples

```python
>>> param = Binomial(Uniform(0.01, 0.2))
```

Uses a varying probability $p$ between 0.01 and 0.2 per sampling.

### Methods

- `copy()` Create a shallow copy of this parameter.
- `deepcopy()` Create a deep copy of this parameter.
- `draw_distribution_graph([title, size, bins])` Generate a plot (image) that shows the parameter’s distribution of values.
Table 8 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>draw_sample([random_state])</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

**class** `imgaug.parameters.ChiSquare(df)`

**Bases:** `imgaug.parameters.StochasticParameter`

Parameter that resembles a (continuous) chi-square distribution.

This is a wrapper around numpy’s `numpy.random.chisquare()`.

**Parameters**

- `df` *(int or tuple of two int or list of int or imgaug.parameters.StochasticParameter)* – Degrees of freedom (must be 1 or higher).
  - If a single int, this int will be used as a constant value.
  - If a tuple of two ints `(a, b)`, the value will be sampled from the discrete range `[a..b]` once per call to `imgaug.parameters.ChiSquare._draw_samples()`.
  - If a list of ints, a random value will be picked from the list per call.
  - If a StochasticParameter, that parameter will be queried once per call.

**Examples**

```python
>>> param = ChiSquare(df=2)
```

A chi-square distribution with two degrees of freedom.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td>draw_distribution_graph([title, size, bins])</td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td>draw_sample([random_state])</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

**class** `imgaug.parameters.Choice(a, replace=True, p=None)`

**Bases:** `imgaug.parameters.StochasticParameter`

Parameter that samples value from a list of allowed values.

**Parameters**

- `a` *(iterable)* – List of allowed values. Usually expected to be integers, floats or strings.
- `replace` *(bool, optional)* – Whether to perform sampling with or without replacing.
- `p` *(None or iterable of number, optional)* – Optional probabilities of each element in `a`. Must have the same length as `a` (if provided).
Examples

```python
>>> param = Choice([0.25, 0.5, 0.75], p=[0.25, 0.5, 0.25])
```

Parameter of which 50 percent of all sampled values will be 0.5. The other 50 percent will be either 0.25 or 0.75.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td><code>draw_distribution_graph()</code></td>
<td>Generate a plot (image) that shows the parameter's</td>
</tr>
<tr>
<td></td>
<td>distribution of values.</td>
</tr>
<tr>
<td><code>draw_sample()</code></td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td><code>draw_samples(size[, random_state])</code></td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

```python
class imgaug.parameters.Clip(other_param, minval=None, maxval=None)
```

Bases: `imgaug.parameters.StochasticParameter`

Clips another parameter to a defined value range.

Parameters

- **other_param** (`imgaug.parameters.StochasticParameter`) – The other parameter, which’s values are to be clipped.
- **minval** (`None or number, optional`) – The minimum value to use. If None, no minimum will be used.
- **maxval** (`None or number, optional`) – The maximum value to use. If None, no maximum will be used.

Examples

```python
>>> param = Clip(Normal(0, 1.0), minval=-2.0, maxval=2.0)
```

Defines a standard normal distribution, which’s values never go below -2.0 or above 2.0. Note that this will lead to small “bumps” of higher probability at -2.0 and 2.0, as values below/above these will be clipped to them.

Methods

```python
class imgaug.parameters.Deterministic(value)
```

Bases: `imgaug.parameters.StochasticParameter`
Parameter that is a constant value.

If \( N \) values are sampled from this parameter, it will return \( N \) times \( V \), where \( V \) is the constant value.

**Parameters**

value *(number or str or imgaug.parameters.StochasticParameter)* – A constant value to use. A string may be provided to generate arrays of strings. If this is a StochasticParameter, a single value will be sampled from it exactly once and then used as the constant value.

**Examples**

```python
>>> param = Deterministic(10)
```

Will always sample the value 10.

**Methods**

- `copy()` Create a shallow copy of this parameter.
- `deepcopy()` Create a deep copy of this parameter.
- `draw_distribution_graph([title, size, bins])` Generate a plot (image) that shows the parameter’s distribution of values.
- `draw_sample([random_state])` Draws a single sample value from this parameter.
- `draw_samples(size[, random_state])` Draws one or more sample values from the parameter.

**class** `imgaug.parameters.DiscreteUniform(a, b)`

Bases: `imgaug.parameters.StochasticParameter`

Parameter that resembles a discrete range of values \([a .. b]\).

**Parameters**

- `a` *(int or imgaug.parameters.StochasticParameter)* – Lower bound of the sampling range. Values will be sampled from \( a<=x<=b \). All sampled values will be discrete. If \( a \) is a StochasticParameter, it will be queried once per sampling to estimate the value of \( a \). If \( a>b \), the values will automatically be flipped. If \( a==b \), all generated values will be identical to \( a \).
- `b` *(int or imgaug.parameters.StochasticParameter)* – Upper bound of the sampling range. Values will be sampled from \( a<=x<=b \). All sampled values will be discrete. If \( b \) is a StochasticParameter, it will be queried once per sampling to estimate the value of \( b \). If \( a>b \), the values will automatically be flipped. If \( a==b \), all generated values will be identical to \( a \).

**Examples**

```python
>>> param = DiscreteUniform(10, Choice([20, 30, 40]))
```

Sampled values will be discrete and come from the either \([10..20]\) or \([10..30]\) or \([10..40]\).

**Methods**

- `copy()` Create a shallow copy of this parameter.
- `deepcopy()` Create a deep copy of this parameter.
### Table 13 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>draw_distribution_graph</code></td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td><code>draw_sample</code></td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td><code>draw_samples</code></td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

```python
>>> param = Discretize(Normal(0, 1.0))
```

Generates a discrete standard normal distribution.

### Methods

- `copy()`: Create a shallow copy of this parameter.
- `deepcopy()`: Create a deep copy of this parameter.
- `draw_distribution_graph(title, size, bins)`: Generate a plot (image) that shows the parameter’s distribution of values.
- `draw_sample([random_state])`: Draws a single sample value from this parameter.
- `draw_samples(size[, random_state])`: Draws one or more sample values from the parameter.

```python
>>> param = Discretize(Normal(0, 1.0))
```

Generates a discrete standard normal distribution.

### Examples

```python
>>> param = Discretize(Normal(0, 1.0))
```

Generates a discrete standard normal distribution.

### Methods

- `copy()`: Create a shallow copy of this parameter.
- `deepcopy()`: Create a deep copy of this parameter.
- `draw_distribution_graph(title, size, bins)`: Generate a plot (image) that shows the parameter’s distribution of values.
- `draw_sample([random_state])`: Draws a single sample value from this parameter.
- `draw_samples(size[, random_state])`: Draws one or more sample values from the parameter.

```python
>>> param = Discretize(Normal(0, 1.0))
```

Generates a discrete standard normal distribution.

### Methods

- `copy()`: Create a shallow copy of this parameter.
- `deepcopy()`: Create a deep copy of this parameter.
- `draw_distribution_graph(title, size, bins)`: Generate a plot (image) that shows the parameter’s distribution of values.
- `draw_sample([random_state])`: Draws a single sample value from this parameter.
- `draw_samples(size[, random_state])`: Draws one or more sample values from the parameter.

```python
>>> param = Discretize(Normal(0, 1.0))
```

Generates a discrete standard normal distribution.
Examples

```python
>>> param = Divide(Uniform(0.0, 1.0), 2)
```

Converts a uniform range \([0.0, 1.0)\) to \([0, 0.5)\).

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td>draw_distribution_graph(title, size, bins)</td>
<td>Generate a plot (image) that shows the parameter's distribution of values.</td>
</tr>
<tr>
<td>draw_sample(random_state)</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

```python
class imgaug.parameters.ForceSign(other_param, positive=False, mode='invert', reroll_count_max=2)
```

Bases: `imgaug.parameters.StochasticParameter`

Converts another parameter’s results to positive or negative values.

Parameters

- `other_param` (*imgaug.parameters.StochasticParameter*) – Other parameter which’s sampled values are to be modified.
- `positive` (*bool*) – Whether to force all signs to be positive/+ (True) or negative/- (False).
- `mode` (*{'invert', 'reroll'}, optional*) – How to change the signs. Valid values are invert and reroll. `invert` means that wrong signs are simply flipped. `reroll` means that all samples with wrong signs are sampled again, optionally many times, until they randomly end up having the correct sign.
- `reroll_count_max` (*int, optional*) – If `mode` is set to `reroll`, this determines how often values may be rerolled before giving up and simply flipping the sign (as in `mode="invert"`). This shouldn’t be set too high, as rerolling is expensive.

Examples

```python
>>> param = ForceSign(Poisson(1), positive=False)
```

Generates a poisson distribution with \(\text{alpha}=1\) that is flipped towards negative values.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td>draw_distribution_graph(title, size, bins)</td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td>draw_sample(random_state)</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

Continued on next page
**class** `imgaug.parameters.FrequencyNoise(exponent=(-4, 4), size_px_max=(4, 32), upscale_method=['linear', 'nearest'])`

**Bases:** `imgaug.parameters.StochasticParameter`

Parameter to generate noise of varying frequencies.

This parameter expects to sample noise for 2d planes, i.e. for sizes \((H, W)\) and will return a value in the range \([0.0, 1.0]\) per location in that plane.

The exponent controls the frequencies and therefore noise patterns. Low values (around -4.0) will result in large blobs. High values (around 4.0) will result in small, repetitive patterns.

The noise is sampled from low resolution planes and upscaled to the requested height and width. The size of the low resolution plane may be defined (high values can be slow) and the interpolation method for upscaling can be set.

**Parameters**

- **exponent** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Exponent to use when scaling in the frequency domain. Sane values are in the range -4 (large blobs) to 4 (small patterns). To generate cloud-like structures, use roughly -2.
  - If number, then that number will be used as the exponent for all iterations.
  - If tuple of two numbers \((a, b)\), then a value will be sampled per iteration from the range \([a, b]\).
  - If a list of numbers, then a value will be picked per iteration at random from that list.
  - If a StochasticParameter, then a value will be sampled from that parameter per iteration.

- **size_px_max** *(int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional)* – The frequency noise is generated in a low resolution environment. This parameter defines the maximum size of that environment (in pixels). The environment is initialized at the same size as the input image and then downscaled, so that no side exceeds \(size\_px\_max\) (aspect ratio is kept).
  - If int, then that number will be used as the size for all iterations.
  - If tuple of two ints \((a, b)\), then a value will be sampled per iteration from the discrete range \([a..b]\).
  - If a list of ints, then a value will be picked per iteration at random from that list.
  - If a StochasticParameter, then a value will be sampled from that parameter per iteration.

- **upscale_method** *(None or imgaug.ALL or str or list of str or imgaug.parameters.StochasticParameter, optional)* – After generating the noise maps in low resolution environments, they have to be upscaled to the input image size. This parameter controls the upscaling method. See also `imgaug.imaug.imresize_many_images()` for a description of possible values.
  - If None, then either ‘nearest’ or ‘linear’ or ‘cubic’ is picked. Most weight is put on linear, followed by cubic.
  - If imgaug.ALL, then either ‘nearest’ or ‘linear’ or ‘area’ or ‘cubic’ is picked per iteration (all same probability).
– If string, then that value will be used as the method (must be ‘nearest’ or ‘linear’ or ‘area’ or ‘cubic’).
– If list of string, then a random value will be picked from that list per iteration.
– If StochasticParameter, then a random value will be sampled from that parameter per iteration.

Examples

```python
>>> param = FrequencyNoise(exponent=-2, size_px_max=(16, 32), upscale_method="linear")
```

Generates noise with cloud-like patterns.

Methods

```python
copy()
deepcopy()
draw_distribution_graph([title, size, bins])
draw_sample([random_state])
draw_samples(size[, random_state])
```

class imgaug.parameters.FromLowerResolution(other_param, size_percent=None, size_px=None, method='nearest', min_size=1)

Bases: imgaug.parameters.StochasticParameter

A meta parameter used to sample other parameter values on a low resolution 2d plane.
Here, ‘2d’ denotes shapes of (H, W, C).

This parameter is intended to be used with parameters that would usually sample one value per pixel (or one value per pixel and channel). With this parameter, the sampling can be made more coarse, i.e. the result will become rectangles instead of single pixels.

Parameters

- **other_param** (imgaug.parameters.StochasticParameter) – The other parameter which is to be sampled on a coarser image.

- **size_percent** (None or number or iterable of number or imgaug.parameters.StochasticParameter, optional) – Size of the 2d sampling plane in percent of the requested size. I.e. this is relative to the size provided in the call to draw_samples(size). Lower values will result in smaller sampling planes, which are then upsampled to size. This means that lower values will result in larger rectangles. The size may be provided as a constant value or a tuple \((a, b)\), which will automatically be converted to the continuous uniform range \([a, b)\) or a StochasticParameter, which will be queried per call to draw_samples().

- **size_px** (None or number or iterable of numbers or imgaug.parameters.StochasticParameter, optional) – Size of the 2d sampling plane in pixels. Lower values will result in smaller sampling planes, which are then upsampled to the input size of draw_samples(size). This means that lower values will result
in larger rectangles. The size may be provided as a constant value or a tuple \((a, b)\), which will automatically be converted to the discrete uniform range \([a..b]\) or a StochasticParameter, which will be queried per call to `draw_samples()`.

- **method** *(str or int or imgaug.parameters.StochasticParameter, optional)* – Upsampling/interpolation method to use. This is used after the sampling is finished and the low resolution plane has to be upsampled to the requested `size` in `draw_samples(size, ...)` The method may be the same as in `imgaug.imgaug.imresize_many_images()`. Usually nearest or linear are good choices. nearest will result in rectangles with sharp edges and linear in rectangles with blurry and round edges. The method may be provided as a StochasticParameter, which will be queried per call to `draw_samples()`.

- **min_size** *(int, optional)* – Minimum size in pixels of the low resolution sampling plane.

**Examples**

```python
def param = FromLowerResolution(Binomial(0.05), size_px=(2, 16), method=Choice([    ←"nearest", "linear"]))
```

Samples from a binomial distribution with \(p=0.05\). The sampling plane will always have a size \(H \times W \times C\) with \(H\) and \(W\) being independently sampled from \([2..16]\) (i.e. it may range from \(2 \times 2 \times C\) up to \(16 \times 16 \times C\) max, but may also be e.g. \(4 \times 8 \times C\). The upsampled method will be nearest in 50 percent of all cases and linear in the other 50 percent. The result will sometimes be rectangular patches of sharp `1`'s surrounded by `0`'s and sometimes blurry blobs of `1`'s, surrounded by values `<1.0`.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td><code>draw_distribution_graph([title, size, bins])</code></td>
<td>Generate a plot (image) that shows the parameter's distribution of values.</td>
</tr>
<tr>
<td><code>draw_sample([random_state])</code></td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td><code>draw_samples(size[, random_state])</code></td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

**class** `imgaug.parameters.IterativeNoiseAggregator` *(other_param, iterations=(1, 3), aggregation_method=['max', 'avg'])`

Parameter to generate noise maps in multiple iterations and aggregate their results.

This is supposed to be used in conjunction with SimplexNoise or FrequencyNoise.

**Parameters**

- **other_param** *(StochasticParameter)* – The noise parameter to iterate multiple times.

- **iterations** *(int or iterable of int or list of int or imgaug.parameters.StochasticParameter, optional)* – The number of iterations. This may be a single integer or a tuple of two integers \((a, b)\), which will result in \([a..b]\) iterations or a list of integers \([a, b, c, \ldots]\), which will result in \(a\ or \ b\ or \ c\, \ldots\) iterations. It may also be a StochasticParameter, in which case the number of iterations will be sampled once per call to `imgaug.parameters.IterativeNoiseAggregator._draw_samples()`.
• **aggregation_method** *(imgaug.ALL or {'min', 'avg', 'max'}) or list of str or imgaug.parameters.StochasticParameter, optional* – The method to use to aggregate the results of multiple iterations. If a string, it must have the value min or max or avg. If min is chosen, the elementwise minimum will be computed over all iterations (pushing the noise towards zeros). max will result in the elementwise maximum and avg in the average over all iterations. If imgaug.ALL is used, it will be randomly either min or max or avg (per call to imgaug.parameters.IterativeNoiseAggregator_draw_samples()). If a list is chosen, it must contain the mentioned strings and a random one will be picked per call. If a StochasticParameter is used, a value will be sampled from it per call.

**Examples**

```python
>>> noise = IterativeNoiseAggregator(SimplexNoise(), iterations=(2, 5),
                                    aggregation_method="max")
```

Generates per call 2 to 5 times simplex noise of a given size. Then combines these noise maps to a single map using elementwise maximum.

**Methods**

- `copy()` Create a shallow copy of this parameter.
- `deepcopy()` Create a deep copy of this parameter.
- `draw_distribution_graph([title, size, bins])` Generate a plot (image) that shows the parameter’s distribution of values.
- `draw_sample([random_state])` Draws a single sample value from this parameter.
- `draw_samples(size[, random_state])` Draws one or more sample values from the parameter.

**class** `imgaug.parameters.Laplace(loc, scale)`

**Bases:** `imgaug.parameters.StochasticParameter`

Parameter that resembles a (continuous) laplace distribution.

This is a wrapper around numpy's `numpy.random.laplace()`.

**Parameters**

- **loc** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* –
  The position of the distribution peak, similar to the mean in normal distributions.
  - If a single number, this number will be used as a constant value.
  - If a tuple of two numbers \((a, b)\), the value will be sampled from the continuous range \([a, b)\) once per call to `imgaug.parameters.Laplace._draw_samples()`.
  - If a list of numbers, a random value will be picked from the list per call.
  - If a StochasticParameter, that parameter will be queried once per call.
- **scale** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* –
  The exponential decay factor, similar to standard deviation in normal distributions.
  - If a single number, this number will be used as a constant value.
– If a tuple of two numbers \((a, b)\), the value will be sampled from the continuous range \([a, b)\), once per call to \texttt{imgaug.parameters.Laplace._draw_samples()} 
– If a list of numbers, a random value will be picked from the list per call. 
– If a StochasticParameter, that parameter will be queried once per call.

**Examples**

```python
>>> param = Laplace(0, 1.0)
```

A laplace distribution, which’s peak is at 0 and decay is 1.0.

**Methods**

- \texttt{copy()} Create a shallow copy of this parameter.
- \texttt{deepcopy()} Create a deep copy of this parameter.
- \texttt{draw_distribution_graph([title, size, bins])} Generate a plot (image) that shows the parameter’s distribution of values.
- \texttt{draw_sample([random_state])} Draws a single sample value from this parameter.
- \texttt{draw_samples(size[, random_state])} Draws one or more sample values from the parameter.

**class** \texttt{imgaug.parameters.Multiply(other_param, val, elementwise=False)}

**Bases:** \texttt{imgaug.parameters.StochasticParameter}

Parameter to multiply other parameter’s results with.

**Parameters**

- \texttt{other_param} \((number\ or\ tuple\ of\ number\ or\ list\ of\ number\ or\ imgaug.parameters.StochasticParameter)\) – Other parameter which’s sampled values are to be multiplied.
- \texttt{val} \((number\ or\ tuple\ of\ number\ or\ list\ of\ number\ or\ imgaug.parameters.StochasticParameter)\) – Multiplier to use. If this is a StochasticParameter, either a single or multiple values will be sampled and used as the multiplier(s).
- \texttt{elementwise} \((bool,\ optional)\) – Controls the sampling behaviour when \texttt{val} is a StochasticParameter. If set to False, a single value will be sampled from \texttt{val} and used as the constant multiplier. If set to True and \texttt{_draw_samples(size=S)} is called, \(S\) values will be sampled from \texttt{val} and multiplied elementwise with the results of \texttt{other_param}.

**Examples**

```python
>>> param = Multiply(Uniform(0.0, 1.0), -1)
```

Converts a uniform range \([0.0, 1.0)\) to \((-1.0, 0.0)\).

**Methods**
### Negative

Converting another parameter’s results to negative values.

**Parameters**

- **other_param** *(imgaug.parameters.StochasticParameter)* – Other parameter whose sampled values are to be modified.

- **mode** *(‘invert’, ‘reroll’, optional)* – How to change the signs. Valid values are invert and reroll.invert means that wrong signs are simply flipped. reroll means that all samples with wrong signs are sampled again, optionally many times, until they randomly end up having the correct sign.

- **reroll_count_max** *(int, optional)* – If mode is set to reroll, this determines how often values may be rerolled before giving up and simply flipping the sign (as in mode="invert"). This shouldn’t be set too high, as rerolling is expensive.

**Examples**

```python
>>> param = Negative(Normal(0, 1), mode="reroll")
```

Generates a normal distribution that has only negative values.

### Normal

Parameter that resembles a (continuous) normal distribution.

This is a wrapper around numpy’s random.normal().

**Parameters**

- **loc** *(number or imgaug.parameters.StochasticParameter)* – The mean of the normal distribution. If StochasticParameter, the mean will be sampled once per call to imgaug.parameters.Normal._draw_samples().

- **scale** *(number or imgaug.parameters.StochasticParameter)* – The standard deviation of the normal distribution. If StochasticParameter, the scale will be sampled once per call to imgaug.parameters.Normal._draw_samples().

**Examples**

```python
>>> param = Normal(Choice([-1.0, 1.0]), 1.0)
```

A standard normal distribution, which’s mean is shifted either 1.0 to the left or 1.0 to the right.
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td>draw_distribution_graph(title, size, bins)</td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td>draw_sample([random_state])</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

class imgaug.parameters.Poisson(lam)

Bases: imgaug.parameters.StochasticParameter

Parameter that resembles a poisson distribution.

A poisson distribution with lambda=0 has its highest probability at point 0 and decreases quickly from there. Poisson distributions are discrete and never negative.

Parameters

- lam (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Lambda parameter of the poisson distribution.
  - If a number, this number will be used as a constant value.
  - If a tuple of two numbers (a, b), the value will be sampled from the range \([a, b)\) once per call to imgaug.parameters.Poisson._draw_samples().
  - If a list of numbers, a random value will be picked from the list per call.
  - If a StochasticParameter, that parameter will be queried once per call.

Examples

```python
>>> param = Poisson(1)
```

Sample from a poisson distribution with lambda=1.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td>draw_distribution_graph(title, size, bins)</td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td>draw_sample([random_state])</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

imgaug.parameters.Positive(other_param, mode='invert', reroll_count_max=2)

Converts another parameter’s results to positive values.

Parameters

- other_param (imgaug.parameters.StochasticParameter) – Other parameter which’s sampled values are to be modified.
• **mode** (*str*, optional) – How to change the signs. Valid values are `invert` and `reroll`. `invert` means that wrong signs are simply flipped. `reroll` means that all samples with wrong signs are sampled again, optionally many times, until they randomly end up having the correct sign.

• **reroll_count_max** (*int*, optional) – If **mode** is set to `reroll`, this determines how often values may be rerolled before giving up and simply flipping the sign (as in **mode**="invert"). This shouldn’t be set too high, as rerolling is expensive.

### Examples

```python
>>> param = Positive(Normal(0, 1), mode="reroll")
```

Generates a normal distribution that has only positive values.

```
class imgaug.parameters.Power(other_param, val, elementwise=False)
Bases: imgaug.parameters.StochasticParameter
```

Parameter to exponentiate another parameter’s results with.

**Parameters**

• **other_param** (*number or tuple of number or list of number or imgaug.parameters.StochasticParameter*) – Other parameter which’s sampled values are to be modified.

• **val** (*number or tuple of number or list of number or imgaug.parameters.StochasticParameter*) – Value to use exponentiate the other parameter’s results with. If this is a StochasticParameter, either a single or multiple values will be sampled and used as the exponents.

• **elementwise** (*bool*, optional) – Controls the sampling behaviour when **val** is a StochasticParameter. If set to False, a single value will be sampled from **val** and used as the exponent for all values generated by **other_param**. If set to True and `_draw_samples(size=S)` is called, **S** values will be sampled from **val** and used as the exponents for the results of **other_param**.

### Examples

```python
>>> param = Power(Uniform(0.0, 1.0), 2)
```

Converts a uniform range \([0.0, 1.0)\) to a distribution that is peaked towards 1.0.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td><code>draw_distribution_graph([title, size, bins])</code></td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td><code>draw_sample([random_state])</code></td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td><code>draw_samples(size[, random_state])</code></td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>
class imgaug.parameters.RandomSign(\texttt{other\_param, p\_positive=0.5})

Bases: imgaug.parameters.StochasticParameter

Converts another parameter’s results randomly to positive or negative values.

Parameters

- \texttt{other\_param} (imgaug.parameters.StochasticParameter) – Other parameter which’s sampled values are to be modified.
- \texttt{p\_positive} (number) – Fraction of values that are supposed to be turned to positive values.

Examples

```python
>>> param = RandomSign(Poisson(1))
```

Generates a poisson distribution with \texttt{alpha=1} that is mirrored at the y-axis.

Methods

- \texttt{copy()} Create a shallow copy of this parameter.
- \texttt{deepcopy()} Create a deep copy of this parameter.
- \texttt{draw\_distribution\_graph([title, size, bins])} Generate a plot (image) that shows the parameter’s distribution of values.
- \texttt{draw\_sample([random\_state])} Draws a single sample value from this parameter.
- \texttt{draw\_samples(size[, random\_state])} Draws one or more sample values from the parameter.

class imgaug.parameters.Sigmoid(\texttt{other\_param, threshold=(-10, 10), activated=True, mul=1, add=0})

Bases: imgaug.parameters.StochasticParameter

Applies a sigmoid function to the outputs of another parameter.

This is intended to be used in combination with SimplexNoise or FrequencyNoise. It pushes the noise values away from \(-0.5\) and towards \(0.0\) or \(1.0\), making the noise maps more binary.

Parameters

- \texttt{other\_param} (imgaug.parameters.StochasticParameter) – The other parameter to which the sigmoid will be applied.
- \texttt{threshold} (number or tuple of number or iterable of number or \texttt{imgaug.parameters.StochasticParameter, optional}) – Sets the value of the sigmoid’s saddle point, i.e. where values start to quickly shift from 0.0 to 1.0. This may be set using a single number, a tuple \((a, b)\) (will result in a random threshold \(a<=x<b\) per call), a list of numbers (will result in a random threshold drawn from the list per call) or a \texttt{StochasticParameter} (will be queried once per call to determine the threshold).
- \texttt{activated} (bool or number, optional) – Defines whether the sigmoid is activated. If this is False, the results of other_param will not be altered. This may be set to a float value \(p\) with \(0.0\leq p\leq 1.0\), which will result in \texttt{activated} being True in \(p\) percent of all calls.
- \texttt{mul} (number, optional) – The results of \texttt{other\_param} will be multiplied with this value before applying the sigmoid. For noise values (range \([0.0, 1.0]\)) this should be set to about 20.
- **add** (*number, optional*) – This value will be added to the results of *other_param* before applying the sigmoid. For noise values (range \([0.0, 1.0]\)) this should be set to about -10.0, provided *mul* was set to 20.

**Examples**

```python
>>> param = Sigmoid(SimplexNoise(), activated=0.5, mul=20, add=-10)
```

Applies a sigmoid to simplex noise in 50 percent of all calls. The noise results are modified to match the sigmoid’s expected value range. The sigmoid’s outputs are in the range \([0.0, 1.0]\).

**Methods**

- `copy()` Create a shallow copy of this parameter.
- `create_for_noise(*other_param[, threshold, ...])` Creates a Sigmoid that is adjusted to be used with noise parameters, i.e.
- `deepcopy()` Create a deep copy of this parameter.
- `draw_distribution_graph(*title, size, bins)` Generate a plot (image) that shows the parameter’s distribution of values.
- `draw_sample([random_state])` Draws a single sample value from this parameter.
- `draw_samples(size[, random_state])` Draws one or more sample values from the parameter.

**static create_for_noise** (*other_param, threshold=(-10, 10), activated=True*)

Creates a Sigmoid that is adjusted to be used with noise parameters, i.e. with parameters which’s output values are in the range \([0.0, 1.0]\).

**Parameters**

- **other_param** (*imgaug.parameters.StochasticParameter*) – See `imgaug.parameters.Sigmoid.__init__()`.  
- **threshold** (*number or tuple of number or iterable of number or imgaug.parameters.StochasticParameter, optional*) – See `imgaug.parameters.Sigmoid.__init__()`.  
- **activated** (*bool or number, optional*) – See `imgaug.parameters.Sigmoid.__init__()`.  

**Returns** A sigmoid adjusted to be used with noise.

**Return type** `Sigmoid`

**class** `imgaug.parameters.SimplexNoise(size_px_max=(2, 16), upscale_method=['linear', 'nearest'])`

Bases: `imgaug.parameters.StochasticParameter`

A parameter that generates simplex noise of varying resolutions.

This parameter expects to sample noise for 2d planes, i.e. for sizes \((H, W)\) and will return a value in the range \([0.0, 1.0]\) per location in that plane.

The noise is sampled from low resolution planes and upscaled to the requested height and width. The size of the low resolution plane may be defined (high values can be slow) and the interpolation method for upscaling can be set.

**Parameters**
size_px_max (int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional) – Size in pixels of the low resolution plane. A single int will be used as a constant value. A tuple of two ints \((a, b)\) will result in random values sampled from \([a, b]\). A list of ints will result in random values being sampled from that list. A StochasticParameter will be queried once per call to imgaug.parameters.SimplexNoise._draw_samples().

upscale_method (str or int or imgaug.parameters.StochasticParameter, optional) – Upsampling/interpolation method to use. This is used after the sampling is finished and the low resolution plane has to be upsampled to the requested size in _draw_samples(size, ...). The method may be the same as in imgaug.imresize_many_images(). Usually nearest or linear are good choices. nearest will result in rectangles with sharp edges and linear in rectangles with blurry and round edges. The method may be provided as a StochasticParameter, which will be queried per call to _draw_samples().

Examples

```python
>>> param = SimplexNoise(upscale_method="linear")
```

Results in smooth simplex noise of varying sizes.

```python
>>> param = SimplexNoise(size_px_max=(8, 16), upscale_method="nearest")
```

Results in rectangular simplex noise of rather high detail.

Methods

```python
copy() Create a shallow copy of this parameter.
deepecopy() Create a deep copy of this parameter.
draw_distribution_graph([title, size, bins]) Generate a plot (image) that shows the parameter’s distribution of values.
draw_sample([random_state]) Draws a single sample value from this parameter.
draw_samples(size[, random_state]) Draws one or more sample values from the parameter.
```

class imgaug.parameters.StochasticParameter

Bases: object

Abstract parent class for all stochastic parameters.

Stochastic parameters are here all parameters from which values are supposed to be sampled. Usually the sampled values are to a degree random. E.g. a stochastic parameter may be the range \([-10, 10]\), with sampled values being 5.2, -3.7, -9.7 and 6.4.

Methods

```python
copy() Create a shallow copy of this parameter.
deepecopy() Create a deep copy of this parameter.
```
Table 28 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>draw_distribution_graph</code></td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td><code>draw_sample</code></td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td><code>draw_samples</code></td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

```python
copy()  
Create a shallow copy of this parameter.

Returns  Shallow copy.
Return type  `imgaug.parameters.StochasticParameter`

deepcopy()  
Create a deep copy of this parameter.

Returns  Deep copy.
Return type  `imgaug.parameters.StochasticParameter`

draw_distribution_graph(title=None, size=(1000, 1000), bins=100)  
Generate a plot (image) that shows the parameter’s distribution of values.

Parameters

- **title** *(*Optional*) – Title of the plot. None is automatically replaced by a title derived from `str(param)`. If set to False, no title will be shown.
- **size** *(tuple of int)* – Number of points to sample. This is always expected to have at least two values. The first defines the number of sampling runs, the second (and further) dimensions define the size assigned to each `imgaug.parameters.StochasticParameter.draw_samples()` call. E.g. (10, 20, 15) will lead to 10 calls of `draw_samples(size=(20, 15))`. The results will be merged to a single 1d array.
- **bins** *(int)* – Number of bins in the plot histograms.

Returns  data – Image of the plot.
Return type  *(H,W,3) ndarray*

draw_sample(random_state=None)  
Draws a single sample value from this parameter.

Parameters  random_state *(*Optional*) – A random state to use during the sampling process. If None, the libraries global random state will be used.

Returns  A single sample value.
Return type  *any*

draw_samples(size, random_state=None)  
Draws one or more sample values from the parameter.

Parameters

- **size** *(tuple of int or int)* – Number of sample values by dimension.
- **random_state** *(*Optional*) – A random state to use during the sampling process. If None, the libraries global random state will be used.

Returns  samples – Sampled values. Usually a numpy ndarray of basically any dtype, though not strictly limited to numpy arrays. Its shape is expected to match `size`.
Return type  iterable

class imgaug.parameters.Subtract (other_param, val, elementwise=False)

Bases: imgaug.parameters.StochasticParameter

Parameter to subtract from another parameter’s results.

Parameters

• other_param  (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Other parameter which’s sampled values are to be modified.

• val  (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Value to add to the other parameter’s results. If this is a StochasticParameter, either a single or multiple values will be sampled and subtracted.

• elementwise  (bool, optional) – Controls the sampling behaviour when val is a StochasticParameter. If set to False, a single value will be sampled from val and subtracted from all values generated by other_param. If set to True and _draw_samples(size=S) is called, S values will be sampled from val and subtracted from the results of other_param.

Examples

```python
>>> param = Subtract(Uniform(0.0, 1.0), 1.0)
```

Converts a uniform range [0.0, 1.0) to [-1.0, 0.0).

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td>draw_distribution_graph([title, size, bins])</td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td>draw_sample([random_state])</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

class imgaug.parameters.TruncatedNormal (loc, scale, low=-inf, high=inf)

Bases: imgaug.parameters.StochasticParameter

Parameter that resembles a truncated normal distribution.

A truncated normal distribution is very close to a normal distribution except the domain is smoothly bounded. This is a wrapper around scipy.stats.truncnorm.

Parameters

• loc  (number or imgaug.parameters.StochasticParameter) – The mean of the normal distribution. If StochasticParameter, the mean will be sampled once per call to imgaug.parameters.TruncatedNormal._draw_samples().

• scale  (number or imgaug.parameters.StochasticParameter) – The standard deviation of the normal distribution. If StochasticParameter, the scale will be sampled once per call to imgaug.parameters.TruncatedNormal._draw_samples().
• **low** *(number or imgaug.parameters.StochasticParameter)* – The minimum value of the truncated normal distribution. If StochasticParameter, the scale will be sampled once per call to `imgaug.parameters.TruncatedNormal._draw_samples()`.

• **high** *(number or imgaug.parameters.StochasticParameter)* – The maximum value of the truncated normal distribution. If StochasticParameter, the scale will be sampled once per call to `imgaug.parameters.TruncatedNormal._draw_samples()`.

### Examples

```python
>>> param = TruncatedNormal(0, 5.0, low=-10, high=10)
>>> samples = param.draw_samples(100, random_state=np.random.RandomState(0))
>>> assert np.all(samples >= -10)
>>> assert np.all(samples <= 10)
```

### Methods

- **copy()**
  - Create a shallow copy of this parameter.

- **deepcopy()**
  - Create a deep copy of this parameter.

- **draw_distribution_graph([title, size, bins])**
  - Generate a plot (image) that shows the parameter’s distribution of values.

- **draw_sample([random_state])**
  - Draws a single sample value from this parameter.

- **draw_samples(size[, random_state])**
  - Draws one or more sample values from the parameter.

### class `imgaug.parameters.Uniform(a, b)`

**Bases:** `imgaug.parameters.StochasticParameter`

Parameter that resembles a continuous uniform range `[a, b)`.

**Parameters**

- **a** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)*
  - Lower bound of the sampling range. Values will be sampled from `a<=x<`b. All sampled values will be continuous. If `a` is a StochasticParameter, it will be queried once per sampling to estimate the value of `a`. If `a>b`, the values will automatically be flipped. If `a==b`, all generated values will be identical to `a`.

- **b** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)*
  - Upper bound of the sampling range. Values will be sampled from `a<=x<b`. All sampled values will be continuous. If `b` is a StochasticParameter, it will be queried once per sampling to estimate the value of `b`. If `a>b`, the values will automatically be flipped. If `a==b`, all generated values will be identical to `a`.

### Examples

```python
>>> param = Uniform(0, 10.0)
```

Samples random values from the range `[0, 10.0)`.
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td>draw_distribution_graph([title, size, bins])</td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td>draw_sample([random_state])</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

class imgaug.parameters.Weibull(a)

Bases: imgaug.parameters.StochasticParameter

Parameter that resembles a (continuous) weibull distribution.

This is a wrapper around numpy’s numpy.random.weibull().

Parameters:

- **a** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* — Shape parameter of the distribution.
  - If a single number, this number will be used as a constant value.
  - If a tuple of two numbers \((a, b)\), the value will be sampled from the continuous range \([a, b)\) once per call to imgaug.parameters.Weibull._draw_samples().
  - If a list of numbers, a random value will be picked from the list per call.
  - If a StochasticParameter, that parameter will be queried once per call.

Examples

```python
>>> param = Weibull(a=0.5)
```

A weibull distribution with shape 0.5.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this parameter.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this parameter.</td>
</tr>
<tr>
<td>draw_distribution_graph([title, size, bins])</td>
<td>Generate a plot (image) that shows the parameter’s distribution of values.</td>
</tr>
<tr>
<td>draw_sample([random_state])</td>
<td>Draws a single sample value from this parameter.</td>
</tr>
<tr>
<td>draw_samples(size[, random_state])</td>
<td>Draws one or more sample values from the parameter.</td>
</tr>
</tbody>
</table>

imgaug.parameters.both_np_float_if_one_is_float(a, b)

imgaug.parameters.draw_distributions_grid(params, rows=None, cols=None, graph_sizes=(350, 350), sample_sizes=None, titles=None)

imgaug.parameters.force_np_float_dtype(val)
13.3 imgaug.multicore

Classes and functions dealing with multicore augmentation.

class imgaug.multicore.BackgroundAugmenter(batch_loader, augseq, queue_size=50, nb_workers='auto')

Bases: object

Class to augment batches in the background (while training on the GPU).
This is a wrapper around the multiprocessing module.

Parameters

- **batch_loader** (*BatchLoader or multiprocessing.Queue*) – BatchLoader object that loads the data fed into the BackgroundAugmenter, or alternatively a Queue. If a Queue, then it must be made sure that a final None in the Queue signals that the loading is finished and no more batches will follow. Otherwise the BackgroundAugmenter will wait forever for the next batch.

- **augseq** (*Augmenter*) – An augmenter to apply to all loaded images. This may be e.g. a Sequential to apply multiple augmenters.

- **queue_size** (*int*) – Size of the queue that is used to temporarily save the augmentation results. Larger values offer the background processes more room to save results when the main process doesn’t load much, i.e. they can lead to smoother and faster training. For large images, high values can block a lot of RAM though.

- **nb_workers** (*‘auto’ or int*) – Number of background workers to spawn. If auto, it will be set to C-1, where C is the number of CPU cores.

Methods

- **get_batch()**
  Returns a batch from the queue of augmented batches.

- **terminate()**
  Terminates all background processes immediately.

- **all_finished()**
get_batch()
Returns a batch from the queue of augmented batches.

If workers are still running and there are no batches in the queue, it will automatically wait for the next batch.

Returns out – One batch or None if all workers have finished.
Return type None or imgaug.Batch

terminate()
Terminates all background processes immediately.
This will also free their RAM.

class imgaug.multicore.BatchLoader(load_batch_func, queue_size=50, nb_workers=1, threaded=True)
Bases: object
Class to load batches in the background.
Loaded batches can be accesses using imgaug.BatchLoader.queue.

Parameters

• load_batch_func (callable or generator) – Generator or generator function (i.e. function that yields Batch objects) or a function that returns a list of Batch objects. Background loading automatically stops when the last batch was yielded or the last batch in the list was reached.

• queue_size (int, optional) – Maximum number of batches to store in the queue. May be set higher for small images and/or small batches.

• nb_workers (int, optional) – Number of workers to run in the background.

• threaded (bool, optional) – Whether to run the background processes using threads (True) or full processes (False).

Methods

all_finished() Determine whether the workers have finished the loading process.

terminate() Stop all workers.

all_finished()
Determine whether the workers have finished the loading process.

Returns out – True if all workers have finished. Else False.

Return type bool
count_workers_alive()
terminate()
Stop all workers.
class imgaug.multicore.Pool(augseq, processes=None, maxtasksperchild=None, seed=None)
Bases: object
Wrapper around the standard library’s multiprocessing.Pool for multicore augmentation.

**Attributes**

- **pool**  Return the multiprocessing.Pool instance or create it if not done yet.

**Methods**

- **close()**  Close the pool gracefully.
- **imap_batches(batches[, chunksize, ...])**  Augment batches from a generator.
- **imap_batches_unordered(batches[, chunksize, ...])**  Augment batches from a generator in a way that does not guarantee to preserve order.
- **join()**  Wait for the workers to exit.
- **map_batches(batches[, chunksize])**  Augment batches.
- **map_batches_async(batches[, chunksize, ...])**  Augment batches asynchronously.
- **terminate()**  Terminate the pool immediately.

**close()**

Close the pool gracefully.

**imap_batches(batches, chunksize=1, output_buffer_size=None)**

Augment batches from a generator.

Pattern for output buffer constraint is from https://stackoverflow.com/a/47058399.

**Parameters**

- **batches**  (generator of imgaug.augmentables.batches.Batch) – The batches to augment, provided as a generator. Each call to the generator should yield exactly one batch.

- **chunksize**  (None or int, optional) – Rough indicator of how many tasks should be sent to each worker. Increasing this number can improve performance.

- **output_buffer_size**  (None or int, optional) – Max number of batches to handle at the same time in the whole pipeline (including already augmented batches that are waiting to be requested). If the buffer size is reached, no new batches will be loaded from batches until a produced (i.e. augmented) batch is consumed (i.e. requested from this method). The buffer is unlimited if this is set to None. For large datasets, this should be set to an integer value to avoid filling the whole RAM if loading+augmentation happens faster than training.

  New in version 0.3.0.

**Yields**  imgaug.augmentables.batches.Batch – Augmented batch.

**imap_batches_unordered(batches, chunksize=1, output_buffer_size=None)**

Augment batches from a generator in a way that does not guarantee to preserve order.

Pattern for output buffer constraint is from https://stackoverflow.com/a/47058399.

**Parameters**

- **batches**  (generator of imgaug.augmentables.batches.Batch) – The batches to augment, provided as a generator. Each call to the generator should yield exactly one batch.

- **chunksize**  (None or int, optional) – Rough indicator of how many tasks should be sent to each worker. Increasing this number can improve performance.

- **output_buffer_size**  (None or int, optional) – Max number of batches to handle at the same time in the whole pipeline (including already augmented batches that are waiting to
be requested). If the buffer size is reached, no new batches will be loaded from batches until a produced (i.e. augmented) batch is consumed (i.e. requested from this method). The buffer is unlimited if this is set to None. For large datasets, this should be set to an integer value to avoid filling the whole RAM if loading+augmentation happens faster than training.

_New in version 0.3.0._


**join()**

Wait for the workers to exit.

This may only be called after calling `imgaug.multicore.Pool.join()` or `imgaug.multicore.Pool.terminate()`.

**map_batches (batches, chunksize=None)**

Augment batches.

**Parameters**

- **batches** *(list of imgaug.augmentables.batches.Batch)* – The batches to augment.
- **chunksize** *(None or int, optional)* – Rough indicator of how many tasks should be sent to each worker. Increasing this number can improve performance.

**Returns** Augmented batches.

**Return type** list of imgaug.augmentables.batches.Batch

**map_batches_async (batches, chunksize=None, callback=None, error_callback=None)**

Augment batches asynchronously.

**Parameters**

- **batches** *(list of imgaug.augmentables.batches.Batch)* – The batches to augment.
- **chunksize** *(None or int, optional)* – Rough indicator of how many tasks should be sent to each worker. Increasing this number can improve performance.
- **callback** *(None or callable, optional)* – Function to call upon finish. See `multiprocessing.Pool`.
- **error_callback** *(None or callable, optional)* – Function to call upon errors. See `multiprocessing.Pool`.

**Returns** Asynchronous result. See `multiprocessing.Pool`.

**Return type** multiprocessing.MapResult

**pool**

Return the multiprocessing.Pool instance or create it if not done yet.

**Returns** The multiprocessing.Pool used internally by this imgaug.multicore.Pool.

**Return type** multiprocessing.Pool

**terminate()**

Terminate the pool immediately.

## 13.4 imgaug.dtypes

`imgaug.dtypes.clip_(array, min_value, max_value)`
13.5 `imgaug.augmentables.batches`

**class** `imgaug.augmentables.batches.Batch(images=None, heatmaps=None, segmentation_maps=None, keypoints=None, bounding_boxes=None, polygons=None, line_strings=None, data=None)`

Bases: `object`

Class encapsulating a batch before and after augmentation.

**Parameters**

- **images** *(None or (N,H,W,C) ndarray or list of (H,W,C) ndarray)* – The images to augment.
- **heatmaps** *(None or list of `imgaug.augmentables.heatmaps.HeatmapsOnImage`)* – The heatmaps to augment.
- **segmentation_maps** *(None or list of `imgaug.augmentables.segmaps.SegmentationMapOnImage`)* – The segmentation maps to augment.
- **keypoints** *(None or list of `imgaug.augmentables.kps.KeypointOnImage`)* – The keypoints to augment.
- **bounding_boxes** *(None or list of `imgaug.augmentables.bbs.BoundingBoxesOnImage`)* – The bounding boxes to augment.
- **polygons** *(None or list of `imgaug.augmentables.polys.PolygonsOnImage`)* – The polygons to augment.
- **line_strings** *(None or list of `imgaug.augmentables.lines.LineStringsOnImage`)* – The line strings to augment.
- **data** – Additional data that is saved in the batch and may be read out after augmentation. This could e.g. contain filepaths to each image in `images`. As this object is usually used for background augmentation with multiple processes, the augmented Batch objects might not be returned in the original order, making this information useful.

**Attributes**

- **bounding_boxes** *Deprecated*. Use `Batch.bounding_boxes_unaug` instead.
heatmaps Deprecated. Use Batch.heatmaps_unaug instead.

images Deprecated. Use Batch.images_unaug instead.

keypoints Deprecated. Use Batch.keypoints_unaug instead.

segmentation_maps Deprecated. Use Batch.segmentation_maps_unaug instead.

Methods

bounding_boxes
  Deprecated. Use Batch.bounding_boxes_unaug instead.

deprecated: Method to use when deprecated.

deepcopy (images_unaug='DEFAULT', images_aug='DEFAULT', heatmaps_unaug='DEFAULT', heatmaps_aug='DEFAULT', segmentation_maps_unaug='DEFAULT', segmentation_maps_aug='DEFAULT', keypoints_unaug='DEFAULT', keypoints_aug='DEFAULT', bounding_boxes_unaug='DEFAULT', bounding_boxes_aug='DEFAULT', polygons_unaug='DEFAULT', polygons_aug='DEFAULT', line_strings_unaug='DEFAULT', line_strings_aug='DEFAULT')

heatmaps
  Deprecated. Use Batch.heatmaps_unaug instead.

images
  Deprecated. Use Batch.images_unaug instead.

keypoints
  Deprecated. Use Batch.keypoints_unaug instead.

segmentation_maps
  Deprecated. Use Batch.segmentation_maps_unaug instead.

class imgaug.augmentables.batches.UnnormalizedBatch (images=None, heatmaps=None, segmentation_maps=None, keypoints=None, bounding_boxes=None, polygons=None, line_strings=None, data=None)

Bases: object

Class for batches of unnormalized data before and after augmentation.

Parameters

- **images** (None or (N,H,W,C) ndarray or (N,H,W) ndarray or iterable of (H,W,C) ndarray or iterable of (H,W) ndarray) – The images to augment.

- **heatmaps** (None or (N,H,W,C) ndarray or imgaug.augmentables.heatmaps.HeatmapsOnImage or iterable of (H,W,C) ndarray or iterable of imgaug.augmentables.heatmaps.HeatmapsOnImage) – The heatmaps to augment. If anything else than HeatmapsOnImage, then the number of heatmaps must match the number of images provided via parameter images. The number is contained either in N or the first iterable’s size.

- **segmentation_maps** (None or (N,H,W) ndarray or imgaug.augmentables.segmaps.SegmentationMapOnImage or iterable of (H,W) ndarray)
or iterable of 

or iterable of imgaug.augmentables.segmaps.SegmentationMapOnImage) – The segmentation maps to augment. If anything else than SegmentationMapOnImage, then the number of segmaps must match the number of images provided via parameter images. The number is contained either in N or the first iterable’s size.

• keypoints (None or list of (N,K,2) ndarray or tuple of number or imgaug.augmentables.kps.Keypoint or iterable of (K,2) ndarray or iterable of tuple of number or iterable of imgaug.augmentables.kps.Keypoint or iterable of imgaug.augmentables.kps.KeypointOnImage or iterable of iterable of tuple of number or iterable of iterable of imgaug.augmentables.kps.Keypoint) – The keypoints to augment. If a tuple (or iterable(s) of tuple), then interpreted as (x,y) coordinates and must hence contain two numbers. A single tuple represents a single coordinate on one image, an iterable of tuples the coordinates on one image and an iterable of iterable of tuples the coordinates on several images. Analogous if Keypoint objects are used instead of tuples. If an ndarray, then N denotes the number of images and K the number of keypoints on each image. If anything else than KeypointsOnImage is provided, then the number of keypoint groups must match the number of images provided via parameter images. The number is contained e.g. in N or in case of “iterable of iterable of tuples” in the first iterable’s size.

• bounding_boxes (None or (N,B,4) ndarray or tuple of number or imgaug.augmentables.bbs.BoundingBox or imgaug.augmentables.bbs.BoundingBoxesOnImage or iterable of (B,4) ndarray or iterable of tuple of number or iterable of imgaug.augmentables.bbs.BoundingBox or iterable of iterable of imgaug.augmentables.bbs.BoundingBoxesOnImage or iterable of iterable of tuple of number or iterable of iterable of imgaug.augmentables.bbs.BoundingBox) – The bounding boxes to augment. This is analogous to the keypoints parameter. However, each tuple – and also the last index in case of arrays – has size 4, denoting the bounding box coordinates x1, y1, x2 and y2.

• polygons (None or (N,#polys,#points,2) ndarray or imgaug.augmentables.polys.Polygon or imgaug.augmentables.polys.PolygonsOnImage or iterable of (#polys,#points,2) ndarray or iterable of tuple of number or iterable of imgaug.augmentables.polys.Polygon or iterable of imgaug.augmentables.polys.PolygonsOnImage or iterable of iterable of tuple of number or iterable of iterable of imgaug.augmentables.polys.Polygon or iterable of iterable of tuple of number or iterable of iterable of iterable of imgaug.augmentables.polys.Polygon) – The polygons to augment. This is similar to the keypoints parameter. However, each polygon may be made up of several (x, y) coordinates (three or more are required for valid polygons). The following datatypes will be interpreted as a single polygon on a single image:

- imgaug.augmentables.polys.Polygon
- iterable of tuple of number
- iterable of imgaug.augmentables.kps.Keypoint

The following datatypes will be interpreted as multiple polygons on a single image:

- imgaug.augmentables.polys.PolygonsOnImage
- iterable of imgaug.augmentables.polys.Polygon
- iterable of iterable of tuple of number
- iterable of iterable of iterable of imgaug.augmentables.kps.Keypoint
- iterable of iterable of iterable of imgaug.augmentables.polys.Polygon
The following datatypes will be interpreted as multiple polygons on multiple images:

- `(N,#polys,#points,2)` ndarray
- iterable of `(N,#polys,#points,2)` ndarray
- iterable of iterable of `(N,#polys,#points,2)` ndarray
- iterable of iterable of iterable of `tuple of number`
- iterable of iterable of iterable of `tuple of imgaug.augmentables.kps.Keypoint`

• `line_strings` (None or `(N,#lines,#points,2)` ndarray or `imgaug.augmentables.lines.LineString` or `imgaug.augmentables.lines.LineStringOnImage` or iterable of `(N,#lines,#points,2)` ndarray or iterable of `tuple of number` or iterable of `imgaug.augmentables.kps.Keypoint` or iterable of `imgaug.augmentables.lines.LineString` or iterable of `imgaug.augmentables.lines.LineStringOnImage` or iterable of `tuple of number` or iterable of `imgaug.augmentables.kps.Keypoint`) – The line strings to augment. See `polygons` for more details as polygons follow a similar structure to line strings.

• `data` – Additional data that is saved in the batch and may be read out after augmentation. This could e.g. contain filepaths to each image in `images`. As this object is usually used for background augmentation with multiple processes, the augmented Batch objects might not be returned in the original order, making this information useful.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fill_from_augmented_normalized_batch(batch_aug_norm)</code></td>
<td>Fill this batch with (normalized) augmentation results.</td>
</tr>
<tr>
<td><code>to_normalized_batch()</code></td>
<td>Convert this unnormalized batch to an instance of Batch.</td>
</tr>
</tbody>
</table>

`fill_from_augmented_normalized_batch(batch_aug_norm)`

Fill this batch with (normalized) augmentation results.

This method receives a (normalized) Batch instance, takes all `*_aug` attributes out if it and assigns them to this batch in unnormalized form. Hence, the datatypes of all `*_aug` attributes will match the datatypes of the `*_unaug` attributes.

**Parameters**

- `batch_aug_norm` (`imgaug.augmentables.batches.Batch`) – Batch after normalization and augmentation.

**Returns**

New UnnormalizedBatch instance. All `*_unaug` attributes are taken from the old UnnormalizedBatch (without deepcopying them) and all `*_aug` attributes are taken from `batch_normalized` converted to unnormalized form.

**Return type**

`imgaug.augmentables.batches.UnnormalizedBatch`

`to_normalized_batch()`

Convert this unnormalized batch to an instance of Batch.

As this method is intended to be called before augmentation, it assumes that none of the `*_aug` attributes is yet set. It will produce an AssertionError otherwise.
The newly created Batch’s *unaug attributes will match the ones in this batch, just in normalized form.

**Returns** The batch, with *unaug attributes being normalized.

**Return type** `imgaug.augmentables.batches.Batch`

### 13.6 imgaug.augmentables.bbs

class `imgaug.augmentables.bbs.BoundingBox(x1, y1, x2, y2, label=None)`

*Bases:* `object`

Class representing bounding boxes.

Each bounding box is parameterized by its top left and bottom right corners. Both are given as x and y-coordinates. The corners are intended to lie inside the bounding box area. As a result, a bounding box that lies completely inside the image but has maximum extensions would have coordinates \((0.0, 0.0)\) and \((W - \epsilon, H - \epsilon)\). Note that coordinates are saved internally as floats.

**Parameters**

- `x1` *(number)* – X-coordinate of the top left of the bounding box.
- `y1` *(number)* – Y-coordinate of the top left of the bounding box.
- `x2` *(number)* – X-coordinate of the bottom right of the bounding box.
- `y2` *(number)* – Y-coordinate of the bottom right of the bounding box.
- `label` *(None or str, optional)* – Label of the bounding box, e.g. a string representing the class.

**Attributes**

- `area` Estimate the area of the bounding box.
- `center_x` Estimate the x-coordinate of the center point of the bounding box.
- `center_y` Estimate the y-coordinate of the center point of the bounding box.
- `height` Estimate the height of the bounding box.
- `width` Estimate the width of the bounding box.
- `x1_int` Return the x-coordinate of the top left corner as an integer.
- `x2_int` Return the x-coordinate of the bottom left corner as an integer.
- `y1_int` Return the y-coordinate of the top left corner as an integer.
- `y2_int` Return the y-coordinate of the bottom left corner as an integer.

**Methods**

- `clip_out_of_image(image)` Clip off all parts of the bounding box that are outside of the image.
- `contains(other)` Estimate whether the bounding box contains a point.
- `copy([x1, y1, x2, y2, label])` Create a shallow copy of the BoundingBox object.
- `cut_out_of_image(*args, **kwargs)` *Deprecated.*
- `deepcopy([x1, y1, x2, y2, label])` Create a deep copy of the BoundingBox object.
- `draw_on_image(image[, color, alpha, size, ...])` Draw the bounding box on an image.

Continued on next page
Table 37 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>extend(all_sides, top, right, bottom, left)</code></td>
<td>Extend the size of the bounding box along its sides.</td>
</tr>
<tr>
<td><code>extract_from_image(image[, pad, pad_max, ...])</code></td>
<td>Extract the image pixels within the bounding box.</td>
</tr>
<tr>
<td><code>intersection(other[, default])</code></td>
<td>Compute the intersection bounding box of this bounding box and another one.</td>
</tr>
<tr>
<td><code>iou(other)</code></td>
<td>Compute the IoU of this bounding box with another one.</td>
</tr>
<tr>
<td><code>is_fully_within_image(image)</code></td>
<td>Estimate whether the bounding box is fully inside the image area.</td>
</tr>
<tr>
<td><code>is_out_of_image(image[, fully, partly])</code></td>
<td>Estimate whether the bounding box is partially or fully outside of the image area.</td>
</tr>
<tr>
<td><code>is_partly_within_image(image)</code></td>
<td>Estimate whether the bounding box is at least partially inside the image area.</td>
</tr>
<tr>
<td><code>project(from_shape, to_shape)</code></td>
<td>Project the bounding box onto a differently shaped image.</td>
</tr>
<tr>
<td><code>shift([top, right, bottom, left])</code></td>
<td>Shift the bounding box from one or more image sides, i.e.</td>
</tr>
<tr>
<td><code>to_keypoints()</code></td>
<td>Convert the corners of the bounding box to keypoints (clockwise, starting at top left).</td>
</tr>
<tr>
<td><code>union(other)</code></td>
<td>Compute the union bounding box of this bounding box and another one.</td>
</tr>
</tbody>
</table>

---

**area**

Estimate the area of the bounding box.

- **Returns** Area of the bounding box, i.e. `height * width`.
- **Return type** number

**center_x**

Estimate the x-coordinate of the center point of the bounding box.

- **Returns** X-coordinate of the center point of the bounding box.
- **Return type** number

**center_y**

Estimate the y-coordinate of the center point of the bounding box.

- **Returns** Y-coordinate of the center point of the bounding box.
- **Return type** number

**clip_out_of_image(image)**

Clip off all parts of the bounding box that are outside of the image.

- **Parameters** `image ((H,W,...) ndarray or tuple of int)` – Image dimensions to use for the clipping of the bounding box. If an ndarray, its shape will be used. If a tuple, it is assumed to represent the image shape and must contain at least two integers.
- **Returns** `result` – Bounding box, clipped to fall within the image dimensions.
- **Return type** `imgaug.BoundingBox`

**contains(other)**

Estimate whether the bounding box contains a point.

- **Parameters** `other (tuple of number or imgaug.Keypoint)` – Point to check for.
- **Returns** True if the point is contained in the bounding box, False otherwise.
Return type  bool

copy (x1=None, y1=None, x2=None, y2=None, label=None)
Create a shallow copy of the BoundingBox object.

Parameters

- **x1 (None or number)** – If not None, then the x1 coordinate of the copied object will be set to this value.
- **y1 (None or number)** – If not None, then the y1 coordinate of the copied object will be set to this value.
- **x2 (None or number)** – If not None, then the x2 coordinate of the copied object will be set to this value.
- **y2 (None or number)** – If not None, then the y2 coordinate of the copied object will be set to this value.
- **label (None or string)** – If not None, then the label of the copied object will be set to this value.

Returns  Shallow copy.

Return type  imgaug.BoundingBox

cut_out_of_image (*args, **kwargs)
Deprecated. Use BoundingBox.clip_out_of_image() instead. clip_out_of_image() has the exactly same interface.

deepecopy (x1=None, y1=None, x2=None, y2=None, label=None)
Create a deep copy of the BoundingBox object.

Parameters

- **x1 (None or number)** – If not None, then the x1 coordinate of the copied object will be set to this value.
- **y1 (None or number)** – If not None, then the y1 coordinate of the copied object will be set to this value.
- **x2 (None or number)** – If not None, then the x2 coordinate of the copied object will be set to this value.
- **y2 (None or number)** – If not None, then the y2 coordinate of the copied object will be set to this value.
- **label (None or string)** – If not None, then the label of the copied object will be set to this value.

Returns  Deep copy.

Return type  imgaug.BoundingBox

draw_on_image (image, color=(0, 255, 0), alpha=1.0, size=1, copy=True, raise_if_out_of_image=False, thickness=None)
Draw the bounding box on an image.

Parameters

- **image ((H,W,C) ndarray(uint8))** – The image onto which to draw the bounding box.
- **color (iterable of int, optional)** – The color to use, corresponding to the channel layout of the image. Usually RGB.
• **alpha** *(float, optional)* – The transparency of the drawn bounding box, where 1.0 denotes no transparency and 0.0 is invisible.

• **size** *(int, optional)* – The thickness of the bounding box in pixels. If the value is larger than 1, then additional pixels will be added around the bounding box (i.e. extension towards the outside).

• **copy** *(bool, optional)* – Whether to copy the input image or change it in-place.

• **raise_if_out_of_image** *(bool, optional)* – Whether to raise an error if the bounding box is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

• **thickness** *(None or int, optional)* – Deprecated.

**Returns**

- result – Image with bounding box drawn on it.

**Return type** *(H,W,C) ndarray(uint8)*

---

### extend

**extend** *(all_sides=0, top=0, right=0, bottom=0, left=0)*

Extend the size of the bounding box along its sides.

**Parameters**

- **all_sides** *(number, optional)* – Value by which to extend the bounding box size along all sides.

- **top** *(number, optional)* – Value by which to extend the bounding box size along its top side.

- **right** *(number, optional)* – Value by which to extend the bounding box size along its right side.

- **bottom** *(number, optional)* – Value by which to extend the bounding box size along its bottom side.

- **left** *(number, optional)* – Value by which to extend the bounding box size along its left side.

**Returns**

- Extended bounding box.

**Return type** 

- imgaug.BoundingBox

---

### extract_from_image

**extract_from_image** *(image, pad=True, pad_max=None, prevent_zero_size=True)*

Extract the image pixels within the bounding box.

This function will zero-pad the image if the bounding box is partially/fully outside of the image.

**Parameters**

- **image** *((H,W) ndarray or (H,W,C) ndarray)* – The image from which to extract the pixels within the bounding box.

- **pad** *(bool, optional)* – Whether to zero-pad the image if the object is partially/fully outside of it.

- **pad_max** *(None or int, optional)* – The maximum number of pixels that may be zero-padded on any side, i.e. if this has value \( N \) the total maximum of added pixels is \( 4 \times N \). This option exists to prevent extremely large images as a result of single points being moved very far away during augmentation.

- **prevent_zero_size** *(bool, optional)* – Whether to prevent height or width of the extracted image from becoming zero. If this is set to True and height or width of the bounding box is below 1, the height/width will be increased to 1. This can be useful to prevent problems,
e.g. with image saving or plotting. If it is set to False, images will be returned as \((H', W')\) or \((H', W', 3)\) with \(H\) or \(W\) potentially being 0.

**Returns** image – Pixels within the bounding box. Zero-padded if the bounding box is partially/fully outside of the image. If prevent_zero_size is activated, it is guaranteed that \(H' > 0\) and \(W' > 0\), otherwise only \(H' \geq 0\) and \(W' \geq 0\).

**Return type** \((H', W')\) ndarray or \((H', W', C)\) ndarray

**height**
Estimate the height of the bounding box.

**Returns** Height of the bounding box.

**Return type** number

**intersection**(other, default=None)
Compute the intersection bounding box of this bounding box and another one.

Note that in extreme cases, the intersection can be a single point, meaning that the intersection bounding box will exist, but then also has a height and width of zero.

**Parameters**

- **other** (imgaug.BoundingBox) – Other bounding box with which to generate the intersection.
- **default** (any, optional) – Default value to return if there is no intersection.

**Returns** Intersection bounding box of the two bounding boxes if there is an intersection. If there is no intersection, the default value will be returned, which can be anything.

**Return type** imgaug.BoundingBox or any

**iou**(other)
Compute the IoU of this bounding box with another one.

IoU is the intersection over union, defined as:

```
\[
\text{area}(\text{intersection}(A, B)) / \text{area}(\text{union}(A, B))
\]
\[
= \text{area}(\text{intersection}(A, B)) / (\text{area}(A) + \text{area}(B) - \text{area}(\text{intersection}(A, B)))
\]
```

**Parameters** other (imgaug.BoundingBox) – Other bounding box with which to compare.

**Returns** IoU between the two bounding boxes.

**Return type** float

**is_fully_within_image**(image)
Estimate whether the bounding box is fully inside the image area.

**Parameters** image ((H, W, ...) ndarray or tuple of int) – Image dimensions to use. If an ndarray, its shape will be used. If a tuple, it is assumed to represent the image shape and must contain at least two integers.

**Returns** True if the bounding box is fully inside the image area. False otherwise.

**Return type** bool

**is_out_of_image**(image, fully=True, partly=False)
Estimate whether the bounding box is partially or fully outside of the image area.

**Parameters**
• **image** ((H,W,...) ndarray or tuple of int) – Image dimensions to use. If an ndarray, its shape will be used. If a tuple, it is assumed to represent the image shape and must contain at least two integers.

• **fully** (bool, optional) – Whether to return True if the bounding box is fully outside of the image area.

• **partly** (bool, optional) – Whether to return True if the bounding box is at least partially outside of the image area.

**Returns** True if the bounding box is partially/fully outside of the image area, depending on defined parameters. False otherwise.

**Return type** bool

**is_partly_within_image** (image)
Estimate whether the bounding box is at least partially inside the image area.

**Parameters**

- **image** ((H,W,...) ndarray or tuple of int) – Image dimensions to use. If an ndarray, its shape will be used. If a tuple, it is assumed to represent the image shape and must contain at least two integers.

**Returns** True if the bounding box is at least partially inside the image area. False otherwise.

**Return type** bool

**project** (from_shape, to_shape)
Project the bounding box onto a differently shaped image.

E.g. if the bounding box is on its original image at x1=(10 of 100 pixels) and y1=(20 of 100 pixels) and is projected onto a new image with size (width=200, height=200), its new position will be (x1=20, y1=40). (Analogous for x2/y2.)

This is intended for cases where the original image is resized. It cannot be used for more complex changes (e.g. padding, cropping).

**Parameters**

- **from_shape** (tuple of int or ndarray) – Shape of the original image. (Before resize.)

- **to_shape** (tuple of int or ndarray) – Shape of the new image. (After resize.)

**Returns** out – BoundingBox object with new coordinates.

**Return type** imgaug.BoundingBox

**shift** (top=None, right=None, bottom=None, left=None)
Shift the bounding box from one or more image sides, i.e. move it on the x/y-axis.

**Parameters**

- **top** (None or int, optional) – Amount of pixels by which to shift the bounding box from the top.

- **right** (None or int, optional) – Amount of pixels by which to shift the bounding box from the right.

- **bottom** (None or int, optional) – Amount of pixels by which to shift the bounding box from the bottom.

- **left** (None or int, optional) – Amount of pixels by which to shift the bounding box from the left.

**Returns** result – Shifted bounding box.

**Return type** imgaug.BoundingBox
to_keypoints()
Convert the corners of the bounding box to keypoints (clockwise, starting at top left).

Returns Corners of the bounding box as keypoints.
Return type list of imgaug.Keypoint

union (other)
Compute the union bounding box of this bounding box and another one.
This is equivalent to drawing a bounding box around all corners points of both bounding boxes.

Parameters other (imgaug.BoundingBox) – Other bounding box with which to generate the union.

Returns Union bounding box of the two bounding boxes.
Return type imgaug.BoundingBox

width
Estimate the width of the bounding box.

Returns Width of the bounding box.
Return type number

x1_int
Return the x-coordinate of the top left corner as an integer.

Returns X-coordinate of the top left corner, rounded to the closest integer.
Return type int

x2_int
Return the x-coordinate of the bottom left corner as an integer.

Returns X-coordinate of the bottom left corner, rounded to the closest integer.
Return type int

y1_int
Return the y-coordinate of the top left corner as an integer.

Returns Y-coordinate of the top left corner, rounded to the closest integer.
Return type int

y2_int
Return the y-coordinate of the bottom left corner as an integer.

Returns Y-coordinate of the bottom left corner, rounded to the closest integer.
Return type int

class imgaug.augmentables.bbs.BoundingBoxesOnImage (bounding_boxes, shape)
Bases: object

Object that represents all bounding boxes on a single image.

Parameters

• bounding_boxes (list of imgaug.BoundingBox) – List of bounding boxes on the image.

• shape (tuple of int) – The shape of the image on which the bounding boxes are placed.
Examples

```python
>>> image = np.zeros((100, 100))
>>> bbs = [
    BoundingBox(x1=10, y1=20, x2=20, y2=30),
    BoundingBox(x1=25, y1=50, x2=30, y2=70)
]
>>> bbs_oi = BoundingBoxesOnImage(bbs, shape=image.shape)
```

Attributes

- **empty** Returns whether this object contains zero bounding boxes.
- **height** Get the height of the image on which the bounding boxes fall.
- **width** Get the width of the image on which the bounding boxes fall.

Methods

- `clip_out_of_image()` Clip off all parts from all bounding boxes that are outside of the image.
- `copy()` Create a shallow copy of the `BoundingBoxesOnImage` object.
- `deepcopy()` Create a deep copy of the `BoundingBoxesOnImage` object.
- `draw_on_image(image[, color, alpha, size,...])` Draw all bounding boxes onto a given image.
- `from_xyxy_array(xyxy, shape)` Convert an (N,4) ndarray to a `BoundingBoxesOnImage` object.
- `on(image)` Project bounding boxes from one image to a new one.
- `remove_out_of_image([fully, partly])` Remove all bounding boxes that are fully or partially outside of the image.
- `shift([top, right, bottom, left])` Shift all bounding boxes from one or more image sides, i.e.
- `to_xyxy_array([dtype])` Convert the `BoundingBoxesOnImage` object to an (N,4) ndarray.
deepcopy()

Create a deep copy of the BoundingBoxesOnImage object.

Returns
Deep copy.

Return type
imgaug.BoundingBoxesOnImage
draw_on_image(image, color=(0, 255, 0), alpha=1.0, size=1, copy=True, raise_if_out_of_image=False, thickness=None)

Draw all bounding boxes onto a given image.

Parameters

• image ((H,W,3) ndarray) – The image onto which to draw the bounding boxes. This image should usually have the same shape as set in BoundingBoxesOnImage.shape.

• color (int or list of int or tuple of int or (3,) ndarray, optional) – The RGB color of all bounding boxes. If a single int C, then that is equivalent to (C, C, C).

• alpha (float, optional) – Alpha/opacity of the bounding box.

• size (int, optional) – Thickness in pixels.

• copy (bool, optional) – Whether to copy the image before drawing the bounding boxes.

• raise_if_out_of_image (bool, optional) – Whether to raise an exception if any bounding box is outside of the image.

• thickness (None or int, optional) – Deprecated.

Returns
image – Image with drawn bounding boxes.

Return type
(H,W,3) ndarray
empty

Returns whether this object contains zero bounding boxes.

Returns
True if this object contains zero bounding boxes.

Return type
bool
classmethod from_xyxy_array(xyxy, shape)

Convert an (N,4) ndarray to a BoundingBoxesOnImage object.

This is the inverse of imgaug.BoundingBoxesOnImage.to_xyxy_array().

Parameters

• xyxy ((N,4) ndarray) – Array containing the corner coordinates (top-left, bottom-right) of N bounding boxes in the form (x1, y1, x2, y2). Should usually be of dtype float32.

• shape (tuple of int) – Shape of the image on which the bounding boxes are placed. Should usually be (H, W, 3) or (H, W).

Returns
Object containing a list of BoundingBox objects following the provided corner coordinates.

Return type
imgaug.BoundingBoxesOnImage
height

Get the height of the image on which the bounding boxes fall.

Returns
Image height.

Return type
int
on(image)
Project bounding boxes from one image to a new one.

**Parameters**
- **image** *(ndarray or tuple of int)* – New image onto which the bounding boxes are to be projected. May also simply be that new image’s shape tuple.

**Returns**
- **bounding_boxes** – Object containing all projected bounding boxes.

**Return type**
imgaug.BoundingBoxesOnImage

remove_out_of_image(fully=True, partly=False)
Remove all bounding boxes that are fully or partially outside of the image.

**Parameters**
- **fully** *(bool, optional)* – Whether to remove bounding boxes that are fully outside of the image.
- **partly** *(bool, optional)* – Whether to remove bounding boxes that are partially outside of the image.

**Returns**
Reduced set of bounding boxes, with those that were fully/partially outside of the image removed.

**Return type**
imgaug.BoundingBoxesOnImage

shift(top=None, right=None, bottom=None, left=None)
Shift all bounding boxes from one or more image sides, i.e. move them on the x/y-axis.

**Parameters**
- **top** *(None or int, optional)* – Amount of pixels by which to shift all bounding boxes from the top.
- **right** *(None or int, optional)* – Amount of pixels by which to shift all bounding boxes from the right.
- **bottom** *(None or int, optional)* – Amount of pixels by which to shift all bounding boxes from the bottom.
- **left** *(None or int, optional)* – Amount of pixels by which to shift all bounding boxes from the left.

**Returns**
Shifted bounding boxes.

**Return type**
imgaug.BoundingBoxesOnImage

to_xyxy_array(dtype=<class 'numpy.float32'>)
Convert the BoundingBoxesOnImage object to an (N,4) ndarray.

This is the inverse of imgaug.BoundingBoxesOnImage.from_xyxy_array().

**Parameters**
- **dtype** *(numpy.dtype, optional)* – Desired output datatype of the ndarray.

**Returns**
(N,4) ndarray array, where N denotes the number of bounding boxes and 4 denotes the top-left and bottom-right bounding box corner coordinates in form \((x_1, y_1, x_2, y_2)\).

**Return type**
ndarray

width
Get the width of the image on which the bounding boxes fall.

**Returns**
Image width.

**Return type**
int
13.7 imgaug.augmentables.heatmaps

```python
class imgaug.augmentables.heatmaps.HeatmapsOnImage(arr, shape, min_value=0.0, max_value=1.0)
```

Object representing heatmaps on images.

Parameters:
- **arr** ((H,W) ndarray or (H,W,C) ndarray) – Array representing the heatmap(s). Must be of dtype float32. If multiple heatmaps are provided, then C is expected to denote their number.
- **shape** (tuple of int) – Shape of the image on which the heatmap(s) is/are placed. NOT the shape of the heatmap(s) array, unless it is identical to the image shape (note the likely difference between the arrays in the number of channels). If there is not a corresponding image, use the shape of the heatmaps array.
- **min_value** (float, optional) – Minimum value for the heatmaps that `arr` represents. This will usually be 0.0.
- **max_value** (float, optional) – Maximum value for the heatmaps that `arr` represents. This will usually be 1.0.

Methods:

- `avg_pool(block_size)` Resize the heatmap(s) array using average pooling of a given block/kernel size.
- `change_normalization(arr, source, target)` Change the value range of a heatmap from one min-max to another min-max.
- `copy()` Create a shallow copy of the Heatmaps object.
- `deepcopy()` Create a deep copy of the Heatmaps object.
- `draw([size, cmap])` Render the heatmaps as RGB images.
- `draw_on_image(image[, alpha, cmap, resize])` Draw the heatmaps as overlays over an image.
- `from_0to1(arr_0to1, shape[, min_value, ...])` Create a heatmaps object from an heatmap array containing values ranging from 0.0 to 1.0.
- `from_uint8(arr_uint8, shape[, min_value, ...])` Create a heatmaps object from an heatmap array containing values ranging from 0 to 255.
- `get_arr()` Get the heatmap's array within the value range originally provided in `__init__()`.
- `invert()` Inverts each value in the heatmap, shifting low towards high values and vice versa.
- `max_pool(block_size)` Resize the heatmap(s) array using max-pooling of a given block/kernel size.
- `pad([top, right, bottom, left, mode, cval])` Pad the heatmaps on their top/right/bottom/left side.
- `pad_to_aspect_ratio(aspect_ratio[, mode, ...])` Pad the heatmaps on their sides so that they match a target aspect ratio.
- `resize(sizes[, interpolation])` Resize the heatmap(s) array to the provided size given the provided interpolation.
- `scale(*args, **kwargs)` Deprecated.
- `to_uint8()` Convert this heatmaps object to a 0-to-255 array.

`avg_pool(block_size)`
Resize the heatmap(s) array using average pooling of a given block/kernel size.
Parameters block_size (int or tuple of int) – Size of each block of values to pool, aka kernel size. See imgaug.pool() for details.

Returns Heatmaps after average pooling.
Return type imgaug.HeatmapsOnImage
classmethod change_normalization (arr, source, target)
Change the value range of a heatmap from one min-max to another min-max.
E.g. the value range may be changed from min=0.0, max=1.0 to min=-1.0, max=1.0.

Parameters
  • arr (ndarray) – Heatmap array to modify.
  • source (tuple of float) – Current value range of the input array, given as (min, max), where both are float values.
  • target (tuple of float) – Desired output value range of the array, given as (min, max), where both are float values.

Returns arr_target – Input array, with value range projected to the desired target value range.
Return type ndarray
copy ()
Create a shallow copy of the Heatmaps object.

Returns Shallow copy.
Return type imgaug.HeatmapsOnImage
deepcopy ()
Create a deep copy of the Heatmaps object.

Returns Deep copy.
Return type imgaug.HeatmapsOnImage
draw (size=None, cmap='jet')
Render the heatmaps as RGB images.

Parameters
  • size (None or float or iterable of int or iterable of float, optional) – Size of the rendered RGB image as (height, width). See imgaug.imgaug.imresize_single_image() for details. If set to None, no resizing is performed and the size of the heatmaps array is used.
  • cmap (str or None, optional) – Color map of matplotlib to use in order to convert the heatmaps to RGB images. If set to None, no color map will be used and the heatmaps will be converted to simple intensity maps.

Returns heatmaps_drawn – Rendered heatmaps. One per heatmap array channel. Dtype is uint8.
Return type list of (H,W,3) ndarray
draw_on_image (image, alpha=0.75, cmap='jet', resize='heatmaps')
Draw the heatmaps as overlays over an image.

Parameters
  • image ((H,W,3) ndarray) – Image onto which to draw the heatmaps. Expected to be of dtype uint8.
• **alpha** *(float, optional)* – Alpha/opacity value to use for the mixing of image and heatmaps. Higher values mean that the heatmaps will be more visible and the image less visible.

• **cmap** *(str or None, optional)* – Color map to use. See `imgaug.HeatmapsOnImage.draw()` for details.

• **resize** *(‘heatmaps’, ‘image’), optional* – In case of size differences between the image and heatmaps, either the image or the heatmaps can be resized. This parameter controls which of the two will be resized to the other’s size.

Returns **mix** – Rendered overlays. One per heatmap array channel. Dtype is uint8.

Return type `list of (H,W,3) ndarray`

---

**static from_0to1** *(arr_0to1, shape, min_value=0.0, max_value=1.0)*

Create a heatmaps object from an heatmap array containing values ranging from 0.0 to 1.0.

Parameters

• **arr_0to1** *(H,W) or (H,W,C) ndarray* – Heatmap(s) array, where \( H \) is height, \( W \) is width and \( C \) is the number of heatmap channels. Expected dtype is float32.

• **shape** *(tuple of ints)* – Shape of the image on which the heatmap(s) is/are placed. NOT the shape of the heatmap(s) array, unless it is identical to the image shape (note the likely difference between the arrays in the number of channels). If there is not a corresponding image, use the shape of the heatmaps array.

• **min_value** *(float, optional)* – Minimum value for the heatmaps that the 0-to-1 array represents. This will usually be 0.0. It is used when calling `imgaug.HeatmapsOnImage.get_arr()`, which converts the underlying \((0.0, 1.0)\) array to value range \((\text{min\_value}, \text{max\_value})\). E.g. if you started with heatmaps in the range \((-1.0, 1.0)\) and projected these to \((0.0, 1.0)\), you should call this function with `min_value=-1.0, max_value=1.0` so that `imgaug.HeatmapsOnImage.get_arr()` returns heatmap arrays having value range \((-1.0, 1.0)\).

• **max_value** *(float, optional)* – Maximum value for the heatmaps that 0-to-255 array represents. See parameter `min_value` for details.

Returns **heatmaps** – Heatmaps object.

Return type `imgaug.HeatmapsOnImage`

---

**static from_uint8** *(arr_uint8, shape, min_value=0.0, max_value=1.0)*

Create a heatmaps object from an heatmap array containing values ranging from 0 to 255.

Parameters

• **arr_uint8** *(H,W) or (H,W,C) ndarray* – Heatmap(s) array, where \( H \) is height, \( W \) is width and \( C \) is the number of heatmap channels. Expected dtype is uint8.

• **shape** *(tuple of int)* – Shape of the image on which the heatmap(s) is/are placed. NOT the shape of the heatmap(s) array, unless it is identical to the image shape (note the likely difference between the arrays in the number of channels). If there is not a corresponding image, use the shape of the heatmaps array.

• **min_value** *(float, optional)* – Minimum value for the heatmaps that the 0-to-255 array represents. This will usually be 0.0. It is used when calling `imgaug.HeatmapsOnImage.get_arr()`, which converts the underlying \((0, 255)\) array to value range \((\text{min\_value}, \text{max\_value})\).

• **max_value** *(float, optional)* – Maximum value for the heatmaps that 0-to-255 array represents. See parameter `min_value` for details.
Returns Heatmaps object.

Return type imgaug.HeatmapsOnImage

get_arr()
Get the heatmap's array within the value range originally provided in __init__().

The HeatmapsOnImage object saves heatmaps internally in the value range \(\text{min}=0.0, \text{max}=1.0\). This function converts the internal representation to \(\text{min}=\text{min\_value}, \text{max}=\text{max\_value}\), where \text{min\_value} and \text{max\_value} are provided upon instantiation of the object.

Returns result – Heatmap array. Dtype is float32.

Return type (H,W) ndarray or (H,W,C) ndarray

invert()
Inverts each value in the heatmap, shifting low towards high values and vice versa.

This changes each value to:

\[ v' = \text{max} - (v - \text{min}) \]

where \(v\) is the value at some spatial location, \(\text{min}\) is the minimum value in the heatmap and \(\text{max}\) is the maximum value. As the heatmap uses internally a 0.0 to 1.0 representation, this simply becomes \(v' = 1.0 - v\).

Note that the attributes \text{min\_value} and \text{max\_value} are not switched. They both keep their values.

This function can be useful e.g. when working with depth maps, where algorithms might have an easier time representing the furthest away points with zeros, requiring an inverted depth map.

Returns arr_inv – Inverted heatmap.

Return type imgaug.HeatmapsOnImage

max_pool(block_size)
Resize the heatmap(s) array using max-pooling of a given block/kernel size.

Parameters block_size (int or tuple of int) – Size of each block of values to pool, aka kernel size. See imgaug.pool() for details.

Returns Heatmaps after max-pooling.

Return type imgaug.HeatmapsOnImage

pad(top=0, right=0, bottom=0, left=0, mode='constant', cval=0.0)
Pad the heatmaps on their top/right/bottom/left side.

Parameters

- top (int, optional) – Amount of pixels to add at the top side of the heatmaps. Must be 0 or greater.
- right (int, optional) – Amount of pixels to add at the right side of the heatmaps. Must be 0 or greater.
- bottom (int, optional) – Amount of pixels to add at the bottom side of the heatmaps. Must be 0 or greater.
- left (int, optional) – Amount of pixels to add at the left side of the heatmaps. Must be 0 or greater.
- mode (string, optional) – Padding mode to use. See numpy.pad() for details.
- cval (number, optional) – Value to use for padding if mode is constant. See numpy.pad() for details.
Returns Padded heatmaps of height $H' = H + \text{top} + \text{bottom}$ and width $W' = W + \text{left} + \text{right}$.

Return type imgaug.HeatmapsOnImage

pad_to_aspect_ratio (aspect_ratio, mode='constant', cval=0.0, return_pad_amounts=False)
Pad the heatmaps on their sides so that they match a target aspect ratio.
Depending on which dimension is smaller (height or width), only the corresponding sides (left/right or top/bottom) will be padded. In each case, both of the sides will be padded equally.

Parameters

- **aspect_ratio (float)** – Target aspect ratio, given as width/height. E.g. 2.0 denotes the image having twice as much width as height.
- **mode (str, optional)** – Padding mode to use. See numpy.pad() for details.
- **cval (number, optional)** – Value to use for padding if mode is constant. See numpy.pad() for details.
- **return_pad_amounts (bool, optional)** – If False, then only the padded image will be returned. If True, a tuple with two entries will be returned, where the first entry is the padded image and the second entry are the amounts by which each image side was padded. These amounts are again a tuple of the form (top, right, bottom, left), with each value being an integer.

Returns

- **heatmaps (imgaug.HeatmapsOnImage)** – Padded heatmaps as HeatmapsOnImage object.
- **pad_amounts (tuple of int)** – Amounts by which the heatmaps were padded on each side, given as a tuple (top, right, bottom, left). This tuple is only returned if return_pad_amounts was set to True.

resize (sizes, interpolation='cubic')
Resize the heatmap(s) array to the provided size given the provided interpolation.

Parameters

- **sizes (float or iterable of int or iterable of float)** – New size of the array in (height, width). See imgaug.imgaug.imresize_single_image() for details.
- **interpolation (None or str or int, optional)** – The interpolation to use during resize. See imgaug.imgaug.imresize_single_image() for details.

Returns Resized heatmaps object.

Return type imgaug.HeatmapsOnImage

scale(*args,**kwargs)
Deprecated. Use HeatmapsOnImage.resize() instead. resize() has the exactly same interface.

to_uint8()
Convert this heatmaps object to a 0-to-255 array.

Returns arr_uint8 – Heatmap as a 0-to-255 array (dtype is uint8).

Return type (H,W,C) ndarray
A single keypoint (aka landmark) on an image.

**Parameters**

- `x (number)` – Coordinate of the keypoint on the x axis.
- `y (number)` – Coordinate of the keypoint on the y axis.

**Attributes**

- `x_int` Return the keypoint’s x-coordinate, rounded to the closest integer.
- `y_int` Return the keypoint’s y-coordinate, rounded to the closest integer.

**Methods**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy([x, y])</code></td>
<td>Create a shallow copy of the Keypoint object.</td>
</tr>
<tr>
<td><code>deepcopy([x, y])</code></td>
<td>Create a deep copy of the Keypoint object.</td>
</tr>
<tr>
<td><code>draw_on_image(image[, color, alpha, size, ...])</code></td>
<td>Draw the keypoint onto a given image.</td>
</tr>
<tr>
<td><code>generate_similar_points_manhattan(nb_steps, ...)</code></td>
<td>Generate nearby points to this keypoint based on manhattan distance.</td>
</tr>
<tr>
<td><code>project(from_shape, to_shape)</code></td>
<td>Project the keypoint onto a new position on a new image.</td>
</tr>
<tr>
<td><code>shift([x, y])</code></td>
<td>Move the keypoint around on an image.</td>
</tr>
</tbody>
</table>

**copy** *(x=None, y=None)*

Create a shallow copy of the Keypoint object.

**Parameters**

- `x (None or number, optional)` – Coordinate of the keypoint on the x axis. If `None`, the instance’s value will be copied.
- `y (None or number, optional)` – Coordinate of the keypoint on the y axis. If `None`, the instance’s value will be copied.

**Returns** Shallow copy.

**Return type** `imgaug.Keypoint`

**deepcopy** *(x=None, y=None)*

Create a deep copy of the Keypoint object.

**Parameters**

- `x (None or number, optional)` – Coordinate of the keypoint on the x axis. If `None`, the instance’s value will be copied.
- `y (None or number, optional)` – Coordinate of the keypoint on the y axis. If `None`, the instance’s value will be copied.

**Returns** Deep copy.

**Return type** `imgaug.Keypoint`

**draw_on_image** *(image, color=(0, 255, 0), alpha=1.0, size=3, copy=True, raise_if_out_of_image=False)*

Draw the keypoint onto a given image.

The keypoint is drawn as a square.

**Parameters**
• **image** *(ndarray)* – The image onto which to draw the keypoint.

• **color** *(int or list or tuple of int or (3,) ndarray, optional)* – The RGB color of the keypoint. If a single int \(C\), then that is equivalent to \((C, C, C)\).

• **alpha** *(float, optional)* – The opacity of the drawn keypoint, where \(1.0\) denotes a fully visible keypoint and \(0.0\) an invisible one.

• **size** *(int, optional)* – The size of the keypoint. If set to \(S\), each square will have size \(S \times S\).

• **copy** *(bool, optional)* – Whether to copy the image before drawing the keypoint.

• **raise_if_out_of_image** *(bool, optional)* – Whether to raise an exception if the keypoint is outside of the image.

**Returns** image – Image with drawn keypoint.

**Return type** *(H,W,3) ndarray*

**generate_similar_points_manhattan** *(nb_steps, step_size, return_array=False)*
Generate nearby points to this keypoint based on manhattan distance.

To generate the first neighbouring points, a distance of \(S\) (step size) is moved from the center point (this keypoint) to the top, right, bottom and left, resulting in four new points. From these new points, the pattern is repeated. Overlapping points are ignored.

The resulting points have a shape similar to a square rotated by 45 degrees.

**Parameters**

• **nb_steps** *(int)* – The number of steps to move from the center point. \(nb\_steps=1\) results in a total of 5 output points (1 center point + 4 neighbours).

• **step_size** *(number)* – The step size to move from every point to its neighbours.

• **return_array** *(bool, optional)* – Whether to return the generated points as a list of keypoints or an array of shape \((N, 2)\), where \(N\) is the number of generated points and the second axis contains the \(x\)- (first value) and \(y\)- (second value) coordinates.

**Returns** points – If \(return\_array\) was \(False\), then a list of Keypoint. Otherwise a numpy array of shape \((N, 2)\), where \(N\) is the number of generated points and the second axis contains the \(x\)- (first value) and \(y\)- (second value) coordinates. The center keypoint (the one on which this function was called) is always included.

**Return type** list of imgaug.Keypoint or \((N,2)\) ndarray

**project** *(from_shape, to_shape)*
Project the keypoint onto a new position on a new image.

E.g. if the keypoint is on its original image at \(x=(10 of 100 pixels)\) and \(y=(20 of 100 pixels)\) and is projected onto a new image with size \((width=200, height=200)\), its new position will be \((20, 40)\).

This is intended for cases where the original image is resized. It cannot be used for more complex changes (e.g. padding, cropping).

**Parameters**

• **from_shape** *(tuple of int)* – Shape of the original image. (Before resize.)

• **to_shape** *(tuple of int)* – Shape of the new image. (After resize.)

**Returns** Keypoint object with new coordinates.

**Return type** imgaug.Keypoint
**shift** \((x=0, y=0)\)
Move the keypoint around on an image.

**Parameters**
- `x` *(number, optional)* – Move by this value on the x axis.
- `y` *(number, optional)* – Move by this value on the y axis.

**Returns** Keypoint object with new coordinates.

**Return type** `imgaug.Keypoint`

**x_int**
Return the keypoint’s x-coordinate, rounded to the closest integer.

**Returns** `result` – Keypoint’s x-coordinate, rounded to the closest integer.

**Return type** `int`

**y_int**
Return the keypoint’s y-coordinate, rounded to the closest integer.

**Returns** `result` – Keypoint’s y-coordinate, rounded to the closest integer.

**Return type** `int`

### class `imgaug.augmentables.kps.KeypointsOnImage(keypoints, shape)`

**Bases:** `object`

Object that represents all keypoints on a single image.

**Parameters**
- `keypoints` *(list of imgaug.Keypoint)* – List of keypoints on the image.
- `shape` *(tuple of int)* – The shape of the image on which the keypoints are placed.

**Examples**

```python
>>> image = np.zeros((70, 70))
>>> kps = [Keypoint(x=10, y=20), Keypoint(x=34, y=60)]
>>> kps_oi = KeypointsOnImage(kps, shape=image.shape)
```

**Attributes**
- `empty` Returns whether this object contains zero keypoints.
- `height`
- `width`

**Methods**

- **copy** *(keypoints, shape)* Create a shallow copy of the KeypointsOnImage object.
- **deepcopy** *(keypoints, shape)* Create a deep copy of the KeypointsOnImage object.
- **draw_on_image** *(image, color, alpha, size, ...)* Draw all keypoints onto a given image.
- **from_coords_array** *(coords, shape)* Deprecated.
Table 41 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_distance_maps(distance_maps[,...])</code></td>
<td>Converts maps generated by <code>to_distance_maps()</code> back to a KeypointsOnImage object.</td>
</tr>
<tr>
<td><code>from_keypoint_image(image[,...])</code></td>
<td>Converts an image generated by <code>to_keypoint_image()</code> back to a KeypointsOnImage object.</td>
</tr>
<tr>
<td><code>from_xy_array(xy, shape)</code></td>
<td>Convert an array (N,2) with a given image shape to a KeypointsOnImage object.</td>
</tr>
<tr>
<td><code>get_coords_array()</code></td>
<td>Deprecated.</td>
</tr>
<tr>
<td><code>on(image)</code></td>
<td>Project keypoints from one image to a new one.</td>
</tr>
<tr>
<td><code>shift([x, y])</code></td>
<td>Move the keypoints around on an image.</td>
</tr>
<tr>
<td><code>to_distance_maps([inverted])</code></td>
<td>Generates a ((H, W, K)) output containing (K) distance maps for (K) keypoints.</td>
</tr>
<tr>
<td><code>to_keypoint_image([size])</code></td>
<td>Draws a new black image of shape ((H, W, N)) in which all keypoint coordinates are set to 255.</td>
</tr>
<tr>
<td><code>to_xy_array()</code></td>
<td>Convert keypoint coordinates to ((N, 2)) array.</td>
</tr>
</tbody>
</table>

**copy (keypoints=None, shape=None)**

Create a shallow copy of the KeypointsOnImage object.

**Parameters**

- **keypoints** *(None or list of imgaug.Keypoint, optional)* – List of keypoints on the image. If `None`, the instance’s keypoints will be copied.

- **shape** *(tuple of int, optional)* – The shape of the image on which the keypoints are placed. If `None`, the instance’s shape will be copied.

**Returns** Shallow copy.

**Return type** `imgaug.KeypointsOnImage`

**deepcopy (keypoints=None, shape=None)**

Create a deep copy of the KeypointsOnImage object.

**Parameters**

- **keypoints** *(None or list of imgaug.Keypoint, optional)* – List of keypoints on the image. If `None`, the instance’s keypoints will be copied.

- **shape** *(tuple of int, optional)* – The shape of the image on which the keypoints are placed. If `None`, the instance’s shape will be copied.

**Returns** Deep copy.

**Return type** `imgaug.KeypointsOnImage`

**draw_on_image (image, color=(0, 255, 0), alpha=1.0, size=3, copy=True, raise_if_out_of_image=False)**

Draw all keypoints onto a given image.

Each keypoint is marked by a square of a chosen color and size.

**Parameters**

- **image** *(\((H, W, 3)\) ndarray)* – The image onto which to draw the keypoints. This image should usually have the same shape as set in KeypointsOnImage.shape.

- **color** *(int or list of int or tuple of int or \((3,)\) ndarray, optional)* – The RGB color of all keypoints. If a single int \(C\), then that is equivalent to \((C, C, C)\).
• **alpha** (*float, optional*) – The opacity of the drawn keypoint, where 1.0 denotes a fully visible keypoint and 0.0 an invisible one.

• **size** (*int, optional*) – The size of each point. If set to `C`, each square will have size `C x C`.

• **copy** (*bool, optional*) – Whether to copy the image before drawing the points.

• **raise_if_out_of_image** (*bool, optional*) – Whether to raise an exception if any keypoint is outside of the image.

**Returns** `image` – Image with drawn keypoints.

**Return type** *(H,W,3) ndarray*

**empty**

Returns whether this object contains zero keypoints.

**Returns** `result` – True if this object contains zero keypoints.

**Return type** `bool`

**static from_coords_array** (*coords, shape*)

```
Deprecated. Use KeypointsOnImage.from_xy_array() instead.
```

Convert an array (N,2) with a given image shape to a KeypointsOnImage object.

**Parameters**

• **coords** *((N, 2) ndarray)* – Coordinates of N keypoints on the original image. Each first entry `coords[i, 0]` is expected to be the x coordinate. Each second entry `coords[i, 1]` is expected to be the y coordinate.

• **shape** (*tuple*) – Shape tuple of the image on which the keypoints are placed.

**Returns** KeypointsOnImage object that contains all keypoints from the array.

**Return type** `KeypointsOnImage`

**static from_distance_maps** (*distance_maps, inverted=False, if_not_found_coords={'x': -1, 'y': -1}, threshold=None, nb_channels=None*)

Converts maps generated by `to_distance_maps()` back to a KeypointsOnImage object.

**Parameters**

• **distance_maps** *((H,W,N) ndarray)* – The distance maps. N is the number of keypoints.

• **inverted** (*bool, optional*) – Whether the given distance maps were generated in inverted or normal mode.

• **if_not_found_coords** *(tuple or list or dict or None, optional)* – Coordinates to use for keypoints that cannot be found in `distance_maps`. If this is a list/tuple, it must have two integer values. If it is a dictionary, it must have the keys `x` and `y`, with each containing one integer value. If this is None, then the keypoint will not be added to the final KeypointsOnImage object.

• **threshold** (*float, optional*) – The search for keypoints works by searching for the argmin (non-inverted) or argmax (inverted) in each channel. This parameters contains the maximum (non-inverted) or minimum (inverted) value to accept in order to view a hit as a keypoint. Use None to use no min/max.

• **nb_channels** *(None or int, optional)* – Number of channels of the image on which the keypoints are placed. Some keypoint augmenters require that information. If set to None, the keypoint’s shape will be set to `(height, width)`, otherwise `(height, width, nb_channels)`.

**Returns** The extracted keypoints.
static from_keypoint_image(image, if_not_found_coords={'x': -1, 'y': -1}, threshold=1, nb_channels=None)

Converts an image generated by to_keypoint_image() back to a KeypointsOnImage object.

Parameters

- **image** ((H,W,N) ndarray) – The keypoints image. N is the number of keypoints.
- **if_not_found_coords** (tuple or list or dict or None, optional) – Coordinates to use for keypoints that cannot be found in image. If this is a list/tuple, it must have two integer values. If it is a dictionary, it must have the keys x and y with each containing one integer value. If this is None, then the keypoint will not be added to the final KeypointsOnImage object.
- **threshold** (int, optional) – The search for keypoints works by searching for the argmax in each channel. This parameters contains the minimum value that the max must have in order to be viewed as a keypoint.
- **nb_channels** (None or int, optional) – Number of channels of the image on which the keypoints are placed. Some keypoint augmenters require that information. If set to None, the keypoint’s shape will be set to (height, width), otherwise (height, width, nb_channels).

Returns **out** – The extracted keypoints.

Return type KeypointsOnImage

classmethod from_xy_array(xy, shape)

Convert an array (N,2) with a given image shape to a KeypointsOnImage object.

Parameters

- **xy** ((N, 2) ndarray) – Coordinates of N keypoints on the original image, given as (N, 2) array of xy-coordinates.
- **shape** (tuple of int or ndarray) – Shape tuple of the image on which the keypoints are placed.

Returns KeypointsOnImage object that contains all keypoints from the array.

Return type KeypointsOnImage

get_coords_array()

Deprecated. Use KeypointsOnImage.to_xy_array() instead.

Convert the coordinates of all keypoints in this object to an array of shape (N,2).

Returns **result** – Where N is the number of keypoints. Each first value is the x coordinate, each second value is the y coordinate.

Return type (N, 2) ndarray

height

on (image)

Project keypoints from one image to a new one.

Parameters **image** (ndarray or tuple of int) – New image onto which the keypoints are to be projected. May also simply be that new image’s shape tuple.

Returns **keypoints** – Object containing all projected keypoints.

Return type imgaug.KeypointsOnImage
**shift** \((x=0, y=0)\)
Move the keypoints around on an image.

**Parameters**
- \(x\) (number, optional) – Move each keypoint by this value on the x axis.
- \(y\) (number, optional) – Move each keypoint by this value on the y axis.

**Returns** out – Keypoints after moving them.

**Return type** KeypointsOnImage

**to_distance_maps** \((\text{inverted} = \text{False})\)
Generates a \((H, W, K)\) output containing \(K\) distance maps for \(K\) keypoints.

The \(k\)-th distance map contains at every location \((y, x)\) the euclidean distance to the \(k\)-th keypoint.

This function can be used as a helper when augmenting keypoints with a method that only supports the augmentation of images.

**Parameters**
- **inverted** (bool, optional) – If True, inverted distance maps are returned where each distance value \(d\) is replaced by \(d/(d+1)\), i.e. the distance maps have values in the range \((0.0, 1.0]\) with 1.0 denoting exactly the position of the respective keypoint.

**Returns**
- **distance_maps** – A float32 array containing \(K\) distance maps for \(K\) keypoints. Each location \((y, x, k)\) in the array denotes the euclidean distance at \((y, x)\) to the \(k\)-th keypoint. In inverted mode the distance \(d\) is replaced by \(d/(d+1)\). The height and width of the array match the height and width in KeypointsOnImage.shape.

**Return type** \((H,W,K)\) ndarray

**to_keypoint_image** \((\text{size} = 1)\)
Draws a new black image of shape \((H, W, N)\) in which all keypoint coordinates are set to 255. \((H=\text{shape height}, W=\text{shape width}, N=\text{number of keypoints})\)

This function can be used as a helper when augmenting keypoints with a method that only supports the augmentation of images.

**Parameters**
- **size** (int) – Size of each (squared) point.

**Returns**
- **image** – Image in which the keypoints are marked. \(H\) is the height, defined in KeypointsOnImage.shape[0] (analogous \(W\)). \(N\) is the number of keypoints.

**Return type** \((H,W,N)\) ndarray

**to_xy_array**
Convert keypoint coordinates to \((N, 2)\) array.

**Returns**
- Array containing the coordinates of all keypoints. Shape is \((N, 2)\) with coordinates in xy-form.

**Return type** \((N, 2)\) ndarray

**width**
imgaug.augmentables.kps.compute_geometric_median\((X, \text{eps}=1e-05)\)
Estimate the geometric median of points in 2D.

Code from https://stackoverflow.com/a/30305181

**Parameters**
- \(X\) \(\text{(N,2) ndarray}\) – Points in 2D. Second axis must be given in xy-form.
- \(\text{eps}\) (float, optional) – Distance threshold when to return the median.
Returns Geometric median as xy-coordinate.

Return type (2,) ndarray

## 13.9 `imgaug.augmentables.lines`  

```python
class imgaug.augmentables.lines.LineString(coords, label=None)
```

Class representing line strings.

A line string is a collection of connected line segments, each having a start and end point. Each point is given as its \((x, y)\) absolute (sub-)pixel coordinates. The end point of each segment is also the start point of the next segment.

The line string is not closed, i.e. start and end point are expected to differ and will not be connected in drawings.

**Parameters**

- `coords` *(iterable of tuple of number or ndarray)* – The points of the line string.
- `label` *(None or str, optional)* – The label of the line string.

**Attributes**

- `height` Get the height of a bounding box encapsulating the line.
- `length` Get the total euclidean length of the line string.
- `width` Get the width of a bounding box encapsulating the line.
- `xx` Get an array of x-coordinates of all points of the line string.
- `xx_int` Get an array of discrete x-coordinates of all points.
- `yy` Get an array of y-coordinates of all points of the line string.
- `yy_int` Get an array of discrete y-coordinates of all points.

**Methods**

- `almost_equals(other[, max_distance, ...])` Compare this and another LineString.
- `clip_out_of_image(image)` Clip off all parts of the line_string that are outside of the image.
- `compute_distance(other[, default])` Compute the minimal distance between the line string and `other`.
- `compute_neighbour_distances()` Get the euclidean distance between each two consecutive points.
- `compute_pointwise_distances(other[, default])` Compute the minimal distance between each point on self and other.
- `concatenate(other)` Concatenate this line string with another one.
- `contains(other[, max_distance])` Estimate whether the bounding box contains a point.
- `coords_almost_equals(other[, max_distance, ...])` Compare this and another LineString’s coordinates.
- `copy([coords, label])` Create a shallow copy of the LineString object.
- `deepcopy([coords, label])` Create a deep copy of the BoundingBox object.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>draw_heatmap_array(image_shape[...])</code></td>
<td>Draw the line segments and points of the line string as a heatmap array.</td>
</tr>
<tr>
<td><code>draw_lines_heatmap_array(image_shape[, ...])</code></td>
<td>Draw the line segments of the line string as a heatmap array.</td>
</tr>
<tr>
<td><code>draw_lines_on_image(image[, color, alpha, ...])</code></td>
<td>Draw the line segments of the line string on a given image.</td>
</tr>
<tr>
<td><code>draw_mask(image_shape[, size_lines, ...])</code></td>
<td>Draw this line segment as a binary image mask.</td>
</tr>
<tr>
<td><code>draw_on_image(image[, color, color_lines, ...])</code></td>
<td>Draw the line string on an image.</td>
</tr>
<tr>
<td><code>draw_points_heatmap_array(image_shape[, ...])</code></td>
<td>Draw the points of the line string as a heatmap array.</td>
</tr>
<tr>
<td><code>draw_points_on_image(image[, color, alpha, ...])</code></td>
<td>Draw the points of the line string on a given image.</td>
</tr>
<tr>
<td><code>extract_from_image(image[, size, pad, ...])</code></td>
<td>Extract the image pixels covered by the line string.</td>
</tr>
<tr>
<td><code>find_intersections_with(other)</code></td>
<td>Find all intersection points between the line string and other.</td>
</tr>
<tr>
<td><code>get_pointwise_inside_image_mask(image)</code></td>
<td>Get for each point whether it is inside of the given image plane.</td>
</tr>
<tr>
<td><code>is_fully_within_image(image[, default])</code></td>
<td>Estimate whether the line string is fully inside the image area.</td>
</tr>
<tr>
<td><code>is_out_of_image(image[, fully, partly, default])</code></td>
<td>Estimate whether the line is partially/fully outside of the image area.</td>
</tr>
<tr>
<td><code>is_partly_within_image(image[, default])</code></td>
<td>Estimate whether the line string is at least partially inside the image.</td>
</tr>
<tr>
<td><code>project(from_shape, to_shape)</code></td>
<td>Project the line string onto a differently shaped image.</td>
</tr>
<tr>
<td><code>shift([top, right, bottom, left])</code></td>
<td>Shift/move the line string from one or more image sides.</td>
</tr>
<tr>
<td><code>subdivide(points_per_edge)</code></td>
<td>Adds N interpolated points with uniform spacing to each edge.</td>
</tr>
<tr>
<td><code>to_bounding_box()</code></td>
<td>Generate a bounding box encapsulating the line string.</td>
</tr>
<tr>
<td><code>to_heatmap(image_shape[, size_lines, ...])</code></td>
<td>Generate a heatmap object from the line string.</td>
</tr>
<tr>
<td><code>to_keypoints()</code></td>
<td>Convert the line string points to keypoints.</td>
</tr>
<tr>
<td><code>to_polygon()</code></td>
<td>Generate a polygon from the line string points.</td>
</tr>
<tr>
<td><code>to_segmentation_map(image_shape[, ...])</code></td>
<td>Generate a segmentation map object from the line string.</td>
</tr>
</tbody>
</table>

**almost_equal** *(other, max_distance=0.0001, points_per_edge=8)*

Compare this and another LineString.

**Parameters**

- **other** *(imgaug.augmentables.lines.LineString)* – The other line string. Must be a LineString instance, not just its coordinates.

- **max_distance** *(float, optional)* – See `imgaug.augmentables.lines.LineString.coords_almost_equals()`.

- **points_per_edge** *(int, optional)* – See `imgaug.augmentables.lines.LineString.coords_almost_equals()`.

**Returns** True if the coordinates are almost equal according to `imgaug.augmentables.lines.LineString.coords_almost_equals()` and additionally the labels are identical. Otherwise False.
clip_out_of_image(image)
Clip off all parts of the line_string that are outside of the image.

Parameters

- **image** (ndarray or tuple of int) – Either an image with shape \((H, W, [C])\) or a tuple denoting such an image shape.

Returns
Line strings, clipped to the image shape. The result may contain any number of line strings, including zero.

Return type
list of imgaug.augmentables.lines.LineString

compute_distance(\(\text{other}, \text{default}={None}\))
Compute the minimal distance between the line string and \(\text{other}\).

Parameters

- **other** (tuple of number or imgaug.augmentables.kps.Keypoint or imgaug.augmentables.LineString) – Other object to which to compute the distance.
- **default** – Value to return if this line string or \(\text{other}\) contain no points.

Returns
Distance to \(\text{other}\) or \(\text{default}\) if not distance could be computed.

Return type
float

compute_neighbour_distances()
Get the euclidean distance between each two consecutive points.

Returns
Euclidean distances between point pairs. Same order as in \(\text{coords}\). For \(N\) points, \(N-1\) distances are returned.

Return type
ndarray

compute_pointwise_distances(\(\text{other}, \text{default}={None}\))
Compute the minimal distance between each point on self and \(\text{other}\).

Parameters

- **other** (tuple of number or imgaug.augmentables.kps.Keypoint or imgaug.augmentables.LineString) – Other object to which to compute the distances.
- **default** – Value to return if \(\text{other}\) contains no points.

Returns
Distances to \(\text{other}\) or \(\text{default}\) if not distance could be computed.

Return type
list of float

concatenate(\(\text{other}\))
Concatenate this line string with another one.

This will add a line segment between the end point of this line string and the start point of \(\text{other}\).

Parameters

- **other** (imgaug.augmentables.lines.LineString or ndarray or iterable of tuple of number) – The points to add to this line string.

Returns
New line string with concatenated points. The label of this line string will be kept.

Return type
imgaug.augmentables.lines.LineString

contains(\(\text{other}, \text{max\_distance}=0.0001\))
Estimate whether the bounding box contains a point.

Parameters

- **other** (tuple of number or imgaug.augmentables.kps.Keypoint) – Point to check for.
• **max_distance** (*float*) – Maximum allowed euclidean distance between the point and the closest point on the line. If the threshold is exceeded, the point is not considered to be contained in the line.

**Returns** True if the point is contained in the line string, False otherwise. It is contained if its distance to the line or any of its points is below a threshold.

**Return type** `bool`

`coords_almost_equal` (*other, max_distance=1e-06, points_per_edge=8*)

Compare this and another LineString’s coordinates.

This is an approximate method based on pointwise distances and can in rare corner cases produce wrong outputs.

**Parameters**

- **other** (`imgaug.augmentables.lines.LineString or tuple of number or ndarray or list of ndarray or list of tuple of number`) – The other line string or its coordinates.

- **max_distance** (*float*) – Max distance of any point from the other line string before the two line strings are evaluated to be unequal.

- **points_per_edge** (*int, optional*) – How many points to interpolate on each edge.

**Returns** Whether the two LineString’s coordinates are almost identical, i.e. the max distance is below the threshold. If both have no coordinates, True is returned. If only one has no coordinates, False is returned. Beyond that, the number of points is not evaluated.

**Return type** `bool`

`copy` (*coords=None, label=None*)

Create a shallow copy of the LineString object.

**Parameters**

- **coords** (`None or iterable of tuple of number or ndarray`) – If not None, then the coords of the copied object will be set to this value.

- **label** (`None or str`) – If not None, then the label of the copied object will be set to this value.

**Returns** Shallow copy.

**Return type** `imgaug.augmentables.lines.LineString`

`deepcopy` (*coords=None, label=None*)

Create a deep copy of the BoundingBox object.

**Parameters**

- **coords** (`None or iterable of tuple of number or ndarray`) – If not None, then the coords of the copied object will be set to this value.

- **label** (`None or str`) – If not None, then the label of the copied object will be set to this value.

**Returns** Deep copy.

**Return type** `imgaug.augmentables.lines.LineString`

`draw_heatmap_array` (*image_shape, alpha_lines=1.0, alpha_points=1.0, size_lines=1, size_points=0, antialiased=True, raise_if_out_of_image=False*)

Draw the line segments and points of the line string as a heatmap array.

**Parameters**
• **image_shape** ([tuple of int]) – The shape of the image onto which to draw the line mask.

• **alpha_lines** ([float, optional]) – Opacity of the line string. Higher values denote a more visible line string.

• **alpha_points** ([float, optional]) – Opacity of the line string points. Higher values denote a more visible points.

• **size_lines** ([int, optional]) – Thickness of the line segments.

• **size_points** ([int, optional]) – Size of the points in pixels.

• **anti_allied** ([bool, optional]) – Whether to draw the line with anti-aliasing activated.

• **raise_if_out_of_image** ([bool, optional]) – Whether to raise an error if the line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

**Returns** Float array of shape `image_shape` (no channel axis) with drawn line segments and points. All values are in the interval `[0.0, 1.0]`.

**Return type** ndarray

draw_lines_heatmap_array function:

```python
image_shape, alpha=1.0, size=1, anti_allied=True, raise_if_out_of_image=False)
```

Draw the line segments of the line string as a heatmap array.

**Parameters**

• **image_shape** ([tuple of int]) – The shape of the image onto which to draw the line mask.

• **alpha** ([float, optional]) – Opacity of the line string. Higher values denote a more visible line string.

• **size** ([int, optional]) – Thickness of the line segments.

• **anti_allied** ([bool, optional]) – Whether to draw the line with anti-aliasing activated.

• **raise_if_out_of_image** ([bool, optional]) – Whether to raise an error if the line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

**Returns** Float array of shape `image_shape` (no channel axis) with drawn line string. All values are in the interval `[0.0, 1.0]`.

**Return type** ndarray

draw_lines_on_image function:

```python
image, color=(0, 255, 0), alpha=1.0, size=3, anti_allied=True, raise_if_out_of_image=False)
```

Draw the line segments of the line string on a given image.

**Parameters**

• **image** ([ndarray or tuple of int]) – The image onto which to draw. Expected to be `uint8` and of shape `(H, W, C)` with `C` usually being 3 (other values are not tested). If a tuple, expected to be `(H, W, C)` and will lead to a new `uint8` array of zeros being created.

• **color** ([int or iterable of int]) – Color to use as RGB, i.e. three values.

• **alpha** ([float, optional]) – Opacity of the line string. Higher values denote a more visible line string.

• **size** ([int, optional]) – Thickness of the line segments.

• **anti_allied** ([bool, optional]) – Whether to draw the line with anti-aliasing activated.
• **raise_if_out_of_image** *(bool, optional)* – Whether to raise an error if the line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

Returns *image* with line drawn on it.

Return type *ndarray*

**draw_mask** *(image_shape, size_lines=1, size_points=0, raise_if_out_of_image=False)*

Draw this line segment as a binary image mask.

Parameters

- **image_shape** *(tuple of int)* – The shape of the image onto which to draw the line mask.
- **size_lines** *(int, optional)* – Thickness of the line segments.
- **size_points** *(int, optional)* – Size of the points in pixels.
- **raise_if_out_of_image** *(bool, optional)* – Whether to raise an error if the line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

Returns Boolean line mask of shape *image_shape* (no channel axis).

Return type *ndarray*

**draw_on_image** *(image, color=(0, 255, 0), color_lines=None, color_points=None, alpha=1.0, alpha_lines=None, alpha_points=None, size=1, size_lines=None, size_points=None, antialiased=True, raise_if_out_of_image=False)*

Draw the line string on an image.

Parameters

- **image** *(ndarray)* – The *(H,W,C)* uint8 image onto which to draw the line string.
- **color** *(iterable of int, optional)* – Color to use as RGB, i.e. three values. The color of the line and points are derived from this value, unless they are set.
- **color_lines** *(None or iterable of int)* – Color to use for the line segments as RGB, i.e. three values. If None, this value is derived from color.
- **color_points** *(None or iterable of int)* – Color to use for the points as RGB, i.e. three values. If None, this value is derived from 0.5 * color.
- **alpha** *(float, optional)* – Opacity of the line string. Higher values denote more visible points. The alphas of the line and points are derived from this value, unless they are set.
- **alpha_lines** *(None or float, optional)* – Opacity of the line string. Higher values denote more visible line string. If None, this value is derived from alpha.
- **alpha_points** *(None or float, optional)* – Opacity of the line string points. Higher values denote more visible points. If None, this value is derived from alpha.
- **size** *(int, optional)* – Size of the line string. The sizes of the line and points are derived from this value, unless they are set.
- **size_lines** *(None or int, optional)* – Thickness of the line segments. If None, this value is derived from size.
- **size_points** *(None or int, optional)* – Size of the points in pixels. If None, this value is derived from 3 * size.
- **antialiased** *(bool, optional)* – Whether to draw the line with anti-aliasing activated. This does currently not affect the point drawing.
• **raise_if_out_of_image** *(bool, optional)* – Whether to raise an error if the line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

**Returns** Image with line string drawn on it.

**Return type** `ndarray`

**draw_points_heatmap_array** *(image_shape, alpha=1.0, size=1, raise_if_out_of_image=False)*

Draw the points of the line string as a heatmap array.

**Parameters**

- **image_shape** *(tuple of int)* – The shape of the image onto which to draw the point mask.
- **alpha** *(float, optional)* – Opacity of the line string points. Higher values denote a more visible points.
- **size** *(int, optional)* – Size of the points in pixels.
- **raise_if_out_of_image** *(bool, optional)* – Whether to raise an error if the line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

**Returns** Float array of shape `image_shape` (no channel axis) with drawn line string points. All values are in the interval `[0.0, 1.0]`.

**Return type** `ndarray`

**draw_points_on_image** *(image, color=(0, 128, 0), alpha=1.0, size=3, copy=True, raise_if_out_of_image=False)*

Draw the points of the line string on a given image.

**Parameters**

- **image** *(ndarray or tuple of int)* – The image onto which to draw. Expected to be `uint8` and of shape `(H, W, C)` with `C` usually being 3 (other values are not tested). If a tuple, expected to be `(H, W, C)` and will lead to a new `uint8` array of zeros being created.
- **color** *(iterable of int)* – Color to use as RGB, i.e. three values.
- **alpha** *(float, optional)* – Opacity of the line string points. Higher values denote a more visible points.
- **size** *(int, optional)* – Size of the points in pixels.
- **copy** *(bool, optional)* – Whether it is allowed to draw directly in the input array (False) or it has to be copied (True). The routine may still have to copy, even if `copy=False` was used. Always use the return value.
- **raise_if_out_of_image** *(bool, optional)* – Whether to raise an error if the line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

**Returns** Float array of shape `image_shape` (no channel axis) with drawn line string points. All values are in the interval `[0.0, 1.0]`.

**Return type** `ndarray`

**extract_from_image** *(image, size=1, pad=True, pad_max=None, antialiased=True, prevent_zero_size=True)*

Extract the image pixels covered by the line string.

It will only extract pixels overlapped by the line string.
This function will by default zero-pad the image if the line string is partially/fully outside of the image. This is for consistency with the same implementations for bounding boxes and polygons.

**Parameters**

- **image** (*ndarray*) – The image of shape \((H, W, [C])\) from which to extract the pixels within the line string.

- **size** (*int, optional*) – Thickness of the line.

- **pad** (*bool, optional*) – Whether to zero-pad the image if the object is partially/fully outside of it.

- **pad_max** (*None or int, optional*) – The maximum number of pixels that may be zero-padded on any side, i.e. if this has value \(N\) the total maximum of added pixels is \(4 \times N\). This option exists to prevent extremely large images as a result of single points being moved very far away during augmentation.

- **antialiased** (*bool, optional*) – Whether to apply anti-aliasing to the line string.

- **prevent_zero_size** (*bool, optional*) – Whether to prevent height or width of the extracted image from becoming zero. If this is set to True and height or width of the line string is below 1, the height/width will be increased to 1. This can be useful to prevent problems, e.g. with image saving or plotting. If it is set to False, images will be returned as \((H', W')\) or \((H', W', 3)\) with \(H\) or \(W\) potentially being 0.

**Returns**

- **image** – Pixels overlapping with the line string. Zero-padded if the line string is partially/fully outside of the image and \(\text{pad} = \text{True}\). If \(\text{prevent_zero_size}\) is activated, it is guaranteed that \(H' > 0\) and \(W' > 0\), otherwise only \(H' >= 0\) and \(W' >= 0\).

**Return type** \((H', W')\) ndarray or \((H', W', C)\) ndarray

### find_intersections_with (other)

Find all intersection points between the line string and other.

**Parameters**

- **other** (*tuple of number or list of tuple of number or list of LineString or LineString*) – The other geometry to use during intersection tests.

**Returns**

- All intersection points. One list per pair of consecutive start and end point, i.e. \(N - 1\) lists of \(N\) points. Each list may be empty or may contain multiple points.

**Return type** list of list of tuple of number

### get_pointwise_inside_image_mask (image)

Get for each point whether it is inside of the given image plane.

**Parameters**

- **image** (*ndarray or tuple of int*) – Either an image with shape \((H, W, [C])\) or a tuple denoting such an image shape.

**Returns**

- Boolean array with one value per point indicating whether it is inside of the provided image plane \((\text{True})\) or not \((\text{False})\).

**Return type** ndarray

### height

Get the height of a bounding box encapsulating the line.

### is_fully_within_image (image, default=False)

Estimate whether the line string is fully inside the image area.

**Parameters**

- **image** (*ndarray or tuple of int*) – Either an image with shape \((H, W, [C])\) or a tuple denoting such an image shape.
• **default** – Default value to return if the line string contains no points.

**Returns**  True if the line string is fully inside the image area. False otherwise.

**Return type**  bool

**is_out_of_image**  *(image, fully=True, partly=False, default=True)*

Estimate whether the line is partially/fully outside of the image area.

**Parameters**

• **image**  *(ndarray or tuple of int)*  – Either an image with shape \((H, W, [C])\) or a tuple denoting such an image shape.

• **fully**  *(bool, optional)*  – Whether to return True if the bounding box is fully outside of the image area.

• **partly**  *(bool, optional)*  – Whether to return True if the bounding box is at least partially outside of the image area.

• **default** – Default value to return if the line string contains no points.

**Returns**  *default* if the line string has no points. True if the line string is partially/fully outside of the image area, depending on defined parameters. False otherwise.

**Return type**  bool

**is_partly_within_image**  *(image, default=False)*

Estimate whether the line string is at least partially inside the image.

**Parameters**

• **image**  *(ndarray or tuple of int)*  – Either an image with shape \((H, W, [C])\) or a tuple denoting such an image shape.

• **default** – Default value to return if the line string contains no points.

**Returns**  True if the line string is at least partially inside the image area. False otherwise.

**Return type**  bool

**length**

Get the total euclidean length of the line string.

**Returns**  The length based on euclidean distance.

**Return type**  float

**project**  *(from_shape, to_shape)*

Project the line string onto a differently shaped image.

E.g. if a point of the line string is on its original image at \(x=(10 \text{ of } 100 \text{ pixels})\) and \(y=(20 \text{ of } 100 \text{ pixels})\) and is projected onto a new image with size \((width=200, height=200)\), its new position will be \((x=20, y=40)\).

This is intended for cases where the original image is resized. It cannot be used for more complex changes (e.g. padding, cropping).

**Parameters**

• **from_shape**  *(tuple of int or ndarray)*  – Shape of the original image. (Before resize.)

• **to_shape**  *(tuple of int or ndarray)*  – Shape of the new image. (After resize.)

**Returns**  *out* – Line string with new coordinates.

**Return type**  *imgaug.augmentables.lines.LineString*
**shift** *(top=None, right=None, bottom=None, left=None)*

Shift/move the line string from one or more image sides.

**Parameters**

- **top** *(None or int, optional)* – Amount of pixels by which to shift the bounding box from the top.
- **right** *(None or int, optional)* – Amount of pixels by which to shift the bounding box from the right.
- **bottom** *(None or int, optional)* – Amount of pixels by which to shift the bounding box from the bottom.
- **left** *(None or int, optional)* – Amount of pixels by which to shift the bounding box from the left.

**Returns** result – Shifted line string.

**Return type** `imgaug.augmentables.lines.LineString`

**subdivide** *(points_per_edge)*

Add \( N \) interpolated points with uniform spacing to each edge.

For each edge between points \( A \) and \( B \) this adds points at \( A + (i/(1+N)) \ast (B - A) \), where \( i \) is the index of the added point and \( N \) is the number of points to add per edge.

Calling this method two times will split each edge at its center and then again split each newly created edge at their center. It is equivalent to calling `subdivide(3)`.

**Parameters**

- **points_per_edge** *(int)* – Number of points to interpolate on each edge.

**Returns** Line string with subdivided edges.

**Return type** `LineString`

**to_bounding_box** *

Generate a bounding box encapsulating the line string.

**Returns** Bounding box encapsulating the line string. None if the line string contained no points.

**Return type** None or `imgaug.augmentables.bbs.BoundingBox`

**to_heatmap** *(image_shape, size_lines=1, size_points=0, antialiased=True, raise_if_out_of_image=False)*

Generate a heatmap object from the line string.

This is similar to `imgaug.augmentables.lines.LineString.draw_lines_heatmap_array()` executed with `alpha=1.0`. The result is wrapped in a `HeatmapsOnImage` object instead of just an array. No points are drawn.

**Parameters**

- **image_shape** *(tuple of int)* – The shape of the image onto which to draw the line mask.
- **size_lines** *(int, optional)* – Thickness of the line.
- **size_points** *(int, optional)* – Size of the points in pixels.
- **antialiased** *(bool, optional)* – Whether to draw the line with anti-aliasing activated.
- **raise_if_out_of_image** *(bool, optional)* – Whether to raise an error if the line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

**Returns** Heatmap object containing drawn line string.
Return type: imgaug.augmentables.heatmaps.HeatmapOnImage

to_keypoints()

Convert the line string points to keypoints.

Returns: Points of the line string as keypoints.

Return type: list of imgaug.augmentables.kps.Keypoint

to_polygon()

Generate a polygon from the line string points.

Returns: Polygon with the same corner points as the line string. Note that the polygon might be invalid, e.g. contain less than 3 points or have self-intersections.

Return type: imgaug.augmentables.polys.Polygon

to_segmentation_map(image_shape, size_lines=1, size_points=0, raise_if_out_of_image=False)

Generate a segmentation map object from the line string.

This is similar to imgaug.augmentables.lines.LineString.draw_mask(). The result is wrapped in a SegmentationMapOnImage object instead of just an array.

Parameters:

- image_shape (tuple of int) – The shape of the image onto which to draw the line mask.
- size_lines (int, optional) – Thickness of the line.
- size_points (int, optional) – Size of the points in pixels.
- raise_if_out_of_image (bool, optional) – Whether to raise an error if the line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

Returns: Segmentation map object containing drawn line string.

Return type: imgaug.augmentables.segmaps.SegmentationMapOnImage

width

Get the width of a bounding box encapsulating the line.

xx

Get an array of x-coordinates of all points of the line string.

xx_int

Get an array of discrete x-coordinates of all points.

yy

Get an array of y-coordinates of all points of the line string.

yy_int

Get an array of discrete y-coordinates of all points.

class imgaug.augmentables.lines.LineStringsOnImage(line_strings, shape)

Bases: object

Object that represents all line strings on a single image.

Parameters:

- line_strings (list of imgaug.augmentables.lines.LineString) – List of line strings on the image.
- shape (tuple of int or ndarray) – The shape of the image on which the objects are placed. Either an image with shape \((H, W, [C])\) or a tuple denoting such an image shape.
Examples

```python
>>> import imgaug.augmentables.lines as lines
>>> image = np.zeros((100, 100))
>>> iss = [
    lines.LineString([(0, 0), (10, 0)]),
    lines.LineString([(10, 20), (30, 30), (50, 70)])
]
>>> lsoi = lines.LineStringsOnImage(iss, shape=image.shape)
```

Attributes

- **empty** Returns whether this object contains zero line strings.

Methods

- `clip_out_of_image()` Clip off all parts of the line strings that are outside of the image.
- `copy(line_strings=None, shape=None)` Create a shallow copy of the LineStringsOnImage object.
- `deepcopy(line_strings=None, shape=None)` Create a deep copy of the LineStringsOnImage object.
- `draw_on_image(image, color, color_lines, ...)` Draw all line strings onto a given image.
- `from_xy_arrays(xy, shape)` Convert an \((N,M,2)\) ndarray to a LineStringsOnImage object.
- `on(image)` Project bounding boxes from one image to a new one.
- `remove_out_of_image(fully, partly)` Remove all line strings that are fully/partially outside of the image.
- `shift(top, right, bottom, left)` Shift/move the line strings from one or more image sides.
- `to_xy_arrays([dtype])` Convert this object to an iterable of \((M, 2)\) arrays of points.

**clip_out_of_image()**

Clip all parts of the line strings that are outside of the image.

**Returns** Line strings, clipped to fall within the image dimensions.

**Return type** `imgaug.augmentables.lines.LineStringsOnImage`

**copy**(line_strings=None, shape=None)

Create a shallow copy of the LineStringsOnImage object.

**Parameters**

- `line_strings` (None or list of imgaug.augmentables.lines.LineString, optional) – List of line strings on the image. If not `None`, then the line_strings attribute of the copied object will be set to this value.

- `shape` (None or tuple of int or ndarray, optional) – The shape of the image on which the objects are placed. Either an image with shape \((H, W, \ldots)\) or a tuple denoting such an image shape. If not `None`, then the shape attribute of the copied object will be set to this value.

**Returns** Shallow copy.
Return type `imgaug.augmentables.lines.LineStringsOnImage`

deeplcopy (line_strings=None, shape=None)
Create a deep copy of the LineStringsOnImage object.

Parameters

- **line_strings** *(None or list of imgaug.augmentables.lines.LineString, optional)* – List of line strings on the image. If not None, then the line_strings attribute of the copied object will be set to this value.
- **shape** *(None or tuple of int or ndarray, optional)* – The shape of the image on which the objects are placed. Either an image with shape \((H, W, [C])\) or a tuple denoting such an image shape. If not None, then the shape attribute of the copied object will be set to this value.

Returns Deep copy.

Return type `imgaug.augmentables.lines.LineStringsOnImage`

draw_on_image (image, color=(0, 255, 0), color_lines=None, color_points=None, alpha=1.0, alpha_lines=None, alpha_points=None, size=1, size_lines=None, size_points=None, antialiased=True, raise_if_out_of_image=False)
Draw all line strings onto a given image.

Parameters

- **image** *(ndarray)* – The \((H,W,C)\) uint8 image onto which to draw the line strings.
- **color** *(iterable of int, optional)* – Color to use as RGB, i.e. three values. The color of the lines and points are derived from this value, unless they are set.
- **color_lines** *(None or iterable of int)* – Color to use for the line segments as RGB, i.e. three values. If None, this value is derived from color.
- **color_points** *(None or iterable of int)* – Color to use for the points as RGB, i.e. three values. If None, this value is derived from \(0.5 \times \text{color}\).
- **alpha** *(float, optional)* – Opacity of the line strings. Higher values denote more visible points. The alphas of the line and points are derived from this value, unless they are set.
- **alpha_lines** *(None or float, optional)* – Opacity of the line strings. Higher values denote more visible line string. If None, this value is derived from alpha.
- **alpha_points** *(None or float, optional)* – Opacity of the line string points. Higher values denote more visible points. If None, this value is derived from alpha.
- **size** *(int, optional)* – Size of the line strings. The sizes of the line and points are derived from this value, unless they are set.
- **size_lines** *(None or int, optional)* – Thickness of the line segments. If None, this value is derived from size.
- **size_points** *(None or int, optional)* – Size of the points in pixels. If None, this value is derived from \(3 \times \text{size}\).
- **antialiased** *(bool, optional)* – Whether to draw the lines with anti-aliasing activated. This does currently not affect the point drawing.
- **raise_if_out_of_image** *(bool, optional)* – Whether to raise an error if a line string is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

Returns Image with line strings drawn on it.
Return type: ndarray

empty
Returns whether this object contains zero line strings.

Returns: True if this object contains zero line strings.
Return type: bool

classmethod from_xy_arrays(xy, shape)
Convert an \((N,M,2)\) ndarray to a LineStringsOnImage object.
This is the inverse of imgaug.augmentables.lines.LineStringsOnImage.
to_xy_array().

Parameters
- \(xy\) ((\(N,M,2\) ndarray or iterable of \((M,2)\) ndarray)) – Array containing the point coordinates
  \(N\) line strings with each \(M\) points given as \((x, y)\) coordinates. \(M\) may differ if an iterable
  of arrays is used. Each array should usually be of dtype float32.
- \(shape\) (tuple of int) – \((H, W, [C])\) shape of the image on which the line strings are placed.

Returns: Object containing a list of LineString objects following the provided point coordinates.
Return type: imgaug.augmentables.lines.LineStringsOnImage

on(image)
Project bounding boxes from one image to a new one.

Parameters:
- \(image\) (ndarray or tuple of int) – The new image onto which to project. Either an
  image with shape \((H, W, [C])\) or a tuple denoting such an image shape.

Returns: line_strings – Object containing all projected line strings.
Return type: imgaug.augmentables.lines.LineStrings

remove_out_of_image(fully=True, partly=False)
Remove all line strings that are fully/partially outside of the image.

Parameters
- \(fully\) (bool, optional) – Whether to remove line strings that are fully outside of the image.
- \(partly\) (bool, optional) – Whether to remove line strings that are partially outside of the image.

Returns: Reduced set of line strings, with those that were fully/partially outside of the image
removed.
Return type: imgaug.augmentables.lines.LineStringsOnImage

shift(top=None, right=None, bottom=None, left=None)
Shift/move the line strings from one or more image sides.

Parameters
- \(top\) (None or int, optional) – Amount of pixels by which to shift all bounding boxes from
  the top.
- \(right\) (None or int, optional) – Amount of pixels by which to shift all bounding boxes from
  the right.
- \(bottom\) (None or int, optional) – Amount of pixels by which to shift all bounding boxes from
  the bottom.
• **left** *(None or int, optional)* – Amount of pixels by which to shift all bounding boxes from the left.

  **Returns**  Shifted line strings.

  **Return type**  `imgaug.augmentables.lines.LineStringsOnImage`

  **to_xy_arrays**(dtypes=`numpy.float32`)  Convert this object to an iterable of \((M, 2)\) arrays of points.

  This is the inverse of `imgaug.augmentables.lines.LineStringsOnImage.from_xy_array()`.

  **Parameters dtypes** *(numpy.dtype, optional)* – Desired output datatype of the ndarray.

  **Returns**  The arrays of point coordinates, each given as \((M, 2)\).

  **Return type**  list of ndarray

### 13.10 `imgaug.augmentables.normalization`

`imgaug.augmentables.normalization.estimate_bounding_boxes_norm_type`(bounding_boxes)

`imgaug.augmentables.normalization.estimate_heatmaps_norm_type`(heatmaps)

`imgaug.augmentables.normalization.estimate_keypoints_norm_type`(keypoints)

`imgaug.augmentables.normalization.estimate_line_strings_norm_type`(line_strings)

`imgaug.augmentables.normalization.estimate_normalization_type`(inputs)

`imgaug.augmentables.normalization.estimate_polygons_norm_type`(polygons)

`imgaug.augmentables.normalization.estimate_segmaps_norm_type`(segmentation_maps)

`imgaug.augmentables.normalization.find_first_nonempty`(attr, parents=None)

`imgaug.augmentables.normalization.invert_normalize_bounding_boxes`(bounding_boxes, bounding_boxes_old)

`imgaug.augmentables.normalization.invert_normalize_heatmaps`(heatmaps, heatmaps_old)

`imgaug.augmentables.normalization.invert_normalize_images`(images, images_old)

`imgaug.augmentables.normalization.invert_normalize_keypoints`(keypoints, keypoints_old)

`imgaug.augmentables.normalization.invert_normalize_line_strings`(line_strings, line_strings_old)

`imgaug.augmentables.normalization.invert_normalize_polygons`(polygons, polygons_old)

`imgaug.augmentables.normalization.invert_normalize_segmentation_maps`(segmentation_maps, segmentation_maps_old)

`imgaug.augmentables.normalization.normalize_bounding_boxes`(inputs, shapes=None)

`imgaug.augmentables.normalization.normalize_heatmaps`(inputs, shapes=None)

`imgaug.augmentables.normalization.normalize_images`(images)
`imgaug.augmentables.normalization.normalize_keypoints(inputs, shapes=None)`

`imgaug.augmentables.normalization.normalize_line_strings(inputs, shapes=None)`

`imgaug.augmentables.normalization.normalize_polygons(inputs, shapes=None)`

`imgaug.augmentables.normalization.normalize_segmentation_maps(inputs, shapes=None)`

`imgaug.augmentables.normalization.restore_dtype_and_merge(arr, input_dtype)`

### 13.11 `imgaug.augmentables.polys`

**class `imgaug.augmentables.polys.MultiPolygon(geoms)`**

Bases: `object`

Class that represents several polygons.

**Parameters**

- **geoms** (list of `imgaug.Polygon`) – List of the polygons.

**Methods**

*static* `from_shapely(geometry[, label])`

Create a MultiPolygon from a Shapely MultiPolygon, a Shapely Polygon or a Shapely GeometryCollection.

**Parameters**

- **geometry** (shapely.geometry.MultiPolygon or shapely.geometry.Polygon or shapely.geometry.collection.GeometryCollection) – The object to convert to a MultiPolygon.

- **label** (None or str, optional) – A label assigned to all Polygons within the MultiPolygon.

**Returns** The derived MultiPolygon.

**Return type** `imgaug.MultiPolygon`

**class `imgaug.augmentables.polys.Polygon(exterior, label=None)`**

Bases: `object`

Class representing polygons.

Each polygon is parameterized by its corner points, given as absolute x- and y-coordinates with sub-pixel accuracy.

**Parameters**

- **exterior** (list of `imgaug.Keypoint` or list of tuple of float or (N,2) ndarray) – List of points defining the polygon. May be either a list of Keypoint objects or a list of tuples in xy-form or a numpy array of shape (N,2) for N points in xy-form. All coordinates are expected to be the absolute coordinates in the image, given as floats, e.g. x=10.7 and y=3.4 for a point at
coordinates (10.7, 3.4). Their order is expected to be clock-wise. They are expected to not be closed (i.e. first and last coordinate differ).

- **label** (*None or str, optional*) – Label of the polygon, e.g. a string representing the class.

**Attributes**

- **area** Estimate the area of the polygon.
- **height** Estimate the height of the polygon.
- **is_valid** Estimate whether the polygon has a valid shape.
- **width** Estimate the width of the polygon.
- **xx** Return the x-coordinates of all points in the exterior.
- **xx_int** Return the x-coordinates of all points in the exterior, rounded to the closest integer value.
- **yy** Return the y-coordinates of all points in the exterior.
- **yy_int** Return the y-coordinates of all points in the exterior, rounded to the closest integer value.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>almost_equals(other[, max_distance, ...])</code></td>
<td>Estimate if this polygon’s and another’s geometry/labels are similar.</td>
</tr>
<tr>
<td><code>change_first_point_by_coords(x, y[, ...])</code></td>
<td>Set the first point of the exterior to the given point based on its coordinates.</td>
</tr>
<tr>
<td><code>change_first_point_by_index(point_idx)</code></td>
<td>Set the first point of the exterior to the given point based on its index.</td>
</tr>
<tr>
<td><code>clip_out_of_image(image)</code></td>
<td>Cut off all parts of the polygon that are outside of the image.</td>
</tr>
<tr>
<td><code>copy([exterior, label])</code></td>
<td>Create a shallow copy of the Polygon object.</td>
</tr>
<tr>
<td><code>cut_out_of_image(image)</code></td>
<td>Create a deep copy of the Polygon object.</td>
</tr>
<tr>
<td><code>deepcopy([exterior, label])</code></td>
<td>Create a deep copy of the Polygon object.</td>
</tr>
<tr>
<td><code>draw_on_image(image[, color, color_face, ...])</code></td>
<td>Draw the polygon on an image.</td>
</tr>
<tr>
<td><code>exterior_almost_equals(other[, ...])</code></td>
<td>Estimate if this and other polygon’s exterior are almost identical.</td>
</tr>
<tr>
<td><code>extract_from_image(image)</code></td>
<td>Extract the image pixels within the polygon.</td>
</tr>
<tr>
<td><code>find_closest_point_index(x, y[, return_distance])</code></td>
<td>Find the index of the point within the exterior that is closest to the given coordinates.</td>
</tr>
<tr>
<td><code>from_shapely(polygon_shapely[, label])</code></td>
<td>Create a polygon from a Shapely polygon.</td>
</tr>
<tr>
<td><code>is_fully_within_image(image)</code></td>
<td>Estimate whether the polygon is fully inside the image area.</td>
</tr>
<tr>
<td><code>is_out_of_image(image[, fully, partly])</code></td>
<td>Estimate whether the polygon is partially or fully outside of the image area.</td>
</tr>
<tr>
<td><code>is_partly_within_image(image)</code></td>
<td>Estimate whether the polygon is at least partially outside of the image area.</td>
</tr>
<tr>
<td><code>project(from_shape, to_shape)</code></td>
<td>Project the polygon onto an image with different shape.</td>
</tr>
<tr>
<td><code>shift([top, right, bottom, left])</code></td>
<td>Shift the polygon from one or more image sides, i.e.</td>
</tr>
<tr>
<td><code>to_bounding_box()</code></td>
<td>Convert this polygon to a bounding box tightly containing the whole polygon.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>to_keypoints()</td>
<td>Convert this polygon’s exterior to Keypoint instances.</td>
</tr>
<tr>
<td>to_line_string([closed])</td>
<td>Convert this polygon’s exterior to a LineString instance.</td>
</tr>
<tr>
<td>to_shapely_line_string([closed, interpolate])</td>
<td>Convert this polygon to a Shapely LineString object.</td>
</tr>
<tr>
<td>to_shapely_polygon()</td>
<td>Convert this polygon to a Shapely polygon.</td>
</tr>
</tbody>
</table>

**almost_equal** *(other, max_distance=1e-06, points_per_edge=8)*

Estimate if this polygon’s and another’s geometry/labels are similar.

This is the same as `imgaug.Polygon.exterior_almost_equal()` but additionally compares the labels.

**Parameters**

- `other` – The object to compare against. If not a Polygon, then False will be returned.
- `max_distance (float, optional)` – See `imgaug.augmentables.polys.Polygon.exterior_almost_equal()`.
- `points_per_edge (int, optional)` – See `imgaug.augmentables.polys.Polygon.exterior_almost_equal()`.

**Returns** Whether the two polygons can be viewed as equal. In the case of the exteriors this is an approximate test.

**Return type** bool

**area**

Estimate the area of the polygon.

**Returns** Area of the polygon.

**Return type** number

**change_first_point_by_coords (x, y, max_distance=0.0001, raise_if_too_far_away=True)**

Set the first point of the exterior to the given point based on its coordinates.

If multiple points are found, the closest one will be picked. If no matching points are found, an exception is raised.

**Note**: This method does not work in-place.

**Parameters**

- `x (number)` – X-coordinate of the point.
- `y (number)` – Y-coordinate of the point.
- `max_distance (None or number, optional)` – Maximum distance past which possible matches are ignored. If None the distance limit is deactivated.
- `raise_if_too_far_away (bool, optional)` – Whether to raise an exception if the closest found point is too far away (True) or simply return an unchanged copy if this object (False).

**Returns** Copy of this polygon with the new point order.

**Return type** imgaug.Polygon
**change_first_point_by_index** (*point_idx*)

Set the first point of the exterior to the given point based on its index.

Note: This method does not work in-place.

- **Parameters** 
  - *point_idx* (*int*) – Index of the desired starting point.

- **Returns** 
  - Copy of this polygon with the new point order.

- **Return type** 
  - `imgaug.Polygon`

**clip_out_of_image** (*image*)

Cut off all parts of the polygon that are outside of the image.

This operation may lead to new points being created. As a single polygon may be split into multiple new polygons, the result is always a list, which may contain more than one output polygon.

This operation will return an empty list if the polygon is completely outside of the image plane.

- **Parameters** 
  - *image* (*ndarray or tuple of int*) – Image dimensions to use for the clipping of the polygon. If an ndarray, its shape will be used. If a tuple, it is assumed to represent the image shape and must contain at least two integers.

- **Returns** 
  - Polygon, clipped to fall within the image dimensions. Returned as a list, because the clipping can split the polygon into multiple parts. The list may also be empty, if the polygon was fully outside of the image plane.

- **Return type** 
  - list of `imgaug.Polygon`

**copy** (*exterior=None, label=None*)

Create a shallow copy of the Polygon object.

- **Parameters**
  - *exterior* (*list of Keypoint or list of tuple or (N,2) ndarray, optional*) – List of points defining the polygon. See `imgaug.Polygon.__init__()` for details.
  - *label* (*None or str, optional*) – If not None, then the label of the copied object will be set to this value.

- **Returns** 
  - Shallow copy.

- **Return type** 
  - `imgaug.Polygon`

**cut_out_of_image** (*image*)

Deprecated. Use `Polygon.clip_out_of_image()` instead. `clip_out_of_image()` has the exactly same interface.

**deepcopy** (*exterior=None, label=None*)

Create a deep copy of the Polygon object.

- **Parameters**
  - *exterior* (*list of Keypoint or list of tuple or (N,2) ndarray, optional*) – List of points defining the polygon. See `imgaug.Polygon.__init__()` for details.
  - *label* (*None or str*) – If not None, then the label of the copied object will be set to this value.

- **Returns** 
  - Deep copy.

- **Return type** 
  - `imgaug.Polygon`
**draw_on_image** (*image, color=(0, 255, 0), color_face=None, color_lines=None, color_points=None, alpha=1.0, alpha_face=None, alpha_lines=None, alpha_points=None, size=1, size_lines=None, size_points=None, raise_if_out_of_image=False*)

Draw the polygon on an image.

**Parameters**

- **image** (*H,W,C* ndarray) – The image onto which to draw the polygon. Usually expected to be of dtype `uint8`, though other dtypes are also handled.
- **color** (*iterable of int, optional*) – The color to use for the whole polygon. Must correspond to the channel layout of the image. Usually RGB. The values for `color_face`, `color_lines` and `color_points` will be derived from this color if they are set to `None`. This argument has no effect if `color_face`, `color_lines` and `color_points` are all set anything other than `None`.
- **color_face** (*None or iterable of int, optional*) – The color to use for the inner polygon area (excluding perimeter). Must correspond to the channel layout of the image. Usually RGB. If this is `None`, it will be derived from `color * 1.0`.
- **color_lines** (*None or iterable of int, optional*) – The color to use for the line (aka perimeter/border) of the polygon. Must correspond to the channel layout of the image. Usually RGB. If this is `None`, it will be derived from `color * 0.5`.
- **color_points** (*None or iterable of int, optional*) – The color to use for the corner points of the polygon. Must correspond to the channel layout of the image. Usually RGB. If this is `None`, it will be derived from `color * 0.5`.
- **alpha** (*float, optional*) – The opacity of the whole polygon, where 1.0 denotes a completely visible polygon and 0.0 an invisible one. The values for `alpha_face`, `alpha_lines` and `alpha_points` will be derived from this alpha value if they are set to `None`. This argument has no effect if `alpha_face`, `alpha_lines` and `alpha_points` are all set anything other than `None`.
- **alpha_face** (*None or number, optional*) – The opacity of the polygon’s inner area (excluding the perimeter), where 1.0 denotes a completely visible inner area and 0.0 an invisible one. If this is `None`, it will be derived from `alpha * 0.5`.
- **alpha_lines** (*None or number, optional*) – The opacity of the polygon’s line (aka perimeter/border), where 1.0 denotes a completely visible line and 0.0 an invisible one. If this is `None`, it will be derived from `alpha * 1.0`.
- **alpha_points** (*None or number, optional*) – The opacity of the polygon’s corner points, where 1.0 denotes completely visible corners and 0.0 invisible ones. If this is `None`, it will be derived from `alpha * 1.0`.
- **size** (*int, optional*) – Size of the polygon. The sizes of the line and points are derived from this value, unless they are set.
- **size_lines** (*None or int, optional*) – Thickness of the polygon’s line (aka perimeter/border). If `None`, this value is derived from `size`.
- **size_points** (*int, optional*) – Size of the points in pixels. If `None`, this value is derived from `3 * size`.
- **raise_if_out_of_image** (*bool, optional*) – Whether to raise an error if the polygon is fully outside of the image. If set to False, no error will be raised and only the parts inside the image will be drawn.

**Returns** `result` – Image with polygon drawn on it. Result dtype is the same as the input dtype.

**Return type** (*H,W,C* ndarray)
exterior_almost_equals \((\text{other, max\_distance}=1e-06, \text{points\_per\_edge}=8)\)

Estimate if this and other polygon’s exterior are almost identical.

The two exteriors can have different numbers of points, but any point randomly sampled on the exterior of one polygon should be close to the closest point on the exterior of the other polygon.

Note that this method works approximately. One can come up with polygons with fairly different shapes that will still be estimated as equal by this method. In practice however this should be unlikely to be the case. The probability for something like that goes down as the interpolation parameter is increased.

**Parameters**

- **other** (`imgaug.Polygon or (N,2) ndarray or list of tuple`) – The other polygon with which to compare the exterior. If this is an ndarray, it is assumed to represent an exterior. It must then have `dtype float32` and shape \((N, 2)\) with the second dimension denoting xy-coordinates. If this is a list of tuples, it is assumed to represent an exterior. Each tuple then must contain exactly two numbers, denoting xy-coordinates.

- **max\_distance** (`number, optional`) – The maximum euclidean distance between a point on one polygon and the closest point on the other polygon. If the distance is exceeded for any such pair, the two exteriors are not viewed as equal. The points are other the points contained in the polygon’s exterior ndarray or interpolated points between these.

- **points\_per\_edge** (`int, optional`) – How many points to interpolate on each edge.

**Returns** Whether the two polygon’s exteriors can be viewed as equal (approximate test).

**Return type** `bool`

extract_from_image \((image)\)

Extract the image pixels within the polygon.

This function will zero-pad the image if the polygon is partially/fully outside of the image.

**Parameters** `image` \((H,W)\) `ndarray or (H,W,C) ndarray\) – The image from which to extract the pixels within the polygon.

**Returns** `result` – Pixels within the polygon. Zero-padded if the polygon is partially/fully outside of the image.

**Return type** \((H',W')\) `ndarray or (H',W',C) ndarray`

find_closest_point_index \((x, y, return\_distance=False)\)

Find the index of the point within the exterior that is closest to the given coordinates.

“Closeness” is here defined based on euclidean distance. This method will raise an AssertionError if the exterior contains no points.

**Parameters**

- **x** (`number`) – X-coordinate around which to search for close points.

- **y** (`number`) – Y-coordinate around which to search for close points.

- **return\_distance** (`bool, optional`) – Whether to also return the distance of the closest point.

**Returns**

- **int** – Index of the closest point.

- **number** – Euclidean distance to the closest point. This value is only returned if `return\_distance` was set to True.

static from_shapely \((\text{polygon\_shapely, label=None})\)

Create a polygon from a Shapely polygon.
Note: This will remove any holes in the Shapely polygon.

**Parameters**

- **polygon_shapely** (*shapely.geometry.Polygon*) – The shapely polygon.
- **label** (*None or str, optional*) – The label of the new polygon.

**Returns** A polygon with the same exterior as the Shapely polygon.

**Return type** `imgaug.Polygon`

**height**

Estimate the height of the polygon.

**Returns** Height of the polygon.

**Return type** `number`

**is_fully_within_image** (*image*)

Estimate whether the polygon is fully inside the image area.

**Parameters**

- **image** (*((H,W,...) ndarray or tuple of int]*) – Image dimensions to use. If an ndarray, its shape will be used. If a tuple, it is assumed to represent the image shape and must contain at least two integers.

**Returns** True if the polygon is fully inside the image area. False otherwise.

**Return type** `bool`

**is_out_of_image** (*image, fully=True, partly=False*)

Estimate whether the polygon is partially or fully outside of the image area.

**Parameters**

- **image** (*((H,W,...) ndarray or tuple of int]*) – Image dimensions to use. If an ndarray, its shape will be used. If a tuple, it is assumed to represent the image shape and must contain at least two integers.

- **fully** (*bool, optional*) – Whether to return True if the polygon is fully outside of the image area.

- **partly** (*bool, optional*) – Whether to return True if the polygon is at least partially outside the image area.

**Returns** True if the polygon is partially/fully outside of the image area, depending on defined parameters. False otherwise.

**Return type** `bool`

**is_partly_within_image** (*image*)

Estimate whether the polygon is at least partially inside the image area.

**Parameters**

- **image** (*((H,W,...) ndarray or tuple of int]*) – Image dimensions to use. If an ndarray, its shape will be used. If a tuple, it is assumed to represent the image shape and must contain at least two integers.

**Returns** True if the polygon is at least partially inside the image area. False otherwise.

**Return type** `bool`

**is_valid**

Estimate whether the polygon has a valid shape.

To be considered valid, the polygons must be made up of at least 3 points and have concave shape. Multiple consecutive points are allowed to have the same coordinates.
Returns True if polygon has at least 3 points and is concave, otherwise False.

Return type bool

project (from_shape, to_shape)
Project the polygon onto an image with different shape.

The relative coordinates of all points remain the same. E.g. a point at (x=20, y=20) on an image (width=100, height=200) will be projected on a new image (width=200, height=100) to (x=40, y=10).

This is intended for cases where the original image is resized. It cannot be used for more complex changes (e.g. padding, cropping).

Parameters

• from_shape (tuple of int) – Shape of the original image. (Before resize.)
• to_shape (tuple of int) – Shape of the new image. (After resize.)

Returns Polygon object with new coordinates.

Return type imgaug.Polygon

shift (top=None, right=None, bottom=None, left=None)
Shift the polygon from one or more image sides, i.e. move it on the x/y-axis.

Parameters

• top (None or int, optional) – Amount of pixels by which to shift the polygon from the top.
• right (None or int, optional) – Amount of pixels by which to shift the polygon from the right.
• bottom (None or int, optional) – Amount of pixels by which to shift the polygon from the bottom.
• left (None or int, optional) – Amount of pixels by which to shift the polygon from the left.

Returns Shifted polygon.

Return type imgaug.Polygon

to_bounding_box ()
Convert this polygon to a bounding box tightly containing the whole polygon.

Returns Tight bounding box around the polygon.

Return type imgaug.BoundingBox

to_keypoints ()
Convert this polygon’s exterior to Keypoint instances.

Returns Exterior vertices as Keypoint instances.

Return type list of imgaug.Keypoint

to_line_string (closed=True)
Convert this polygon’s exterior to a LineString instance.

Parameters closed (bool, optional) – Whether to close the line string, i.e. to add the first point of the exterior also as the last point at the end of the line string. This has no effect if the polygon has a single point or zero points.

Returns Exterior of the polygon as a line string.

Return type imgaug.augmentables.lines.LineString
to_shapely_line_string\( (closed=False, interpolate=0) \)
Convert this polygon to a Shapely LineString object.

Parameters

- closed (bool, optional) – Whether to return the line string with the last point being identical to the first point.
- interpolate (int, optional) – Number of points to interpolate between any pair of two consecutive points. These points are added to the final line string.

Returns The Shapely LineString matching the polygon’s exterior.

Return type shapely.geometry.LineString

to_shapely_polygon()
Convert this polygon to a Shapely polygon.

Returns The Shapely polygon matching this polygon’s exterior.

Return type shapely.geometry.Polygon

width
Estimate the width of the polygon.

Returns Width of the polygon.

Return type number

xx
Return the x-coordinates of all points in the exterior.

Returns X-coordinates of all points in the exterior as a float32 ndarray.

Return type (N,2) ndarray

xx_int
Return the x-coordinates of all points in the exterior, rounded to the closest integer value.

Returns X-coordinates of all points in the exterior, rounded to the closest integer value. Result dtype is int32.

Return type (N,2) ndarray

yy
Return the y-coordinates of all points in the exterior.

Returns Y-coordinates of all points in the exterior as a float32 ndarray.

Return type (N,2) ndarray

yy_int
Return the y-coordinates of all points in the exterior, rounded to the closest integer value.

Returns Y-coordinates of all points in the exterior, rounded to the closest integer value. Result dtype is int32.

Return type (N,2) ndarray

class imgaug.augmentables.polys.PolygonsOnImage\( (polys, shape) \)
Bases: object

Object that represents all polygons on a single image.

Parameters

- polys (list of imgaug.Polygon) – List of polygons on the image.
• **shape** *(tuple of int)* – The shape of the image on which the polygons are placed.

### Examples

```python
>>> import numpy as np
>>> import imgaug as ia
>>> image = np.zeros((100, 100))
>>> polys = [
    ia.Polygon([(0, 0), (100, 0), (100, 100), (0, 100)]),
    ia.Polygon([(50, 0), (100, 50), (50, 100), (0, 50)])
]
>>> polys_oi = ia.PolygonsOnImage(polys, shape=image.shape)
```

### Attributes

- **empty** Returns whether this object contains zero polygons.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clip_out_of_image()</code></td>
<td>Clip off all parts from all polygons that are outside of the image.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of the PolygonsOnImage object.</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of the PolygonsOnImage object.</td>
</tr>
<tr>
<td><code>draw_on_image(image[, color, color_face,...])</code></td>
<td>Draw all polygons onto a given image.</td>
</tr>
<tr>
<td><code>on(image)</code></td>
<td>Project polygons from one image to a new one.</td>
</tr>
<tr>
<td><code>remove_out_of_image([fully, partly])</code></td>
<td>Remove all polygons that are fully or partially outside of the image.</td>
</tr>
<tr>
<td><code>shift([top, right, bottom, left])</code></td>
<td>Shift all polygons from one or more image sides, i.e.</td>
</tr>
</tbody>
</table>

**clip_out_of_image()**

Clip off all parts from all polygons that are outside of the image.

**NOTE:** The result can contain less polygons than the input did. That happens when a polygon is fully outside of the image plane.

**NOTE:** The result can also contain more polygons than the input did. That happens when distinct parts of a polygon are only connected by areas that are outside of the image plane and hence will be clipped off, resulting in two or more unconnected polygon parts that are left in the image plane.

**Returns** Polygons, clipped to fall within the image dimensions. Count of output polygons may differ from the input count.

**Return type** `imgaug.PolygonsOnImage`

**copy()**

Create a shallow copy of the PolygonsOnImage object.

**Returns** Shallow copy.

**Return type** `imgaug.PolygonsOnImage`

**deepcopy()**

Create a deep copy of the PolygonsOnImage object.

**Returns** Deep copy.
**Return type**  
`imgaug.PolygonsOnImage`

**draw_on_image** (*image, color=(0, 255, 0), color_face=None, color_lines=None, color_points=None, alpha=1.0, alpha_face=None, alpha_lines=None, alpha_points=None, size=1, size_lines=None, size_points=None, raise_if_out_of_image=False*)

Draw all polygons onto a given image.

**Parameters**

- **image** (*(`H,W,C`) ndarray*) – The image onto which to draw the bounding boxes. This image should usually have the same shape as set in `PolygonsOnImage.shape`.

- **color** (*iterable of int, optional*) – The color to use for the whole polygons. Must correspond to the channel layout of the image. Usually RGB. The values for `color_face`, `color_lines` and `color_points` will be derived from this color if they are set to `None`. This argument has no effect if `color_face`, `color_lines` and `color_points` are all set anything other than `None`.

- **color_face** (*None or iterable of int, optional*) – The color to use for the inner polygon areas (excluding perimeters). Must correspond to the channel layout of the image. Usually RGB. If this is `None`, it will be derived from `color * 1.0`.

- **color_lines** (*None or iterable of int, optional*) – The color to use for the lines (aka perimeters/borders) of the polygons. Must correspond to the channel layout of the image. Usually RGB. If this is `None`, it will be derived from `color * 0.5`.

- **color_points** (*None or iterable of int, optional*) – The color to use for the corner points of the polygons. Must correspond to the channel layout of the image. Usually RGB. If this is `None`, it will be derived from `color * 0.5`.

- **alpha** (*float, optional*) – The opacity of the whole polygons, where 1.0 denotes completely visible polygons and 0.0 invisible ones. The values for `alpha_face`, `alpha_lines` and `alpha_points` will be derived from this alpha value if they are set to `None`. This argument has no effect if `alpha_face`, `alpha_lines` and `alpha_points` are all set anything other than `None`.

- **alpha_face** (*None or number, optional*) – The opacity of the polygon’s inner areas (excluding the perimeters), where 1.0 denotes completely visible inner areas and 0.0 invisible ones. If this is `None`, it will be derived from `alpha * 0.5`.

- **alpha_lines** (*None or number, optional*) – The opacity of the polygon’s lines (aka perimeters/borders), where 1.0 denotes completely visible perimeters and 0.0 invisible ones. If this is `None`, it will be derived from `alpha * 1.0`.

- **alpha_points** (*None or number, optional*) – The opacity of the polygon’s corner points, where 1.0 denotes completely visible corners and 0.0 invisible ones. Currently this is an on/off choice, i.e. only 0.0 or 1.0 are allowed. If this is `None`, it will be derived from `alpha * 1.0`.

- **size** (*int, optional*) – Size of the polygons. The sizes of the line and points are derived from this value, unless they are set.

- **size_lines** (*None or int, optional*) – Thickness of the polygon lines (aka perimeter/border). If `None`, this value is derived from `size`.

- **size_points** (*int, optional*) – The size of all corner points. If set to `C`, each corner point will be drawn as a square of size `C x C`.

- **raise_if_out_of_image** (*bool, optional*) – Whether to raise an error if any polygon is fully outside of the image. If set to `False`, no error will be raised and only the parts inside the image will be drawn.

**Returns**  
`image` – Image with drawn polygons.
Return type  (H,W,C) ndarray

empty
Returns whether this object contains zero polygons.
Returns  True if this object contains zero polygons.
Return type  bool

on (image)
Project polygons from one image to a new one.
Parameters  image (ndarray or tuple of int) – New image onto which the polygons are to be projected. May also simply be that new image’s shape tuple.
Returns  Object containing all projected polygons.
Return type  imgaug.PolygonsOnImage

remove_out_of_image (fully=True, partly=False)
Remove all polygons that are fully or partially outside of the image.
Parameters
•  fully (bool, optional) – Whether to remove polygons that are fully outside of the image.
•  partly (bool, optional) – Whether to remove polygons that are partially outside of the image.
Returns  Reduced set of polygons, with those that were fully/partially outside of the image removed.
Return type  imgaug.PolygonsOnImage

shift (top=None, right=None, bottom=None, left=None)
Shift all polygons from one or more image sides, i.e. move them on the x/y-axis.
Parameters
•  top (None or int, optional) – Amount of pixels by which to shift all polygons from the top.
•  right (None or int, optional) – Amount of pixels by which to shift all polygons from the right.
•  bottom (None or int, optional) – Amount of pixels by which to shift all polygons from the bottom.
•  left (None or int, optional) – Amount of pixels by which to shift all polygons from the left.
Returns  Shifted polygons.
Return type  imgaug.PolygonsOnImage

13.12  imgaug.augmentables.segmaps

class  imgaug.augmentables.segmaps.SegmentationMapOnImage (arr, shape, nb_classes=None)

Bases: object

Object representing a segmentation map associated with an image.

Variables  DEFAULT_SEGMENT_COLORS (list of tuple of int) – Standard RGB colors to use during drawing, ordered by class index.
Parameters

- **arr** *(H,W) ndarray or (H,W,1) ndarray or (H,W,C) ndarray* – Array representing the segmentation map. May have datatypes bool, integer or float.
  - If bool: Assumed to be of shape (H,W), (H,W,1) or (H,W,C). If (H,W) or (H,W,1) it is assumed to be for the case of having a single class (where any False denotes background). Otherwise there are assumed to be C channels, one for each class, with each of them containing a mask for that class. The masks may overlap.
  - If integer: Assumed to be of shape (H,W) or (H,W,1). Each pixel is assumed to contain an integer denoting the class index. Classes are assumed to be non-overlapping. The number of classes cannot be guessed from this input, hence nb_classes must be set.
  - If float: Assumed to be of shape (H,W), (H,W,1) or (H,W,C) with meanings being similar to the case of bool. Values are expected to fall always in the range 0.0 to 1.0 and are usually expected to be either 0.0 or 1.0 upon instantiation of a new segmentation map. Classes may overlap.

- **shape** *(iterable of int)* – Shape of the corresponding image (NOT the segmentation map array). This is expected to be *(H, W)* or *(H, W, C)* with C usually being 3. If there is no corresponding image, then use the segmentation map’s shape instead.

- **nb_classes** *(int or None)* – Total number of unique classes that may appear in an segmentation map, i.e. the max class index plus 1. This may be None if the input array is of type bool or float. The number of classes however must be provided if the input array is of type int, as then the number of classes cannot be guessed.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>copy()</strong></td>
<td>Create a shallow copy of the segmentation map object.</td>
</tr>
<tr>
<td><strong>deepcopy()</strong></td>
<td>Create a deep copy of the segmentation map object.</td>
</tr>
<tr>
<td><strong>draw([size, background_threshold, ...])</strong></td>
<td>Render the segmentation map as an RGB image.</td>
</tr>
<tr>
<td><strong>draw_on_image(image[, alpha, resize, ...])</strong></td>
<td>Draw the segmentation map as an overlay over an image.</td>
</tr>
<tr>
<td><strong>from_heatmaps(heatmaps[, class_indices, ...])</strong></td>
<td>Convert heatmaps to segmentation map.</td>
</tr>
<tr>
<td><strong>get_arr_int([background_threshold, ...])</strong></td>
<td>Get the segmentation map array as an integer array of shape <em>(H, W).</em></td>
</tr>
<tr>
<td><strong>pad([top, right, bottom, left, mode, cval])</strong></td>
<td>Pad the segmentation map on its top/right/bottom/left side.</td>
</tr>
<tr>
<td><strong>pad_to_aspect_ratio(aspect_ratio[, mode, ...])</strong></td>
<td>Pad the segmentation map on its sides so that its matches a target aspect ratio.</td>
</tr>
<tr>
<td><strong>resize(sizes[, interpolation])</strong></td>
<td>Resize the segmentation map array to the provided size given the provided interpolation.</td>
</tr>
<tr>
<td>**scale(*args, <strong>kwargs)</strong></td>
<td>Deprecated.</td>
</tr>
<tr>
<td><strong>to_heatmaps([only_nonempty, ...])</strong></td>
<td>Convert segmentation map to heatmaps object.</td>
</tr>
</tbody>
</table>

**DEFAULT_SEGMENT_COLORS** = [(0, 0, 0), (230, 25, 75), (60, 180, 75), (255, 225, 25), (0, 130, 200), (245, 130, 48), (145, 238, 127), (85, 55, 20), (127, 125, 100), (64, 0, 0), (85, 127, 97), (64, 64, 0), (127, 107, 90), (0, 0, 64), (64, 64, 64)]

Create a shallow copy of the segmentation map object.

**Returns** Shallow copy.

**Return type** `imgaug.SegmentationMapOnImage`
deepcopy()
Create a deep copy of the segmentation map object.

Returns Deep copy.

Return type imgaug.SegmentationMapOnImage
draw(size=None, background_threshold=0.01, background_class_id=None, colors=None, return_foreground_mask=False)
Render the segmentation map as an RGB image.

Parameters

• size (None or float or iterable of int or iterable of float, optional) – Size of the rendered RGB image as (height, width). See imgaug.imresize_single_image() for details. If set to None, no resizing is performed and the size of the segmentation map array is used.

• background_threshold (float, optional) – See imgaug.SegmentationMapOnImage.get_arr_int().

• background_class_id (None or int, optional) – See imgaug.SegmentationMapOnImage.get_arr_int().

• colors (None or list of tuple of int, optional) – Colors to use. One for each class to draw. If None, then default colors will be used.

• return_foreground_mask (bool, optional) – Whether to return a mask of the same size as the drawn segmentation map, containing True at any spatial location that is not the background class and False everywhere else.

Returns

• segmap_drawn ((H,W,3) ndarray) – Rendered segmentation map (dtype is uint8).

• foreground_mask ((H,W) ndarray) – Mask indicating the locations of foreground classes (dtype is bool). This value is only returned if return_foreground_mask is True.

draw_on_image(image, alpha=0.75, resize='segmentation_map', background_threshold=0.01, background_class_id=None, colors=None, draw_background=False)
Draw the segmentation map as an overlay over an image.

Parameters

• image ((H,W,3) ndarray) – Image onto which to draw the segmentation map. Dtype is expected to be uint8.

• alpha (float, optional) – Alpha/opacity value to use for the mixing of image and segmentation map. Higher values mean that the segmentation map will be more visible and the image less visible.

• resize (‘segmentation_map’, ‘image’), optional) – In case of size differences between the image and segmentation map, either the image or the segmentation map can be resized. This parameter controls which of the two will be resized to the other’s size.

• background_threshold (float, optional) – See imgaug.SegmentationMapOnImage.get_arr_int().

• background_class_id (None or int, optional) – See imgaug.SegmentationMapOnImage.get_arr_int().

• colors (None or list of tuple of int, optional) – Colors to use. One for each class to draw. If None, then default colors will be used.
• **draw_background** *(bool, optional)* – If True, the background will be drawn like any other class. If False, the background will not be drawn, i.e. the respective background pixels will be identical with the image’s RGB color at the corresponding spatial location and no color overlay will be applied.

**Returns** *mix* – Rendered overlays (dtype is uint8).

**Return type** *(H,W,3) ndarray*

**static from_heatmaps** *(heatmaps, class_indices=None, nb_classes=None)*

Convert heatmaps to segmentation map.

Assumes that each class is represented as a single heatmap channel.

**Parameters**

• **heatmaps** *(imgaug.HeatmapsOnImage)* – Heatmaps to convert.

• **class_indices** *(None or list of int, optional)* – List of class indices represented by each heatmap channel. See also the secondary output of *imgaug.SegmentationMapOnImage.to_heatmap()* If this is provided, it must have the same length as the number of heatmap channels.

• **nb_classes** *(None or int, optional)* – Number of classes. Must be provided if *class_indices* is set.

**Returns** Segmentation map derived from heatmaps.

**Return type** *imgaug.SegmentationMapOnImage*

**get_arr_int** *(background_threshold=0.01, background_class_id=None)*

Get the segmentation map array as an integer array of shape *(H, W)*.

Each pixel in that array contains an integer value representing the pixel’s class. If multiple classes overlap, the one with the highest local float value is picked. If that highest local value is below *background_threshold*, the method instead uses the background class id as the pixel’s class value. By default, class id 0 is the background class. This may only be changed if the original input to the segmentation map object was an integer map.

**Parameters**

• **background_threshold** *(float, optional)* – At each pixel, each class-heatmap has a value between 0.0 and 1.0. If none of the class-heatmaps has a value above this threshold, the method uses the background class id instead.

• **background_class_id** *(None or int, optional)* – Class id to fall back to if no class-heatmap passes the threshold at a spatial location. May only be provided if the original input was an integer mask and in these cases defaults to 0. If the input were float or boolean masks, the background class id may not be set as it is assumed that the background is implicitly defined as ‘any spatial location that has zero-like values in all masks’.

**Returns** *result* – Segmentation map array (int32). If the original input consisted of boolean or float masks, then the highest possible class id is 1+C, where *C* is the number of provided float/boolean masks. The value 0 in the integer mask then denotes the background class.

**Return type** *(H,W) ndarray*

**pad** *(top=0, right=0, bottom=0, left=0, mode=’constant’, cval=0.0)*

Pad the segmentation map on its top/right/bottom/left side.

**Parameters**

• **top** *(int, optional)* – Amount of pixels to add at the top side of the segmentation map. Must be 0 or greater.
• **right** (*int, optional*) – Amount of pixels to add at the right side of the segmentation map. Must be 0 or greater.

• **bottom** (*int, optional*) – Amount of pixels to add at the bottom side of the segmentation map. Must be 0 or greater.

• **left** (*int, optional*) – Amount of pixels to add at the left side of the segmentation map. Must be 0 or greater.

• **mode** (*str, optional*) – Padding mode to use. See `numpy.pad()` for details.

• **cval** (*number, optional*) – Value to use for padding if `mode` is `constant`. See `numpy.pad()` for details.

**Returns**

segmap – Padded segmentation map of height \( H' = H + \text{top} + \text{bottom} \) and width \( W' = W + \text{left} + \text{right} \).

**Return type** `imgaug.SegmentationMapOnImage`

**pad_to_aspect_ratio** (*aspect_ratio*, `mode='constant'`, `cval=0.0`, `return_pad_amounts=False`)

Pad the segmentation map on its sides so that its matches a target aspect ratio.

Depending on which dimension is smaller (height or width), only the corresponding sides (left/right or top/bottom) will be padded. In each case, both of the sides will be padded equally.

**Parameters**

• **aspect_ratio** (*float*) – Target aspect ratio, given as width/height. E.g. 2.0 denotes the image having twice as much width as height.

• **mode** (*str, optional*) – Padding mode to use. See `numpy.pad()` for details.

• **cval** (*number, optional*) – Value to use for padding if `mode` is `constant`. See `numpy.pad()` for details.

• **return_pad_amounts** (*bool, optional*) – If False, then only the padded image will be returned. If True, a tuple with two entries will be returned, where the first entry is the padded image and the second entry are the amounts by which each image side was padded. These amounts are again a tuple of the form (top, right, bottom, left), with each value being an integer.

**Returns**

• **segmap** (*imgaug.SegmentationMapOnImage*) – Padded segmentation map as SegmentationMapOnImage object.

• **pad_amounts** (*tuple of int*) – Amounts by which the segmentation map was padded on each side, given as a tuple (top, right, bottom, left). This tuple is only returned if `return_pad_amounts` was set to True.

**resize** (*sizes, interpolation='cubic'*)

Resize the segmentation map array to the provided size given the provided interpolation.

**Parameters**

• **sizes** (*float or iterable of int or iterable of float*) – New size of the array in (height, width). See `imgaug.imgaug.imresize_single_image()` for details.

• **interpolation** (*None or str or int, optional*) – The interpolation to use during resize. See `imgaug.imgaug.imresize_single_image()` for details. Note: The segmentation map is internally stored as multiple float-based heatmaps, making smooth interpolations potentially more reasonable than nearest neighbour interpolation.

**Returns**

segmap – Resized segmentation map object.
**Return type**  `imgaug.SegmentationMapOnImage`

**scale**

*Deprecated.* Use `SegmentationMapOnImage.resize()` instead. `resize()` has the exactly same interface.

**to_heatmaps**  
Return a heatmaps object from the segmentation map.

Each segmentation map class will be represented as a single heatmap channel.

**Parameters**

- **only_nonempty** (bool, optional) – If True, then only heatmaps for classes that appear in the segmentation map will be generated. Additionally, a list of these class ids will be returned.

- **not_none_if_no_nonempty** (bool, optional) – If `only_nonempty` is True and for a segmentation map no channel was non-empty, this function usually returns None as the heatmaps object. If however this parameter is set to True, a heatmaps object with one channel (representing class 0) will be returned as a fallback in these cases.

**Returns**

- **imgaug.HeatmapsOnImage or None** – Segmentation map as a heatmaps object. If `only_nonempty` was set to True and no class appeared in the segmentation map, then this is None.

- **class_indices** (list of int) – Class ids (0 to C-1) of the classes that were actually added to the heatmaps. Only returned if `only_nonempty` was set to True.

### 13.13 imgaug.augmentables.utils

**imgaug.augmentables.utils.interpolate_point_pair**

Interpolate between two points.

**Parameters**

- **point_a** (ndarray or tuple of number) – Point A.

- **point_b** (ndarray or tuple of number) – Point B.

- **nb_steps** (int) – Number of steps.

**Returns**

**imgaug.augmentables.utils.interpolate_points**

Interpolate between multiple points.

**Parameters**

- **points** (ndarray or tuple of number) – Points to interpolate.

- **nb_steps** (int) – Number of steps.

- **closed** (bool) – Whether to close the path.

**Returns**

**imgaug.augmentables.utils.interpolate_points_by_max_distance**

Interpolate points by maximum distance.

**Parameters**

- **points** (ndarray or tuple of number) – Points to interpolate.

- **max_distance** (float) – Maximum distance.

- **closed** (bool) – Whether to close the path.

**Returns**

**imgaug.augmentables.utils.normalize_shape**

Normalize a shape.

**Parameters**

- **shape** (tuple of int or ndarray) – Shape to normalize.

**Returns**

**imgaug.augmentables.utils.project_coords**

Project coordinates from one image to another.

**Parameters**

- **coords** (ndarray or tuple of number) – Coordinates to project. Either a (N, 2) numpy array or a tuple of (x, y) coordinates.

- **from_shape** (tuple of int or ndarray) – Old image shape.

- **to_shape** (tuple of int or ndarray) – New image shape.
• **to_shape** *(tuple of int or ndarray)* – New image shape.

**Returns** Projected coordinates as *(N, 2)* float32 numpy array.

**Return type** ndarray

13.14 imgaug.augmenters.meta

Augmenters that don’t apply augmentations themselves, but are needed for meta usage.

Do not import directly from this file, as the categorization is not final. Use instead

```python
from imgaug import augmenters as iaa
```

and then e.g.

```python
seq = iaa.Sequential([...])
```

List of augmenters:

• Augmenter (base class for all augmenters)
• Sequential
• SomeOf
• OneOf
• Sometimes
• WithChannels
• Noop
• Lambda
• AssertLambda
• AssertShape
• ChannelShuffle

Note: WithColorspace is in color.py.

imgaug.augmenters.meta.AssertLambda *(func_images=None, func_heatmaps=None, func_keypoints=None, func_polygons=None, name=None, deterministic=False, random_state=None)*

Augmenter that runs an assert on each batch of input images using a lambda function as condition.

This is useful to make generic assumption about the input images and error out early if they aren’t met.

**dtype support:**

* `'uint8'`: yes; fully tested
* `'uint16'`: yes; tested
* `'uint32'`: yes; tested
* `'uint64'`: yes; tested
* `'int8'`: yes; tested
* `'int16'`: yes; tested
* `'int32'`: yes; tested
* `'int64'`: yes; tested
* `'float16'`: yes; tested

(continues on next page)
Parameters

• **func_images** *(None or callable, optional)* – The function to call for each batch of images. It must follow the form `function(images, random_state, parents, hooks)` and return either `True` (valid input) or `False` (invalid input). It essentially reuses the interface of `imgaug.augmenters.meta.Augmenter._augment_images()`.

• **func_heatmaps** *(None or callable, optional)* – The function to call for each batch of heatmaps. It must follow the form `function(heatmaps, random_state, parents, hooks)` and return either `True` (valid input) or `False` (invalid input). It essentially reuses the interface of `imgaug.augmenters.meta.Augmenter._augment_heatmaps()`.

• **func_keypoints** *(None or callable, optional)* – The function to call for each batch of keypoints. It must follow the form `function(keypoints_on_images, random_state, parents, hooks)` and return either `True` (valid input) or `False` (invalid input). It essentially reuses the interface of `imgaug.augmenters.meta.Augmenter._augment_keypoints()`.

• **func_polygons** *(None or callable, optional)* – The function to call for each batch of polygons. It must follow the form `function(polygons_on_images, random_state, parents, hooks)` and return either `True` (valid input) or `False` (invalid input). It essentially reuses the interface of `imgaug.augmenters.meta.Augmenter._augment_polygons()`.

• **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

• **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

• **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

`imgaug.augmenters.meta.AssertShape(shape, check_images=True, check_heatmaps=True, check_keypoints=True, check_polygons=True, name=None, deterministic=False, random_state=None)`

Augmenter to make assumptions about the shape of input image(s), heatmaps and keypoints.

dtype support:

* `'uint8'`: yes; fully tested
* `'uint16'`: yes; tested
* `'uint32'`: yes; tested
* `'uint64'`: yes; tested
* `'int8'`: yes; tested
* `'int16'`: yes; tested
* `'int32'`: yes; tested
* `'int64'`: yes; tested
* `'float16'`: yes; tested
* `'float32'`: yes; tested
* `'float64'`: yes; tested

(continues on next page)
Parameters

- **shape (tuple)** – The expected shape, given as a tuple. The number of entries in the tuple must match the number of dimensions, i.e. it must contain four entries for \((N, H, W, C)\). If only a single image is augmented via `augment_image()`, then \(N\) is viewed as 1 by this augmenter. If the input image(s) don’t have a channel axis, then \(C\) is viewed as 1 by this augmenter. Each of the four entries may be None or a tuple of two ints or a list of ints.
  - If an entry is None, any value for that dimensions is accepted.
  - If an entry is int, exactly that integer value will be accepted or no other value.
  - If an entry is a tuple of two ints with values \(a\) and \(b\), only a value \(x\) with \(a \leq x < b\) will be accepted for the dimension.
  - If an entry is a list of ints, only a value for the dimension will be accepted which is contained in the list.

- **check_images (bool, optional)** – Whether to validate input images via the given shape.

- **check_heatmaps (bool, optional)** – Whether to validate input heatmaps via the given shape. The number of heatmaps will be checked and for each Heatmaps instance its array’s height and width, but not the channel count as the channel number denotes the expected number of channels in images.

- **check_keypoints (bool, optional)** – Whether to validate input keypoints via the given shape. This will check (a) the number of keypoints and (b) for each KeypointsOnImage instance the `.shape`, i.e. the shape of the corresponding image.

- **check_polygons (bool, optional)** – Whether to validate input keypoints via the given shape. This will check (a) the number of polygons and (b) for each PolygonsOnImage instance the `.shape`, i.e. the shape of the corresponding image.

- **name (None or str, optional)** – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **deterministic (bool, optional)** – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **random_state (None or int or numpy.random.RandomState, optional)** – See `imgaug.augmenters.meta.Augmenter.__init__()`.

Examples

```python
>>> seq = iaa.Sequential([
    iaa.AssertShape((None, 32, 32, 3)),
    iaa.Fliplr(0.5)
])
```

will first check for each image batch, if it contains a variable number of 32x32 images with 3 channels each. Only if that check succeeds, the horizontal flip will be executed (otherwise an assertion error will be thrown).
```python
>>> seq = iaa.Sequential([
  iaa.AssertShape(((None, (32, 64), 32, [1, 3])),
  iaa.Fliplr(0.5)
])
```

like above, but now the height may be in the range $32 \leq H < 64$ and the number of channels may be either 1 or 3.

```python
class imgaug.augmenters.meta.Augmenter(name=None, deterministic=False, random_state=None)
```

Bases: object

Base class for Augmenter objects. All augmenters derive from this class.

Methods

```python
__call__(*args, **kwargs) Alias for imgaug.augmenters.meta.Augmenter.augment().
```

```python
augment([return_batch, hooks]) Augment data.
```

```python
augment_batch(batch[, hooks]) Augment a single batch.
```

```python
augment_batches(batches[, hooks, background]) Augment multiple batches.
```

```python
augment_bounding_boxes(bounding_boxes_on_images) Augment bounding boxes.
```

```python
augment_heatmaps(heatmaps[, parents, hooks]) Augment a heatmap.
```

```python
augment_image(image[, hooks]) Augment a single image.
```

```python
augment_images(images[, parents, hooks]) Augment multiple images.
```

```python
augment_keypoints(keypoints_on_images[, ...]) Augment image keypoints.
```

```python
augment_line_strings(line_strings_on_images) Augment line strings.
```

```python
augment_polygons(polygons_on_images[, ...]) Augment polygons.
```

```python
augment_segmentation_maps(segmaps[, ...]) Augment segmentation maps.
```

```python
copy() Create a shallow copy of this Augmenter instance.
```

```python
copy_random_state(source[, recursive, ...]) Copy the random states from a source augmenter sequence.
```

```python
copy_random_state_(source[, recursive, ...]) Copy the random states from a source augmenter sequence (inplace).
```

```python
deepcopy() Create a deep copy of this Augmenter instance.
```

```python
draw_grid(images, rows, cols) Apply this augmenter to the given images and return a grid image of the results.
```

```python
find_augmenters(func[, parents, flat]) Find augmenters that match a condition.
```

```python
find_augmenters_by_name(name[, regex, flat]) Find augmenter(s) by name.
```

```python
find_augmenters_by_names(names[, regex, flat]) Find augmenter(s) by names.
```

```python
get_all_children([flat]) Returns all children of this augmenter as a list.
```

```python
get_children_lists() Get a list of lists of children of this augmenter.
```

```python
localize_random_state([recursive]) Converts global random states to local ones.
```

```python
localize_random_state_(recursive) Converts global random states to local ones.
```

```python
pool([processes, maxtasksperchild, seed]) Create a pool used for multicore augmentation from this augmenter.
```

Continued on next page
Table 48 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`remove_augmenters(func[, copy,</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>noop_if_topmost])`</td>
<td></td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**augment**(return_batch=False, hooks=None, **kwargs)

Augment data.

This method is a wrapper around `imgaug.augmentables.batches.UnnormalizedBatch` and `augment_batch()`. Hence, it supports the same datatypes as `UnnormalizedBatch`.

If return_batch was not set to True, the method will return a tuple of augmentables. It will return the same types of augmentables (only augmented) as input into the method. This behaviour is partly specific to the python version:

- **In python 3.6+ (if return_batch=False):**
  - Three or more augmentables may be used as input.
  - None of the provided named arguments has to be image or images.
  - The return order matches the order of the named arguments, e.g. b, d, c = augment(b=b, d=d, c=c).

- **In python <3.6 (if return_batch=False):**
  - One or two augmentables may be used as input arguments, not more than that.
  - One the input arguments has to be image or images.
  - The augmented images are always* returned first, independent of the input argument order. E.g. a, b = augment(b=b, images=a).

If return_batch was not set to False, an instance of UnnormalizedBatch will be returned. The output is the same for all python version and any number or combination of augmentables may be provided.

All augmentables must be provided as named arguments. E.g. `augment(<array>)` will crash, but `augment(images=<array>)` will work.

**Parameters**

- **image** *(None)* – or (H,W,C) ndarray or (H,W) ndarray, optional The image to augment. Only this or images can be set, not both. If return_batch is False and the python version is below 3.6, either this or images _must_ be provided.

- **images** *(None or (N,H,W) ndarray or (N,W,H) ndarray or iterable of (H,W,C) ndarray or iterable of (H,W) ndarray, optional)* – The images to augment. Only this or image can be set, not both. If return_batch is False and the python version is below 3.6, either this or image _must_ be provided.
• **heatmaps** *(None or (N,H,W,C) ndarray or imgaug.augmentables.heatmaps.HeatmapsOnImage or iterable of (H,W,C) ndarray or iterable of imgaug.augmentables.heatmaps.HeatmapsOnImage, optional)* – The heatmaps to augment. If anything else than HeatmapsOnImage, then the number of heatmaps must match the number of images provided via parameter `images`. The number is contained either in `N` or the first iterable’s size.

• **segmentation_maps** *(None or (N,H,W) ndarray or imgaug.augmentables.segmaps.SegmentationMapOnImage or iterable of (H,W) ndarray or iterable of imgaug.augmentables.segmaps.SegmentationMapOnImage, optional)* – The segmentation maps to augment. If anything else than SegmentationMapOnImage, then the number of segmaps must match the number of images provided via parameter `images`. The number is contained either in `N` or the first iterable’s size.

• **keypoints** *(None or list of (N,K,2) ndarray or tuple of number or imgaug.augmentables.kps.Keypoint or iterable of (K,2) ndarray or iterable of tuple of number or iterable of imgaug.augmentables.kps.Keypoint or iterable of imgaug.augmentables.kps.KeypointOnImage or iterable of iterable of tuple of number or iterable of iterable of imgaug.augmentables.kps.Keypoint, optional)* – The keypoints to augment. If a tuple (or iterable(s) of tuple), then interpreted as (x,y) coordinates and must hence contain two numbers. A single tuple represents a single coordinate on one image, an iterable of tuples the coordinates on one image and an iterable of iterable of tuples the coordinates on several images. Analogous if Keypoint objects are used instead of tuples. If an ndarray, then `N` denotes the number of images and `K` the number of keypoints on each image. If anything else than KeypointsOnImage is provided, then the number of keypoint groups must match the number of images provided via parameter images. The number is contained e.g. in `N` or in case of “iterable of iterable of tuples” in the first iterable’s size.

• **bounding_boxes** *(None or (N,B,4) ndarray or tuple of number or imgaug.augmentables.bbs.BoundingBox or imgaug.augmentables.bbs.BoundingBoxesOnImage or iterable of (B,4) ndarray or iterable of tuple of number or iterable of imgaug.augmentables.bbs.BoundingBox or iterable of imgaug.augmentables.bbs.BoundingBoxesOnImage or iterable of iterable of tuple of number or iterable of iterable imgaug.augmentables.bbs.BoundingBox, optional)* – The bounding boxes to augment. This is analogous to the keypoints parameter. However, each tuple – and also the last index in case of arrays – has size 4, denoting the bounding box coordinates x1, y1, x2 and y2.

• **polygons** *(None or (N,#polys,#points,2) ndarray or imgaug.augmentables.polys.Polygon or imgaug.augmentables.polys.PolygonsOnImage or iterable of (#polys,#points,2) ndarray or iterable of iterable of tuple of number or iterable of imgaug.augmentables.polys.Polygon or iterable of imgaug.augmentables.polys.PolygonsOnImage or iterable of iterable of tuple of number or iterable of iterable of imgaug.augmentables.polys.Polygon, optional)* – The polygons to augment. This is similar to the keypoints parameter. However, each polygon may be made up of several (x,y) coordinates (three or more are required for valid polygons). The following datatypes will be interpreted as a single polygon on a single image:

  - imgaug.augmentables.polys.Polygon
  - iterable of tuple of number
The following datatypes will be interpreted as multiple polygons on a single image:

- `imgaug.augmentables.polys.PolygonsOnImage`
- iterable of `imgaug.augmentables.polys.Polygon`
- iterable of iterable of tuple of number
- iterable of iterable of `imgaug.augmentables.kps.Keypoint`
- iterable of iterable of `imgaug.augmentables.polys.Polygon`

The following datatypes will be interpreted as multiple polygons on multiple images:

- `(N,#polys,#points,2)` ndarray
- iterable of `(polys,#points,2)` ndarray
- iterable of iterable of `(points,2)` ndarray
- iterable of iterable of iterable of tuple of number
- iterable of iterable of iterable of `imgaug.augmentables.kps.Keypoint`

- `line_strings` (None or `(N,#lines,#points,2)` ndarray or `imgaug.augmentables.lines.LineString` or `imgaug.augmentables.lines.LineStringOnImage` or iterable of `(polys,#points,2)` ndarray or iterable of tuple of number or iterable of `imgaug.augmentables.kps.Keypoint` or iterable of iterable of `imgaug.augmentables.lines.LineString` or iterable of iterable of `imgaug.augmentables.lines.LineStringOnImage` or iterable of iterable of `(points,2)` ndarray or iterable of iterable of tuple of number or iterable of iterable of `imgaug.augmentables.kps.Keypoint`, optional) – The line strings to augment. See `polygons`, which behaves similarly.

- `return_batch` (bool, optional) – Whether to return an instance of `imgaug.augmentables.batches.UnnormalizedBatch`. If the python version is below 3.6 and more than two augmentables were provided (e.g. images, keypoints and polygons), then this must be set to True. Otherwise an error will be raised.

- `hooks` (None or `imgaug.imgaug.HooksImages`, optional) – Hooks object to dynamically interfere with the augmentation process.

Returns If `return_batch` was set to True, a instance of `UnnormalizedBatch` will be returned. If `return_batch` was set to False, a tuple of augmentables will be returned, e.g. `(augmented images, augmented keypoints)`. The datatypes match the input datatypes of the corresponding named arguments. In python <3.6, augmented images are always the first entry in the returned tuple. In python 3.6+ the order matches the order of the named arguments.

Return type tuple or `imgaug.augmentables.batches.UnnormalizedBatch`
This creates a single image and a set of two keypoints on it, then augments both by applying a random rotation between -25deg and +25deg. The sampled rotation value is automatically aligned between image and keypoints. Note that in python <3.6, augmented images will always be returned first, independent of the order of the named input arguments. So `keypoints_aug, images_aug = aug.augment(keypoints=keypoints, image=image)` would _not_ work (except in python 3.6+).

This creates two images of size 64x64 and 32x32, two sets of keypoints (each containing two keypoints) and two sets of bounding boxes (the first containing one bounding box, the second two bounding boxes). These augmentables are then augmented by applying random rotations between -25deg and +25deg to them. The rotation values are sampled by image and aligned between all augmentables on the same image. The method finally returns an instance of `UnnormalizedBatch` from which the augmented data can be retrieved via `batch_aug.images_aug`, `batch_aug.keypoints_aug`, and `batch_aug.bounding_boxes_aug`. In python 3.6+, `return_batch` can be kept at `False` and the augmented data can be retrieved as `images_aug, keypoints_aug, bbs_aug = augment(...)`. 

**augment_batch** *(batch, hooks=None)*

Augment a single batch.

**Parameters**

- **batch** *(imgaug.augmentables.batches.Batch or imgaug.augmentables.batches.UnnormalizedBatch)*
  
  A single batch to augment.

- **hooks** *(None or imgaug.HooksImages, optional)* – HooksImages object to dynamically interfere with the augmentation process.

**Returns** Augmented batch.

**Return type** imgaug.augmentables.batches.Batch or imgaug.augmentables.batches.UnnormalizedBatch

**augment_batches** *(batch, hooks=None, background=False)*

Augment multiple batches.

In contrast to other augment functions, this function _yields_ batches instead of just returning a full list. This is more suited for most training loops. It also supports augmentation on multiple cpu cores, activated via the `background` flag.
Parameters

- **batches** *(imgaug.augmentables.batches.Batch or imgaug.augmentables.batches.UnnormalizedBatch or iterable of imgaug.augmentables.batches.Batch or iterable of imgaug.augmentables.batches.UnnormalizedBatch)* – A single batch or a list of batches to augment.

- **hooks** *(None or imgaug.HooksImages, optional)* – HooksImages object to dynamically interfere with the augmentation process.

- **background** *(bool, optional)* – Whether to augment the batches in background processes. If true, hooks can currently not be used as that would require pickling functions. Note that multicore augmentation distributes the batches onto different CPU cores. It does not split the data within batches. It is therefore not sensible to use `background=True` for a single batch. Note also that multicore augmentation needs some time to start. It is therefore not recommended to use it for very few batches.


**augment_bounding_boxes** *(bounding_boxes_on_images, hooks=None)*

Augment bounding boxes.

This is the corresponding function to `augment_keypoints()`, just for bounding boxes. Usually you will want to call `augment_images()` with a list of images, e.g. `augment_images([A, B, C])` and then `augment_bounding_boxes()` with the corresponding list of bounding boxes on these images, e.g. `augment_bounding_boxes([Abb, Bbb, Cbb])`, where `Abb` are the bounding boxes on image `A`.

Make sure to first convert the augmenter(s) to deterministic states before augmenting images and their corresponding bounding boxes, e.g. by

```python
>>> A = B = C = np.ones((10, 10), dtype=np.uint8)
>>> Abb = Bbb = Cbb = ia.BoundingBoxesOnImage([ia.BoundingBox(1, 1, 9, 9)], shape=(10, 10))
>>> seq = iaa.Fliplr(0.5)
>>> seq_det = seq.to_deterministic()
>>> imgs_aug = seq_det.augment_images([A, B, C])
>>> bbs_aug = seq_det.augment_bounding_boxes([Abb, Bbb, Cbb])
```

Otherwise, different random values will be sampled for the image and bounding box augmentations, resulting in different augmentations (e.g. images might be rotated by 30deg and bounding boxes by -10deg). Also make sure to call `to_deterministic()` again for each new batch, otherwise you would augment all batches in the same way.

Parameters

- **bounding_boxes_on_images** *(imgaug.BoundingBoxesOnImage or list of imgaug.BoundingBoxesOnImage)* – The bounding boxes to augment. Expected is an instance of imgaug.BoundingBoxesOnImage or a list of imgaug.BoundingBoxesOnImage objects, with each such object containing the bounding boxes of a single image.

- **hooks** *(None or imgaug.HooksKeypoints, optional)* – HooksKeypoints object to dynamically interfere with the augmentation process.

Returns `result` – Augmented bounding boxes.

Return type `imgaug.BoundingBoxesOnImage` or list of `imgaug.BoundingBoxesOnImage`
**augment_heatmaps** *(heatmaps, parents=None, hooks=None)*

Augment a heatmap.

**Parameters**

- **heatmaps** *(imgaug.HeatmapsOnImage or list of imgaug.HeatmapsOnImage)* – Heatmap(s) to augment. Either a single heatmap or a list of heatmaps.

- **parents** *(None or list of imgaug.augmenters.meta.Augmenter, optional)* – Parent augmenters that have previously been called before the call to this function. Usually you can leave this parameter as None. It is set automatically for child augmenters.

- **hooks** *(None or imgaug.HooksHeatmaps, optional)* – HooksHeatmaps object to dynamically interfere with the augmentation process.

**Returns** heatmap_result – Corresponding augmented heatmap(s).

**Return type** imgaug.HeatmapsOnImage or list of imgaug.HeatmapsOnImage

**augment_image** *(image, hooks=None)*

Augment a single image.

**Parameters**

- **image** *((H,W,C) ndarray or (H,W) ndarray)* – The image to augment. Channel-axis is optional, but expected to be the last axis if present. In most cases, this array should be of dtype `uint8`, which is supported by all augmenters. Support for other dtypes varies by augmenter – see the respective augmenter-specific documentation for more details.

- **hooks** *(None or imgaug.HooksImages, optional)* – HooksImages object to dynamically interfere with the augmentation process.

**Returns** img – The corresponding augmented image.

**Return type** ndarray

**augment_images** *(images, parents=None, hooks=None)*

Augment multiple images.

**Parameters**

- **images** *((N,H,W,C) ndarray or (N,H,W) ndarray or list of (H,W,C) ndarray or list of (H,W) ndarray)* – Images to augment. The input can be a list of numpy arrays or a single array. Each array is expected to have shape `(H, W, C)` or `(H, W)`, where `H` is the height, `W` is the width and `C` are the channels. Number of channels may differ between images. If a list is chosen, height and width may differ per between images. In most cases, this array (or these arrays) should be of dtype `uint8`, which is supported by all augmenters. Support for other dtypes varies by augmenter – see the respective augmenter-specific documentation for more details.

- **parents** *(None or list of imgaug.augmenters.Augmenter, optional)* – Parent augmenters that have previously been called before the call to this function. Usually you can leave this parameter as None. It is set automatically for child augmenters.

- **hooks** *(None or imgaug.HooksImages, optional)* – HooksImages object to dynamically interfere with the augmentation process.

**Returns** images_result – Corresponding augmented images.

**Return type** ndarray or list

**augment_keypoints** *(keypoints_on_images, parents=None, hooks=None)*

Augment image keypoints.
This is the corresponding function to `augment_images()`, just for keypoints/landmarks (i.e. coordinates on the image). Usually you will want to call `augment_images()` with a list of images, e.g. `augment_images([A, B, C])` and then `augment_keypoints()` with the corresponding list of keypoints on these images, e.g. `augment_keypoints([Ak, Bk, Ck])`, where Ak are the keypoints on image A.

Make sure to first convert the augmenter(s) to deterministic states before augmenting images and their corresponding keypoints, e.g. by

```python
>>> A = B = C = np.zeros((10, 10), dtype=np.uint8)
>>> Ak = Bk = Ck = ia.KeypointsOnImage([ia.Keypoint(2, 2)], (10, 10))
>>> seq = iaa.Fliplr(0.5)
>>> seq_det = seq.to_deterministic()
>>> imgs_aug = seq_det.augment_images([A, B, C])
>>> kps_aug = seq_det.augment_keypoints([Ak, Bk, Ck])
```

Otherwise, different random values will be sampled for the image and keypoint augmentations, resulting in different augmentations (e.g. images might be rotated by $30\text{deg}$ and keypoints by $-10\text{deg}$). Also make sure to call `to_deterministic()` again for each new batch, otherwise you would augment all batches in the same way.

**Parameters**

- **keypoints_on_images** (imgaug.KeypointsOnImage or list of imgaug.KeypointsOnImage)
  
  The keypoints/landmarks to augment. Expected is an instance of imgaug.KeypointsOnImage or a list of imgaug.KeypointsOnImage objects, with each such object containing the keypoints of a single image.

- **parents** (None or list of imgaug.augmenters.meta.Augmenter, optional)
  
  Parent augmenters that have previously been called before the call to this function. Usually you can leave this parameter as None. It is set automatically for child augmenters.

- **hooks** (None or imgaug.HooksKeypoints, optional)
  
  HooksKeypoints object to dynamically interfere with the augmentation process.

**Returns** `keypoints_on_images_result` – Augmented keypoints.

**Return type** imgaug.KeypointsOnImage or list of imgaug.KeypointsOnImage

---

**augment_line_strings** (line_strings_on_images, parents=None, hooks=None)

Augment line strings.

This is the corresponding function to `augment_keypoints()`, just for line strings. Usually you will want to call `augment_images()` with a list of images, e.g. `augment_images([A, B, C])` and then `augment_line_strings()` with the corresponding list of line strings on these images, e.g. `augment_line_strings([A_line, B_line, C_line])`, where A_line are the line strings on image A.

Make sure to first convert the augmenter(s) to deterministic states before augmenting images and their corresponding line strings, e.g. by

```python
>>> import imgaug as ia
>>> import imgaug.augmenters as iaa
>>> A = B = C = np.ones((10, 10), dtype=np.uint8)
>>> A_line = B_line = C_line = ia.LineStringsOnImage([ia.LineString([(0, 0), (1, 0), (1, 1), (0, 1)])], shape=(10, 10))
>>> seq = iaa.Fliplr(0.5)
>>> seq_det = seq.to_deterministic()
```
Otherwise, different random values will be sampled for the image and line string augmentations, resulting in different augmentations (e.g. images might be rotated by $30^\circ$ and line strings by $-10^\circ$). Also make sure to call `to_deterministic()` again for each new batch, otherwise you would augment all batches in the same way.

**Parameters**

- **line_strings_on_images** *(imgaug.augmentables.lines.LineStringsOnImage or list of imgaug.augmentables.lines.LineStringsOnImage)* – The line strings to augment. Expected is an instance of LineStringsOnImage or a list of LineStringsOnImage objects, with each such object containing the line strings of a single image.

- **parents** *(None or list of imgaug.augmenters.meta.Augmenter, optional)* – Parent augmenters that have previously been called before the call to this function. Usually you can leave this parameter as None. It is set automatically for child augmenters.

- **hooks** *(None or imgaug.HooksKeypoints, optional)* – HooksKeypoints object to dynamically interfere with the augmentation process.

**Returns** Augmented line strings.

**Return type** `imgaug.augmentables.lines.LineStringsOnImage` or list of `imgaug.augmentables.lines.LineStringsOnImage`

### augment_polygons(polygons_on_images, parents=None, hooks=None)

Augment polygons.

This is the corresponding function to `augment_keypoints()`, just for polygons. Usually you will want to call `augment_images()` with a list of images, e.g. `augment_images([A, B, C])` and then `augment_polygons()` with the corresponding list of polygons on these images, e.g. `augment_polygons([A_poly, B_poly, C_poly])`, where `A_poly` are the polygons on image `A`.

Make sure to first convert the augmenter(s) to deterministic states before augmenting images and their corresponding polygons, e.g. by

```python
>>> import imgaug as ia
>>> import imgaug.augmenters as iaa
>>> A = B = C = np.ones((10, 10), dtype=np.uint8)
>>> Apoly = Bpoly = Cpoly = ia.PolygonsOnImage([ia.Polygon([(0, 0), (1, 0), (1, 1), (0, 1)])], shape=(10, 10))
>>> seq = iaa.Fliplr(0.5)
>>> seq_det = seq.to_deterministic()
>>> imgs_aug = seq_det.augment_images([A, B, C])
>>> polys_aug = seq_det.augment_polygons([Apoly, Bpoly, Cpoly])
```

Otherwise, different random values will be sampled for the image and polygon augmentations, resulting in different augmentations (e.g. images might be rotated by $30^\circ$ and polygons by $-10^\circ$). Also make sure to call `to_deterministic()` again for each new batch, otherwise you would augment all batches in the same way.

**Parameters**

- **polygons_on_images** *(imgaug.PolygonsOnImage or list of imgaug.PolygonsOnImage)* – The polygons to augment. Expected is an instance of imgaug.PolygonsOnImage or a list of PolygonsOnImage objects, with each such object containing the polygons of a single image.

- **parents** *(None or list of imgaug.augmenters.meta.Augmenter, optional)* – Parent augmenters that have previously been called before the call to this function. Usually you can leave this parameter as None. It is set automatically for child augmenters.

- **hooks** *(None or imgaug.HooksKeypoints, optional)* – HooksKeypoints object to dynamically interfere with the augmentation process.

**Returns** Augmented line strings.

**Return type** `imgaug.augmentables.lines.LineStringsOnImage` or list of `imgaug.augmentables.lines.LineStringsOnImage`
of imgaug.PolygonsOnImage objects, with each such object containing the polygons of a single image.

- **parents** *(None or list of imgaug.augmenters.meta.Augmenter, optional)* – Parent augmenters that have previously been called before the call to this function. Usually you can leave this parameter as None. It is set automatically for child augmenters.

- **hooks** *(None or imgaug.HooksKeypoints, optional)* – HooksKeypoints object to dynamically interfere with the augmentation process.

**Returns**

- **result** – Augmented polygons.

**Return type**

imgaug.PolygonsOnImage or list of imgaug.PolygonsOnImage

### augment_segmentation_maps

#### Parameters

- **segmaps** *(imgaug.SegmentationMapOnImage or list of imgaug.SegmentationMapOnImage)* – Segmentation map(s) to augment. Either a single heatmap or a list of segmentation maps.

- **parents** *(None or list of imgaug.augmenters.meta.Augmenter, optional)* – Parent augmenters that have previously been called before the call to this function. Usually you can leave this parameter as None. It is set automatically for child augmenters.

- **hooks** *(None or imgaug.HooksHeatmaps, optional)* – HooksHeatmaps object to dynamically interfere with the augmentation process.

**Returns**

- **segmaps_aug** – Corresponding augmented segmentation map(s).

**Return type**

imgaug.SegmentationMapOnImage or list of imgaug.SegmentationMapOnImage

### copy

Create a shallow copy of this Augmenter instance.

**Returns**

Shallow copy of this Augmenter instance.

**Return type**

imgaug.augmenters.meta.Augmenter

### copy_random_state

#### Parameters


**Returns**

- **aug** – Copy of the augmenter(s) with the same random state(s) as in the source augmenter(s).

**Return type**

imgaug.augmenters.meta.Augmenter
**copy_random_state** (source, recursive=True, matching='position', matching_tolerant=True, copy_determinism=False)

Copy the random states from a source augmenter sequence (inplace).

**Parameters**

- **source** (imgaug.augmenters.meta.Augmenter) – The source augmenter from where to copy the random_state(s). May have children (e.g. a Sequential). May not use the global random state. This is used by default by all augmenters. Call imgaug.augmenters.meta.Augmenter.localize_random_state_() once on the source to localize all random states.

- **recursive** (bool, optional) – Whether to copy the random states of the source augmenter and all of its children (True) or just the source augmenter (False).

- **matching** ('position', 'name'), optional) – Defines the matching mode to use during recursive copy. This is used to associate source augmenters with target augmenters. If position then the target and source sequences of augmenters are turned into flattened lists and are associated based on their list indices. If name then the target and source augmenters are matched based on their names (i.e. augmenter.name).

- **matching_tolerant** (bool, optional) – Whether to use tolerant matching between source and target augmenters. If set to False: Name matching will raise an exception for any target augmenter which’s name does not appear among the source augmeters. Position matching will raise an exception if source and target augmenter have an unequal number of children.

- **copy_determinism** (bool, optional) – Whether to copy the deterministic flags from source to target augmenters too.

**Returns** self – Returns itself (after random state copy).

**Return type** imgaug.augmenters.meta.Augmenter

**deepcopy** ()

Create a deep copy of this Augmenter instance.

**Returns** Deep copy of this Augmenter instance.

**Return type** imgaug.augmenters.meta.Augmenter

**draw_grid** (images, rows, cols)

Apply this augmenter to the given images and return a grid image of the results. Each cell in the grid contains a single augmented variation of an input image.

If multiple images are provided, the row count is multiplied by the number of images and each image gets its own row. E.g. for images = [A, B], rows=2, cols=3:

<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

for images = [A], rows=2, cols=3:

<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

**Parameters**
• **images** ([N,H,W,3] ndarray or [H,W,3] ndarray or [H,W] ndarray or list of [H,W,3] ndarray or list of [H,W] ndarray) – List of images of which to show the augmented versions. If a list, then each element is expected to have shape (H, W) or (H, W, 3). If a single array, then it is expected to have shape (N, H, W, 3) or (H, W, 3) or (H, W).

• **rows** (int) – Number of rows in the grid. If N input images are given, this value will automatically be multiplied by N to create rows for each image.

• **cols** (int) – Number of columns in the grid.

**Returns** grid – The generated grid image with augmented versions of the input images. Here, Hg and Wg reference the output size of the grid, and not the sizes of the input images.

**Return type** (Hg, Wg, 3) ndarray

### find_augmenters

**find_augmenters** (**func**, **parents**=**None**, **flat**=**True**)

Find augmenters that match a condition. This function will compare this augmenter and all of its children with a condition. The condition is a lambda function.

**Parameters**

• **func** (callable) – A function that receives an Augmenter instance and a list of parent Augmenter instances and must return True, if that augmenter is valid match.

• **parents** (None or list of imgaug.augmenters.meta.Augmenter, optional) – List of parent augmenters. Intended for nested calls and can usually be left as None.

• **flat** (bool, optional) – Whether to return the result as a flat list (True) or a nested list (False). In the latter case, the nesting matches each augmenters position among the children.

**Returns** augmenters – Nested list if flat was set to False. Flat list if flat was set to True.

**Return type** list of imgaug.augmenters.meta.Augmenter

### Examples

```python
>>> aug = iaa.Sequential(
    
    iaa.Fliplr(0.5, name="fliplr"),
    iaa.Flipud(0.5, name="flipud")
)

>>> print(aug.find_augmenters(lambda a, parents: a.name == "fliplr"))
```

This will return the first child augmenter (Fliplr instance).

### find_augmenters_by_name

**find_augmenters_by_name** (**name**, **regex**=**False**, **flat**=**True**)

Find augmenter(s) by name.

**Parameters**

• **name** (str) – Name of the augmenter(s) to search for.

• **regex** (bool, optional) – Whether name parameter is a regular expression.

• **flat** (bool, optional) – See **imgaug.augmenters.meta.Augmenter. find_augmenters()**.

**Returns** augmenters – Nested list if flat was set to False. Flat list if flat was set to True.

**Return type** list of imgaug.augmenters.meta.Augmenter

### find_augmenters_by_names

**find_augmenters_by_names** (**names**, **regex**=**False**, **flat**=**True**)

Find augmenter(s) by names.
Parameters

- **names** *(list of str)* – Names of the augmenter(s) to search for.
- **regex** *(bool, optional)* – Whether `names` is a list of regular expressions.
- **flat** *(boolean, optional)* – See `imgaug.augmenters.meta.Augmenter.find_augmenters()`.

**Returns**

- **augmenters** – Nested list if flat was set to False. Flat list if flat was set to True.

**Return type**

- list of `imgaug.augmenters.meta.Augmenter`

### `get_all_children` *(flat=False)*

Returns all children of this augmenter as a list.

If the augmenter has no children, the returned list is empty.

**Parameters**

- **flat** *(bool)* – If set to True, the returned list will be flat.

**Returns**

- **result** – The children as a nested or flat list.

**Return type**

- list of `imgaug.augmenters.meta.Augmenter`

### `get_children_lists()`

Get a list of lists of children of this augmenter.

For most augmenters, the result will be a single empty list. For augmenters with children it will often be a list with one sublist containing all children. In some cases the augmenter will contain multiple distinct lists of children, e.g. an if-list and an else-list. This will lead to a result consisting of a single list with multiple sublists, each representing the respective sublist of children.

E.g. for an if/else-augmenter that executes the children A1, A2 if a condition is met and otherwise executes the children B1, B2, B3 the result will be `[[A1, A2], [B1, B2, B3]]`.

IMPORTANT: While the topmost list may be newly created, each of the sublist must be editable inplace resulting in a changed children list of the augmenter. E.g. if an Augmenter IfElse(condition, [A1, A2], [B1, B2, B3]) returns `[[A1, A2], [B1, B2, B3]]` for a call to `imgaug.augmenters.meta.Augmenter.get_children_lists()` and A2 is removed inplace from [A1, A2], then the children lists of IfElse(...) must also change to [A1], [B1, B2, B3]. This is used in `imgaug.augmenters.meta.Augmenter.remove_augmenters_inplace()`.

**Returns**

- **children** – One or more lists of child augmenter. Can also be a single empty list.

**Return type**

- list of list of `imgaug.augmenters.meta.Augmenter`

### `get_parameters()`

Converts global random states to local ones. See `Augmenter.localize_random_state()` for more details.

**Parameters**


**Returns**

- **aug** – Returns copy of augmenter and children, with localized random states.

**Return type**

- `imgaug.augmenters.meta.Augmenter`

### `localize_random_state_` *(recursive=True)*

Converts global random states to local ones.
A global random state exists exactly once. Many augmenters can point to it (and thereby use it to sample random numbers). Local random states usually exist for exactly one augmenter and are saved within that augmenter.

Usually there is no need to change global into local random states. The only noteworthy exceptions are

- whenever you want to use determinism (so that the global random state is not accidentally reverted)
- whenever you want to copy random states from one augmenter to another. (Copying the global random state doesn’t help very much. If you copy the state from A to B, then execute A and then B, B’s (global) random state has already changed because of A’s sampling.)

The case of determinism is handled automatically by `imgaug.augmenters.meta.Augmenter.to_deterministic()`. Only when you copy random states (via `imgaug.augmenters.meta.Augmenter.copy_random_state()`, you need to call this function first.

**Parameters**

- `recursive (bool, optional)` – Whether to localize the random states of children too.

**Returns**

- `self` – Returns itself (with localized random states).

**Return type**

*imgaug.augmenters.meta.Augmenter*

---

### pool

Create a pool used for multicore augmentation from this augmenter.

**Parameters**

- `processes (None or int, optional)` – Same as for `imgaug.multicore.Pool.__init__()`. The number of background workers, similar to the same parameter in multiprocessing.Pool. If `None`, the number of the machine’s CPU cores will be used (this counts hyperthreads as CPU cores). If this is set to a negative value `p`, then `P - abs(p)` will be used, where `P` is the number of CPU cores. E.g. `-1` would use all cores except one (this is useful to e.g. reserve one core to feed batches to the GPU).

- `maxtasksperchild (None or int, optional)` – Same as for `imgaug.multicore.Pool.__init__()`. The number of tasks done per worker process before the process is killed and restarted, similar to the same parameter in multiprocessing.Pool. If `None`, worker processes will not be automatically restarted.

- `seed (None or int, optional)` – Same as for `imgaug.multicore.Pool.__init__()`. The seed to use for child processes. If `None`, a random seed will be used.

**Returns**

- `Pool for multicore augmentation.`

**Return type**

*imgaug.multicore.Pool*

---

### Examples

```python
>>> import imgaug as ia
>>> from imgaug import augmenters as iaa
>>> import numpy as np
>>> aug = iaa.Add(1)
>>> images = np.zeros((16, 128, 128, 3), dtype=np.uint8)
>>> batches = [ia.Batch(images=np.copy(images)) for _ in range(100)]
>>> with aug.pool(processes=-1, seed=2) as pool:
...     batches_aug = pool.map_batches(batches, chunksize=8)
>>> print(np.sum(batches_aug[0].images_aug[0]))
49152
```
Creates 100 batches of empty images. Each batch contains 16 images of size 128x128. The batches are then augmented on all CPU cores except one (processes=-1). After augmentation, the sum of pixel values from the first augmented image is printed.

```python
>>> import imgaug as ia
>>> from imgaug import augmenters as iaa
>>> import numpy as np

>>> aug = iaa.Add(1)
>>> images = np.zeros((16, 128, 128, 3), dtype=np.uint8)

>>> def generate_batches():
...     for _ in range(100):
...         yield ia.Batch(images=np.copy(images))

>>> with aug.pool(processes=-1, seed=2) as pool:
...     batches_aug = pool.imap_batches(generate_batches(), chunksize=8)
...     batch_aug = next(batches_aug)

>>> print(np.sum(batch_aug.images_aug[0]))
49152
```

Same as above. This time, a generator is used to generate batches of images. Again, the first augmented image’s sum of pixels is printed.

```python
remove_augmenters (func, copy=True, noop_if_topmost=True)
```
Remove this augmenter or its children that match a condition.

#### Parameters

- **func** *(callable)* – Condition to match per augmenter. The function must expect the augmenter itself and a list of parent augmenters and returns True if that augmenter is to be removed, or False otherwise. E.g. `lambda a, parents: a.name == "fliplr" and len(parents) == 1` removes an augmenter with name “fliplr” if it is the direct child of the augmenter upon which `remove_augmenters()` was initially called.
- **copy** *(bool, optional)* – Whether to copy this augmenter and all if its children before removing. If False, removal is performed in-place.
- **noop_if_topmost** *(bool, optional)* – If True and the condition (lambda function) leads to the removal of the topmost augmenter (the one this function is called on initially), then that topmost augmenter will be replaced by a Noop instance (i.e. an object that will still offer `augment_images()`, but does not change images). If False, None will be returned in these cases. This can only be False if copy is set to True.

#### Returns

- **aug** – This augmenter after the removal was performed. Is None iff condition was matched for the topmost augmenter, copy was set to True and `noop_if_topmost` was set to False.

#### Return type

`imgaug.augmenters.meta.Augmenter` or None

#### Examples

```python
>>> seq = iaa.Sequential([
...     iaa.Fliplr(0.5, name="fliplr"),
...     iaa.Flipud(0.5, name="flipud"),
... ])

>>> seq = seq.remove_augmenters(lambda a, parents: a.name == "fliplr")
```

This removes the augmenter Fliplr from the Sequential object’s children.
remove_augmenters_inplace(func, parents=None)

Remove in-place children of this augmenter that match a condition.

This is functionally identical to remove_augmenters() with copy=False, except that it does not affect the topmost augmenter (the one on which this function is initially called on).

Parameters

- **func**: (callable) – See imgaug.augmenters.meta.Augmenter.remove_augmenters().
- **parents** (None or list of imgaug.augmenters.meta.Augmenter, optional) – List of parent Augmenter instances that lead to this Augmenter. If None, an empty list will be used. This parameter can usually be left empty and will be set automatically for children.

Examples

```python
>>> seq = iaa.Sequential([
>>>     iaa.Fliplr(0.5, name="fliplr"),
>>>     iaa.Flipud(0.5, name="flipud"),
>>> ])
>>> seq.remove_augmenters_inplace(lambda a, parents: a.name == "fliplr")
```

This removes the augmenter Fliplr from the Sequential object’s children.

reseed(random_state=None, deterministic_too=False)

Reseed this augmenter and all of its children (if it has any).

This function is useful, when augmentations are run in the background (i.e. on multiple cores). It should be called before sending this Augmenter object to a background worker or once within each worker with different seeds (i.e., if \(N\) workers are used, the function should be called \(N\) times). Otherwise, all background workers will use the same seeds and therefore apply the same augmentations.

If this augmenter or any child augmenter had a random state that pointed to the global random state, it will automatically be replaced with a local random state. This is similar to what imgaug.augmenters.meta.Augmenter.localize_random_state() does.

Parameters

- **random_state** (None or int or numpy.RandomState, optional) – A RandomState that is used to sample seeds per augmenter. If int, the parameter will be used as a seed for a new RandomState. If None, a new RandomState will automatically be created.
- **deterministic_too** (bool, optional) – Whether to also change the seed of an augmenter \(A\), if \(A\) is deterministic. This is the case both when this augmenter object is \(A\) or one of its children is \(A\).

show_grid(images, rows, cols)

Apply this augmenter to the given images and show/plot the results as a grid of images.

If multiple images are provided, the row count is multiplied by the number of images and each image gets its own row. E.g. for images = [A, B], rows=2, cols=3:

```
A A A
B B B
A A A
B B B
```

for images = [A], rows=2, cols=3:

```
A
A
```
Parameters

- **images** ((N,H,W,3) ndarray or (H,W,3) ndarray or (H,W) ndarray or list of (H,W,3) ndarray or list of (H,W) ndarray) – List of images of which to show the augmented versions. If a list, then each element is expected to have shape (H, W) or (H, W, 3). If a single array, then it is expected to have shape (N, H, W, 3) or (H, W, 3) or (H, W).

- **rows** (int) – Number of rows in the grid. If N input images are given, this value will automatically be multiplied by N to create rows for each image.

- **cols** (int) – Number of columns in the grid.

**to_deterministic** *(n=None)*

Converts this augmenter from a stochastic to a deterministic one.

A stochastic augmenter samples new values for each parameter per image. Feed a new batch of images into the augmenter and you will get a new set of transformations. A deterministic augmenter also samples new values for each parameter per image, but starts each batch with the same RandomState (i.e. seed). Feed two batches of images into the augmenter and you get the same transformations both times (same number of images assumed; some augmenter’s results are also dependend on image height, width and channel count).

Using determinism is useful for keypoint augmentation, as you will usually want to augment images and their corresponding keypoints in the same way (e.g. if an image is rotated by 30deg, then also rotate its keypoints by 30deg).

**Parameters** **n** *(None or int, optional)* – Number of deterministic augmenters to return. If None then only one Augmenter object will be returned. If 1 or higher, then a list containing n Augmenter objects will be returned.

**Returns** **det** – A single Augmenter object if n was None, otherwise a list of Augmenter objects (even if n was 1).

**Return type** `imgaug.augmenters.meta.Augmenter` or list of `imgaug.augmenters.meta.Augmenter`

class `imgaug.augmenters.meta.ChannelShuffle` *(p=1.0, channels=None, name=None, deterministic=False, random_state=None)*

**Bases**: `imgaug.augmenters.meta.Augmenter`

Augmenter that randomly shuffles the channels in images.

dtype support:

- `'uint8'`: yes; fully tested
- `'uint16'`: yes; tested
- `'uint32'`: yes; tested
- `'uint64'`: yes; tested
- `'int8'`: yes; tested
- `'int16'`: yes; tested
- `'int32'`: yes; tested
- `'int64'`: yes; tested
- `'float16'`: yes; tested
- `'float32'`: yes; tested
- `'float64'`: yes; tested
- `'float128'`: yes; tested
- `'bool'`: yes; tested
Parameters

- **p** *(float or imgaug.parameters.StochasticParameter, optional)* – Probability of shuffling channels in any given image. May be a fixed probability as a float, or a StochasticParameter that returns 0s and 1s.

- **channels** *(None or imgaug.ALL or list of int, optional)* – Which channels are allowed to be shuffled with each other. If this is None or imgaug.ALL, then all channels may be shuffled. If it is a list of integers, then only the channels with indices in that list may be shuffled. (Values start at 0. All channel indices in the list must exist in each image.)

- **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> aug = iaa.ChannelShuffle(0.25)
Shuffles channels for 25% of all images.
```

```python
>>> aug = iaa.ChannelShuffle(0.25, channels=[0, 1])
Shuffles channels 0 and 1 with each other for 25% of all images.
```

Methods

- **__call__(**args, **kwargs)**
  
  Alias for imgaug.augmenters.meta.Augmenter.augment().

- **augment([return_batch, hooks])**
  
  Augment data.

- **augment_batch(batch[, hooks])**
  
  Augment a single batch.

- **augment_batches(batches[, hooks, background])**
  
  Augment multiple batches.

- **augment_bounding_boxes(bounding_boxes_on_images)**
  
  Augment bounding boxes.

- **augment_heatmaps(heatmaps[, parents, hooks])**
  
  Augment a heatmap.

- **augment_image(image[, hooks])**
  
  Augment a single image.

- **augment_images(images[, parents, hooks])**
  
  Augment multiple images.

- **augment_keypoints(keypoints_on_images[, ...])**
  
  Augment image keypoints.

- **augment_line_strings(line_strings_on_images)**
  
  Augment line strings.

- **augment_polygons(polygons_on_images[, ...])**
  
  Augment polygons.

- **augment_segmentation_maps(segmaps[, ...])**
  
  Augment segmentation maps.

- **copy()**
  
  Create a shallow copy of this Augmenter instance.

- **copy_random_state(source[, recursive, ...])**
  
  Copy the random states from a source augmenter sequence.
Table 49 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy_random_state_(source[, recursive, …])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters()

class imgaug.augmenters.meta.Lambda (func_images=None, func_heatmaps=None, func_keypoints=None, func_polygons='keypoints', name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter that calls a lambda function for each batch of input image.

This is useful to add missing functions to a list of augmenters.

dtype support:

* `'uint8'`: yes; fully tested
* `'uint16'`: yes; tested
* `'uint32'`: yes; tested
* `'uint64'`: yes; tested
* `'int8'`: yes; tested
* `'int16'`: yes; tested
* `'int32'`: yes; tested
* `'int64'`: yes; tested
* `'float16'`: yes; tested
* `'float32'`: yes; tested
* `'float64'`: yes; tested

(continues on next page)
Parameters

- **func_images** *(None or callable, optional)* – The function to call for each batch of images. It must follow the form

  ```python
  function(images, random_state, parents, hooks)
  ```

  and return the changed images (may be transformed in-place). This is essentially the interface of `imgaug.augmenters.meta.Augmenter._augment_images()`. If this is None instead of a function, the images will not be altered.

- **func_heatmaps** *(None or callable, optional)* – The function to call for each batch of heatmaps. It must follow the form

  ```python
  function(heatmaps, random_state, parents, hooks)
  ```

  and return the changed heatmaps (may be transformed in-place). This is essentially the interface of `imgaug.augmenters.meta.Augmenter._augment_heatmaps()`. If this is None instead of a function, the heatmaps will not be altered.

- **func_keypoints** *(None or callable, optional)* – The function to call for each batch of image keypoints. It must follow the form

  ```python
  function(keypoints_on_images, random_state, parents, hooks)
  ```

  and return the changed keypoints (may be transformed in-place). This is essentially the interface of `imgaug.augmenters.meta.Augmenter._augment_keypoints()`. If this is None instead of a function, the keypoints will not be altered.

- **func_polygons** *("keypoints" or None or callable, optional)* – The function to call for each batch of image polygons. It must follow the form

  ```python
  function(polygons_on_images, random_state, parents, hooks)
  ```

  and return the changed polygons (may be transformed in-place). This is essentially the interface of `imgaug.augmenters.meta.Augmenter._augment_polygons()`. If this is None instead of a function, the polygons will not be altered. If this is the string "keypoints" instead of a function, the polygons will automatically be augmented by transforming their corner vertices to keypoint and calling `func_keypoints`.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

**Examples**
Replaces every second row in images with black pixels. Leaves heatmaps and keypoints unchanged.

Replaces every second row in images with black pixels, sets every second row in heatmaps to zero and leaves keypoints unchanged (same for bounding boxes and polygons).

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, **kwargs)</td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter augment()</code>.</td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 50 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy_random_state_(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

---

### get_parameters()

**class** `imgaug.augmenters.meta.Noop(name=None, deterministic=False, random_state=None)`

**Bases:** `imgaug.augmenters.meta.Augmenter`

Augmenter that never changes input images (“no operation”).

This augmenter is useful when you just want to use a placeholder augmenter in some situation, so that you can continue to call `imgaug.augmenters.meta.Augmenter.augment_images()`, without actually changing them (e.g. when switching from training to test).

**dtype support:**

- `uint8`: yes; fully tested
- `uint16`: yes; tested
- `uint32`: yes; tested
- `uint64`: yes; tested
- `int8`: yes; tested
- `int16`: yes; tested
- `int32`: yes; tested
- `int64`: yes; tested
- `float16`: yes; tested
- `float32`: yes; tested
- `float64`: yes; tested

(continues on next page)
Parameters

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()``.`
- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()``.`
- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()``.`

Methods

``__call__(*args, **kwargs)`` Alias for `imgaug.augmenters.meta.Augmenter.augment()`.

``augment([return_batch, hooks])`` Augment data.

``augment_batch(batch[, hooks])`` Augment a single batch.

``augment_batches(batches[, hooks, background])`` Augment multiple batches.

``augment_bounding_boxes(bounding_boxes_on_images)`` Augment bounding boxes.

``augment_heatmaps(heatmaps[, parents, hooks])`` Augment a heatmap.

``augment_image(image[, hooks])`` Augment a single image.

``augment_images(images[, parents, hooks])`` Augment multiple images.

``augment_keypoints(keypoints_on_images[, ...])`` Augment image keypoints.

``augment_line_strings(line_strings_on_images)`` Augment line strings.

``augment_polygons(polygons_on_images[, ...])`` Augment polygons.

``augment_segmentation_maps(segmaps[, ...])`` Augment segmentation maps.

``copy()`` Create a shallow copy of this Augmenter instance.

``copy_random_state(source[, recursive, ...])`` Copy the random states from a source augmenter sequence.

``copy_random_state_(source[, recursive, ...])`` Copy the random states from a source augmenter sequence (inplace).

``deepcopy()`` Create a deep copy of this Augmenter instance.

``draw_grid(images, rows, cols)`` Apply this augmenter to the given images and return a grid image of the results.

``find_augmenters(func[, parents, flat])`` Find augmenters that match a condition.

``find_augmenters_by_name(name[, regex, flat])`` Find augmenter(s) by name.

``find_augmenters_by_names(names[, regex, flat])`` Find augmenter(s) by names.

``get_all_children([flat])`` Returns all children of this augmenter as a list.

``get_children_lists()`` Get a list of lists of children of this augmenter.

``localize_random_state([recursive])`` Converts global random states to local ones.

``localize_random_state_( [recursive] )`` Converts global random states to local ones.

(continued on next page)
Table 51 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy,</td>
<td>Remove this augmenter or its children that match a</td>
</tr>
<tr>
<td>noop_if_topmost])</td>
<td>condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Recess this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([n])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters()

```
get_parameters()

imgaug.augmenters.meta.OneOf(children, name=None, deterministic=False, random_state=None)
```

Augmenter that always executes exactly one of its children.

dtype support:

See ```imgaug.augmenters.meta.SomeOf```.

Parameters

- `children` (list of imgaug.augmenters.meta.Augmenter) – The choices of augmenters to apply.
- `name` (None or str, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.
- `deterministic` (bool, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.
- `random_state` (None or int or numpy.random.RandomState, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

Examples

```python
>>> imgs = [np.ones((10, 10))]
>>> seq = iaa.OneOf([
...     iaa.Fliplr(1.0),
...     iaa.Flipud(1.0)
... ])
>>> imgs_aug = seq.augment_images(imgs)
```

flips each image either horizontally or vertically.

```python
>>> seq = iaa.OneOf([
...     iaa.Fliplr(1.0),
...     iaa.Sequential([ 
...         iaa.GaussianBlur(1.0),
...         iaa.Dropout(0.05),
...     ])
... ])
```

(continues on next page)
either flips each image horizontally, or adds blur+dropout+noise or does nothing.

class imgaug.augmenters.meta.Sequential(children=None, random_order=False, name=None, deterministic=False, random_state=None)
Bases: imgaug.augmenters.meta.Augmenter, list

List augmenter that may contain other augmenters to apply in sequence or random order.

NOTE: You are not forced to use Sequential in order to use other augmenters. Each augmenter can be used on its own, e.g. the following defines an augmenter for horizontal flips and then augments a single image:

```
aug = iaa.Fliplr(0.5)
image_aug = aug.augment_image(image)
```

dtype support:

- `'uint8'`: yes; fully tested
- `'uint16'`: yes; tested
- `'uint32'`: yes; tested
- `'uint64'`: yes; tested
- `'int8'`: yes; tested
- `'int16'`: yes; tested
- `'int32'`: yes; tested
- `'int64'`: yes; tested
- `'float16'`: yes; tested
- `'float32'`: yes; tested
- `'float64'`: yes; tested
- `'float128'`: yes; tested
- `'bool'`: yes; tested

Parameters

- **children** (imgaug.augmenters.meta.Augmenter or list of imgaug.augmenters.meta.Augmenter or None, optional) – The augmenters to apply to images.
- **random_order** (bool, optional) – Whether to apply the child augmenters in random order per image. The order is resampled for each image.
- **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
- **deterministic** (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
- **random_state** (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
Examples

```python
>>> imgs = [np.random.rand(10, 10)]
>>> seq = iaa.Sequential([
>>>     iaa.Fliplr(0.5),
>>>     iaa.Flipud(0.5)
>>> ], random_order=True)
>>> imgs_aug = seq.augment_images(imgs)
```

Calls always first the horizontal flip augmenter and then the vertical flip augmenter (each having a probability of 50 percent to be used).

```python
>>> seq = iaa.Sequential([iaa.Fliplr(0.5), iaa.Flipud(0.5)], random_order=True)
>>> imgs_aug = seq.augment_images(imgs)
```

Calls sometimes first the horizontal flip augmenter and sometimes first the vertical flip augmenter (each again with 50 percent probability to be used).

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, *<em>kwargs)</em></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td>add(augmenter)</td>
<td>Add an augmenter to the list of child augmenters.</td>
</tr>
<tr>
<td>append($self, object, /)</td>
<td>Append object to the end of the list.</td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>clear($self, /)</td>
<td>Remove all items from list.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>count($self, value, /)</td>
<td>Return number of occurrences of value.</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 52 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>extend($self, iterable, /)</code></td>
<td>Extend list by appending elements from the iterable.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>index($self, value[, start, stop])</code></td>
<td>Return first index of value.</td>
</tr>
<tr>
<td><code>insert($self, index, object, /)</code></td>
<td>Insert object before index.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([]recursive)</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>pop($self[, index])</code></td>
<td>Remove and return item at index (default last).</td>
</tr>
<tr>
<td><code>remove($self, value, /)</code></td>
<td>Remove first occurrence of value.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Ressed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>reverse($self, /)</code></td>
<td>Reverse IN PLACE.</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>sort($self, /, *[key, reverse])</code></td>
<td>Stable sort IN PLACE.</td>
</tr>
<tr>
<td><code>to_deterministic(n)</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

---

**add** *(augmenter)*

Add an augmenter to the list of child augmenters.

**Parameters**

*augmenter* *(imgaug.augmenters.meta.Augmenter)* – The augmenter to add.

**get_children_lists()**

Get a list of lists of children of this augmenter.

For most augmenters, the result will be a single empty list. For augmenters with children it will often be a list with one sublist containing all children. In some cases the augmenter will contain multiple distinct lists of children, e.g. an if-list and an else-list. This will lead to a result consisting of a single list with multiple sublists, each representing the respective sublist of children.

E.g. for an if/else-augmenter that executes the children A1, A2 if a condition is met and otherwise executes the children B1, B2, B3 the result will be `[[A1, A2], [B1, B2, B3]].`

**IMPORTANT:** While the topmost list may be newly created, each of the sublist must be editable inplace resulting in a changed children list of the augmenter. E.g. if an Augmenter `IfElse(condition, [A1, A2], [B1, B2, B3])` returns `[[A1, A2], [B1, B2, B3]]` for a call to `imgaug.augmenters.meta.Augmenter.get_children_lists()` and A2 is removed inplace from `[A1, A2]`, then the children lists of `IfElse(...)` must also change to `[A1], [B1, B2, B3]`. This is used in `imgaug.augmenters.meta.Augmenter.remove_augmenters_inplace()`. 
Returns children – One or more lists of child augmenter. Can also be a single empty list.

Return type list of list of imgaug.augmenters.meta.Augmenter

get_parameters()

class imgaug.augmenters.meta.SomeOf(n=None, children=None, random_order=False, name=None, deterministic=False, random_state=None)
Bases: imgaug.augmenters.meta.Augmenter, list

List augmenter that applies only some of its children to images.

E.g. this allows to define a list of 20 augmenters, but only apply a random selection of 5 of them to each image.

This augmenter currently does not support replacing (i.e. picking the same child multiple times) due to implementation difficulties in connection with deterministic augmenters.

dtype support:

* `"uint8"`: yes; fully tested
* `"uint16"`: yes; tested
* `"uint32"`: yes; tested
* `"uint64"`: yes; tested
* `"int8"`: yes; tested
* `"int16"`: yes; tested
* `"int32"`: yes; tested
* `"int64"`: yes; tested
* `"float16"`: yes; tested
* `"float32"`: yes; tested
* `"float64"`: yes; tested
* `"float128"`: yes; tested
* `"bool"`: yes; tested

Parameters

• **n** (int or tuple of int or list of int or imgaug.parameters.StochasticParameter or None, optional) –
  
  Count of augmenters to apply.
  
  – If int, then exactly n of the child augmenters are applied to every image.
  
  – If tuple of two ints (a, b), then a <= x <= b augmenters are picked and applied to every image. Here, b may be set to None, then it will automatically replaced with the total number of available children.
  
  – If StochasticParameter, then N numbers will be sampled for N images. The parameter is expected to be discrete.
  
  – If None, then the total number of available children will be used (i.e. all children will be applied).

• **children** (imgaug.augmenters.meta.Augmenter or list of imgaug.augmenters.meta.Augmenter or None, optional) – The augmenters to apply to images.

• **random_order** (boolean, optional) – Whether to apply the child augmenters in random order per image. The order is resampled for each image.

• **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
• **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  
• **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

### Examples

```python
def imgs = [np.random.rand(10, 10)]
def seq = iaa.SomeOf(1, [  
  iaa.Fliplr(1.0),  
  iaa.Flipud(1.0)  
])
```

Applies either Fliplr or Flipud to images.

```python
def seq = iaa.SomeOf((1, 3), [  
  iaa.Fliplr(1.0),  
  iaa.Flipud(1.0),  
  iaa.GaussianBlur(1.0)  
])
```

Applies one to three of the listed augmenters (Fliplr, Flipud, GaussianBlur) to images. They are always applied in the order (1st) Fliplr, (2nd) Flipud, (3rd) GaussianBlur.

```python
def seq = iaa.SomeOf(1, None, [  
  iaa.Fliplr(1.0),  
  iaa.Flipud(1.0),  
  iaa.GaussianBlur(1.0)  
], random_order=True)
```

Applies one to all of the listed augmenters (Fliplr, Flipud, GaussianBlur) to images. They are applied in random order, i.e. sometimes Blur first, followed by Fliplr, sometimes Fliplr follow by Flipud followed by Blur, sometimes Flipud follow by Blur, etc.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(*args, **kwargs)</code></td>
<td>Alias for <code>iaaug.augmenters.meta.Augmenter.augment()</code>.</td>
</tr>
<tr>
<td><code>add(augmenter)</code></td>
<td>Add an augmenter to the list of child augmenters.</td>
</tr>
<tr>
<td><code>append(self, object, /)</code></td>
<td>Append object to the end of the list.</td>
</tr>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment_bounding_boxes(bounding_boxes_on_images[, hooks])</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps(heatmaps[, parents, hooks])</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
<tr>
<td><code>augment_images(images[, parents, hooks])</code></td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>augment_keypoints(keypoints_on_images[, ...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
<tr>
<td><code>augment_segmentation_maps(segmaps[, ...])</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>clear($self, /)</code></td>
<td>Remove all items from list.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>count($self, value, /)</code></td>
<td>Return number of occurrences of value.</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>extend($self, iterable, /)</code></td>
<td>Extend list by appending elements from the iterable.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>index($self, value[, start, stop])</code></td>
<td>Return first index of value.</td>
</tr>
<tr>
<td><code>insert($self, index, object, /)</code></td>
<td>Insert object before index.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>pop($self[, index])</code></td>
<td>Remove and return item at index (default last).</td>
</tr>
<tr>
<td><code>remove($self, value, /)</code></td>
<td>Remove first occurrence of value.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseeds this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>reverse($self, /)</code></td>
<td>Reverse IN PLACE.</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>sort($self, /, *[key, reverse])</code></td>
<td>Stable sort IN PLACE.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_parameters**

`add(augmenter)`

Add an augmenter to the list of child augmenters.
Parameters **augmenter** *(imgaug.augmenters.meta.Augmenter) – The augmenter to add.*

**get_children_lists()**
Get a list of lists of children of this augmenter.

For most augmenters, the result will be a single empty list. For augmenters with children it will often be a list with one sublist containing all children. In some cases the augmenter will contain multiple distinct lists of children, e.g. an if-list and an else-list. This will lead to a result consisting of a single list with multiple sublists, each representing the respective sublist of children.

E.g. for an if/else-augmenter that executes the children \(A_1, A_2\) if a condition is met and otherwise executes the children \(B_1, B_2, B_3\) the result will be \([[A_1, A_2], [B_1, B_2, B_3]]\).

IMPORTANT: While the topmost list may be newly created, each of the sublist must be editable inplace resulting in a changed children list of the augmenter. E.g. if an Augmenter IfElse(condition, \([A_1, A_2], [B_1, B_2, B_3]\)) returns \([[A_1, A_2], [B_1, B_2, B_3]]\) for a call to *imgaug.augmenters.meta.Augmenter.get_children_lists()* and \(A_2\) is removed inplace from \([A_1, A_2]\), then the children lists of IfElse(...) must also change to \([A_1], [B_1, B_2, B_3]\). This is used in *imgaug.augmeneters.meta.Augmenter.remove_augmenters_inplace()*.

**Returns** children – One or more lists of child augmenter. Can also be a single empty list.

**Return type** list of list of imgaug.augmenters.meta.Augmenter

**get_parameters()**

```python
class imgaug.augmenters.meta.Sometimes(p=0.5, then_list=None, else_list=None, name=None, deterministic=False, random_state=None)
```

Bases: *imgaug.augmenters.meta.Augmenter*

Augment only \(p\) percent of all images with one or more augmenters.

Let \(C\) be one or more child augmenters given to Sometimes. Let \(p\) be the percent of images to augment. Let \(I\) be the input images. Then (on average) \(p\) percent of all images in \(I\) will be augmented using \(C\).

dtype support:

```python
* `\'uint8\'`: yes; fully tested
* `\'uint16\'`: yes; tested
* `\'uint32\'`: yes; tested
* `\'uint64\'`: yes; tested
* `\'int8\'`: yes; tested
* `\'int16\'`: yes; tested
* `\'int32\'`: yes; tested
* `\'int64\'`: yes; tested
* `\'float16\'`: yes; tested
* `\'float32\'`: yes; tested
* `\'float64\'`: yes; tested
* `\'float128\'`: yes; tested
* `\'bool\'`: yes; tested
```

Parameters

- **p** *(float or imgaug.parameters.StochasticParameter, optional) – Sets the probability with which the given augmenters will be applied to input images. E.g. a value of 0.5 will result in 50 percent of all input images being augmented.*

- **then_list** *(None or imgaug.augmenters.meta.Augmenter or list of imgaug.augmenters.meta.Augmenter, optional) – Augmenter(s) to apply to \(p\) percent of*
all images.

- **else_list** *(None or imgaug.augmenters.meta.Augmenter or list of imgaug.augmenters.meta.Augmenter, optional)* – Augmenter(s) to apply to \((1-p)\) percent of all images. These augmenters will be applied only when the ones in then_list are NOT applied (either-or-relationship).

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

**Examples**

```python
>>> aug = iaa.Sometimes(0.5, iaa.GaussianBlur(0.3))
```

when calling `aug.augment_images()`, only (on average) 50 percent of all images will be blurred.

```python
>>> aug = iaa.Sometimes(0.5, iaa.GaussianBlur(0.3), iaa.Fliplr(1.0))
```

when calling `aug.augment_images()`, (on average) 50 percent of all images will be blurred, the other (again, on average) 50 percent will be horizontally flipped.

**Methods**

```python
__call__(*args, **kwargs) Alias for imgaug.augmenters.meta.Augmenter.augment().
```

```python
augment([return_batch, hooks]) Augment data.
augment_batch(batch[, hooks]) Augment a single batch.
augment_batches(batches[, hooks, background]) Augment multiple batches.
augment_bounding_boxes(bounding_boxes_on_images) Augment bounding boxes.
augment_heatmaps(heatmaps[, parents, hooks]) Augment a heatmap.
augment_image(image[, hooks]) Augment a single image.
augment_images(images[, parents, hooks]) Augment multiple images.
augment_keypoints(keypoints_on_images[, ...]) Augment image keypoints.
augment_line_strings(line_strings_on_images) Augment line strings.
augment_polygons(polygons_on_images[, ...]) Augment polygons.
augment_segmentation_maps(segmaps[, ...]) Augment segmentation maps.
copy() Create a shallow copy of this Augmenter instance.
copy_random_state(source[, recursive, ...]) Copy the random states from a source augmenter sequence.
copy_random_state_inplace(source[, recursive, ...]) Copy the random states from a source augmenter sequence (inplace).
deepcopy() Create a deep copy of this Augmenter instance.
```

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Rseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

### get_children_lists()

Get a list of lists of children of this augmenter.

For most augmenters, the result will be a single empty list. For augmenters with children it will often be a list with one sublist containing all children. In some cases the augmenter will contain multiple distinct lists of children, e.g. an if-list and an else-list. This will lead to a result consisting of a single list with multiple sublists, each representing the respective sublist of children.

E.g. for an if/else-augmenter that executes the children A1, A2 if a condition is met and otherwise executes the children B1, B2, B3 the result will be `[[A1, A2], [B1, B2, B3]]`.

IMPORTANT: While the topmost list may be newly created, each of the sublist must be editable inplace resulting in a changed children list of the augmenter. E.g. if an Augmenter IfElse(condition, [A1, A2], [B1, B2, B3]) returns `[[A1, A2], [B1, B2, B3]]` for a call to `imgaug.augmenters.meta.Augmenter.get_children_lists()` and A2 is removed inplace from [A1, A2], then the children lists of IfElse(...) must also change to [A1], [B1, B2, B3]. This is used in `imgaug.augmeneters.meta.Augmenter.remove_augmenters_inplace()`.

Returns children – One or more lists of child augmenter. Can also be a single empty list.

Return type list of list of imgaug.augmenters.meta.Augmenter

### get_parameters()

#### class imgaug.augmenters.meta.WithChannels

**Bases:** imgaug.augmenters.meta.Augmenter Apply child augmenters to specific channels.
Let $C$ be one or more child augmenters given to this augmenter. Let $H$ be a list of channels. Let $I$ be the input images. Then this augmenter will pick the channels $H$ from each image in $I$ (resulting in new images) and apply $C$ to them. The result of the augmentation will be merged back into the original images.

### dtype support:

- `uint8`: yes; fully tested
- `uint16`: yes; tested
- `uint32`: yes; tested
- `uint64`: yes; tested
- `int8`: yes; tested
- `int16`: yes; tested
- `int32`: yes; tested
- `int64`: yes; tested
- `float16`: yes; tested
- `float32`: yes; tested
- `float64`: yes; tested
- `float128`: yes; tested
- `bool`: yes; tested

### Parameters

- **channels** *(None or list of int, optional)* – Sets the channels to be extracted from each image. If None, all channels will be used. Note that this is not stochastic - the extracted channels are always the same ones.

- **children** *(Augmenter or list of imgaug.augmenters.meta.Augmenter or None, optional)* – One or more augmenters to apply to images, after the channels are extracted.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

### Examples

```python
>>> aug = iaa.WithChannels([0], iaa.Add(10))
```

assuming input images are RGB, then this augmenter will add 10 only to the first channel, i.e. make images more red.

### Methods

- **`__call__`** *(args, **kwargs)* – Alias for `imgaug.augmenters.meta.Augmenter.augment()`.

- **`augment`** *(return_batch, hooks)* – Augment data.

- **`augment_batch`** *(batch[, hooks]*) – Augment a single batch.

- **`augment_batches`** *(batches[, hooks, background]*) – Augment multiple batches.

- **`augment_bounding_boxes`** *(bounding_boxes_on_images)* – Augment bounding boxes.

Continued on next page
Table 55 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_(recursive)</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([in])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_children_lists()**

Get a list of lists of children of this augmenter.

For most augmenters, the result will be a single empty list. For augmenters with children it will often be a list with one sublist containing all children. In some cases the augmenter will contain multiple distinct lists of children, e.g. an if-list and an else-list. This will lead to a result consisting of a single list with multiple sublists, each representing the respective sublist of children.

E.g. for an if/else-augmenter that executes the children A1, A2 if a condition is met and otherwise executes the children B1, B2, B3 the result will be [['A1, A2'], ['B1, B2, B3']].
IMPORTANT: While the topmost list may be newly created, each of the sublist must be editable inplace resulting in a changed children list of the augmenter. E.g. if an Augmenter IfElse(condition, [A1, A2], [B1, B2, B3]) returns [[A1, A2], [B1, B2, B3]] for a call to imgaug.augmenters.meta.Augmenter.get_children_lists() and A2 is removed inplace from [A1, A2], then the children lists of IfElse(...) must also change to [A1], [B1, B2, B3]. This is used in imgaug.augmenters.meta.Augmenter.remove_augmenters_inplace().

Returns children – One or more lists of child augmenter. Can also be a single empty list.

Return type list of list of imgaug.augmenters.meta.Augmenter

generate_parameters()

imgaug.augmenters.meta.clip_augmented_image(image, min_value, max_value)
imgaug.augmenters.meta.clip_augmented_image_(image, min_value, max_value)
imgaug.augmenters.meta.clip_augmented_images(images, min_value, max_value)
imgaug.augmenters.meta.clip_augmented_images_(images, min_value, max_value)
imgaug.augmenters.meta.copy_arrays(arrays)
imgaug.augmenters.meta.estimate_max_number_of_channels(images)
imgaug.augmenters.meta.handle_children_list(lst, augmenter_name, lst_name, default='sequential')
imgaug.augmenters.meta.invert_reduce_to_nonempty(objs, ids, objs_reduced)
imgaug.augmenters.meta.reduce_to_nonempty(objs)
imgaug.augmenters.meta.shuffle_channels(image, random_state, channels=None)

Randomize the order of (color) channels in an image.

dtype support:

* `"uint8"`: yes; fully tested
* `"uint16"`: yes; indirectly tested (1)
* `"uint32"`: yes; indirectly tested (1)
* `"uint64"`: yes; indirectly tested (1)
* `"int8"`: yes; indirectly tested (1)
* `"int16"`: yes; indirectly tested (1)
* `"int32"`: yes; indirectly tested (1)
* `"int64"`: yes; indirectly tested (1)
* `"float16"`: yes; indirectly tested (1)
* `"float32"`: yes; indirectly tested (1)
* `"float64"`: yes; indirectly tested (1)
* `"float128"`: yes; indirectly tested (1)
* `"bool"`: yes; indirectly tested (1)

- (1) Indirectly tested via `"ChannelShuffle"`.

Parameters

- image ((H,W,[C]) ndarray) – Image of any dtype for which to shuffle the channels.
- random_state (numpy.random.RandomState) – The random state to use for this shuffling operation.
- channels (None or imgaug.ALL or list of int, optional) – Which channels are allowed to be shuffled with each other. If this is None or imgaug.ALL, then all channels may be
shuffled. If it is a list of integers, then only the channels with indices in that list may be shuffled. (Values start at 0. All channel indices in the list must exist in each image.)

Returns
The input image with shuffled channels.

Return type
ndarray

13.15 imgaug.augmenters.arithmetic

Augmenters that perform simple arithmetic changes.
Do not import directly from this file, as the categorization is not final. Use instead:

```
from imgaug import augmenters as iaa
```

and then e.g.:

```
seq = iaa.Sequential([iaa.Add((-5, 5)), iaa.Multiply((0.9, 1.1))])
```

List of augmenters:

- Add
- AddElementwise
- AdditiveGaussianNoise
- AdditiveLaplaceNoise
- AdditivePoissonNoise
- Multiply
- MultiplyElementwise
- Dropout
- CoarseDropout
- ReplaceElementwise
- ImpulseNoise
- SaltAndPepper
- CoarseSaltAndPepper
- Salt
- CoarseSalt
- Pepper
- CoarsePepper
- Invert
- ContrastNormalization
- JpegCompression

```
class imgaug.augmenters.arithmetic.Add (value=0, per_channel=False, name=None, deterministic=False, random_state=None)
```

Add a value to all pixels in an image.
dtype support:

* `uint8`: yes; fully tested
* `uint16`: yes; tested
* `uint32`: no
* `uint64`: no
* `int8`: yes; tested
* `int16`: yes; tested
* `int32`: no
* `int64`: no
* `float16`: yes; tested
* `float32`: yes; tested
* `float64`: no
* `float128`: no
* `bool`: yes; tested

Parameters

- **value** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Value to add to all pixels.
  - If a number, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value from the discrete range \([a, b]\) will be used.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then a value will be sampled per image from that parameter.
- **per_channel** *(bool or float, optional)* – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images \(per\_channel\) will be treated as True, otherwise as False.
- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter._init__()`.
- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter._init__()`.

Examples

```python
>>> aug = iaa.Add(10)
```
always adds a value of 10 to all pixels in the image.

```python
>>> aug = iaa.Add((-10, 10))
```
adds a value from the discrete range \([-10 .. 10]\) to all pixels of the input images. The exact value is sampled per image.

```python
>>> aug = iaa.Add((-10, 10), per_channel=True)
```
adds a value from the discrete range \([-10 .. 10]\) to all pixels of the input images. The exact value is sampled per image AND channel, i.e. to a red-channel it might add 5 while subtracting 7 from the blue channel of the same image.
```python
>>> aug = iaa.Add((-10, 10), per_channel=0.5)
```

same as previous example, but the `per_channel` feature is only active for 50 percent of all images.

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, **kwargs)</td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segments[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
</tbody>
</table>
converts this augmenter from a stochastic to a deterministic one.

get_parameters()

class imgaug.augmenters.arithmetic.AddElementwise(value=0, per_channel=False,
name=None, deterministic=False,
random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Add values to the pixels of images with possibly different values for neighbouring pixels.

While the Add Augmenter adds a constant value per image, this one can add different values (sampled per pixel).

dtype support:

* `uint8`: yes; fully tested
* `uint16`: yes; tested
* `uint32`: no
* `int8`: no;
* `int16`: yes; tested
* `int32`: no
* `int64`: no
* `float16`: yes; tested
* `float32`: yes; tested
* `float64`: no
* `float128`: no
* `bool`: yes; tested

Parameters

- **value** *(int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional)*

  Value to add to the pixels.

  - If an int, then that value will be used for all images.
  - If a tuple `(a, b)`, then values from the discrete range `[a .. b]` will be sampled.
  - If a list of integers, a random value will be sampled from the list per image.
  - If a StochasticParameter, then values will be sampled per pixel (and possibly channel) from that parameter.

- **per_channel** *(bool or float, optional)* – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float `p`, then for `p` percent of all images per_channel will be treated as True, otherwise as False.

- **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
Examples

>>> aug = iaa.AddElementwise(10)
always adds a value of 10 to all pixels in the image.

>>> aug = iaa.AddElementwise((-10, 10))
samples per pixel a value from the discrete range [-10 .. 10] and adds that value to the pixel.

>>> aug = iaa.AddElementwise((-10, 10), per_channel=True)
samples per pixel and channel a value from the discrete range [-10 .. 10] ands it to the pixel’s value. Therefore, added values may differ between channels of the same pixel.

>>> aug = iaa.AddElementwise((-10, 10), per_channel=0.5)
same as previous example, but the per_channel feature is only active for 50 percent of all images.

Methods

__call__(*args, **kwargs) Alias for imgaug.augmenters.meta.Augmenter.augment().

augment([return_batch, hooks]) Augment data.
augment_batch(batch[, hooks]) Augment a single batch.
augment_batches(batches[, hooks, background]) Augment multiple batches.
augment_bounding_boxes(bounding_boxes_on_images) Augment bounding boxes.
augment_heatmaps(heatmaps[, parents, hooks]) Augment a heatmap.
augment_image(image[, hooks]) Augment a single image.
augment_images(images[, parents, hooks]) Augment multiple images.
augment_keypoints(keypoints_on_images[, ...]) Augment image keypoints.
augment_line_strings(line_strings_on_images) Augment line strings.
augment_polygons(polygons_on_images[, ...]) Augment polygons.
augment_segmentation_maps(segmaps[, ...]) Augment segmentation maps.
copy() Create a shallow copy of this Augmenter instance.
copy_random_state(source[, recursive, ...]) Copy the random states from a source augmenter sequence.
copy_random_state_(source[, recursive, ...]) Copy the random states from a source augmenter sequence (inplace).
deepcopy() Create a deep copy of this Augmenter instance.
draw_grid(images, rows, cols) Apply this augmenter to the given images and return a grid image of the results.
find_augmenters(func[, parents, flat]) Find augmenters that match a condition.
find_augmenters_by_name(name[, regex, flat]) Find augmenter(s) by name.
find_augmenters_by_names(names[, regex, flat]) Find augmenter(s) by names.
Table 57 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_all_children()</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters()

```python
imgaug.augmenters.arithmetic.AdditiveGaussianNoise(loc=0, scale=0, per_channel=False, name=None, deterministic=False, random_state=None)
```

Add gaussian noise (aka white noise) to images.

dtype support:

```
See `''imgaug.augmenters.arithmetic.AddElementwise```
```

Parameters

- **loc** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* –
  Mean of the normal distribution that generates the noise.
  - If a number, exactly that value will be used.
  - If a tuple `(a, b)`, a random value from the range `a <= x <= b` will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **scale** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* –
  Standard deviation of the normal distribution that generates the noise. Must be `>= 0`. If 0 then only `loc` will be used.
  - If an int or float, exactly that value will be used.
  - If a tuple `(a, b)`, a random value from the range `a <= x <= b` will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
– If a StochasticParameter, a value will be sampled from the parameter per image.

• **per_channel** *(bool or float, optional)* – Whether to use the same noise value per pixel for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images per_channel will be treated as True, otherwise as False.

• **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

• **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

• **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

### Examples

```python
>>> aug = iaa.AdditiveGaussianNoise(scale=0.1*255)
```

adds gaussian noise from the distribution \(N(0, 0.1\times255)\) to images.

```python
>>> aug = iaa.AdditiveGaussianNoise(scale=(0, 0.1*255))
```

adds gaussian noise from the distribution \(N(0, s)\) to images, where \(s\) is sampled per image from the range \(0 \leq s \leq 0.1\times255\).

```python
>>> aug = iaa.AdditiveGaussianNoise(scale=0.1*255, per_channel=True)
```

adds gaussian noise from the distribution \(N(0, 0.1\times255)\) to images, where the noise value is different per pixel and channel (e.g. a different one for red, green and blue channels for the same pixel).

```python
>>> aug = iaa.AdditiveGaussianNoise(scale=0.1*255, per_channel=0.5)
```

adds gaussian noise from the distribution \(N(0, 0.1\times255)\) to images, where the noise value is sometimes (50 percent of all cases) the same per pixel for all channels and sometimes different (other 50 percent).  

```python
imgaug.augmenters.arithmetic.AdditiveLaplaceNoise(loc=0, scale=0, per_channel=False, name=None, deterministic=False, random_state=None)
```

Add laplace noise to images.

The laplace distribution is similar to the gaussian distribution, but has puts weight on the long tail. Hence, this noise will add more outliers (very high/low values). It is somewhere between gaussian noise and salt and pepper noise.

Values of around \(255 \times 0.05\) for `scale` lead to visible noise (for uint8). Values of around \(255 \times 0.10\) for `scale` lead to very visible noise (for uint8). It is recommended to usually set `per_channel` to True.

**dtype support:**

See `'imgaug.augmenters.arithmetic.AddElementwise'``.

### Parameters

- **loc** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* –  

  Mean of the laplace distribution that generates the noise.
If a number, exactly that value will be used.
- If a tuple \((a, b)\), a random value from the range \(a <= x <= b\) will be sampled per image.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, a value will be sampled from the parameter per image.

- **scale** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – Standard deviation of the laplace distribution that generates the noise. Must be \(>= 0\). If 0 then only \(loc\) will be used. Recommended to be around \(255 * 0.05\).
- If an int or float, exactly that value will be used.
- If a tuple \((a, b)\), a random value from the range \(a <= x <= b\) will be sampled per image.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, a value will be sampled from the parameter per image.

- **per_channel** (bool or float, optional) – Whether to use the same noise value per pixel for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images \(per\_channel\) will be treated as True, otherwise as False.

- **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> aug = iaa.AdditiveLaplaceNoise(scale=0.1*255)
```

Adds laplace noise from the distribution \(\text{Laplace}(0, 0.1*255)\) to images.

```python
>>> aug = iaa.AdditiveLaplaceNoise(scale=(0, 0.1*255))
```

Adds laplace noise from the distribution \(\text{Laplace}(0, s)\) to images, where \(s\) is sampled per image from the range \(0 <= s <= 0.1*255\).

```python
>>> aug = iaa.AdditiveLaplaceNoise(scale=0.1*255, per_channel=True)
```

Adds laplace noise from the distribution \(\text{Laplace}(0, 0.1*255)\) to images, where the noise value is different per pixel and channel (e.g. a different one for red, green and blue channels for the same pixel).

```python
>>> aug = iaa.AdditiveLaplaceNoise(scale=0.1*255, per_channel=0.5)
```

Adds laplace noise from the distribution \(\text{Laplace}(0, 0.1*255)\) to images, where the noise value is sometimes (50 percent of all cases) the same per pixel for all channels and sometimes different (other 50 percent).

*Create an augmenter to add poisson noise to images.*

```python
imgaug.augmenters.arithmetic.AdditivePoissonNoise(lam=0, per_channel=False, name=None, deterministic=False, random_state=None)
```
Poisson noise is comparable to gaussian noise as in `AdditiveGaussianNoise`, but the values are sampled from a poisson distribution instead of a gaussian distribution. As poisson distributions produce only positive numbers, the sign of the sampled values are here randomly flipped.

Values of around 10.0 for \( \lambda \) lead to visible noise (for uint8). Values of around 20.0 for \( \lambda \) lead to very visible noise (for uint8). It is recommended to usually set `per_channel` to True.

dtype support:

See `''imgaug.augmenters.arithmetic.AddElementwise``.

**Parameters**

- **\( \lambda \)** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – Lambda parameter of the poisson distribution. Recommended values are around 0.0 to 10.0.
  - If a number, exactly that value will be used.
  - If a tuple \((a, b)\), a random value from the range \(a \leq x \leq b\) will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **per_channel** (bool or float, optional) – Whether to use the same noise value per pixel for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images `per_channel` will be treated as True, otherwise as False.

**Examples**

```
>>> aug = iaa.AdditivePoissonNoise(lam=5.0)
```

Adds poisson noise sampled from \( \text{Poisson}(5.0) \) to images.

```
>>> aug = iaa.AdditivePoissonNoise(lam=(0.0, 10.0))
```

Adds poisson noise sampled from \( \text{Poisson}(x) \) to images, where \( x \) is randomly sampled per image from the interval \([0.0, 10.0]\).

```
>>> aug = iaa.AdditivePoissonNoise(lam=5.0, per_channel=True)
```

Adds poisson noise sampled from \( \text{Poisson}(5.0) \) to images, where the values are different per pixel and channel (e.g. a different one for red, green and blue channels for the same pixel).

```
>>> aug = iaa.AdditivePoissonNoise(lam=(0.0, 10.0), per_channel=True)
```
Adds poisson noise sampled from $\text{Poisson}(x)$ to images, with $x$ being sampled from $\text{uniform}(0.0, 10.0)$ per image, pixel and channel. This is the recommended configuration.

```python
>>> aug = iaa.AdditivePoissonNoise(lam=2, per_channel=0.5)
```

Adds poisson noise sampled from the distribution $\text{Poisson}(2)$ to images, where the values are sometimes (50 percent of all cases) the same per pixel for all channels and sometimes different (other 50 percent).

```python
imgaug.augmenters.arithmetic.CoarseDropout(p=0, size_px=None, size_percent=None, per_channel=False, min_size=4, name=None, deterministic=False, random_state=None)
```

Augmenter that sets rectangular areas within images to zero.

In contrast to Dropout, these areas can have larger sizes. (E.g. you might end up with three large black rectangles in an image.) Note that the current implementation leads to correlated sizes, so when there is one large area that is dropped, there is a high likelihood that all other dropped areas are also large.

This method is implemented by generating the dropout mask at a lower resolution (than the image has) and then upsampling the mask before dropping the pixels.

**dtype support:**

See ```imgaug.augmenters.arithmetic.MultiplyElementwise```.

**Parameters**

- **p** (float or tuple of float or imgaug.parameters.StochasticParameter, optional) –
  The probability of any pixel being dropped (i.e. set to zero).
  - If a float, then that value will be used for all pixels. A value of 1.0 would mean, that all pixels will be dropped. A value of 0.0 would lead to no pixels being dropped.
  - If a tuple $(a, b)$, then a value $p$ will be sampled from the range $a \leq p \leq b$ per image and be used as the pixel’s dropout probability.
  - If a StochasticParameter, then this parameter will be used to determine per pixel whether it should be dropped (sampled value of 0) or shouldn’t (sampled value of 1).

- **size_px** (int or tuple of int or imgaug.parameters.StochasticParameter, optional) – The size of the lower resolution image from which to sample the dropout mask in absolute pixel dimensions.
  - If an integer, then that size will be used for both height and width. E.g. a value of 3 would lead to a $3 \times 3$ mask, which is then upsampled to $H \times W$, where $H$ is the image size and $W$ the image width.
  - If a tuple $(a, b)$, then two values $M, N$ will be sampled from the range $[a..b]$ and the mask will be generated at size $M \times N$, then upsampled to $H \times W$.
  - If a StochasticParameter, then this parameter will be used to determine the sizes. It is expected to be discrete.

- **size_percent** (float or tuple of float or imgaug.parameters.StochasticParameter, optional) – The size of the lower resolution image from which to sample the dropout mask in percent of the input image.
  - If a float, then that value will be used as the percentage of the height and width (relative to the original size). E.g. for value $p$, the mask will be sampled from $(p \times H) \times (p \times W)$ and later upsampled to $H \times W$.  

---

13.15. `imgaug.augmenters.arithmetic` 291
– If a tuple \((a, b)\), then two values \(m, n\) will be sampled from the interval \((a, b)\) and used as the percentages, i.e. the mask size will be \((m\times H)\times(n\times W)\).

– If a StochasticParameter, then this parameter will be used to sample the percentage values. It is expected to be continuous.

• **per_channel** (bool or float, optional) – Whether to use the same value (is dropped / is not dropped) for all channels of a pixel (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images *per_channel* will be treated as True, otherwise as False.

• **min_size** (int, optional) – Minimum size of the low resolution mask, both width and height. If *size_percent* or *size_px* leads to a lower value than this, *min_size* will be used instead. This should never have a value of less than 2, otherwise one may end up with a 1x1 low resolution mask, leading easily to the whole image being dropped.

• **name** (None or str, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

• **deterministic** (bool, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

• **random_state** (None or int or numpy.random.RandomState, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

Examples

```python
>>> aug = iaa.CoarseDropout(0.02, size_percent=0.5)
```
drops 2 percent of all pixels on an lower-resolution image that has 50 percent of the original image’s size, leading to dropped areas that have roughly 2x2 pixels size.

```python
>>> aug = iaa.CoarseDropout((0.0, 0.05), size_percent=(0.05, 0.5))
```
generates a dropout mask at 5 to 50 percent of image’s size. In that mask, 0 to 5 percent of all pixels are dropped (random per image).

```python
>>> aug = iaa.CoarseDropout((0.0, 0.05), size_px=(2, 16))
```
same as previous example, but the lower resolution image has 2 to 16 pixels size.

```python
>>> aug = iaa.CoarseDropout(0.02, size_percent=0.5, per_channel=True)
```
drops 2 percent of all pixels at 50 percent resolution (2x2 sizes) in a channel-wise fashion, i.e. it is unlikely for any pixel to have all channels set to zero (black pixels).

```python
>>> aug = iaa.CoarseDropout(0.02, size_percent=0.5, per_channel=0.5)
```
same as previous example, but the *per_channel* feature is only active for 50 percent of all images.

`imgaug.augmenters.arithmetic.CoarsePepper(p=0, size_px=None, size_percent=None, per_channel=False, min_size=4, name=None, deterministic=False, random_state=None)`

Adds coarse pepper noise to an image, i.e. rectangles that contain noisy black-ish pixels.

dtype support:
Parameters

- **p** *(float or tuple of float or list of float or imgaug.parameters.StochasticParameter, optional)*
  - Probability of changing a pixel to pepper noise.
  - If a float, then that value will be used for all images as the probability.
  - If a tuple \((a, b)\), then a probability will be sampled per image from the range \(a \leq x \leq b\).
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then this parameter will be used as the mask, i.e. it is expected to contain values between 0.0 and 1.0, where 1.0 means that pepper is to be added at that location.

- **size_px** *(int or tuple of int or imgaug.parameters.StochasticParameter, optional)* – The size of the lower resolution image from which to sample the noise mask in absolute pixel dimensions.
  - If an integer, then that size will be used for both height and width. E.g. a value of 3 would lead to a 3x3 mask, which is then upsampled to \(H\times W\), where \(H\) is the image size and \(W\) the image width.
  - If a tuple \((a, b)\), then two values \(M, N\) will be sampled from the range \([a..b]\) and the mask will be generated at size \(M\times N\), then upsampled to \(H\times W\).
  - If a StochasticParameter, then this parameter will be used to determine the sizes. It is expected to be discrete.

- **size_percent** *(float or tuple of float or imgaug.parameters.StochasticParameter, optional)* – The size of the lower resolution image from which to sample the noise mask in percent of the input image.
  - If a float, then that value will be used as the percentage of the height and width (relative to the original size). E.g. for value \(p\), the mask will be sampled from \((p\times H)\times(p\times W)\) and later upsampled to \(H\times W\).
  - If a tuple \((a, b)\), then two values \(m, n\) will be sampled from the interval \((a, b)\) and used as the percentages, i.e. the mask size will be \((m\times H)\times(n\times W)\).
  - If a StochasticParameter, then this parameter will be used to sample the percentage values. It is expected to be continuous.

- **per_channel** *(bool or float, optional)* – Whether to use the same value (is dropped / is not dropped) for all channels of a pixel (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images **per_channel** will be treated as True, otherwise as False.

- **min_size** *(int, optional)* – Minimum size of the low resolution mask, both width and height. If **size_percent** or **size_px** leads to a lower value than this, **min_size** will be used instead. This should never have a value of less than 2, otherwise one may end up with a 1x1 low resolution mask, leading easily to the whole image being replaced.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

See `imgaug.augmenters.arithmetic.ReplaceElementwise`. 

13.15. `imgaug.augmenters.arithmetic` 293
• **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

• **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

## Examples

```python
>>> aug = iaa.CoarsePepper(0.05, size_percent=(0.01, 0.1))
```

Replaces 5 percent of all pixels with pepper in an image that has 1 to 10 percent of the input image size, then upscales the results to the input image size, leading to large rectangular areas being replaced.

```python
imgaug.augmenters.arithmetic.CoarseSalt(p=0, size_px=None, size_percent=None, per_channel=False, min_size=4, name=None, deterministic=False, random_state=None)
```

Adds coarse salt noise to an image, i.e. rectangles containing noisy white-ish pixels.

dtype support:

```python
See ```imgaug.augmenters.arithmetic.ReplaceElementwise```
```

### Parameters

- **p** *(float or tuple of float or list of float or **imgaug.parameters.StochasticParameter**, optional)*
  
  Probability of changing a pixel to salt noise.
  
  - If a float, then that value will be used for all images as the probability.
  - If a tuple `(a, b)`, then a probability will be sampled per image from the range `a <= x <= b`.
  - If a list, then a random value will be sampled from that list per image.
  - If a `StochasticParameter`, then this parameter will be used as the mask, i.e. it is expected to contain values between 0.0 and 1.0, where 1.0 means that salt is to be added at that location.

- **size_px** *(int or tuple of int or **imgaug.parameters.StochasticParameter**, optional)* – The size of the lower resolution image from which to sample the noise mask in absolute pixel dimensions.
  
  - If an integer, then that size will be used for both height and width. E.g. a value of 3 would lead to a 3x3 mask, which is then upsampled to HxW, where H is the image size and W the image width.
  - If a tuple `(a, b)`, then two values M, N will be sampled from the range `[a..b]` and the mask will be generated at size MxN, then upsampled to HxW.
  - If a `StochasticParameter`, then this parameter will be used to determine the sizes. It is expected to be discrete.

- **size_percent** *(float or tuple of float or **imgaug.parameters.StochasticParameter**, optional)* – The size of the lower resolution image from which to sample the noise mask in percent of the input image.
  
  - If a float, then that value will be used as the percentage of the height and width (relative to the original size). E.g. for value p, the mask will be sampled from `(p*H) x (p*W)` and later upsampled to HxW.
– If a tuple \((a, b)\), then two values \(m, n\) will be sampled from the interval \((a, b)\) and used as the percentages, i.e., the mask size will be \((m*H) \times (n*W)\).

– If a StochasticParameter, then this parameter will be used to sample the percentage values. It is expected to be continuous.

• **per_channel** *(bool or float, optional)* – Whether to use the same value (is dropped / is not dropped) for all channels of a pixel (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images per_channel will be treated as True, otherwise as False.

• **min_size** *(int, optional)* – Minimum size of the low resolution mask, both width and height. If size_percent or size_px leads to a lower value than this, min_size will be used instead. This should never have a value of less than 2, otherwise one may end up with a 1x1 low resolution mask, leading easily to the whole image being replaced.

• **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

• **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

• **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

### Examples

```python
>>> aug = iaa.CoarseSalt(0.05, size_percent=(0.01, 0.1))
```

Replaces 5 percent of all pixels with salt in an image that has 1 to 10 percent of the input image size, then upscales the results to the input image size, leading to large rectangular areas being replaced.

```python
imgaug.augmenters.arithmetic.CoarseSaltAndPepper(p=0, size_px=None, size_percent=None, per_channel=False, min_size=4, name=None, deterministic=False, random_state=None)
```

Adds coarse salt and pepper noise to an image, i.e., rectangles that contain noisy white-ish and black-ish pixels.

TODO replace dtype support with uint8 only, because replacement is geared towards that value range.

dtype support:

```python
See ```imgaug.augmenters.arithmetic.ReplaceElementwise````.
```

### Parameters

• **p** *(float or tuple of float or list of float or imgaug.parameters.StochasticParameter, optional)*

  Probability of changing a pixel to salt/pepper noise.

  – If a float, then that value will be used for all images as the probability.

  – If a tuple \((a, b)\), then a probability will be sampled per image from the range \(a <= x <= b\).

  – If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then this parameter will be used as the mask, i.e. it is expected to contain values between 0.0 and 1.0, where 1.0 means that salt/pepper is to be added at that location.

- **size_px** *(int or tuple of int or imgaug.parameters.StochasticParameter, optional)* – The size of the lower resolution image from which to sample the noise mask in absolute pixel dimensions.
  - If an integer, then that size will be used for both height and width. E.g. a value of 3 would lead to a 3×3 mask, which is then upsampled to H×W, where H is the image size and W the image width.
  - If a tuple (a, b), then two values M, N will be sampled from the range [a..b] and the mask will be generated at size M×N, then upsampled to H×W.
  - If a StochasticParameter, then this parameter will be used to determine the sizes. It is expected to be discrete.

- **size_percent** *(float or tuple of float or imgaug.parameters.StochasticParameter, optional)* – The size of the lower resolution image from which to sample the noise mask in percent of the input image.
  - If a float, then that value will be used as the percentage of the height and width (relative to the original size). E.g. for value p, the mask will be sampled from (p×H)×(p×W) and later upsampled to H×W.
  - If a tuple (a, b), then two values m, n will be sampled from the interval (a, b) and used as the percentages, i.e. the mask size will be (m×H)×(n×W).
  - If a StochasticParameter, then this parameter will be used to sample the percentage values. It is expected to be continuous.

- **per_channel** *(bool or float, optional)* – Whether to use the same value (is dropped / is not dropped) for all channels of a pixel (False) or to sample a new value for each channel (True). If this value is a float p, then for p percent of all images per_channel will be treated as True, otherwise as False.

- **min_size** *(int, optional)* – Minimum size of the low resolution mask, both width and height. If size_percent or size_px leads to a lower value than this, min_size will be used instead. This should never have a value of less than 2, otherwise one may end up with a 1x1 low resolution mask, leading easily to the whole image being replaced.

- **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

### Examples

```python
g = imgaug.CoarseSaltAndPepper(0.05, size_percent=(0.01, 0.1))
```

Replaces 5 percent of all pixels with salt/pepper in an image that has 1 to 10 percent of the input image size, then upscales the results to the input image size, leading to large rectangular areas being replaced.
Augmenter that changes the contrast of images.

dtype support:
See `imgaug.augmenters.contrast.LinearContrast`.

Parameters

- **alpha** (number or tuple of number or list of number or `imgaug.parameters.StochasticParameter`, optional) – Strength of the contrast normalization. Higher values than 1.0 lead to higher contrast, lower values decrease the contrast.
  - If a number, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value will be sampled per image from the range \(a \leq x \leq b\) and be used as the alpha value.
  - If a list, then a random value will be sampled per image from that list.
  - If a StochasticParameter, then this parameter will be used to sample the alpha value per image.

- **per_channel** (bool or float, optional) – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images `per_channel` will be treated as True, otherwise as False.

- **name** (None or str, optional) – See `imgaug.augmenters.meta.Augmenter.__init__`.

- **deterministic** (bool, optional) – See `imgaug.augmenters.meta.Augmenter.__init__`.

- **random_state** (None or int or `numpy.random.RandomState`, optional) – See `imgaug.augmenters.meta.Augmenter.__init__`.

Examples

```python
>>> iaa.ContrastNormalization((0.5, 1.5))
```

Decreases oder improves contrast per image by a random factor between 0.5 and 1.5. The factor 0.5 means that any difference from the center value (i.e. 128) will be halved, leading to less contrast.

```python
>>> iaa.ContrastNormalization((0.5, 1.5), per_channel=0.5)
```

Same as before, but for 50 percent of all images the normalization is done independently per channel (i.e. factors can vary per channel for the same image). In the other 50 percent of all images, the factor is the same for all channels.

Augmenter that sets a certain fraction of pixels in images to zero.

dtype support:
See ```imgaug.augmenters.arithmetic.MultiplyElementwise```. 

Parameters
• **p** (*float or tuple of float or imgaug.parameters.StochasticParameter, optional*) – The probability of any pixel being dropped (i.e. set to zero).

  – If a float, then that value will be used for all images. A value of 1.0 would mean that all pixels will be dropped and 0.0 that no pixels would be dropped. A value of 0.05 corresponds to 5 percent of all pixels dropped.

  – If a tuple *(a, b)*, then a value *p* will be sampled from the range *a* <= *p* <= *b* per image and be used as the pixel’s dropout probability.

  – If a StochasticParameter, then this parameter will be used to determine per pixel whether it should be dropped (sampled value of 0) or shouldn’t (sampled value of 1). If you instead want to provide the probability as a stochastic parameter, you can usually do `imgaug.parameters.Binomial(1-p)` to convert parameter *p* to a 0/1 representation.

• **per_channel** (*bool or float, optional*) – Whether to use the same value (is dropped / is not dropped) for all channels of a pixel (False) or to sample a new value for each channel (True). If this value is a float *p*, then for *p* percent of all images `per_channel` will be treated as True, otherwise as False.

• **name** (*None or str, optional*) – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

• **deterministic** (*bool, optional*) – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

• **random_state** (*None or int or numpy.random.RandomState, optional*) – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

### Examples

```python
group Dropout(0.02) 
```
drops 2 percent of all pixels.

```python
group Dropout((0.0, 0.05)) 
```
drops in each image a random fraction of all pixels, where the fraction is in the range 0.0 <= x <= 0.05.

```python
group Dropout(0.02, per_channel=True) 
```
drops 2 percent of all pixels in a channel-wise fashion, i.e. it is unlikely for any pixel to have all channels set to zero (black pixels).

```python
group Dropout(0.02, per_channel=0.5) 
```
same as previous example, but the `per_channel` feature is only active for 50 percent of all images.

`imgaug.augmenters.arithmetic.ImpulseNoise(p=0, name=None, deterministic=False, random_state=None)`  

Creates an augmenter to apply impulse noise to an image.

This is identical to `SaltAndPepper`, except that `per_channel` is always set to True.

Dtype support:

```python
See `imgaug.augmenters.arithmetic.SaltAndPepper`
```
class imgaug.augmenters.arithmetic.Invert(p=0, per_channel=False, min_value=None, max_value=None, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter that inverts all values in images.

For the standard value range of 0-255 it converts 0 to 255, 255 to 0 and 10 to \((255-10)=245\).

Let \(M\) be the maximum value possible, \(m\) the minimum value possible, \(v\) a value. Then the distance of \(v\) to \(m\) is \(d=\text{abs}(v-m)\) and the new value is given by \(v'=M-d\).

dtype support:

```python
if (min_value=None and max_value=None):
    * `\'uint8\'`: yes; fully tested
    * `\'uint16\'`: yes; tested
    * `\'uint32\'`: yes; tested
    * `\'uint64\'`: yes; tested
    * `\'int8\'`: yes; tested
    * `\'int16\'`: yes; tested
    * `\'int32\'`: yes; tested
    * `\'int64\'`: yes; tested
    * `\'float16\'`: yes; tested
    * `\'float32\'`: yes; tested
    * `\'float64\'`: yes; tested
    * `\'float128\'`: yes; tested
    * `\'bool\'`: yes; tested

if (min_value!=None or max_value!=None):
    * `\'uint8\'`: yes; fully tested
    * `\'uint16\'`: yes; tested
    * `\'uint32\'`: yes; tested
    * `\'uint64\'`: yes; tested
    * `\'int8\'`: yes; tested
    * `\'int16\'`: yes; tested
    * `\'int32\'`: yes; tested
    * `\'int64\'`: no (1)
    * `\'float16\'`: yes; tested
    * `\'float32\'`: yes; tested
    * `\'float64\'`: no (1)
    * `\'float128\'`: no (2)
    * `\'bool\'`: no (3)

- (1) Not allowed as int/float have to be increased in resolution when using \(--\)min/max values.
- (2) Not tested.
- (3) Makes no sense when using min/max values.
```

Parameters

- **p** *(float or imgaug.parameters.StochasticParameter, optional)* –
  The probability of an image to be inverted.
  - If a float, then that probability will be used for all images.
  - If a StochasticParameter, then that parameter will queried per image and is expected to return values in the range \([0.0, 1.0]\), where values >0.5 mean that the image/channel
is supposed to be inverted. Recommended to be some form of `imgaug.parameters.Binomial`.

- **per_channel** (bool or float, optional) – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float $p$, then for $p$ percent of all images `per_channel` will be treated as True, otherwise as False.

- **min_value** (None or number, optional) – Minimum of the value range of input images, e.g. 0 for uint8 images. If set to None, the value will be automatically derived from the image’s dtype.

- **max_value** (int or float, optional) – Maximum of the value range of input images, e.g. 255 for uint8 images. If set to None, the value will be automatically derived from the image’s dtype.

- **name** (None or str, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **deterministic** (bool, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **random_state** (None or int or numpy.random.RandomState, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

**Examples**

```python
>>> aug = iaa.Invert(0.1)
```

Inverts the colors in 10 percent of all images.

```python
>>> aug = iaa.Invert(0.1, per_channel=0.5)
```

For 50 percent of all images, it inverts all channels with a probability of 10 percent (same as the first example). For the other 50 percent of all images, it inverts each channel individually with a probability of 10 percent (so some channels of an image may end up inverted, others not).

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(*args, **kwargs*)</code></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment_bounding_boxes(bounding_boxes_on_images)</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps(heatmaps[, parents, hooks])</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
<tr>
<td><code>augment_images(images[, parents, hooks])</code></td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td><code>augment_keypoints(keypoints_on_images[, ...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 58 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>augment_segmentation_maps</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([In])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters

ALLOW_DTYPES_CUSTOM_MINMAX = [dtype('uint8'), dtype('uint16'), dtype('uint32'), dtype('float16'), dtype('float32')]

class imgaug.augmenters.arithmetic.JpegCompression(compression=50, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Degrade image quality by applying JPEG compression to it.

During JPEG compression, high frequency components (e.g. edges) are removed. With low compression (strength) only the highest frequency components are removed, while very high compression (strength) will lead to only the lowest frequency components "surviving". This lowers the image quality. For more details, see https://en.wikipedia.org/wiki/Compression_artifact.

Note that this augmenter still returns images as numpy arrays (i.e. saves the images with JPEG compression and then reloads them into arrays). It does not return the raw JPEG file content.

dtype support:
Parameters

- **compression** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Degree of compression used during jpeg compression within value range \([0, 100]\). Higher values denote stronger compression and will cause low-frequency components to disappear. Note that JPEG’s compression strength is also often set as a *quality*, which is the inverse of this parameter. Common choices for the *quality* setting are around 80 to 95, depending on the image. This translates here to a *compression* parameter of around 20 to 5.
  
  - If a single number, then that value will be used for the compression degree.
  
  - If a tuple of two number \((a, b)\), then the compression will be a value sampled from the interval \([a..b]\).
  
  - If a list, then a random value will be sampled and used as the compression per image.
  
  - If a StochasticParameter, then \(N\) samples will be drawn from that parameter per \(N\) input images, each representing the compression for the \(n\)th image. Expected to be discrete.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

Examples

```python
>>> aug = iaa.JpegCompression(compression=(80, 95))
```

Removes high frequency components in images based on JPEG compression with a *compression strength* between 80 and 95 (randomly sampled per image). This corresponds to a (very low) *quality* setting of 5 to 20.

Methods

```python
__call__(*args, **kwargs)
```  

Alias for `imgaug.augmenters.meta.Augmenter.augment()`.  

Continued on next page
### Table 59 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment_bounding_boxes(bounding_boxes_on_images)</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps(heatmaps[, parents, hooks])</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
<tr>
<td><code>augment_images(images[, parents, hooks])</code></td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td><code>augment_keypoints(keypoints_on_images[, ...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
<tr>
<td><code>augment_segmentation_maps(segmaps[, ...])</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive,...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive,...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

#### get_parameters()
Multiply all pixels in an image with a specific value.

This augmenter can be used to make images lighter or darker.

dtype support:

* `uint8`: yes; fully tested
* `uint16`: yes; tested
* `uint32`: no
* `uint64`: no
* `int8`: yes; tested
* `int16`: yes; tested
* `int32`: no
* `int64`: no
* `float16`: yes; tested
* `float32`: yes; tested
* `float64`: no
* `float128`: no
* `bool`: yes; tested

Note: tests were only conducted for rather small multipliers, around -10.0 to +10.

In general, the multipliers sampled from `mul` must be in a value range that corresponds to the input image's dtype. E.g. if the input image has dtype uint16 and the samples generated from `mul` are float64, this augmenter will still force all samples to be within the value range of float16, as it has the same number of bytes (two) as uint16. This is done to make overflows less likely to occur.

**Parameters**

- **mul** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – The value with which to multiply the pixel values in each image.
  - If a number, then that value will always be used.
  - If a tuple (a, b), then a value from the range a <= x <= b will be sampled per image and used for all pixels.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then that parameter will be used to sample a new value per image.

- **per_channel** *(bool or float, optional)* – Whether to use the same multiplier per pixel for all channels (False) or to sample a new value for each channel (True). If this value is a float p, then for p percent of all images per_channel will be treated as True, otherwise as False.

- **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
Examples

```python
>>> aug = iaa.Multiply(2.0)
```

would multiply all images by a factor of 2, making the images significantly brighter.

```python
>>> aug = iaa.Multiply((0.5, 1.5))
```

would multiply images by a random value from the range $0.5 \leq x \leq 1.5$, making some images darker and others brighter.

Methods

```
__call__(*args, **kwargs) Alias for imgaug.augmenters.meta.Augmenter.augment().
```

```
augment([return_batch, hooks]) Augment data.
augment_batch(batch[, hooks]) Augment a single batch.
augment_batches(batches[, hooks, background]) Augment multiple batches.
augment_bounding_boxes(bounding_boxes_on_images) Augment bounding boxes.
augment_heatmaps(heatmaps[, parents, hooks]) Augment a heatmap.
augment_image(image[, hooks]) Augment a single image.
augment_images(images[, parents, hooks]) Augment multiple images.
augment_keypoints(keypoints_on_images[, ...]) Augment image keypoints.
augment_line_strings(line_strings_on_images) Augment line strings.
augment_polygons(polygons_on_images[, ...]) Augment polygons.
augment_segmentation_maps(segmaps[, ...]) Augment segmentation maps.
```

```
copy() Create a shallow copy of this Augmenter instance.
copy_random_state(source[, recursive, ...]) Copy the random states from a source augmenter sequence.
copy_random_state_(source[, recursive, ...]) Copy the random states from a source augmenter sequence (inplace).
depdeepcopy() Create a deep copy of this Augmenter instance.
draw_grid(images, rows, cols) Apply this augmenter to the given images and return a grid image of the results.
find_augmenters(func[, parents, flat]) Find augmenters that match a condition.
find_augmenters_by_name(name[, regex, flat]) Find augmenter(s) by name.
find_augmenters_by_names(names[, regex, flat]) Find augmenter(s) by names.
get_all_children([flat]) Returns all children of this augmenter as a list.
get_children_lists() Get a list of lists of children of this augmenter.
localize_random_state([recursive]) Converts global random states to local ones.
localize_random_state_(recursive) Converts global random states to local ones.
pool([processes, maxtasksperchild, seed]) Create a pool used for multicore augmentation from this augmenter.
remove_augmenters(func[, copy, noop_if_topmost]) Remove this augmenter or its children that match a condition.
```

Continued on next page
Table 60 – continued from previous page

<table>
<thead>
<tr>
<th>remove_augmenters_inplace(func[, parents])</th>
<th>Remove in-place children of this augmenter that match a condition.</th>
</tr>
</thead>
<tbody>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic(n)</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

```python
get_parameters()
```

**class** `imgaug.augmenters.arithmetic.MultiplyElementwise (mul=1.0, per_channel=False, name=None, deterministic=False, random_state=None)`

**Bases:** `imgaug.augmenters.meta.Augmenter`

Multiply values of pixels with possibly different values for neighbouring pixels.

While the Multiply Augmenter uses a constant multiplier per image, this one can use different multipliers per pixel.

**dtype support:**

```
* `'uint8'`: yes; fully tested  
* `'uint16'`: yes; tested  
* `'uint32'`: no  
* `'uint64'`: no  
* `'int8'`: yes; tested  
* `'int16'`: yes; tested  
* `'int32'`: no  
* `'int64'`: no  
* `'float16'`: yes; tested  
* `'float32'`: yes; tested  
* `'float64'`: no  
* `'float128'`: no  
* `'bool'`: yes; tested
```

Note: tests were only conducted for rather small multipliers, around -10.0 to +10.0.

In general, the multipliers sampled from `mul` must be in a value range that corresponds to the input image's dtype. E.g. if the input image has dtype uint16 and the samples generated from `mul` are float64, this augmenter will still force all samples to be within the value range of float16, as it has the same number of bytes (two) as uint16. This is done to make overflows less likely to occur.

**Parameters**
• **mul** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – The value by which to multiply the pixel values in the image.
  – If a number, then that value will always be used.
  – If a tuple (a, b), then a value from the range \( a \leq x \leq b \) will be sampled per image and pixel.
  – If a list, then a random value will be sampled from that list per image.
  – If a StochasticParameter, then that parameter will be used to sample a new value per image and pixel.

• **per_channel** (bool or float, optional) – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \( p \), then for \( p \) percent of all images **per_channel** will be treated as True, otherwise as False.

• **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

• **deterministic** (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

• **random_state** (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> from imgaug import augmenters as iaa
>>> aug = iaa.MultiplyElementwise(2.0)
```

multiply all images by a factor of 2.0, making them significantly brighter.

```python
>>> aug = iaa.MultiplyElementwise((0.5, 1.5))
```

samples per pixel a value from the range \( 0.5 \leq x \leq 1.5 \) and multiplies the pixel with that value.

```python
>>> aug = iaa.MultiplyElementwise((0.5, 1.5), per_channel=True)
```

samples per pixel and channel a value from the range \( 0.5 \leq x \leq 1.5 \) and multiplies the pixel by that value. Therefore, added multipliers may differ between channels of the same pixel.

```python
>>> aug = iaa.MultiplyElementwise((0.5, 1.5), per_channel=0.5)
```

same as previous example, but the **per_channel** feature is only active for 50 percent of all images.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, **kwargs)</td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code>.</td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 61 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>augment_bounding_boxes</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_inplace</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_inplace</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters()  

`get_parameters()`  

`get_parameters()` returns a list of parameters that can be used to configure this augmenter.

Imgaug.augmenters.arithmetic.**Pepper**(*p=0, per_channel=False, name=None, deterministic=False, random_state=None*)

Adds pepper noise to an image, i.e. black-ish pixels.

This is similar to dropout, but slower and the black pixels are not uniformly black.

dtype support:
Parameters

- **p** *(float or tuple of float or list of float or imgaug.parameters.StochasticParameter, optional) – Probability of changing a pixel to pepper noise.*
  - If a float, then that value will be used for all images as the probability.
  - If a tuple \((a, b)\), then a probability will be sampled per image from the range \(a \leq x \leq b\).
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then this parameter will be used as the *mask*, i.e. it is expected to contain values between 0.0 and 1.0, where 1.0 means that pepper is to be added at that location.

- **per_channel** *(bool or float, optional) – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images per_channel will be treated as True, otherwise as False.*

- **name** *(None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().*

- **deterministic** *(bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().*

- **random_state** *(None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().*

Examples

```python
>>> aug = iaa.Pepper(0.05)
```

Replaces 5 percent of all pixels with pepper.

class imgaug.augmenters.arithmetic.ReplaceElementwise(mask, replacement, per_channel=False, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Replace pixels in an image with new values.

dtype support:

- `'uint8'`: yes; fully tested
- `'uint16'`: yes; tested
- `'uint32'`: yes; tested
- `'uint64'`: no (1)
- `'int8'`: yes; tested
- `'int16'`: yes; tested
- `'int32'`: yes; tested
- `'int64'`: yes; tested
- `'float16'`: yes; tested

(continues on next page)
Parameters

- **mask** *(float or tuple of float or list of float or imgaug.parameters.StochasticParameter)* –
  Mask that indicates the pixels that are supposed to be replaced. The mask will be thresholded with 0.5. A value of 1 then indicates a pixel that is supposed to be replaced.
  - If this is a float, then that value will be used as the probability of being a 1 per pixel.
  - If a tuple \((a, b)\), then the probability will be sampled per image from the range \(a \leq x \leq b\).
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then this parameter will be used to sample a mask.

- **replacement** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* –
  The replacement to use at all locations that are marked as 1 in the mask.
  - If this is a number, then that value will always be used as the replacement.
  - If a tuple \((a, b)\), then the replacement will be sampled pixelwise from the range \(a \leq x \leq b\).
  - If a list of number, then a random value will be picked from that list as the replacement per pixel.
  - If a StochasticParameter, then this parameter will be used to sample pixelwise replacement values.

- **per_channel** *(bool or float, optional)* – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images *per channel* will be treated as True, otherwise as False.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

Examples

```python
>>> aug = ReplaceElementwise(0.05, [0, 255])
```

Replace 5 percent of all pixels in each image by either 0 or 255.
### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(*)args, **kwargs)</code></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment_bounding_boxes(bounding_boxes_on_images)</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps(heatmaps[, parents, hooks])</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
<tr>
<td><code>augment_images(images[, parents, hooks])</code></td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td><code>augment keypoints(keypoints_on_images[, ...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
<tr>
<td><code>augment_segmentation_maps(segmaps[, ...])</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive,...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive,...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([in])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>
get_parameters()

`imgaug.augmenters.arithmetic.Salt(p=0, per_channel=False, name=None, deterministic=False, random_state=None)`

Adds salt noise to an image, i.e. white-ish pixels.

dtype support:

See `''imgaug.augmenters.arithmetic.ReplaceElementwise''`.

**Parameters**

- **p** *(float or tuple of float or list of float or imgaug.parameters.StochasticParameter, optional)*
  - Probability of changing a pixel to salt noise.
    - If a float, then that value will be used for all images as the probability.
    - If a tuple \((a, b)\), then a probability will be sampled per image from the range \(a \leq x \leq b\).
    - If a list, then a random value will be sampled from that list per image.
    - If a StochasticParameter, then this parameter will be used as the *mask*, i.e. it is expected to contain values between 0.0 and 1.0, where 1.0 means that salt is to be added at that location.

- **per_channel** *(bool or float, optional)*
  - Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images per_channel will be treated as True, otherwise as False.

- **name** *(None or str, optional)*
  - See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **deterministic** *(bool, optional)*
  - See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **random_state** *(None or int or numpy.random.RandomState, optional)*
  - See `imgaug.augmenters.meta.Augmenter.__init__()`.

**Examples**

```
>>> aug = iaa.Salt(0.05)
```

Replaces 5 percent of all pixels with salt.

`imgaug.augmenters.arithmetic.SaltAndPepper(p=0, per_channel=False, name=None, deterministic=False, random_state=None)`

Adds salt and pepper noise to an image, i.e. some white-ish and black-ish pixels.

dtype support:

See `''imgaug.augmenters.arithmetic.ReplaceElementwise''`.

**Parameters**

- **p** *(float or tuple of float or list of float or imgaug.parameters.StochasticParameter, optional)*
  - Probability of changing a pixel to salt/pepper noise.
- If a float, then that value will be used for all images as the probability.
- If a tuple \((a, b)\), then a probability will be sampled per image from the range \(a \leq x \leq b\).
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then this parameter will be used as the mask, i.e. it is expected to contain values between 0.0 and 1.0, where 1.0 means that salt/pepper is to be added at that location.

- **per_channel** (bool or float, optional) – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images **per_channel** will be treated as True, otherwise as False.
- **name** (None or str, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.
- **deterministic** (bool, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.
- **random_state** (None or int or numpy.random.RandomState, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

Examples

```python
>>> aug = iaa.SaltAndPepper(0.05)
```

Replaces 5 percent of all pixels with salt/pepper.

### 13.16 `imgaug.augmenters.blend`

Augmenters that blend two images with each other.

Do not import directly from this file, as the categorization is not final. Use instead

```python
from imgaug import augmenters as iaa
```

and then e.g.

```python
seq = iaa.Sequential([
    iaa.Alpha(0.5, iaa.Add((-5, 5)))
])
```

List of augmenters:

- Alpha
- AlphaElementwise
- SimplexNoiseAlpha
- FrequencyNoiseAlpha

```python
class imgaug.augmenters.blend.Alpha(factor=0, first=None, second=None, per_channel=False, name=None, deterministic=False, random_state=None)
```

Augmenter to blend two image sources using an alpha/transparency value.
The two image sources can be imagined as branches. If a source is not given, it is automatically the same as the input. Let A be the first branch and B be the second branch. Then the result images are defined as \( \text{factor} \times A + (1-\text{factor}) \times B \), where \( \text{factor} \) is an overlay factor.

For keypoint augmentation this augmenter will pick the keypoints either from the first or the second branch. The first one is picked if \( \text{factor} \geq 0.5 \) is true (per image). It is recommended to \textit{not} use augmenters that change keypoint positions with this class.

dtype support:

> See :func:`imgaug.augmenters.blend.blend_alpha`.

**Parameters**

- **factor** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Weighting of the results of the first branch. Values close to 0 mean that the results from the second branch (see parameter \( \text{second} \)) make up most of the final image.
  - If float, then that value will be used for all images.
  - If tuple \((a, b)\), then a random value from range \(a \leq x \leq b\) will be sampled per image.
  - If a list, then a random value will be picked from that list per image.
  - If StochasticParameter, then that parameter will be used to sample a value per image.

- **first** *(None or imgaug.augmenters.meta.Augmenter or iterable of imgaug.augmenters.meta.Augmenter, optional)* – Augmenter(s) that make up the first of the two branches.
  - If None, then the input images will be reused as the output of the first branch.
  - If Augmenter, then that augmenter will be used as the branch.
  - If iterable of Augmenter, then that iterable will be converted into a Sequential and used as the augmenter.

- **second** *(None or imgaug.augmenters.meta.Augmenter or iterable of imgaug.augmenters.meta.Augmenter, optional)* – Augmenter(s) that make up the second of the two branches.
  - If None, then the input images will be reused as the output of the second branch.
  - If Augmenter, then that augmenter will be used as the branch.
  - If iterable of Augmenter, then that iterable will be converted into a Sequential and used as the augmenter.

- **per_channel** *(bool or float, optional)* – Whether to use the same factor for all channels (False) or to sample a new value for each channel (True). If this value is a float \( p \), then for \( p \) percent of all images \( \text{per_channel} \) will be treated as True, otherwise as False.

- **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
### Examples

```python
>>> aug = iaa.Alpha(0.5, iaa.Grayscale(1.0))
```

Converts each image to grayscale and alpha-blends it by 50 percent with the original image, thereby removing about 50 percent of all color. This is equivalent to `iaa.Grayscale(0.5)`.

```python
>>> aug = iaa.Alpha((0.0, 1.0), iaa.Grayscale(1.0))
```

Converts each image to grayscale and alpha-blends it by a random percentage (sampled per image) with the original image, thereby removing a random percentage of all colors. This is equivalent to `iaa.Grayscale((0.0, 1.0))`.

```python
>>> aug = iaa.Alpha((0.0, 1.0), iaa.Affine(rotate=(-20, 20)), per_channel=0.5)
```

Rotates each image by a random degree from the range \([-20, 20]\). Then alpha-blends that new image with the original one by a random factor from the range \([0.0, 1.0]\). In 50 percent of all cases, the blending happens channel-wise and the factor is sampled independently per channel. As a result, e.g. the red channel may look visible rotated (factor near 1.0), while the green and blue channels may not look rotated (factors near 0.0). NOTE: It is not recommended to use Alpha with augmenters that change the positions of pixels if you also want to augment keypoints, as it is unclear which of the two keypoint results (first or second branch) should be used as the final result.

```python
>>> aug = iaa.Alpha((0.0, 1.0), first=iaa.Add(10), second=iaa.Multiply(0.8))
```

(A) Adds 10 to each image and (B) multiplies each image by 0.8. Then per image a blending factor is sampled from the range \([0.0, 1.0]\). If it is close to 1.0, the results from (A) are mostly used, otherwise the ones from (B). This is equivalent to `iaa.Sequential([iaa.Multiply(0.8), iaa.Alpha((0.0, 1.0), iaa.Add(10))])`.

```python
>>> aug = iaa.Alpha(iap.Choice([0.25, 0.75]), iaa.MedianBlur((3, 7)))
```

Applies a random median blur to each image and alpha-blends the result with the original image by either 25 or 75 percent strength.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `__call__(*args, **kwargs)` | Alias for `imgaug.augmenters.meta.Augmenter.augment()`.
| `augment([return_batch, hooks])` | Augment data. |
| `augment_batch(batch[, hooks])` | Augment a single batch. |
| `augment_batches(batches[, hooks, background])` | Augment multiple batches. |
| `augment_bounding_boxes(bounding_boxes_on_images)` | Augment bounding boxes. |
| `augment_heatmaps(heatmaps[, parents, hooks])` | Augment a heatmap. |
| `augment_image(image[, hooks])` | Augment a single image. |
| `augment_images(images[, parents, hooks])` | Augment multiple images. |
| `augment_keypoints(keypoints_on_images[, ...])` | Augment image keypoints. |
| `augment_line_strings(line_strings_on_images)` | Augment line strings. |
| `augment_polygons(polygons_on_images[, ...])` | Augment polygons. |
Table 63 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive,...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive,...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([, flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy,noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic()</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_parameters**

**get_children_lists()**

Get a list of lists of children of this augmenter.

For most augmenters, the result will be a single empty list. For augmenters with children it will often be a list with one sublist containing all children. In some cases the augmenter will contain multiple distinct lists of children, e.g. an if-list and an else-list. This will lead to a result consisting of a single list with multiple sublists, each representing the respective sublist of children.

E.g. for an if/else-augmenter that executes the children $A1,A2$ if a condition is met and otherwise executes the children $B1,B2,B3$ the result will be $[[A1, A2], [B1, B2, B3]]$.

IMPORTANT: While the topmost list may be newly created, each of the sublist must be editable inplace resulting in a changed children list of the augmenter. E.g. if an Augmenter IfElse(condition, [A1, A2], [B1, B2, B3]) returns $[[A1, A2], [B1, B2, B3]]$ for a call to imgaug.augmenters.meta.Augmenter.get_children_lists() and A2 is removed inplace from [A1, A2], then the children lists of IfElse(...) must also change to [A1], [B1, B2, B3]. This is used in imgaug.augmenters.meta.Augmenter.remove_augmenters_inplace().
Returns children – One or more lists of child augmenter. Can also be a single empty list.

Return type list of list of imgaug.augmenters.meta.Augmenter

get_parameters()

class imgaug.augmenters.blend.AlphaElementwise(factor=0, first=None, second=None, per_channel=False, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.blend.Alpha

Augmenter to blend two image sources pixelwise alpha/transparency values.

This is the same as Alpha, except that the transparency factor is sampled per pixel instead of once per image (or a few times per image, if per_channel is True).

See Alpha for more description.

dtype support:

See :func:`imgaug.augmenters.blend.blend_alpha`.

Parameters

• factor (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – Weighting of the results of the first branch. Values close to 0 mean that the results from the second branch (see parameter second) make up most of the final image.
  – If float, then that value will be used for all images.
  – If tuple (a, b), then a random value from range \(a \leq x \leq b\) will be sampled per image.
  – If a list, then a random value will be picked from that list per image.
  – If StochasticParameter, then that parameter will be used to sample a value per image.

• first (None or imgaug.augmenters.meta.Augmenter or iterable of imgaug.augmenters.meta.Augmenter, optional) –
  Augmenter(s) that make up the first of the two branches.
  – If None, then the input images will be reused as the output of the first branch.
  – If Augmenter, then that augmenter will be used as the branch.
  – If iterable of Augmenter, then that iterable will be converted into a Sequential and used as the augmenter.

• second (None or imgaug.augmenters.meta.Augmenter or iterable of imgaug.augmenters.meta.Augmenter, optional) –
  Augmenter(s) that make up the second of the two branches.
  – If None, then the input images will be reused as the output of the second branch.
  – If Augmenter, then that augmenter will be used as the branch.
  – If iterable of Augmenter, then that iterable will be converted into a Sequential and used as the augmenter.

• per_channel (bool or float, optional) – Whether to use the same factor for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images per_channel will be treated as True, otherwise as False.
• **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  
• **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  
• **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

**Examples**

```python
>>> aug = iaa.AlphaElementwise(0.5, iaa.Grayscale(1.0))
```

Converts each image to grayscale and overlays it by 50 percent with the original image, thereby removing about 50 percent of all color. This is equivalent to `iaa.Grayscale(0.5)`. This is also equivalent to `iaa.Alpha(0.5, iaa.Grayscale(1.0))`, as the transparency factor is the same for all pixels.

```python
>>> aug = iaa.AlphaElementwise((0, 1.0), iaa.Grayscale(1.0))
```

Converts each image to grayscale and alpha-blends it by a random percentage (sampled per pixel) with the original image, thereby removing a random percentage of all colors per pixel.

```python
>>> aug = iaa.AlphaElementwise((0.0, 1.0), iaa.Affine(rotate=(-20, 20)), per_channel=0.5)
```

Rotates each image by a random degree from the range \([-20, 20]\). Then alpha-blends that new image with the original one by a random factor from the range \([0.0, 1.0]\), sampled per pixel. In 50 percent of all cases, the blending happens channel-wise and the factor is sampled independently per channel. As a result, e.g. the red channel may look visible rotated (factor near 1.0), while the green and blue channels may not look rotated (factors near 0.0). NOTE: It is not recommended to use Alpha with augmenters that change the positions of pixels if you also want to augment keypoints, as it is unclear which of the two keypoint results (first or second branch) should be used as the final result.

```python
>>> aug = iaa.AlphaElementwise((0.0, 1.0), first=iaa.Add(10), second=iaa.Multiply(0.8))
```

(A) Adds 10 to each image and (B) multiplies each image by 0.8. Then per pixel a blending factor is sampled from the range \([0.0, 1.0]\). If it is close to 1.0, the results from (A) are mostly used, otherwise the ones from (B).

```python
>>> aug = iaa.AlphaElementwise(iap.Choice([0.25, 0.75]), iaa.MedianBlur((3, 7)))
```

Applies a random median blur to each image and alpha-blends the result with the original image by either 25 or 75 percent strength (sampled per pixel).

**Methods**

```python
__call__(*args, **kwargs) 
```

Alias for `imgaug.augmenters.meta.Augmenter.augment()`.  

```python
augment([return_batch, hooks])
```

Augment data.  

```python
augment_batch(batch[, hooks])
```

Augment a single batch.  

```python
augment_batches(batches[, hooks, background])
```

Augment multiple batches.
Table 64 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>augment_bounding_boxes</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
<tr>
<td><code>augment_images(images[, parents, hooks])</code></td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td><code>augment_keypoints(keypoints_on_images[, ...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
<tr>
<td><code>augment_segmentation_maps(segmaps[, ...])</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

```
imgaug.augmenters.blend.FrequencyNoiseAlpha(exponent=(-4, 4), first=None, second=None, per_channel=False, size_px_max=(4, 16), upscale_method=None, iterations=(1, 3), aggregation_method=['avg', 'max'], sigmoid=0.5, sigmoid_thresh=None, name=None, deterministic=False, random_state=None)

Augmenter to alpha-blend two image sources using frequency noise masks.
```
The alpha masks are sampled using frequency noise of varying scales, which can sometimes create large connected blobs of 1s surrounded by 0s and other times results in smaller patterns. If nearest neighbour upsampling is used, these blobs can be rectangular with sharp edges.

dtype support:

See `imgaug.augmenters.blend.AlphaElementwise`.

**Parameters**

- **exponent** *(number or tuple of number of list of number or imgaug.parameters.StochasticParameter, optional)* – Exponent to use when scaling in the frequency domain. Sane values are in the range -4 (large blobs) to 4 (small patterns). To generate cloud-like structures, use roughly -2.
  - If number, then that number will be used as the exponent for all iterations.
  - If tuple of two numbers \((a, b)\), then a value will be sampled per iteration from the range \([a, b]\).
  - If a list of numbers, then a value will be picked per iteration at random from that list.
  - If a StochasticParameter, then a value will be sampled from that parameter per iteration.

- **first** *(None or imgaug.augmenters.meta.Augmenter or iterable of imgaug.augmenters.meta.Augmenter, optional)* – Augmenter(s) that make up the first of the two branches.
  - If None, then the input images will be reused as the output of the first branch.
  - If Augmenter, then that augmenter will be used as the branch.
  - If iterable of Augmenter, then that iterable will be converted into a Sequential and used as the augmenter.

- **second** *(None or imgaug.augmenters.meta.Augmenter or iterable of imgaug.augmenters.meta.Augmenter, optional)* – Augmenter(s) that make up the second of the two branches.
  - If None, then the input images will be reused as the output of the second branch.
  - If Augmenter, then that augmenter will be used as the branch.
  - If iterable of Augmenter, then that iterable will be converted into a Sequential and used as the augmenter.

- **per_channel** *(bool or float, optional)* – Whether to use the same factor for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images per channel will be treated as True, otherwise as False.

- **size_px_max** *(int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional)* – The noise is generated in a low resolution environment. This parameter defines the maximum size of that environment (in pixels). The environment is initialized at the same size as the input image and then downscaled, so that no side exceeds \(size\_px\_max\) (aspect ratio is kept).
  - If int, then that number will be used as the size for all iterations.
  - If tuple of two ints \((a, b)\), then a value will be sampled per iteration from the discrete range \([a..b]\).
  - If a list of ints, then a value will be picked per iteration at random from that list.
If a StochasticParameter, then a value will be sampled from that parameter per iteration.

**upscale_method** *(None or imgaug.ALL or str or list of str or imgaug.parameters.StochasticParameter, optional) –* After generating the noise maps in low resolution environments, they have to be upscaled to the input image size. This parameter controls the upscaling method.

- If None, then either nearest or linear or cubic is picked. Most weight is put on linear, followed by cubic.
- If imgaug.ALL, then either nearest or linear or area or cubic is picked per iteration (all same probability).
- If string, then that value will be used as the method (must be nearest or linear or area or cubic).
- If list of string, then a random value will be picked from that list per iteration.
- If StochasticParameter, then a random value will be sampled from that parameter per iteration.

**iterations** *(int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional) –* How often to repeat the simplex noise generation process per image.

- If int, then that number will be used as the iterations for all images.
- If tuple of two ints \((a, b)\), then a value will be sampled per image from the discrete range \([a..b]\).
- If a list of ints, then a value will be picked per image at random from that list.
- If a StochasticParameter, then a value will be sampled from that parameter per image.

**aggregation_method** *(imgaug.ALL or str or list of str or imgaug.parameters.StochasticParameter, optional) –* The noise maps (from each iteration) are combined to one noise map using an aggregation process. This parameter defines the method used for that process. Valid methods are min, max or avg, where ‘min’ combines the noise maps by taking the (elementwise) minimum over all iteration’s results, max the (elementwise) maximum and avg the (elementwise) average.

- If imgaug.ALL, then a random value will be picked per image from the valid ones.
- If a string, then that value will always be used as the method.
- If a list of string, then a random value will be picked from that list per image.
- If a StochasticParameter, then a random value will be sampled from that parameter per image.

**sigmoid** *(bool or number, optional) –* Whether to apply a sigmoid function to the final noise maps, resulting in maps that have more extreme values (close to 0.0 or 1.0).

- If bool, then a sigmoid will always (True) or never (False) be applied.
- If a number \(p\) with \(0\leq p\leq 1\), then a sigmoid will be applied to \(p\) percent of all final noise maps.

**sigmoid_thresh** *(None or number or tuple of number or imgaug.parameters.StochasticParameter, optional) –* Threshold of the sigmoid, when applied. Thresholds above zero (e.g. 5.0) will move the saddle point towards the right, leading to more values close to 0.0.

- If None, then Normal(0, 5.0) will be used.
- If number, then that threshold will be used for all images.
- If tuple of two numbers \((a, \ b)\), then a random value will be sampled per image from the range \([a, \ b]\).

- If StochasticParameter, then a random value will be sampled from that parameter per image.

  - **name** *(None or str, optional)* — See `imgaug.augmenters.meta.Augmenter.__init__()`

  - **deterministic** *(bool, optional)* — See `imgaug.augmenters.meta.Augmenter.__init__()`

  - **random_state** *(None or int or numpy.random.RandomState, optional)* — See `imgaug.augmenters.meta.Augmenter.__init__()`

### Examples

```python
>>> aug = iaa.FrequencyNoiseAlpha(first=iaa.EdgeDetect(1.0))
```

Detects per image all edges, marks them in a black and white image and then alpha-blends the result with the original image using frequency noise masks.

```python
>>> aug = iaa.FrequencyNoiseAlpha(first=iaa.EdgeDetect(1.0), upscale_method="linear")
```

Same as the first example, but uses only (smooth) linear upscaling to scale the frequency noise masks to the final image sizes, i.e. no nearest neighbour upsampling is used, which would result in rectangles with hard edges.

```python
>>> aug = iaa.FrequencyNoiseAlpha(first=iaa.EdgeDetect(1.0), upscale_method="linear", exponent=-2, sigmoid=False)
```

Same as the previous example, but also limits the exponent to -2 and deactivates the sigmoid, resulting in cloud-like patterns without sharp edges.

```python
>>> aug = iaa.FrequencyNoiseAlpha(first=iaa.EdgeDetect(1.0), sigmoid_thresh=iap.Normal(10.0, 5.0))
```

Same as the first example, but uses a threshold for the sigmoid function that is further to the right. This is more conservative, i.e. the generated noise masks will be mostly black (values around 0.0), which means that most of the original images (parameter/branch `second`) will be kept, rather than using the results of the augmentation (parameter/branch `first`).

### imgaug.augmenters.blend.SimplexNoiseAlpha

```python
imgaug.augmenters.blend.SimplexNoiseAlpha(first=None, second=None, per_channel=False, size_px_max=(2, 16), upscale_method=None, iterations=(1, 3), aggregation_method='max', sigmoid=True, sigmoid_threshold=None, name=None, deterministic=False, random_state=None)
```

Augmenter to alpha-blend two image sources using simplex noise alpha masks.

The alpha masks are sampled using a simplex noise method, roughly creating connected blobs of 1s surrounded by 0s. If nearest neighbour upsampling is used, these blobs can be rectangular with sharp edges.

**dtype support:**

See `"imgaug.augmenters.blend.AlphaElementwise"`.

### Parameters
• **first** (None or imgaug.augmenters.meta.Augmenter or iterable of imgaug.augmenters.meta.Augmenter, optional) –
  Augmenter(s) that make up the first of the two branches.
  – If None, then the input images will be reused as the output of the first branch.
  – If Augmenter, then that augmenter will be used as the branch.
  – If iterable of Augmenter, then that iterable will be converted into a Sequential and used as the augmenter.

• **second** (None or imgaug.augmenters.meta.Augmenter or iterable of imgaug.augmenters.meta.Augmenter, optional) –
  Augmenter(s) that make up the second of the two branches.
  – If None, then the input images will be reused as the output of the second branch.
  – If Augmenter, then that augmenter will be used as the branch.
  – If iterable of Augmenter, then that iterable will be converted into a Sequential and used as the augmenter.

• **per_channel** (bool or float, optional) – Whether to use the same factor for all channels (False) or to sample a new value for each channel (True). If this value is a float \( p \), then for \( p \) percent of all images **per_channel** will be treated as True, otherwise as False.

• **size_px_max** (int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional) – The simplex noise is always generated in a low resolution environment. This parameter defines the maximum size of that environment (in pixels). The environment is initialized at the same size as the input image and then downscaled, so that no side exceeds **size_px_max** (aspect ratio is kept).
  – If int, then that number will be used as the size for all iterations.
  – If tuple of two ints \((a, b)\), then a value will be sampled per iteration from the discrete range \([a..b]\).
  – If a list of ints, then a value will be picked per iteration at random from that list.
  – If a StochasticParameter, then a value will be sampled from that parameter per iteration.

• **upscale_method** (None or imgaug.ALL or str or list of str or imgaug.parameters.StochasticParameter, optional) – After generating the noise maps in low resolution environments, they have to be upsampled to the input image size. This parameter controls the upscaling method.
  – If None, then either nearest or linear or cubic is picked. Most weight is put on linear, followed by cubic.
  – If ia.ALL, then either nearest or linear or area or cubic is picked per iteration (all same probability).
  – If string, then that value will be used as the method (must be ‘nearest’ or linear or area or cubic).
  – If list of string, then a random value will be picked from that list per iteration.
  – If StochasticParameter, then a random value will be sampled from that parameter per iteration.

• **iterations** (int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional) –
How often to repeat the simplex noise generation process per image.

- If int, then that number will be used as the iterations for all images.
- If tuple of two ints \((a, b)\), then a value will be sampled per image from the discrete range \([a..b]\).
- If a list of ints, then a value will be picked per image at random from that list.
- If a StochasticParameter, then a value will be sampled from that parameter per image.

- **aggregation_method** (\texttt{imgaug.ALL} or \texttt{str} or \texttt{list of str} or \texttt{imgaug.parameters.StochasticParameter}, optional) – The noise maps (from each iteration) are combined to one noise map using an aggregation process. This parameter defines the method used for that process. Valid methods are \texttt{min}, \texttt{max} or \texttt{avg}, where \texttt{min} combines the noise maps by taking the (elementwise) minimum over all iteration’s results, \texttt{max} the (elementwise) maximum and \texttt{avg} the (elementwise) average.
  - If \texttt{imgaug.ALL}, then a random value will be picked per image from the valid ones.
  - If a string, then that value will always be used as the method.
  - If a list of string, then a random value will be picked from that list per image.
  - If a StochasticParameter, then a random value will be sampled from that parameter per image.

- **sigmoid** (\texttt{bool} or \texttt{number}, optional) – Whether to apply a sigmoid function to the final noise maps, resulting in maps that have more extreme values (close to 0.0 or 1.0).
  - If bool, then a sigmoid will always (True) or never (False) be applied.
  - If a number \(p\) with \(0 \leq p \leq 1\), then a sigmoid will be applied to \(p\) percent of all final noise maps.

- **sigmoid_thresh** (\texttt{None} or \texttt{number} or \texttt{tuple of number} or \texttt{imgaug.parameters.StochasticParameter}, optional) – Threshold of the sigmoid, when applied. Thresholds above zero (e.g. 5.0) will move the saddle point towards the right, leading to more values close to 0.0.
  - If \texttt{None}, then \texttt{Normal(0, 5.0)} will be used.
  - If \texttt{number}, then that threshold will be used for all images.
  - If \texttt{tuple of two numbers} \((a, b)\), then a random value will be sampled per image from the range \([a, b]\).
  - If \texttt{StochasticParameter}, then a random value will be sampled from that parameter per image.

- **name** (\texttt{None} or \texttt{str}, optional) – See \texttt{imgaug.augmenters.meta.Augmenter.__init__()}

- **deterministic** (\texttt{bool}, optional) – See \texttt{imgaug.augmenters.meta.Augmenter.__init__()}

- **random_state** (\texttt{None} or \texttt{int} or \texttt{numpy.random.RandomState}, optional) – See \texttt{imgaug.augmenters.meta.Augmenter.__init__()}

**Examples**

```python
>>> aug = iaa.SimplexNoiseAlpha(iaa.EdgeDetect(1.0))
```
Detects per image all edges, marks them in a black and white image and then alpha-blends the result with the original image using simplex noise masks.

```python
>>> aug = iaa.SimplexNoiseAlpha(iaa.EdgeDetect(1.0), upscale_method="linear")
```

Same as the first example, but uses only (smooth) linear upscaling to scale the simplex noise masks to the final image sizes, i.e. no nearest neighbour upsampling is used, which would result in rectangles with hard edges.

```python
>>> aug = iaa.SimplexNoiseAlpha(iaa.EdgeDetect(1.0), sigmoid_thresh=iaa.Normal(10.0, 5.0))
```

Same as the first example, but uses a threshold for the sigmoid function that is further to the right. This is more conservative, i.e. the generated noise masks will be mostly black (values around 0.0), which means that most of the original images (parameter/branch second) will be kept, rather than using the results of the augmentation (parameter/branch first).

```
imgaug.augmenters.blend.blend_alpha(image_fg, image_bg, alpha, eps=0.01)
```

Blend two images using an alpha blending.

In an alpha blending, the two images are naively mixed. Let \( A \) be the foreground image and \( B \) the background image and \( a \) is the alpha value. Each pixel intensity is then computed as \( a \cdot A_{ij} + (1-a) \cdot B_{ij} \).

### Parameters

- **image_fg** ((\( H,W,C \)) \( \text{ndarray} \)) – Foreground image. Shape and dtype kind must match the one of the background image.
- **image_bg** ((\( H,W,C \)) \( \text{ndarray} \)) – Background image. Shape and dtype kind must match the one of the foreground image.
- **alpha** (number or iterable of number or \( \text{ndarray} \)) – The blending factor, between 0.0 and 1.0. Can be interpreted as the opacity of the foreground image. Values around 1.0 result in...
only the foreground image being visible. Values around 0.0 result in only the background image being visible. Multiple alphas may be provided. In these cases, there must be exactly one alpha per channel in the foreground/background image. Alternatively, for \((H, W, C)\) images, either one \((H, W)\) array or an \((H, W, C)\) array of alphas may be provided, denoting the elementwise alpha value.

- **\(\text{eps} \) (number, optional)** – Controls when an alpha is to be interpreted as exactly 1.0 or exactly 0.0, resulting in only the foreground/background being visible and skipping the actual computation.

**Returns** \(\text{image\_blend}\) – Blend of foreground and background image.

**Return type** \((H, W, C)\) ndarray

### 13.17 imgaug.augmenters.blur

Augmenters that blur images.

Do not import directly from this file, as the categorization is not final. Use instead

```python
from imgaug import augmenters as iaa
```

and then e.g.

```python
seq = iaa.Sequential(
    [iaa.GaussianBlur((0.0, 3.0)),
     iaa.AverageBlur((2, 5))]
)
```

List of augmenters:

- GaussianBlur
- AverageBlur
- MedianBlur
- BilateralBlur
- MotionBlur

**class** `imgaug.augmenters.blur.AverageBlur(k=1, name=None, deterministic=False, random_state=None)`

Bases: `imgaug.augmenters.meta.Augmenter`

Blur an image by computing simple means over neighbourhoods.

The padding behaviour around the image borders is cv2’s BORDER_REFLECT_101.

dtype support:

- ```uint8```: yes; fully tested
- ```uint16```: yes; tested
- ```uint32```: no (1)
- ```uint64```: no (2)
- ```int8```: yes; tested (3)
- ```int16```: yes; tested
- ```int32```: no (4)
- ```int64```: no (5)
- ```float16```: yes; tested (6)

(continues on next page)
Parameters

- **k (int or tuple of int or tuple of tuple of int or imgaug.parameters.StochasticParameter or tuple of StochasticParameter, optional)** – Kernel size to use.
  - If a single int, then that value will be used for the height and width of the kernel.
  - If a tuple of two ints \((a, b)\), then the kernel size will be sampled from the interval \([a..b]\).
  - If a tuple of two tuples of ints \(((a, b), (c, d))\), then per image a random kernel height will be sampled from the interval \([a..b]\) and a random kernel width will be sampled from the interval \([c..d]\).
  - If a StochasticParameter, then \(N\) samples will be drawn from that parameter per \(N\) input images, each representing the kernel size for the nth image.
  - If a tuple \((a, b)\), where either \(a\) or \(b\) is a tuple, then \(a\) and \(b\) will be treated according to the rules above. This leads to different values for height and width of the kernel.

- **name (None or str, optional)** – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **deterministic (bool, optional)** – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **random_state (None or int or numpy.random.RandomState, optional)** – See `imgaug.augmenters.meta.Augmenter.__init__()`.

Examples

```python
>>> aug = iaa.AverageBlur(k=5)
```
Blurs all images using a kernel size of 5x5.

```python
>>> aug = iaa.AverageBlur(k=(2, 5))
```
Blurs images using a varying kernel size per image, which is sampled from the interval \([2..5]\).
Blurs images using a varying kernel size per image, which's height is sampled from the interval [5..7] and which's width is sampled from [1..3].

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, **kwargs)</td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[,...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[,...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[,...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive,...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive,...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children()[flat]</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
</tbody>
</table>
Table 65 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

```python
get_parameters()
```

class `imgaug.augmenters.blur.BilateralBlur` (``d=1``, `sigma_color=(10, 250)`, `sigma_space=(10, 250)`, `name=None`, `deterministic=False`, `random_state=None`)

Blur/Denoise an image using a bilateral filter.

Bilateral filters blur homogenous and textured areas, while trying to preserve edges.

See [http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#bilateralfilter](http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#bilateralfilter) for more information regarding the parameters.

dtype support:

```
* `'uint8'`: yes; not tested
* `'uint16'`: ?
* `'uint32'`: ?
* `'uint64'`: ?
* `'int8'`: ?
* `'int16'`: ?
* `'int32'`: ?
* `'int64'`: ?
* `'float16'`: ?
* `'float32'`: ?
* `'float64'`: ?
* `'float128'`: ?
* `'bool'`: ?
```

Parameters

- **d** (int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional) – Diameter of each pixel neighborhood with value range \([1 .. \infty)\). High values for d lead to significantly worse performance. Values equal or less than 10 seem to be good. Use ≤5 for real-time applications.
  - If a single int, then that value will be used for the diameter.
  - If a tuple of two ints \((a, b)\), then the diameter will be a value sampled from the interval \([a..b]\).
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then \(N\) samples will be drawn from that parameter per \(N\) input images, each representing the diameter for the nth image. Expected to be discrete.

- **sigma_color** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – Filter sigma in the color space with value range \([1, \infty)\). A larger value of the parameter means that farther colors within
the pixel neighborhood (see sigma_space) will be mixed together, resulting in larger areas of semi-equal color.

- If a single int, then that value will be used for the diameter.
- If a tuple of two ints \((a, b)\), then the diameter will be a value sampled from the interval \([a, b]\).
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then \(N\) samples will be drawn from that parameter per \(N\) input images, each representing the diameter for the \(n\)th image. Expected to be discrete.

- **sigma_space** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Filter sigma in the coordinate space with value range \([1, \inf)\). A larger value of the parameter means that farther pixels will influence each other as long as their colors are close enough (see sigma_color).

  - If a single int, then that value will be used for the diameter.
  - If a tuple of two ints \((a, b)\), then the diameter will be a value sampled from the interval \([a, b]\).
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then \(N\) samples will be drawn from that parameter per \(N\) input images, each representing the diameter for the \(n\)th image. Expected to be discrete.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

**Examples**

```python
>>> aug = iaa.BilateralBlur(d=(3, 10), sigma_color=(10, 250), sigma_space=(10, 250))
```

blurs all images using a bilateral filter with max distance 3 to 10 and wide ranges for sigma_color and sigma_space.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(**args, **kwargs)</code></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment_bounding_boxes(bounding_boxes_on_images)</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps(heatmaps[, parents, hooks])</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 66 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>augment_images</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_inplace</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_inplace</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters()

class imgaug.augmenters.blur.GaussianBlur(sigma=0, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter to blur images using gaussian kernels.

dtype support:

See :func:`imgaug.augmenters.blur.blur_gaussian_(backend="auto")`.

Parameters
• **sigma** *(number or tuple of number or list of number or im-
  mgaug.parameters.StochasticParameter, optional)* – Standard deviation of the gaussian
  kernel. Values in the range 0.0 (no blur) to 3.0 (strong blur) are common.

  – If a single float, that value will always be used as the standard deviation.
  – If a tuple \((a, b)\), then a random value from the range \(a <= x <= b\) will be picked
    per image.
  – If a list, then a random value will be sampled per image from that list.
  – If a StochasticParameter, then \(N\) samples will be drawn from that parameter per \(N\) input
    images.

• **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.
  __init__()_.

• **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.
  __init__()_.

• **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.
  augmenters.meta.Augmenter.__init__()_.

Examples

```python
>>> aug = iaa.GaussianBlur(sigma=1.5)
```

blurs all images using a gaussian kernel with standard deviation 1.5.

```python
>>> aug = iaa.GaussianBlur(sigma=(0.0, 3.0))
```

blurs images using a gaussian kernel with a random standard deviation from the range \(0.0 <= x <= 3.0\).
The value is sampled per image.

Methods

```python
__call__(*args, **kwargs)*
```

Alias for `imgaug.augmenters.meta.Augmenter.augment()`.

```python
augment([return_batch, hooks])
```

Augment data.

```python
augment_batch(batch[, hooks])
```

Augment a single batch.

```python
augment_batches(batches[, hooks, background])
```

Augment multiple batches.

```python
augment_bounding_boxes(bounding_boxes_on_images)
```

Augment bounding boxes.

```python
augment_heatmaps(heatmaps[, parents, hooks])
```

Augment a heatmap.

```python
augment_image(image[, hooks])
```

Augment a single image.

```python
augment_images(images[, parents, hooks])
```

Augment multiple images.

```python
augment_keypoints(keypoints_on_images[, ...
```

Augment image keypoints.

```python
augment_line_strings(line_strings_on_images)
```

Augment line strings.

```python
augment_polygons(polygons_on_images[, ...
```

Augment polygons.

```python
augment_segmentation_maps(segments[, ...
```

Augment segmentation maps.

```python
copy()
```

Create a shallow copy of this Augmenter instance.

Continued on next page
Table 67 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy_random_state(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_(recursive)</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Rseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

### get_parameters()

**class** `imgaug.augmenters.blur.MedianBlur(k=1, name=None, deterministic=False, random_state=None)`

**Bases:** `imgaug.augmenters.meta.Augmenter`

Blur an image by computing median values over neighbourhoods.

Median blurring can be used to remove small dirt from images. At larger kernel sizes, its effects have some similarity with Superpixels.

**dtype support:**

- ``uint8``: yes; fully tested
- ``uint16``: ?
- ``uint32``: ?
- ``uint64``: ?
- ``int8``: ?
- ``int16``: ?
- ``int32``: ?
- ``int64``: ?
- ``float16``: ?
- ``float32``: ?

(continues on next page)
Parameters

- **k** *(int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional)* –
  Kernel size.
  - If a single int, then that value will be used for the height and width of the kernel. Must be an odd value.
  - If a tuple of two ints \((a, b)\), then the kernel size will be an odd value sampled from the interval \([a..b]\). \(a\) and \(b\) must both be odd values.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then \(N\) samples will be drawn from that parameter per \(N\) input images, each representing the kernel size for the \(n\)th image. Expected to be discrete. If a sampled value is not odd, then that value will be increased by 1.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

Examples

```python
>>> aug = iaa.MedianBlur(k=5)
```

blurs all images using a kernel size of 5x5.  

```python
>>> aug = iaa.MedianBlur(k=(3, 7))
```

blurs images using a varying kernel size per image, which is and odd value sampled from the interval \([3..7]\), i.e. 3 or 5 or 7.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(*args, **kwargs)</code></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
</tbody>
</table>

(continued on next page)
### Table 68 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>augment_keypoints(keypoints_on_images[, ...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
<tr>
<td><code>augment_segmentation_maps(segmaps[, ...])</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

### get_parameters

```python
imgaug.augmenters.blur.MotionBlur(k=5, angle=(0, 360), direction=(-1.0, 1.0), order=1, name=None, deterministic=False, random_state=None)
```

Augmenter that sharpens images and overlays the result with the original image.

dtype support:

See ```imgaug.augmenters.convolutional.Convolve```.

**Parameters**

- `k` *(int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional)* –
Kernel size to use.

- If a single int, then that value will be used for the height and width of the kernel.
- If a tuple of two ints \((a, b)\), then the kernel size will be sampled from the interval \([a..b]\).
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then \(N\) samples will be drawn from that parameter per \(N\) input images, each representing the kernel size for the \(n\)th image.

- **angle** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* –
  Angle of the motion blur in degrees (clockwise, relative to top center direction).
  - If a number, exactly that value will be used.
  - If a tuple \((a, b)\), a random value from the range \(a <= x <= b\) will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **direction** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* –
  Forward/backward direction of the motion blur. Lower values towards -1.0 will point the motion blur towards the back (with angle provided via `angle`). Higher values towards 1.0 will point the motion blur forward. A value of 0.0 leads to a uniformly (but still angled) motion blur.
  - If a number, exactly that value will be used.
  - If a tuple \((a, b)\), a random value from the range \(a <= x <= b\) will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **order** *(int or iterable of int or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* –
  Interpolation order to use when rotating the kernel according to `angle`. See imgaug.augmenters.geometric.Affine.__init__(). Recommended to be 0 or 1, with 0 being faster, but less continuous/smooth as `angle` is changed, particularly around multiple of 45 degrees.

- **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
- **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
- **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> aug = iaa.MotionBlur(k=15)
```

Create a motion blur augmenter with kernel size of 15x15.
Create a motion blur augmenter with kernel size of 15x15 and a blur angle of either -45 or 45 degrees (randomly picked per image).

```python
>>> aug = iaa.MotionBlur(k=15, angle=[-45, 45])
```

Blur an image using gaussian blurring.

```python
imgaug.augmenters.blur.blur_gaussian_(image, sigma, ksize=None, backend='auto', eps=0.001)
```

This operation might change the input image in-place.

dtype support:

```python
if (backend=="auto"):
    *
    * `uint8`: yes; fully tested (1)
    * `uint16`: yes; tested (1)
    * `uint32`: yes; tested (2)
    * `uint64`: yes; tested (2)
    * `int8`: yes; tested (1)
    * `int16`: yes; tested (1)
    * `int32`: yes; tested (1)
    * `int64`: yes; tested (2)
    * `float16`: yes; tested (1)
    * `float32`: yes; tested (1)
    * `float64`: yes; tested (1)
    * `float128`: no
    * `bool`: yes; tested (1)

- (1) Handled by `cv2`. See `backend="cv2"`.
- (2) Handled by `scipy`. See `backend="scipy"`.

if (backend=="cv2"):
    *
    * `uint8`: yes; fully tested
    * `uint16`: yes; tested
    * `uint32`: no (2)
    * `uint64`: no (3)
    * `int8`: yes; tested (4)
    * `int16`: yes; tested
    * `int32`: yes; tested (5)
    * `int64`: no (6)
    * `float16`: yes; tested (7)
    * `float32`: yes; tested
    * `float64`: yes; tested
    * `float128`: no (8)
    * `bool`: yes; tested (1)

- (1) Mapped internally to `float32`. Otherwise causes `TypeError: src data type = 0 is not supported`.
- (2) Causes `TypeError: src data type = 6 is not supported`.
- (3) Causes `cv2.error: OpenCV(3.4.5) (...)/filter.cpp:2957: error: (-213) The function/feature is not implemented) Unsupported combination of source format (=4), and buffer format (=5) in function 'getLinearRowFilter'`.
- (4) Mapped internally to `int16`. Otherwise causes `cv2.error: OpenCV(3.4.5) (...)/filter.cpp:2957:`
error: (-213: The function/feature is not implemented) Unsupported
combination of source format (=1),
and buffer format (=5) in function 'getLinearRowFilter''.
  - (5) Mapped internally to `float64'. Otherwise causes `cv2.error:
OpenCV(3.4.5) (...)/filter.cpp:2957:
error: (-213: The function/feature is not implemented) Unsupported
combination of source format (=4),
and buffer format (=5) in function 'getLinearRowFilter''.
  - (6) Causes `cv2.error: OpenCV(3.4.5) (...)/filter.cpp:2957: error: (-
213: The function/feature is not implemented) Unsupported combination of source format (=4), and buffer
format (=5) in function 'getLinearRowFilter''.
  - (7) Mapped internally to `float32'. Otherwise causes `TypeError: src
data type = 23 is not supported``.
  - (8) Causes `TypeError: src data type = 13 is not supported``.

if (backend="scipy"):

* `uint8`: yes; fully tested
* `uint16`: yes; tested
* `uint32`: yes; tested
* `uint64`: yes; tested
* `int8`: yes; tested
* `int16`: yes; tested
* `int32`: yes; tested
* `int64`: yes; tested
* `float16`: yes; tested (1)
* `float32`: yes; tested
* `float64`: yes; tested
* `float128`: no (2)
* `bool`: yes; tested (3)

  - (1) Mapped internally to `float32`. Otherwise causes `RuntimeError:
array type dtype('float16')
not supported``.
  - (2) Causes `RuntimeError: array type dtype('float128') not supported``.
  - (3) Mapped internally to `float32`. Otherwise too inaccurate.

Parameters

- **image** (*numpy.ndarray*) – The image to blur. Expected to be of shape (H, W) or (H, W, C).

- **sigma** (*number*) – Standard deviation of the gaussian blur. Larger numbers result in more large-scale blurring, which is overall slower than small-scale blurring.

- **ksize** (*None or int, optional*) – Size in height/width of the gaussian kernel. This argument is only understood by the cv2 backend. If it is set to None, an appropriate value for ksize will automatically be derived from sigma. The value is chosen tighter for larger sigmas to avoid as much as possible very large kernel sizes and thereby improve performance.

- **backend** (*{'auto', 'cv2', 'scipy'}, optional*) – Backend library to use. If auto, then the likely best library will be automatically picked per image. That is usually equivalent to cv2 (OpenCV) and it will fall back to scipy for datatypes not supported by OpenCV.

- **eps** (*number, optional*) – A threshold used to decide whether sigma can be considered zero.
Returns image – The blurred image. Same shape and dtype as the input.
Return type  numpy.ndarray

13.18  imgaug.augmenters.color

Augmenters that apply color space oriented changes.
Do not import directly from this file, as the categorization is not final. Use instead

```python
from imgaug import augmenters as iaa
```

and then e.g.

```python
seq = iaa.Sequential([
    iaa.Grayscale((0.0, 1.0)),
    iaa.AddToHueAndSaturation((-10, 10))
])
```

List of augmenters:

- InColorspace (deprecated)
- WithColorspace
- AddToHueAndSaturation
- ChangeColorspace
- Grayscale

```python
class imgaug.augmenters.color.AddToHueAndSaturation(value=0, per_channel=False, from_colorspace='RGB', name=None, deterministic=False, random_state=None)
```

Bases: `imgaug.augmenters.meta.Augmenter`

Augmenter that increases/decreases hue and saturation by random values.

The augmenter first transforms images to HSV colorspace, then adds random values to the H and S channels and afterwards converts back to RGB.

TODO add float support
dtype support:

- `uint8`
- `uint16`
- `uint32`
- `uint64`
- `int8`
- `int16`
- `int32`
- `int64`
- `float16`
- `float32`
- `float64`
- `float128`
- `bool`
Parameters

- **value** *(int or tuple of int or list of int or imgaug.parameters.StochasticParameter, optional)*
  
  See `imgaug.augmenters.arithmetic.Add.__init__()`.

- **per_channel** *(bool or float, optional)*
  
  See `imgaug.augmenters.arithmetic.Add.__init__()`.

- **from_colorspace** *(str, optional)*
  
  See `imgaug.augmenters.color.ChangeColorspace.__init__()`.

- **channels** *(int or list of int or None, optional)*
  
  See `imgaug.augmenters.meta.WithChannels.__init__()`.

- **name** *(None or str, optional)*
  
  See `imgaug.augmenters.meta.Augmenter.__init__()`.

Examples

```python
>>> aug = AddToHueAndSaturation((-20, 20), per_channel=True)
```

Adds random values between -20 and 20 to the hue and saturation (independently per channel and the same value for all pixels within that channel).

Methods

- `__call__(*args, **kwargs)`
  
  Alias for `imgaug.augmenters.meta.Augmenter.augment()`.

- `augment([return_batch, hooks])`
  
  Augment data.

- `augment_batch(batch[, hooks])`
  
  Augment a single batch.

- `augment_batches(batches[, hooks, background])`
  
  Augment multiple batches.

- `augment_bounding_boxes(bounding_boxes_on_images)`
  
  Augment bounding boxes.

- `augment_heatmaps(heatmaps[, parents, hooks])`
  
  Augment a heatmap.

- `augment_image(image[, hooks])`
  
  Augment a single image.

- `augment_images(images[, parents, hooks])`
  
  Augment multiple images.

- `augment_keypoints(keypoints_on_images[, ...])`
  
  Augment image keypoints.

- `augment_line_strings(line_strings_on_images)`
  
  Augment line strings.

- `augment_polygons(polygons_on_images[, ...])`
  
  Augment polygons.

- `augment_segmentation_maps(segmaps[, ...])`
  
  Augment segmentation maps.

- `copy()`
  
  Create a shallow copy of this Augmenter instance.

- `copy_random_state(source[, recursive, ...])`
  
  Copy the random states from a source augmenter sequence.

- `copy_random_state_(source[, recursive, ...])`
  
  Copy the random states from a source augmenter sequence (inplace).

- `deepcopy()`
  
  Create a deep copy of this Augmenter instance.

- `draw_grid(images, rows, cols)`
  
  Apply this augmenter to the given images and return a grid image of the results.

- `find_augmenters(func[, parents, flat])`
  
  Find augmenters that match a condition.

Continued on next page
Table 69 – continued from previous page

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>find_augmenters_by_name</td>
<td>name[, regex[, flat]]</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names</td>
<td>names[, regex[, flat]]</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children</td>
<td>[flat]</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_all_children_lists</td>
<td>()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state</td>
<td>[recursive]</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_</td>
<td>[recursive]</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool</td>
<td>[processes, maxtasksperchild, seed]</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters</td>
<td>func[, copy[, noop_if_topmost]]</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace</td>
<td>func[, parents]</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed</td>
<td>[random_state, deterministic_too]</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid</td>
<td>images, rows, cols</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic</td>
<td>[in]</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

```python
get_parameters()
```

class imgaug.augmenters.color.ChangeColorspace(to_colorspace, from_colorspace='RGB', alpha=1.0, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter to change the colorspace of images.

NOTE: This augmenter is not tested. Some colorspaces might work, others might not.

NOTE: This augmenter tries to project the colorspace value range on 0-255. It outputs dtype=uint8 images.

TODO check dtype support

dtype support:

* `"uint8"`: yes; not tested
* `"uint16"`: ?
* `"uint32"`: ?
* `"uint64"`: ?
* `"int8"`: ?
* `"int16"`: ?
* `"int32"`: ?
* `"int64"`: ?
* `"float16"`: ?
* `"float32"`: ?
* `"float64"`: ?
* `"float128"`: ?
* `"bool"`: ?

Parameters
• **to_colorspace** *(str or list of str or imgaug.parameters.StochasticParameter)* – The target colorspace. Allowed strings are: RGB, BGR, GRAY, CIE, YCrCb, HSV, HLS, Lab, Luv. These are also accessible via ChangeColorspace.<NAME>, e.g. ChangeColorspace.YCrCb.
  
  – If a string, it must be among the allowed colorspaces.
  – If a list, it is expected to be a list of strings, each one being an allowed colorspace. A random element from the list will be chosen per image.
  – If a StochasticParameter, it is expected to return string. A new sample will be drawn per image.

• **from_colorspace** *(str, optional)* – The source colorspace (of the input images). See to_colorspace. Only a single string is allowed.

• **alpha** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – The alpha value of the new colorspace when overlayed over the old one. A value close to 1.0 means that mostly the new colorspace is visible. A value close to 0.0 means, that mostly the old image is visible.
  
  – If an int or float, exactly that value will be used.
  – If a tuple \((a, b)\), a random value from the range \(a \leq x \leq b\) will be sampled per image.
  
  – If a list, then a random value will be sampled from that list per image.
  
  – If a StochasticParameter, a value will be sampled from the parameter per image.

• **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

• **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

• **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

### Methods

<table>
<thead>
<tr>
<th><strong>call</strong>(**args, **kwargs)</th>
<th>Alias for <strong>imgaug.augmenters.meta.Augmenter.augment()</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>augment_segmentation_maps(segmaps[, ...])</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic(In)</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

### get_parameters

- **BGR** = 'BGR'
- **CIE** = 'CIE'
- **COLORSPACES** = {'BGR', 'CIE', 'GRAY', 'HLS', 'HSV', 'Lab', 'Luv', 'RGB', 'YCrCb'}
- **CV_VARS** = {'BGR2CIE': <MagicMock id='140619346526504'>, 'BGR2GRAY': <MagicMock id='140619346509896'>, ... 'RGB2Luv': <MagicMock id='140619346468424'>, 'RGB2YCrCb': <MagicMock id='140619346418376'>}
- **GRAY** = 'GRAY'
- **HLS** = 'HLS'
- **HSV** = 'HSV'
- **Lab** = 'Lab'
- **Luv** = 'Luv'
- **RGB** = 'RGB'
- **YCrCb** = 'YCrCb'

13.18. imgaug.augmenters.color 343
Augmenter to convert images to their grayscale versions.

NOTE: Number of output channels is still 3, i.e. this augmenter just “removes” color.

TODO check dtype support

dtype support:

* `uint8`: yes; fully tested
* `uint16`: ?
* `uint32`: ?
* `uint64`: ?
* `int8`: ?
* `int16`: ?
* `int32`: ?
* `int64`: ?
* `float16`: ?
* `float32`: ?
* `float64`: ?
* `float128`: ?
* `bool`: ?

Parameters

- **alpha** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – The alpha value of the grayscale image when overlayed over the old image. A value close to 1.0 means, that mostly the new grayscale image is visible. A value close to 0.0 means, that mostly the old image is visible.
  - If a number, exactly that value will always be used.
  - If a tuple \((a, b)\), a random value from the range \(a \leq x \leq b\) will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **from_colorspace** (str, optional) – The source colorspace (of the input images). Allowed strings are: RGB, BGR, GRAY, CIE, YCrCb, HSV, HLS, Lab, Luv. See imgaug.augmenters.color.ChangeColorspace.__init__().

- **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> aug = iaa.Grayscale(alpha=1.0)
```

creates an augmenter that turns images to their grayscale versions.
```python
>>> aug = iaa.Grayscale(alpha=(0.0, 1.0))
```

creates an augmenter that turns images to their grayscale versions with an alpha value in the range \(0 \leq \alpha \leq 1\). An alpha value of 0.5 would mean, that the output image is 50 percent of the input image and 50 percent of the grayscale image (i.e. 50 percent of color removed).

`imgaug.augmenters.color.InColorspace(to_colorspace, from_colorspace='RGB', children=None, name=None, deterministic=False, random_state=None)`

**Deprecated.** Use `WithColorspace` instead.

Convert images to another colorspace.

**class** `imgaug.augmenters.color.WithColorspace(to_colorspace, from_colorspace='RGB', children=None, name=None, deterministic=False, random_state=None)`

**Bases:** `imgaug.augmenters.meta.Augmenter`

Apply child augmenters within a specific colorspace.

This augmenter takes a source colorspace A and a target colorspace B as well as children C. It changes images from A to B, then applies the child augmenters C and finally changes the colorspace back from B to A. See also `ChangeColorspace()` for more.

dtype support:

- `'uint8'`: yes; fully tested
- `'uint16'`: ?
- `'uint32'`: ?
- `'uint64'`: ?
- `'int8'`: ?
- `'int16'`: ?
- `'int32'`: ?
- `'int64'`: ?
- `'float16'`: ?
- `'float32'`: ?
- `'float64'`: ?
- `'float128'`: ?
- `'bool'`: ?

**Parameters**

- **to_colorspace** *(str)* – See `imgaug.augmenters.ChangeColorspace.__init__()`.  
- **children** *(None or Augmenter or list of Augmenters, optional)* – See `imgaug.augmenters.ChangeColorspace.__init__()`.  
- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  
- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  
- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

13.18. `imgaug.augmenters.color`
### Examples

```python
>>> aug = iaa.WithColorspace(to_colorspace="HSV", from_colorspace="RGB",
>>>   children=iaa.WithChannels(0, iaa.Add(10)))
```

This augmenter will add 10 to Hue value in HSV colorspace, then change the colorspace back to the original (RGB).

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(*args, **kwargs)</code></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code>.</td>
</tr>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment_bounding_boxes(bounding_boxes_on_images)</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps(heatmaps[, parents, hooks])</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
<tr>
<td><code>augment_images(images[, parents, hooks])</code></td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td><code>augment_keypoints(keypoints_on_images[,...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
<tr>
<td><code>augment_segmentation_maps(segmaps[, ...])</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive,...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive,...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[,noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[,parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
</tbody>
</table>

Continued on next page
**Table 71** – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([n])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

### get_children_lists()

Get a list of lists of children of this augmenter.

For most augmenters, the result will be a single empty list. For augmenters with children it will often be a list with one sublist containing all children. In some cases the augmenter will contain multiple distinct lists of children, e.g. an if-list and an else-list. This will lead to a result consisting of a single list with multiple sublists, each representing the respective sublist of children.

E.g. for an if/else-augmenter that executes the children \(A_1, A_2\) if a condition is met and otherwise executes the children \(B_1, B_2, B_3\) the result will be \([A_1, A_2], [B_1, B_2, B_3]\).

**IMPORTANT**: While the topmost list may be newly created, each of the sublist must be editable inplace resulting in a changed children list of the augmenter. E.g. if an Augmenter `IfElse(condition, [A_1, A_2], [B_1, B_2, B_3])` returns \([A_1, A_2], [B_1, B_2, B_3]\) for a call to `imgaug.augmenters.meta.Augmenter.get_children_lists()` and \(A_2\) is removed inplace from \([A_1, A_2]\), then the children lists of `IfElse(...)` must also change to \([A_1], [B_1, B_2, B_3]\). This is used in `imgaug.augmenters.meta.Augmenter.remove_augmenters_inplace()`.

**Returns** children

One or more lists of child augmenter. Can also be a single empty list.

**Return type** list of list of `imgaug.augmenters.meta.Augmenter`

### 13.19 imgaug.augmenters.contrast

Augmenters that perform contrast changes.

Do not import directly from this file, as the categorization is not final. Use instead

```
from imgaug import augmenters as iaa
```

and then e.g.

```
seq = iaa.Sequential([iaa.GammaContrast((0.5, 1.5))])
```

List of augmenters:

- GammaContrast
- SigmoidContrast
- LogContrast
- LinearContrast
- AllChannelsHistogramEqualization
• HistogramEqualization
• AllChannelsCLAHE
• CLAHE

class imgaug.augmenters.contrast.AllChannelsCLAHE(clip_limit=40, tile_grid_size_px=8, tile_grid_size_px_min=3, per_channel=False, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Contrast Limited Adaptive Histogram Equalization, applied to all channels of the input images.

CLAHE performs histogram equilization within image patches, i.e. over local neighbourhoods.

dtype support:

* `\`uint8\``: yes; fully tested
* `\`uint16\`": yes; tested
* `\`uint32\`": no (1)
* `\`uint64\`": no (1)
* `\`int8\`": no (2)
* `\`int16\`": no (2)
* `\`int32\`": no (2)
* `\`int64\`": no (2)
* `\`float16\`": no (2)
* `\`float32\`": no (2)
* `\`float64\`": no (2)
* `\`float128\`": no (1)
* `\`bool\`": no (1)

- (1) rejected by cv2
- (2) results in error in cv2: `\`cv2.error: OpenCV(3.4.2) (...)/clahe.cpp:351:\`error: (-215:Assertion failed)
  src.type() == (((0) & ((1 << 3) - 1)) + (((1)-1) << 3))
  || _src.type() == (((2) & ((1 << 3) - 1)) + (((1)-1) << 3))` in function `\`apply\``

Parameters

• `clip_limit` (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – See imgaug.augmenters.contrast.CLAHE.

• `tile_grid_size_px` (int or tuple of int or list of int or imgaug.parameters.StochasticParameter or tuple of tuple of int or tuple of list of int or tuple of list of int or tuple of imgaug.parameters.StochasticParameter, optional) – See imgaug.augmenters.contrast.CLAHE.

• `tile_grid_size_px_min` (int, optional) – See imgaug.augmenters.contrast.CLAHE.

• `per_channel` (bool or float, optional) – Whether to use the same values for all channels (False) or to sample new values for each channel (True). If this parameter is a float p, then for p percent of all images per_channel will be treated as True, otherwise as False.

• `name` (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
• **deterministic** *(bool, optional)* – See imaug.augmenters.meta.Augmenter.__init__().

• **random_state** *(None or int or numpy.random.RandomState, optional)* – See imaug.augmenters.meta.Augmenter.__init__().

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, *<em>kwargs)</em></td>
<td>Alias for imaug.augmenters.meta.Augmenter.augment().</td>
</tr>
<tr>
<td>augment(<em>return_batch, hooks</em>)</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(<em>batch[, hooks]</em>)</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(<em>batches[, hooks, background]</em>)</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(<em>bounding_boxes_on_images</em>)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(<em>heatmaps[, parents, hooks]</em>)</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(<em>image[, hooks]</em>)</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(<em>images[, parents, hooks]</em>)</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(<em>keypoints_on_images</em>[...]*)</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(<em>line_strings_on_images</em>)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(<em>polygons_on_images</em>[...]*)</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(<em>segmaps</em>[...]*)</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(<em>source[, recursive,...]</em>)</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(<em>source[, recursive,...]</em>)</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(<em>images, rows, cols</em>)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(<em>func[, parents, flat]</em>)</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(<em>name[, regex, flat]</em>)</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(<em>names[, regex, flat]</em>)</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children(<em>flat</em>)</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state(<em>[recursive]</em>)</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_(<em>[recursive]</em>)</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool(<em>[processes, maxtasksperchild, seed]</em>)</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(<em>func[, copy, noop_if_topmost]</em>)</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(<em>func[, parents]</em>)</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed(<em>[random_state, deterministic_too]</em>)</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
</tbody>
</table>

Continued on next page
Table 72 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>show_grid</code></td>
<td>Applies this augmenter to the given images and show/plot the results as a</td>
</tr>
<tr>
<td></td>
<td>grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

```python
get_parameters()
```

```python
class imgaug.augmenters.contrast.AllChannelsHistogramEqualization(name=None, deterministic=False, random_state=None)
```

Bases: `imgaug.augmenters.meta.Augmenter`

Augmenter to perform standard histogram equalization on images, applied to all channels of each input image.

dtype support:

- `'uint8'`: yes; fully tested
- `'uint16'`: no (1)
- `'uint32'`: no (2)
- `'uint64'`: no (1)
- `'int8'`: no (1)
- `'int16'`: no (1)
- `'int32'`: no (1)
- `'int64'`: no (1)
- `'float16'`: no (2)
- `'float32'`: no (1)
- `'float64'`: no (1)
- `'float128'`: no (2)
- `'bool'`: no (1)

- (1) causes cv2 error: `
  cv2.error: OpenCV(3.4.5) (...)\n  histogram.cpp:3345: Assertion failed
  src.type() == CV_8UC1 in function 'equalizeHist'
- (2) rejected by cv2

Parameters

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  
- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  
- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

Methods
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multithreaded augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([in])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_parameters()**
class imgaug.augmenters.contrast.CLAHE(clip_limit=40, tile_grid_size_px=8, tile_grid_size_px_min=3, from_colorspace='RGB', to_colorspace='Lab', name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Contrast Limited Adaptive Histogram Equalization.

This augmenter applies CLAHE to images, a form of histogram equalization that normalizes within local image patches. The augmenter transforms input images to a target colorspace (e.g. Lab), extracts an intensity-related channel from the converted images (e.g. L for Lab), applies CLAHE to the channel and then converts the resulting image back to the original colorspace.

Grayscale images (images without channel axis or with only one channel axis) are automatically handled, from_colorspace does not have to be adjusted for them. For images with four channels (e.g. RGBA), the fourth channel is ignored in the colorspace conversion (e.g. from an RGBA image, only the RGB part is converted, normalized, converted back and concatenated with the input A channel). Images with unusual channel numbers (2, 5 or more than 5) are normalized channel-by-channel (same behaviour as AllChannelsCLAHE, though a warning will be raised).

dtype support:

- ('uint8', yes; fully tested
- ('uint16', no (1)
- ('uint32', no (1)
- ('uint64', no (1)
- ('int8', no (1)
- ('int16', no (1)
- ('int32', no (1)
- ('int64', no (1)
- ('float16', no (1)
- ('float32', no (1)
- ('float64', no (1)
- ('float128', no (1)
- ('bool', no (1)

- (1) This augmenter uses ChangeColorspace, which is currently limited to
  └→ 'uint8'.

Parameters

- **clip_limit** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – Clipping limit. Higher values result in stronger contrast. OpenCV uses a default of 40, though values around 5 seem to already produce decent contrast.
  - If a number, then that value will be used for all images.
  - If a tuple (a, b), then a value from the range [a, b] will be used per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then a value will be sampled per image from that parameter.

- **tile_grid_size_px** (int or tuple of int or list of int or imgaug.parameters.StochasticParameter or tuple of tuple of int or tuple of list of int or tuple of imgaug.parameters.StochasticParameter, optional) –
  Kernel size, i.e. size of each local neighbourhood in pixels.
  - If an int, then that value will be used for all images for both kernel height and width.
– If a tuple \((a, b)\), then a value from the discrete range \([a..b]\) will be sampled per image.

– If a list, then a random value will be sampled from that list per image and used for both kernel height and width.

– If a StochasticParameter, then a value will be sampled per image from that parameter per image and used for both kernel height and width.

– If a tuple of int given as \(((a, b), (c, d))\), then two values will be sampled independently from the discrete ranges \([a..b]\) and \([c..d]\) per image and used as the kernel height and width.

– If a tuple of lists of int, then two values will be sampled independently per image, one from the first list and one from the second, and used as the kernel height and width.

– If a tuple of StochasticParameter, then two values will be sampled independently per image, one from the first parameter and one from the second, and used as the kernel height and width.

- **tile_grid_size_px_min** (int, optional) – Minimum kernel size in px, per axis. If the sampling results in a value lower than this minimum, it will be clipped to this value.

- **from_colorspace** (\{“RGB”, “BGR”, “HSV”, “HLS”, “Lab”\}, optional) – Colorspace of the input images. If any input image has only one or zero channels, this setting will be ignored and it will be assumed that the input is grayscale. If a fourth channel is present in an input image, it will be removed before the colorspace conversion and later re-added. See also [imgaug.augmenters.color.ChangeColorspace](https://imgaug.readthedocs.io/en/latest/api/augmenters/color/ChangeColorspace.html) for details.

- **to_colorspace** (\{“Lab”, “HLS”, “HSV”\}, optional) – Colorspace in which to perform CLAHE. For Lab, CLAHE will only be applied to the first channel (L), for HLS to the second (L) and for HSV to the third (V). To apply CLAHE to all channels of an input image (without colorspace conversion), see [imgaug.augmenters.contrast.AllChannelsCLAHE](https://imgaug.readthedocs.io/en/latest/api/augmenters/contrast/AllChannelsCLAHE.html).


### Examples

```python
>>> aug = iaa.CLAHE()
```

Creates a standard CLAHE augmenter.

```python
>>> aug = iaa.CLAHE(clip_limit=(1, 50))
```

Creates a CLAHE augmenter with a clip limit uniformly sampled from \([1..50]\), where 1 is rather low contrast and 50 is rather high contrast.

```python
>>> aug = iaa.CLAHE(tile_grid_size_px=(3, 21))
```

Creates a CLAHE augmenter with kernel sizes of \(S\times S\), where \(S\) is uniformly sampled from \([3..21]\). Sampling happens once per image.
>>> aug = iaa.CLAHE(tile_grid_size_px=iap.Discretize(iap.Normal(loc=7, scale=2)), tile_grid_size_px_min=3)

Creates a CLAHE augmenter with kernel sizes of SxS, where S is sampled from N(7, 2), but does not go below 3.

>>> aug = iaa.CLAHE(tile_grid_size_px=((3, 21), [3, 5, 7]))

Creates a CLAHE augmenter with kernel sizes of HxW, where H is uniformly sampled from [3..21] and W is randomly picked from the list [3, 5, 7].

>>> aug = iaa.CLAHE(from_colorspace=iaa.CLAHE.BGR, to_colorspace=iaa.CLAHE.HSV)

Creates a CLAHE augmenter that converts images from BGR colorspace to HSV colorspace and then applies the local histogram equalization to the V channel of the images (before converting back to BGR). Alternatively, Lab (default) or HLS can be used as the target colorspace. Grayscale images (no channels / one channel) are never converted and are instead directly normalized (i.e. from_colorspace does not have to be changed for them).

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code>.</td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[], hooks)</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[], hooks, background)</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[], parents, hooks)</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[], hooks)</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[], parents, hooks)</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_inplace(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
</tbody>
</table>
Table 74 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic(n)</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_parameters**

```python
BGR = 'BGR'
HLS = 'HLS'
HSV = 'HSV'
Lab = 'Lab'
RGB = 'RGB'
```

**imgaug.augmenters.contrast.GammaContrast**

```python
imgaug.augmenters.contrast.GammaContrast(gamma=1, per_channel=False, name=None, deterministic=False, random_state=None)
```

Adjust contrast by scaling each pixel value to \(255 \times ((I_{ij}/255)^{\text{gamma}})\).

Values in the range `gamma=(0.5, 2.0)` seem to be sensible.

dtype support:

See :func:`imgaug.augmenters.contrast.adjust_contrast_gamma`.

**Parameters**

- **gamma** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Exponent for the contrast adjustment. Higher values darken the image.
  - If a number, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value from the range \([a, b]\) will be used per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then a value will be sampled per image from that parameter.

- **per_channel** *(bool or float, optional)* – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images per_channel will be treated as True, otherwise as False.
name (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

deterministic (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

random_state (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

Returns Augmenter to perform gamma contrast adjustment.

Return type _ContrastFuncWrapper

class imgaug.augmenters.contrast.HistogramEqualization(from_colorspace='RGB',
to_colorspace='Lab',
name=None,
deterministic=False,
random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter to apply standard histogram equalization to images.

This augmenter is similar to imgaug.augmenters.contrast.CLAHE.

The augmenter transforms input images to a target colorspace (e.g. Lab), extracts an intensity-related channel from the converted images (e.g. L for Lab), applies Histogram Equalization to the channel and then converts the resulting image back to the original colorspace.

Grayscale images (images without channel axis or with only one channel axis) are automatically handled, from_colorspace does not have to be adjusted for them. For images with four channels (e.g. RGBA), the fourth channel is ignored in the colorspace conversion (e.g. from an RGBA image, only the RGB part is converted, normalized, converted back and concatenated with the input A channel). Images with unusual channel numbers (2, 5 or more than 5) are normalized channel-by-channel (same behaviour as AllChannelsHistogramEqualization, though a warning will be raised).

dtype support:

```
* `"uint8```: yes; fully tested
* `"uint16```: no (1)
* `"uint32```: no (1)
* `"uint64```: no (1)
* `"int8```: no (1)
* `"int16```: no (1)
* `"int32```: no (1)
* `"int64```: no (1)
* `"float16```: no (1)
* `"float32```: no (1)
* `"float64```: no (1)
* `"float128```: no (1)
* `"bool```: no (1)

- (1) This augmenter uses AllChannelsHistogramEqualization, which only supports `"uint8```.```

Parameters

- from_colorspace (\{“RGB”, “BGR”, “HSV”, “HLS”, “Lab”\}, optional) – Colorspace of the input images. If any input image has only one or zero channels, this setting will be ignored and it will be assumed that the input is grayscale. If a fourth channel is present in
an input image, it will be removed before the colorspace conversion and later re-added. See also imgaug.augmenters.color.ChangeColorspace for details.

- **to_colorspace** ("Lab", "HLS", "HSV", optional) – Colorspace in which to perform Histogram Equalization. For Lab, the equalization will only be applied to the first channel (L), for HLS to the second (L) and for HSV to the third (V). To apply histogram equalization to all channels of an input image (without colorspace conversion), see imgaug.augmenters.contrast.AllChannelsHistogramEqualization.

- **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

**Examples**

```python
>>> aug = iaa.HistogramEqualization()

Creates a standard histogram equalization augmenter.

>>> aug = iaa.HistogramEqualization(from_colorspace=iaa.HistogramEqualization.BGR,

```to_colorspace=iaa.HistogramEqualization.HSV)

Creates a histogram equalization augmenter that converts images from BGR colorspace to HSV colorspace and then applies the local histogram equalization to the V channel of the images (before converting back to BGR). Alternatively, Lab (default) or HLS can be used as the target colorspace. Grayscale images (no channels / one channel) are never converted and are instead directly normalized (i.e. from_colorspace does not have to be changed for them).

**Methods**

```python
__call__(*args, **kwargs) Alias for imgaug.augmenters.meta.Augmenter.augment().

augment([return_batch, hooks]) Augment data.

augment_batch(batch[, hooks]) Augment a single batch.

augment_batchses(batches[, hooks, background]) Augment multiple batches.

augment_bounding_boxes(bounding_boxes_on_images) Augment bounding boxes.

augment_heatmaps(heatmaps[, parents, hooks]) Augment a heatmap.

augment_image(image[, hooks]) Augment a single image.

augment_images(images[, parents, hooks]) Augment multiple images.

augment_keypoints(keypoints_on_images[, ...]) Augment image keypoints.

augment_line_strings(line_strings_on_images) Augment line strings.

augment_polygons(polygons_on_images[, ...]) Augment polygons.

augment_segmentation_maps(segmaps[, ...]) Augment segmentation maps.
```
Table 75 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_(recursive)</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_parameters**

```python
BGR = 'BGR'
HLS = 'HLS'
HSV = 'HSV'
Lab = 'Lab'
RGB = 'RGB'

get_parameters()
```

`imgaug.augmenters.contrast.LinearContrast(alpha=1, per_channel=False, name=None, deterministic=False, random_state=None)`

Adjust contrast by scaling each pixel value to $127 + \alpha(127 - I_{ij})$.

dtype support:

```
See :func:`imgaug.augmenters.contrast.adjust_contrast_linear`
```

**Parameters**
• \textbf{alpha} (number or tuple of number or list of number or \texttt{imgaug.parameters.StochasticParameter}, optional) – Multiplier to linearly pronounce (>1.0), dampen (0.0 to 1.0) or invert (<0.0) the difference between each pixel value and the center value, e.g. 127 for \texttt{uint8}.
  
  – If a number, then that value will be used for all images.
  
  – If a tuple \((a, b)\), then a value from the range \([a, b]\) will be used per image.
  
  – If a list, then a random value will be sampled from that list per image.
  
  – If a StochasticParameter, then a value will be sampled per image from that parameter.

• \textbf{per_channel} (bool or float, optional) – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images \texttt{per_channel} will be treated as True, otherwise as False.

• \textbf{name} (None or str, optional) – See \texttt{imgaug.augmenters.meta.Augmenter.__init__}.

• \textbf{deterministic} (bool, optional) – See \texttt{imgaug.augmenters.meta.Augmenter.__init__}.

• \textbf{random_state} (None or int or \texttt{numpy.random.RandomState}, optional) – See \texttt{imgaug.augmenters.meta.Augmenter.__init__}.

\textbf{Returns} Augmenter to perform contrast adjustment by linearly scaling the distance to 128.

\textbf{Return type} \texttt{_ContrastFuncWrapper}

\texttt{imgaug.augmenters.contrast LogContrast}(\texttt{gain=1, per_channel=False, name=None, deterministic=False, random_state=None})

Adjust contrast by scaling each pixel value to \(255 \times \text{gain} \times \log_2(1 + \text{I}_{ij}/255)\).

dtype support:

\texttt{See :func:`imgaug.augmenters.contrast.adjust_contrast_log`}.

\textbf{Parameters}

• \textbf{gain} (number or tuple of number or list of number or \texttt{imgaug.parameters.StochasticParameter}, optional) – Multiplier for the logarithm result. Values around 1.0 lead to a contrast-adjusted images. Values above 1.0 quickly lead to partially broken images due to exceeding the datatype’s value range.
  
  – If a number, then that value will be used for all images.
  
  – If a tuple \((a, b)\), then a value from the range \([a, b]\) will be used per image.
  
  – If a list, then a random value will be sampled from that list per image.
  
  – If a StochasticParameter, then a value will be sampled per image from that parameter.

• \textbf{per_channel} (bool or float, optional) – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float \(p\), then for \(p\) percent of all images \texttt{per_channel} will be treated as True, otherwise as False.

• \textbf{name} (None or str, optional) – See \texttt{imgaug.augmenters.meta.Augmenter.__init__}.

• \textbf{deterministic} (bool, optional) – See \texttt{imgaug.augmenters.meta.Augmenter.__init__}.
• `random_state` *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

**Returns** Augmenter to perform logarithmic contrast adjustment.  

**Return type** `_ContrastFuncWrapper`

`imgaug.augmenters.contrast.SigmoidContrast` *(gain=10, cutoff=0.5, per_channel=False, name=None, deterministic=False, random_state=None)*

Adjust contrast by scaling each pixel value to $255 \times \frac{1}{1 + \exp(gain \times (cutoff - I_{ij}/255))}$. Values in the range $gain=(5, 20)$ and $cutoff=(0.25, 0.75)$ seem to be sensible.

dtype support:

See :func:`imgaug.augmenters.contrast.adjust_contrast_sigmoid`.

**Parameters**

• `gain` *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Multiplier for the sigmoid function’s output. Higher values lead to quicker changes from dark to light pixels.
  
  – If a number, then that value will be used for all images.
  
  – If a tuple $(a, b)$, then a value from the range $[a, b]$ will be used per image.
  
  – If a list, then a random value will be sampled from that list per image.
  
  – If a StochasticParameter, then a value will be sampled per image from that parameter.

• `cutoff` *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Cutoff that shifts the sigmoid function in horizontal direction. Higher values mean that the switch from dark to light pixels happens later, i.e. the pixels will remain darker.
  
  – If a number, then that value will be used for all images.
  
  – If a tuple $(a, b)$, then a value from the range $[a, b]$ will be used per image.
  
  – If a list, then a random value will be sampled from that list per image.
  
  – If a StochasticParameter, then a value will be sampled per image from that parameter.

• `per_channel` *(bool or float, optional)* – Whether to use the same value for all channels (False) or to sample a new value for each channel (True). If this value is a float $p$, then for $p$ percent of all images `per_channel` will be treated as True, otherwise as False.

• `name` *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

• `deterministic` *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

• `random_state` *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

**Returns** Augmenter to perform sigmoid contrast adjustment.  

**Return type** `_ContrastFuncWrapper`
`imgaug.augmenters.contrast.adjust_contrast_gamma`(`arr`, `gamma`)  
Adjust contrast by scaling each pixel value to $255 \times (I_{ij}/255)^{\gamma}$.

dtype support:

- ``uint8``: yes; fully tested (1) (2) (3)
- ``uint16``: yes; tested (2) (3)
- ``uint32``: yes; tested (2) (3)
- ``uint64``: yes; tested (2) (3) (4)
- ``int8``: limited; tested (2) (3) (5)
- ``int16``: limited; tested (2) (3) (5)
- ``int32``: limited; tested (2) (3) (5)
- ``int64``: limited; tested (2) (3) (4) (5)
- ``float16``: limited; tested (5)
- ``float32``: limited; tested (5)
- ``float64``: limited; tested (5)
- ``float128``: no (6)
- ``bool``: no (7)

- (1) Handled by `cv2`. Other dtypes are handled by `skimage`
- (2) Normalization is done as `$I_{ij}/\max$`, where `$\max$` is the maximum value of the dtype, e.g. 255 for `uint8`. The normalization is reversed afterwards, e.g. `result*255` for `uint8`.
- (3) Integer-like values are not rounded after applying the contrast adjustment equation (before inverting the normalization to 0.0-1.0 space), i.e. projection from continuous space to discrete happens according to floor function.
- (4) Note that scikit-image doc says that integers are converted to `float64` values before applying the contrast normalization method. This might lead to inaccuracies for large 64bit integer values. Tests showed no indication of that happening though.
- (5) Must not contain negative values. Values $\geq 0$ are fully supported.
- (6) Leads to error in scikit-image.
- (7) Does not make sense for contrast adjustments.

Parameters

- `arr` (`numpy.ndarray`) – Array for which to adjust the contrast. Dtype `uint8` is fastest.
- `gamma` (`number`) – Exponent for the contrast adjustment. Higher values darken the image.

Returns Array with adjusted contrast.

Return type `numpy.ndarray`

`imgaug.augmenters.contrast.adjust_contrast_linear`(`arr`, `alpha`)  
Adjust contrast by scaling each pixel value to $127 + \alpha \times (I_{ij}-127)$.

dtype support:

- ``uint8``: yes; fully tested (1) (2)
- ``uint16``: yes; tested (2)
- ``uint32``: yes; tested (2)
- ``uint64``: no (3)
- ``int8``: yes; tested (2)
- ``int16``: yes; tested (2)

(continues on next page)
Parameters

• **arr** (numpy.ndarray) – Array for which to adjust the contrast. Dtype `uint8` is fastest.

• **alpha** (number) – Multiplier to linearly pronounce (>1.0), dampen (0.0 to 1.0) or invert (<0.0) the difference between each pixel value and the center value, e.g. 127 for `uint8`.

**Returns**
Array with adjusted contrast.

**Return type**
numpy.ndarray

`imgaug.augmenters.contrast.adjust_contrast_log(arr, gain)`

Adjust contrast by scaling each pixel value to \(255 \times \text{gain} \times \log_2(1 + \frac{I_{ij}}{255})\).

dtype support:

- (1) Handled by `cv2`. Other dtypes are handled by raw `numpy`.
- (2) Only tested for reasonable alphas with up to a value of around 100.
- (3) Conversion to `float64` is done during augmentation, hence `uint64`, `int64`, and `float128` support cannot be guaranteed.
- (4) Does not make sense for contrast adjustments.

(continues on next page)
applying the contrast normalization method. This might lead to inaccuracies for large 64bit integer values. Tests showed no indication of that happening though.

- (5) Must not contain negative values. Values >=0 are fully supported.
- (6) Leads to error in scikit-image.
- (7) Does not make sense for contrast adjustments.

Parameters

- **arr** (*numpy.ndarray*) – Array for which to adjust the contrast. Dtype `uint8` is fastest.
- **gain** (*number*) – Multiplier for the logarithm result. Values around 1.0 lead to a contrast-adjusted images. Values above 1.0 quickly lead to partially broken images due to exceeding the datatype’s value range.

Returns

Array with adjusted contrast.

Return type

*numpy.ndarray*

```python
imgaug.augmenters.contrast.adjust_contrast_sigmoid(arr, gain, cutoff)
```

Adjust contrast by scaling each pixel value to \(255 \times \frac{1}{1 + \exp(gain \times (\text{cutoff} - I_{ij}/255))}\).

dtype support:

- ```uint8```: yes; fully tested (1) (2) (3)
- ```uint16```: yes; tested (2) (3)
- ```uint32```: yes; tested (2) (3)
- ```uint64```: yes; tested (2) (3) (4)
- ```int8```: limited; tested (2) (3) (5)
- ```int16```: limited; tested (2) (3) (5)
- ```int32```: limited; tested (2) (3) (5)
- ```int64```: limited; tested (2) (3) (4) (5)
- ```float16```: limited; tested (5)
- ```float32```: limited; tested (5)
- ```float64```: limited; tested (5)
- ```float128```: no (6)
- ```bool```: no (7)

- (1) Handled by ```cv2```. Other dtypes are handled by ```skimage```
- (2) Normalization is done as ```I_{ij}/\text{max}``, where ```max``` is the maximum value of the dtype, e.g. 255 for ```uint8```. The normalization is reversed afterwards, e.g. ```result*255``` for ```uint8```
- (3) Integer-like values are not rounded after applying the contrast adjustment equation (before inverting the normalization to 0.0-1.0 space), i.e. projection from continuous space to discrete happens according to floor function.
- (4) Note that scikit-image doc says that integers are converted to ```float64``` values before applying the contrast normalization method. This might lead to inaccuracies for large 64bit integer values. Tests showed no indication of that happening though.
- (5) Must not contain negative values. Values >=0 are fully supported.
- (6) Leads to error in scikit-image.
- (7) Does not make sense for contrast adjustments.

Parameters
• `arr` (**numpy.ndarray**): Array for which to adjust the contrast. Dtype **uint8** is fastest.
• `gain` (**number**): Multiplier for the sigmoid function’s output. Higher values lead to quicker changes from dark to light pixels.
• `cutoff` (**number**): Cutoff that shifts the sigmoid function in horizontal direction. Higher values mean that the switch from dark to light pixels happens later, i.e. the pixels will remain darker.

**Returns**  Array with adjusted contrast.

**Return type**  **numpy.ndarray**

### 13.20  `imgaug.augmenters.convolutional`

Augmenters that apply convolutions to images.

Do not import directly from this file, as the categorization is not final. Use instead

```
from imgaug import augmenters as iaa
```

and then e.g.

```
seq = iaa.Sequential([
    iaa.Sharpen((0.0, 1.0)),
    iaa.Emboss((0.0, 1.0))
])
```

List of augmenters:

- Convolve
- Sharpen
- Emboss
- EdgeDetect
- DirectedEdgeDetect

For MotionBlur, see `blur.py`.

### class  `imgaug.augmenters.convolutional.Convolve`

Bases: `imgaug.augmenters.meta.Augmenter`

Apply a Convolution to input images.

**dtype support:**

```
* `\`uint8\``: yes; fully tested
* `\`uint16\``: yes; tested
* `\`uint32\``: no (1)
* `\`uint64\``: no (2)
* `\`int8\``: yes; tested (3)
* `\`int16\`": yes; tested
* `\`int32\`": no (2)
* `\`int64\`": no (2)
* `\`float16\`": yes; tested (4)
* `\`float32\`": yes; tested
* `\`float64\`": yes; tested
```

(continues on next page)
* ``float128``: no (1)
* ``bool``: yes; tested (4)

- (1) rejected by `cv2.filter2D()`.
- (2) causes error: cv2.error: OpenCV(3.4.2) (/...)/filter.cpp:4487: error: (-213: The function/feature is not implemented) Unsupported combination of source format (=1), and destination format (=1) in function 'getLinearFilter'.
- (3) mapped internally to `int16`.
- (4) mapped internally to `float32`.

**Parameters**

- **matrix** (*None or *(H, W) ndarray or imgaug.parameters.StochasticParameter or callable, optional*) –

  The weight matrix of the convolution kernel to apply.

  - If None, the input images will not be changed.
  - If a numpy array, that array will be used for all images and channels as the kernel.
  - If a callable, the parameter will be called for each image via `param(image, C, random_state)`. The function must either return a list of `C` matrices (i.e. one per channel) or a 2D numpy array (will be used for all channels) or a 3D HxWxC numpy array. If a list is returned, each entry may be None, which will result in no changes to the respective channel.

- **name** (*None or str, optional*) – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

- **deterministic** (*bool, optional*) – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

- **random_state** (*None or int or numpy.random.RandomState, optional*) – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

**Examples**

```python
>>> matrix = np.array([[0, -1, 0],
                      [-1, 4, -1],
                      [0, -1, 0]])
>>> aug = iaa.Convolve(matrix=matrix)
```

Convolves all input images with the kernel shown in the `matrix` variable.

```python
>>> def gen_matrix(image, nb_channels, random_state):
...    matrix_A = np.array([[0, -1, 0],
...                         [-1, 4, -1],
...                         [0, -1, 0]])
...    matrix_B = np.array([[0, 1, 0],
...                         [1, -4, 1],
...                         [0, 1, 0]])
...    if image.shape[0] % 2 == 0:
...        return [matrix_A] * nb_channels
...    else:
...        return [matrix_A, matrix_B] * (nb_channels // 2) + [matrix_B] * (nb_channels % 2)
```

(continues on next page)
>>> return [matrix_B] * nb_channels
>>> aug = iaa.Convolve(matrix=gen_matrix)

convolves images that have an even height with matrix A and images with an odd height with matrix B.

Methods

__call__((*args, **kwargs))

Alias for

imgaug.augmenters.meta.Augmenter.augment().

augment([return_batch, hooks])

Augment data.

augment_batch(batch[, hooks])

Augment a single batch.

augment_batches(batches[, hooks, background])

Augment multiple batches.

augment_boundaries(bounding_boxes_on_images)

Augment boundary boxes.

augment_heatmaps(heatmaps[, parents, hooks])

Augment a heatmap.

augment_image(image[, hooks])

Augment a single image.

augment_images(images[, parents, hooks])

Augment multiple images.

augment_keypoints(keypoints_on_images[, ...])

Augment image keypoints.

augment_lines_on_images(augment_line_strings_on_images)

Augment line strings.

augment_polygons(polygons_on_images[, ...])

Augment polygons.

augment_segmentations(segments[, ...])

Augment segmentation maps.

copy()

Create a shallow copy of this Augmenter instance.

copy_random_state(source[, recursive, ...])

Copy the random states from a source augmenter sequence.

copy_random_state_inplace(source[, recursive, ...])

Copy the random states from a source augmenter sequence (inplace).

deprecated()

Create a deep copy of this Augmenter instance.

draw_grid(images, rows, cols)

Apply this augmenter to the given images and return a grid image of the results.

find_augmenters(func[, parents, flat])

Find augmenters that match a condition.

find_augmenters_by_name(name[, regex, flat])

Find augmenter(s) by name.

find_augmenters_by_names(names[, regex, flat])

Find augmenter(s) by names.

get_all_children([flat])

Returns all children of this augmenter as a list.

get_children_lists()

Get a list of lists of children of this augmenter.

localize_random_state([recursive])

Converts global random states to local ones.

localize_random_state_inplace([recursive])

Converts global random states to local ones.

pool([processes, maxtasksperchild, seed])

Create a pool used for multicore augmentation from this augmenter.

remove_augmenters(func[, copy, noop_if_topmost])

Remove this augmenter or its children that match a condition.

remove_augmenters_inplace(func[, parents])

Remove in-place children of this augmenter that match a condition.

reseed([random_state, deterministic_too])

Reseed this augmenter and all of its children (if it has any).

Continued on next page
**Table 76 – continued from previous page**

<table>
<thead>
<tr>
<th>Augmenter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_parameters()**

```python
imgaug.augmenters.convolutional.DirectedEdgeDetect(alpha=0, direction=(0.0, 1.0), name=None, deterministic=False, random_state=None)
```

Augmenter that detects edges that have certain directions and marks them in a black and white image and then overlays the result with the original image.

**dtype support:**


**Parameters**

- **alpha** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – Visibility of the sharpened image. At 0, only the original image is visible, at 1.0 only its sharpened version is visible.
  - If an int or float, exactly that value will be used.
  - If a tuple `(a, b)`, a random value from the range `a <= x <= b` will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **direction** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – Angle of edges to pronounce, where 0 represents 0 degrees and 1.0 represents 360 degrees (both clockwise, starting at the top). Default value is `(0.0, 1.0)`, i.e. pick a random angle per image.
  - If an int or float, exactly that value will be used.
  - If a tuple `(a, b)`, a random value from the range `a <= x <= b` will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
Examples

```python
>>> aug = DirectedEdgeDetect(alpha=1.0, direction=0)
```

turns input images into edge images in which edges are detected from top side of the image (i.e. the top sides of horizontal edges are added to the output).

```python
>>> aug = DirectedEdgeDetect(alpha=1.0, direction=90/360)
```
same as before, but detecting edges from the right (right side of each vertical edge).

```python
>>> aug = DirectedEdgeDetect(alpha=1.0, direction=(0.0, 1.0))
```
same as before, but detecting edges from a variable direction (anything between 0 and 1.0, i.e. 0 degrees and 360 degrees, starting from the top and moving clockwise).

```python
>>> aug = DirectedEdgeDetect(alpha=(0.0, 0.3), direction=0)
```
generates edge images (edges detected from the top) and overlays them with the input images by a variable amount between 0 and 30 percent (e.g. for 0.3 then 0.7*old_image + 0.3*edge_image).

```python
imgaug.augmenters.convolutional.EdgeDetect(alpha=0, name=None, deterministic=False, random_state=None)
```
Augmenter that detects all edges in images, marks them in a black and white image and then overlays the result with the original image.

dtype support:

See ```imgaug.augmenters.convolutional.Convolve```.

Parameters

- **alpha** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Visibility of the sharpened image. At 0, only the original image is visible, at 1.0 only its sharpened version is visible.
  - If an int or float, exactly that value will be used.
  - If a tuple \((a, b)\), a random value from the range \(a \leq x \leq b\) will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.
- **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
- **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
- **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> aug = EdgeDetect(alpha=(0.0, 1.0))
```
detects edges in an image and overlays the result with a variable alpha in the range $0.0 \leq a \leq 1.0$ over the old image.

```
imgaug.augmenters.convolutional.Emboss(alpha=0, strength=1, name=None, deterministic=False, random_state=None)
```

Augmenter that embosses images and overlays the result with the original image.

The embossed version pronounces highlights and shadows, letting the image look as if it was recreated on a metal plate ("embossed").

dtype support:

See ``imgaug.augmenters.convolutional.Convolve```.

**Parameters**

- **alpha** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Visibility of the sharpened image. At 0, only the original image is visible, at 1.0 only its sharpened version is visible.
  
  - If an int or float, exactly that value will be used.
  
  - If a tuple $(a, b)$, a random value from the range $a \leq x \leq b$ will be sampled per image.
  
  - If a list, then a random value will be sampled from that list per image.
  
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **strength** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Parameter that controls the strength of the embossing. Sane values are somewhere in the range $(0, 2)$ with 1 being the standard embossing effect. Default value is 1.
  
  - If an int or float, exactly that value will be used.
  
  - If a tuple $(a, b)$, a random value from the range $a \leq x \leq b$ will be sampled per image.
  
  - If a list, then a random value will be sampled from that list per image.
  
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

**Examples**

```python
>>> aug = Emboss(alpha=(0.0, 1.0), strength=(0.5, 1.5))
```

embosses an image with a variable strength in the range $0.5 \leq x \leq 1.5$ and overlays the result with a variable alpha in the range $0.0 \leq a \leq 1.0$ over the old image.

```
imgaug.augmenters.convolutional.Sharpen(alpha=0, lightness=1, name=None, deterministic=False, random_state=None)
```

Augmenter that sharpens images and overlays the result with the original image.
dtype support:

See `imgaug.augmenters.convolutional.Convolve`.

**Parameters**

- **alpha** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Visibility of the sharpened image. At 0, only the original image is visible, at 1.0 only its sharpened version is visible.
  - If an int or float, exactly that value will be used.
  - If a tuple \((a, b)\), a random value from the range \(a \leq x \leq b\) will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **lightness** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Parameter that controls the lightness/brightness of the sharpened image. Sane values are somewhere in the range \((0.5, 2)\). The value 0 results in an edge map. Values higher than 1 create bright images. Default value is 1.
  - If an int or float, exactly that value will be used.
  - If a tuple \((a, b)\), a random value from the range \(a \leq x \leq b\) will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, a value will be sampled from the parameter per image.

- **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

**Examples**

```python
>>> aug = Sharpen(alpha=(0.0, 1.0))
```

sharpens input images and overlays the sharpened image by a variable amount over the old image.

```python
>>> aug = Sharpen(alpha=(0.0, 1.0), lightness=(0.75, 2.0))
```

sharpens input images with a variable lightness in the range \(0.75 \leq x \leq 2.0\) and with a variable alpha.

### 13.21 imgaug.augmenters.flip

Augmenters that apply mirroring/flipping operations to images.

Do not import directly from this file, as the categorization is not final. Use instead
```python
from imgaug import augmenters as iaa

and then e.g.

```python
definition = iaa.Sequential([  
    iaa.Fliplr((0.0, 1.0)),
    iaa.Flipud((0.0, 1.0))
])
```

List of augmenters:

- Fliplr
- Flipud

class imgaug.augmenters.flip.Fliplr(p=0, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Flip/mirror input images horizontally.

dtype support:

* `'uint8'`: yes; fully tested
* `'uint16'`: yes; tested
* `'uint32'`: yes; tested
* `'uint64'`: yes; tested
* `'int8'`: yes; tested
* `'int16'`: yes; tested
* `'int32'`: yes; tested
* `'int64'`: yes; tested
* `'float16'`: yes; tested
* `'float32'`: yes; tested
* `'float64'`: yes; tested
* `'float128'`: yes; tested
* `'bool'`: yes; tested

Parameters

- **p** *(number or imgaug.parameters.StochasticParameter, optional)* – Probability of each image to get flipped.
- **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
- **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
- **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> aug = iaa.Fliplr(0.5)
```

would horizontally flip/mirror 50 percent of all input images.

```python
>>> aug = iaa.Fliplr(1.0)
```
would horizontally flip/mirror all input images.

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong> (*args, **kwargs)</td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td>augment((return_batch, hooks))</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Create a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtaskspervchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([in])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>
get_parameters()

class imgaug.augmenters.flip.Flipud(p=0, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Flip/mirror input images vertically.

dtype support:

- `uint8`: yes; fully tested
- `uint16`: yes; tested
- `uint32`: yes; tested
- `uint64`: yes; tested
- `int8`: yes; tested
- `int16`: yes; tested
- `int32`: yes; tested
- `int64`: yes; tested
- `float16`: yes; tested
- `float32`: yes; tested
- `float64`: yes; tested
- `float128`: yes; tested
- `bool`: yes; tested

Parameters

- **p** (number or imgaug.parameters.StochasticParameter, optional) – Probability of each image to get flipped.
- **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
- **deterministic** (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
- **random_state** (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> aug = iaa.Flipud(0.5)
```

would vertically flip/mirror 50 percent of all input images.

```python
>>> aug = iaa.Flipud(1.0)
```

would vertically flip/mirror all input images.

Methods

__call__(*args, **kwargs) Alias for imgaug.augmenters.meta.Augmenter.augment().
augment([return_batch, hooks])  Augment data.
augment_batch(batch[, hooks])  Augment a single batch.
augment_batches(batches[, hooks, background])  Augment multiple batches.
augment_bounding_boxes(bounding_boxes_on_images)  Augment bounding boxes.
augment_heatmaps(heatmaps[, parents, hooks])  Augment a heatmap.
augment_image(image[, hooks])  Augment a single image.
augment_images(images[, parents, hooks])  Augment multiple images.
augment_keypoints(keypoints_on_images[, ...])  Augment image keypoints.
augment_line_strings(line_strings_on_images)  Augment line strings.
augment_polygons(polygons_on_images[, ...])  Augment polygons.
augment_segmentation_maps(segmaps[, ...])  Augment segmentation maps.
copy()  Create a shallow copy of this Augmenter instance.
copy_random_state(source[, recursive,...])  Copy the random states from a source augmenter sequence.
copy_random_state_(source[, recursive,...])  Copy the random states from a source augmenter sequence (inplace).
deepcopy()  Create a deep copy of this Augmenter instance.
draw_grid(images, rows, cols)  Apply this augmenter to the given images and return a grid image of the results.
find_augmenters(func[, parents, flat])  Find augmenters that match a condition.
find_augmenters_by_name(name[, regex, flat])  Find augmenter(s) by name.
find_augmenters_by_names(names[, regex, flat])  Find augmenter(s) by names.
get_all_children([flat])  Returns all children of this augmenter as a list.
get_children_lists()  Get a list of lists of children of this augmenter.
localize_random_state([recursive])  Converts global random states to local ones.
localize_random_state_([recursive])  Converts global random states to local ones.
pool([processes, maxtasksperchild, seed])  Create a pool used for multicore augmentation from this augmenter.
remove_augmenters(func[, copy, noop_if_topmost])  Remove this augmenter or its children that match a condition.
remove_augmenters_inplace(func[, parents])  Remove in-place children of this augmenter that match a condition.
reseed([random_state, deterministic_too])  Reseed this augmenter and all of its children (if it has any).
show_grid(images, rows, cols)  Apply this augmenter to the given images and show/plot the results as a grid of images.
to_deterministic(In)  Converts this augmenter from a stochastic to a deterministic one.

get_parameters()

imgaug.augmenters.flip.HorizontalFlip(*args, **kwargs)
Alias for Fliplr.
imgaug.augmenters.flip.**VerticalFlip**(*args, **kwargs*)  
Alias for Flipud.

### 13.22 imgaug.augmenters.geometric

Augmenters that apply affine transformations or other similar augmentations.

Do not import directly from this file, as the categorization is not final. Use instead

```python
from imgaug import augmenters as iaa
```

and then e.g.

```python
seq = iaa.Sequential([  
    iaa.Affine(...),  
    iaa.PerspectiveTransform(...)  
])
```

#### List of augmenters:

- Affine
- AffineCv2
- PiecewiseAffine
- PerspectiveTransform
- ElasticTransformation
- Rot90

**class** imgaug.augmenters.geometric.**Affine**(scale=1.0, translate_percent=0, translate_px=0, rotate=0, shear=0, order=1, cval=0, mode='constant', fit_output=False, backend='auto', name=None, deterministic=False, random_state=None)

**Bases:** imgaug.augmenters.meta.Augmenter

Augmenter to apply affine transformations to images.

This is mostly a wrapper around skimage’s AffineTransform class and warp function.

Affine transformations involve:

- Translation (‘move’ image on the x-/y-axis)
- Rotation
- Scaling (‘zoom’ in/out)
- Shear (move one side of the image, turning a square into a trapezoid)

All such transformations can create “new” pixels in the image without a defined content, e.g. if the image is translated to the left, pixels are created on the right. A method has to be defined to deal with these pixel values. The parameters `cval` and `mode` of this class deal with this.

Some transformations involve interpolations between several pixels of the input image to generate output pixel values. The parameter `order` deals with the method of interpolation used for this.

dtype support:
if (backend="skimage", order in [0, 1]):

* `uint8`: yes; tested
* `uint16`: yes; tested
* `uint32`: yes; tested (1)
* `uint64`: no (2)
* `int8`: yes; tested
* `int16`: yes; tested
* `int32`: yes; tested (1)
* `int64`: no (2)
* `float16`: yes; tested
* `float32`: yes; tested
* `float64`: yes; tested
* `float128`: no (2)
* `bool`: yes; tested

- (1) scikit-image converts internally to float64, which might affect the
  accuracy of large integers. In tests this seemed to not be an issue.
- (2) results too inaccurate

if (backend="skimage", order in [3, 4]):

* `uint8`: yes; tested
* `uint16`: yes; tested
* `uint32`: yes; tested (1)
* `uint64`: no (2)
* `int8`: yes; tested
* `int16`: yes; tested
* `int32`: yes; tested (1)
* `int64`: no (2)
* `float16`: yes; tested
* `float32`: yes; tested
* `float64`: limited; tested (3)
* `float128`: no (2)
* `bool`: yes; tested

- (1) scikit-image converts internally to float64, which might affect the
  accuracy of large integers. In tests this seemed to not be an issue.
- (2) results too inaccurate
- (3) `NaN` around minimum and maximum of float64 value range

if (backend="skimage", order=5):

* `uint8`: yes; tested
* `uint16`: yes; tested
* `uint32`: yes; tested (1)
* `uint64`: no (2)
* `int8`: yes; tested
* `int16`: yes; tested
* `int32`: yes; tested (1)
* `int64`: no (2)
* `float16`: yes; tested
* `float32`: yes; tested
* `float64`: limited; not tested (3)
* `float128`: no (2)

(continues on next page)
(continued from previous page)

* `bool`: yes; tested
- (1) scikit-image converts internally to float64, which might affect the accuracy of large integers. In tests this seemed to not be an issue.
- (2) results too inaccurate
- (3) `NaN` around minimum and maximum of float64 value range

if (backend="cv2", order=0):

* `uint8`: yes; tested
* `uint16`: yes; tested
* `uint32`: no (1)
* `uint64`: no (2)
* `int8`: yes; tested
* `int16`: yes; tested
* `int32`: yes; tested
* `int64`: no (2)
* `float16`: yes; tested (3)
* `float32`: yes; tested
* `float64`: yes; tested
* `float128`: no (1)
* `bool`: yes; tested (3)

- (1) rejected by cv2
- (2) changed to `int32` by cv2
- (3) mapped internally to `float32`

if (backend="cv2", order=1):

* `uint8`: yes; fully tested
* `uint16`: yes; tested
* `uint32`: no (1)
* `uint64`: no (2)
* `int8`: yes; tested (3)
* `int16`: yes; tested
* `int32`: no (2)
* `int64`: no (2)
* `float16`: yes; tested (4)
* `float32`: yes; tested
* `float64`: yes; tested
* `float128`: no (1)
* `bool`: yes; tested (4)

- (1) rejected by cv2
- (2) causes cv2 error: `cv2.error: OpenCV(3.4.4) (...) imgwarp.cpp:1805: _error: (-215:Assertion failed) ifunc != 0 in function 'remap'`
- (3) mapped internally to `int16`
- (4) mapped internally to `float32`

if (backend="cv2", order=3):

* `uint8`: yes; tested
* `uint16`: yes; tested
* `uint32`: no (1)
* `uint64`: no (2)

(continues on next page)
* `int8`*: yes; tested (3)
* `int16`*: yes; tested
* `int32`*: no (2)
* `int64`*: no (2)
* `float16`*: yes; tested (4)
* `float32`*: yes; tested
* `float64`*: yes; tested
* `float128`*: no (1)
* `bool`*: yes; tested (4)

- (1) rejected by cv2
- (2) causes cv2 error: `cv2.error: OpenCV(3.4.4) (...)imgwarp.cpp:1805: error: (...-215:Assertion failed) ifunc != 0 in function 'remap'`
- (3) mapped internally to `int16`
- (4) mapped internally to `float32`

Parameters

- **scale** *(number or tuple of number or list of number or imageaug.parameters.StochasticParameter or dict ("x": number/tuple/list/StochasticParameter, "y": number/tuple/list/StochasticParameter), optional)* – Scaling factor to use, where 1.0 represents no change and 0.5 is zoomed out to 50 percent of the original size.
  - If a single number, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value will be sampled from the range \(a \leq x \leq b\) per image. That value will be used identically for both x- and y-axis.
  - If a list, then a random value will eb sampled from that list per image.
  - If a StochasticParameter, then from that parameter a value will be sampled per image (again, used for both x- and y-axis).
  - If a dictionary, then it is expected to have the keys “x” and/or “y”. Each of these keys can have the same values as described before for this whole parameter (scale). Using a dictionary allows to set different values for the axis. If they are set to the same ranges, different values may still be sampled per axis.

- **translate_percent** *(None or number or tuple of number or list of number or imageaug.parameters.StochasticParameter or dict ("x": number/tuple/list/StochasticParameter, "y": number/tuple/list/StochasticParameter), optional)* – Translation in percent relative to the image height/width (x-translation, y-translation) to use, where 0 represents no change and 0.5 is half of the image height/width.
  - If None then equivalent to 0 unless translate_px has a non-None value.
  - If a single number, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value will be sampled from the range \(a \leq x \leq b\) per image. That percent value will be used identically for both x- and y-axis.
  - If a list, then a random value will eb sampled from that list per image.
  - If a StochasticParameter, then from that parameter a value will be sampled per image (again, used for both x- and y-axis).
  - If a dictionary, then it is expected to have the keys “x” and/or “y”. Each of these keys can have the same values as described before for this whole parameter (translate_percent).
Using a dictionary allows to set different values for the axis. If they are set to the same ranges, different values may still be sampled per axis.

- **translate_px** *(None or int or tuple of int or list of int or imgaug.parameters.StochasticParameter or dict {“x”: int/tuple/list/StochasticParameter, “y”: int/tuple/list/StochasticParameter}, optional)* – Translation in pixels.
  - If None then equivalent to 0.0 unless translate_percent has a non-None value.
  - If a single int, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value will be sampled from the discrete range \([a..b]\) per image. That number will be used identically for both x- and y-axis.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then from that parameter a value will be sampled per image (again, used for both x- and y-axis).
  - If a dictionary, then it is expected to have the keys “x” and/or “y”. Each of these keys can have the same values as described before for this whole parameter (translate_px). Using a dictionary allows to set different values for the axis. If they are set to the same ranges, different values may still be sampled per axis.

- **rotate** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Rotation in degrees (_NOT_ radians), i.e. expected value range is 0 to 360 for positive rotations (may also be negative). Rotation happens around the _center_ of the image, not the top left corner as in some other frameworks.
  - If a number, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value will be sampled per image from the range \(a <= x <= b\) and be used as the rotation value.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then this parameter will be used to sample the rotation value per image.

- **shear** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Shear in degrees (_NOT_ radians), i.e. expected value range is 0 to 360 for positive shear (may also be negative).
  - If a float/int, then that value will be used for all shear (may also be negative).
  - If a tuple \((a, b)\), then a value will be sampled per image from the range \(a <= x <= b\) and be used as the rotation value.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then this parameter will be used to sample the shear value per image.

- **order** *(int or iterable of int or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* – Interpolation order to use. Same meaning as in skimage:
  - 0: Nearest-neighbor
  - 1: Bi-linear (default)
- 2: Bi-quadratic (not recommended by skimage)
- 3: Bi-cubic
- 4: Bi-quartic
- 5: Bi-quintic

Method 0 and 1 are fast, 3 is a bit slower, 4 and 5 are very slow. If the backend is cv2, the mapping to OpenCV's interpolation modes is as follows:
- 0 -> cv2.INTER_NEAREST
- 1 -> cv2.INTER_LINEAR
- 2 -> cv2.INTER_CUBIC
- 3 -> cv2.INTER_CUBIC
- 4 -> cv2.INTER_CUBIC

As datatypes this parameter accepts:
- If a single int, then that order will be used for all images.
- If an iterable, then for each image a random value will be sampled from that iterable (i.e. list of allowed order values).
- If imgaug.ALL, then equivalent to list [0, 1, 3, 4, 5] in case of backend skimage and otherwise [0, 1, 3].
- If StochasticParameter, then that parameter is queried per image to sample the order value to use.

**cval** *(number or tuple of number or list of number or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* – The constant value used for skimage's transform function. This is the value used to fill up pixels in the result image that didn't exist in the input image (e.g. when translating to the left, some new pixels are created at the right). Such a fill-up with a constant value only happens, when *mode* is “constant”. The expected value range is [0, 255]. It may be a float value.
- If this is a single number, then that value will be used (e.g. 0 results in black pixels).
- If a tuple (a, b), then a random value from the range a <= x <= b is picked per image.
- If a list, then a random value will be sampled from that list per image.
- If imgaug.ALL, a value from the discrete range [0 .. 255] will be sampled per image.
- If a StochasticParameter, a new value will be sampled from the parameter per image.

**fit_output** *(bool, optional)* – Whether the image after affine transformation is completely contained in the output image. If False, parts of the image may be outside of the image plane or the image might make up only a small part of the image plane. Activating this can be useful e.g. for rotations by 45 degrees to avoid that the image corners are outside of the image plane. Note that activating this will negate translation. Note also that activating this may lead to image sizes differing from the input image sizes. To avoid this, wrap Affine in KeepSizeByResize, e.g. KeepSizeByResize(Affine(...)).

**mode** *(str or list of str or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* – Parameter that defines the handling of newly created pixels. Same meaning as in skimage (and numpy.pad):
- constant: Pads with a constant value
- **edge**: Pads with the edge values of array
- **symmetric**: Pads with the reflection of the vector mirrored along the edge of the array.
- **reflect**: Pads with the reflection of the vector mirrored on the first and last values of the vector along each axis.
- **wrap**: Pads with the wrap of the vector along the axis. The first values are used to pad the end and the end values are used to pad the beginning.

If `cv2` is chosen as the backend the mapping is as follows:

- **constant** -> `cv2.BORDER_CONSTANT`
- **edge** -> `cv2.BORDER_REPLICATE`
- **symmetric** -> `cv2.BORDER_REFLECT`
- **reflect** -> `cv2.BORDER_REFLECT_101`
- **wrap** -> `cv2.BORDER_WRAP`

The datatype of the parameter may be:

- If a single string, then that mode will be used for all images.
- If a list of strings, then per image a random mode will be picked from that list.
- If `imgaug.ALL`, then a random mode from all possible modes will be picked.
- If `StochasticParameter`, then the mode will be sampled from that parameter per image, i.e. it must return only the above mentioned strings.

- **backend** (str, optional) – Framework to use as a backend. Valid values are `auto`, `skimage` (scikit-image’s warp) and `cv2` (OpenCV’s warp). If `auto` is used, the augmenter will automatically try to use `cv2` where possible (order must be in `[0, 1, 3]` and image’s dtype uint8, otherwise skimage is chosen). It will silently fall back to skimage if order/dtype is not supported by cv2. cv2 is generally faster than skimage. It also supports RGB cvals, while skimage will resort to intensity cvals (i.e. 3x the same value as RGB). If `cv2` is chosen and order is 2 or 4, it will automatically fall back to order 3.

- **name** (None or str, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **deterministic** (bool, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

- **random_state** (None or int or `numpy.random.RandomState`, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`.

### Examples

```python
>>> aug = iaa.Affine(scale=2.0)
```

zooms all images by a factor of 2.

```python
>>> aug = iaa.Affine(translate_px=16)
```

translates all images on the x- and y-axis by 16 pixels (to the right/top), fills up any new pixels with zero (black values).
translates all images on the x- and y-axis by 10 percent of their width/height (to the right/top), fills up any new pixels with zero (black values).

rotates all images by 35 degrees, fills up any new pixels with zero (black values).

rotates all images by 15 degrees, fills up any new pixels with zero (black values).

translates all images on the x- and y-axis by a random value between -16 and 16 pixels (to the right/top) (same for both axis, i.e. sampled once per image), fills up any new pixels with zero (black values).

translates all images on the x-axis by a random value between -16 and 16 pixels (to the right) and on the y-axis by a random value between -4 and 4 pixels to the top. Even if both ranges were the same, both axis could use different samples. Fills up any new pixels with zero (black values).

same as previously, but uses (randomly) either nearest neighbour interpolation or linear interpolation.

same as previously, but fills up any new pixels with a random brightness (same for the whole image).

same as previously, but fills up the new pixels in only 50 percent of all images with black values. In the other 50 percent of all cases, the value of the nearest edge is used.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| __call__(*args, **kwargs) | Alias for `imgaug.augmenters.meta.Augmenter.augment()`.
| augment([return_batch, hooks]) | Augment data.
| augment_batch(batch[, hooks]) | Augment a single batch.
| augment_batches(batches[, hooks, background]) | Augment multiple batches.
| augment_bounding_boxes(bounding_boxes_on_images) | Augment bounding boxes.
| augment_heatmaps(heatmaps[, parents, hooks]) | Augment a heatmap.
| augment_image(image[, hooks]) | Augment a single image.
| augment_images(images[, parents, hooks]) | Augment multiple images.
| augment_keypoints(keypoints_on_images[, ...]) | Augment image keypoints.
| augment_line_strings(line_strings_on_images) | Augment line strings.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([in])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters

VALID_DTYPES_CV2_ORDER_0 = {'bool', 'float16', 'float32', 'float64', 'int16', 'int32', 'int8', 'uint16', 'uint8'}
VALID_DTYPES_CV2_ORDER_NOT_0 = {'bool', 'float16', 'float32', 'float64', 'int16', 'int32', 'int8', 'uint16', 'uint8'}

class imgaug.augmenters.geometric.AffineCv2(scale=1.0, translate_percent=None, translate_px=None, rotate=0.0, shear=0.0, order=MagicMock(id='140619346153368'), cval=0, mode=MagicMock(id='14061934613368'), name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter to apply affine transformations to images using cv2 (i.e. opencv) backend.
NOTE: This augmenter will likely be removed in the future as Affine() already offers a cv2 backend (use backend="cv2").

Affine transformations involve:

- Translation (“move” image on the x-/y-axis)
- Rotation
- Scaling (“zoom” in/out)
- Shear (move one side of the image, turning a square into a trapezoid)

All such transformations can create “new” pixels in the image without a defined content, e.g. if the image is translated to the left, pixels are created on the right. A method has to be defined to deal with these pixel values. The parameters cval and mode of this class deal with this.

Some transformations involve interpolations between several pixels of the input image to generate output pixel values. The parameter order deals with the method of interpolation used for this.

**dtype support:**

```
* `"uint8``": yes; fully tested
* `"uint16``": ?
* `"uint32``": ?
* `"uint64``": ?
* `"int8``": ?
* `"int16``": ?
* `"int32``": ?
* `"int64``": ?
* `"float16``": ?
* `"float32``": ?
* `"float64``": ?
* `"float128``": ?
* `"bool``": ?
```

**Parameters**

- **scale** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter or dict {“x”: number/tuple/list/StochasticParameter, “y”: number/tuple/list/StochasticParameter}, optional)* – Scaling factor to use, where 1.0 represents no change and 0.5 is zoomed out to 50 percent of the original size.
  - If a single float, then that value will be used for all images.
  - If a tuple (a, b), then a value will be sampled from the range a <= x <= b per image. That value will be used identically for both x- and y-axis.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then from that parameter a value will be sampled per image (again, used for both x- and y-axis).
  - If a dictionary, then it is expected to have the keys “x” and/or “y”. Each of these keys can have the same values as described before for this whole parameter (scale). Using a dictionary allows to set different values for the axis. If they are set to the same ranges, different values may still be sampled per axis.

- **translate_percent** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter or dict {“x”: number/tuple/list/StochasticParameter, “y”: number/tuple/list/StochasticParameter}, optional)* – Translation in percent relative to
the image height/width (x-translation, y-translation) to use, where 0 represents no change and 0.5 is half of the image height/width.

- If a single float, then that value will be used for all images.
- If a tuple \((a, b)\), then a value will be sampled from the range \(a \leq x \leq b\) per image. That percent value will be used identically for both x- and y-axis.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then from that parameter a value will be sampled per image (again, used for both x- and y-axis).
- If a dictionary, then it is expected to have the keys “x” and/or “y”. Each of these keys can have the same values as described before for this whole parameter (\(translate\_\text{percent}\)). Using a dictionary allows to set different values for the axis. If they are set to the same ranges, different values may still be sampled per axis.

- **\(translate\_\text{px}\)** (int or tuple of int or list of int or imgaug.parameters.StochasticParameter or dict \{“x”: int/tuple/list/StochasticParameter, “y”: int/tuple/list/StochasticParameter\}, optional) – Translation in pixels.
  - If a single int, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value will be sampled from the discrete range \([a..b]\) per image. That number will be used identically for both x- and y-axis.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then from that parameter a value will be sampled per image (again, used for both x- and y-axis).
  - If a dictionary, then it is expected to have the keys “x” and/or “y”. Each of these keys can have the same values as described before for this whole parameter \(translate\_\text{px}\). Using a dictionary allows to set different values for the axis. If they are set to the same ranges, different values may still be sampled per axis.

- **\(rotate\)** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – Rotation in degrees (\(_\text{NOT}_\) radians), i.e. expected value range is 0 to 360 for positive rotations (may also be negative). Rotation happens around the \_center\_ of the image, not the top left corner as in some other frameworks.
  - If a float/int, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value will be sampled per image from the range \(a \leq x \leq b\) and be used as the rotation value.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then this parameter will be used to sample the rotation value per image.

- **\(shear\)** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – Shear in degrees (\(_\text{NOT}_\) radians), i.e. expected value range is 0 to 360 for positive shear (may also be negative).
  - If a float/int, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value will be sampled per image from the range \(a \leq x \leq b\) and be used as the rotation value.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then this parameter will be used to sample the shear value per image.

**order** *(int or list of int or str or list of str or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* –
Interpolation order to use. Allowed are:
- cv2.INTER_NEAREST - a nearest-neighbor interpolation
- cv2.INTER_LINEAR - a bilinear interpolation (used by default)
- cv2.INTER_CUBIC - a bicubic interpolation over 4x4 pixel neighborhood
- cv2.INTER_LANCZOS4
- nearest
- linear
- cubic
- lanczos4

The first four are OpenCV constants, the other four are strings that are automatically replaced by the OpenCV constants. INTER_NEAREST (nearest neighbour interpolation) and INTER_NEAREST (linear interpolation) are the fastest.
- If a single int, then that order will be used for all images.
- If a string, then it must be one of: nearest, linear, cubic, lanczos4.
- If an iterable of int/string, then for each image a random value will be sampled from that iterable (i.e. list of allowed order values).
- If imgaug.ALL, then equivalent to list [cv2.INTER_NEAREST, cv2.INTER_LINEAR, cv2.INTER_CUBIC, cv2.INTER_LANCZOS4].
- If StochasticParameter, then that parameter is queried per image to sample the order value to use.

**cval** *(number or tuple of number or list of number or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* – The constant value used to fill up pixels in the result image that didn’t exist in the input image (e.g. when translating to the left, some new pixels are created at the right). Such a fill-up with a constant value only happens, when *mode* is “constant”. The expected value range is [0, 255]. It may be a float value.
- If this is a single int or float, then that value will be used (e.g. 0 results in black pixels).
- If a tuple (a, b), then a random value from the range a <= x <= b is picked per image.
- If a list, then a random value will be sampled from that list per image.
- If imgaug.ALL, a value from the discrete range [0 .. 255] will be sampled per image.
- If a StochasticParameter, a new value will be sampled from the parameter per image.

**mode** *(int or str or list of str or list of int or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* – optional Parameter that defines the handling of newly created pixels. Same meaning as in OpenCV’s border mode. Let abcdefgh be an image’s content and | be an image boundary, then:
- cv2.BORDER_REPLICATE: aaaaaa|abcdefg|hahahah
- `cv2.BORDER_REFLECT`: fedcba|abcdefgh|hgfedcb
- `cv2.BORDER_REFLECT_101`: gfedcb|abcdefgh|gfedcba
- `cv2.BORDER_WRAP`: cdefgh|abcdefg|abcdefg
- `cv2.BORDER_CONSTANT`: iiiiiii|abcdefg|iiiiiiii, where i is the defined cval.
- `replicate`: Same as `cv2.BORDER_REPLICATE`.
- `reflect`: Same as `cv2.BORDER_REFLECT`.
- `reflect_101`: Same as `cv2.BORDER_REFLECT_101`.
- `wrap`: Same as `cv2.BORDER_WRAP`.
- `constant`: Same as `cv2.BORDER_CONSTANT`.

The datatype of the parameter may be:
- If a single int, then it must be one of `cv2.BORDER_*`.
- If a single string, then it must be one of: `replicate`, `reflect`, `reflect_101`, `wrap`, `constant`.
- If a list of ints/strings, then per image a random mode will be picked from that list.
- If `imgaug.ALL`, then a random mode from all possible modes will be picked.
- If StochasticParameter, then the mode will be sampled from that parameter per image, i.e. it must return only the above mentioned strings.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`
- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`
- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`

### Examples

```python
>>> aug = iaa.Affine(scale=2.0)
```
zooms all images by a factor of 2.

```python
>>> aug = iaa.Affine(translate_px=16)
```
translates all images on the x- and y-axis by 16 pixels (to the right/top), fills up any new pixels with zero (black values).

```python
>>> aug = iaa.Affine(translate_percent=0.1)
```
translates all images on the x- and y-axis by 10 percent of their width/height (to the right/top), fills up any new pixels with zero (black values).

```python
>>> aug = iaa.Affine(rotate=35)
```
rotates all images by 35 degrees, fills up any new pixels with zero (black values).
>>> aug = iaa.AffineCv2(shear=15)
rotates all images by 15 degrees, fills up any new pixels with zero (black values).

>>> aug = iaa.AffineCv2(translate_px=(-16, 16))
translates all images on the x- and y-axis by a random value between -16 and 16 pixels (to the right/top) (same for both axis, i.e. sampled once per image), fills up any new pixels with zero (black values).

>>> aug = iaa.AffineCv2(translate_px={"x": (-16, 16), "y": (-4, 4)})
translates all images on the x-axis by a random value between -16 and 16 pixels (to the right) and on the y-axis by a random value between -4 and 4 pixels to the top. Even if both ranges were the same, both axis could use different samples. Fills up any new pixels with zero (black values).

>>> aug = iaa.AffineCv2(scale=2.0, order=[0, 1])
same as previously, but uses (randomly) either nearest neighbour interpolation or linear interpolation.

>>> aug = iaa.AffineCv2(translate_px=16, cval=(0, 255))
same as previously, but fills up any new pixels with a random brightness (same for the whole image).

>>> aug = iaa.AffineCv2(translate_px=16, mode=["constant", "replicate"])
same as previously, but fills up the new pixels in only 50 percent of all images with black values. In the other 50 percent of all cases, the value of the closest edge is used.

Methods

__call__(*args, **kwargs)
Alias for \texttt{imgaug.augmenters.meta.Augmenter.augment().}

augment([\texttt{return\_batch}, \texttt{hooks}])
Augment data.

augment\_batch(batch[, hooks])
Augment a single batch.

augment\_batches(batches[, hooks, \texttt{background}])
Augment multiple batches.

augment\_bounding\_boxes(bounding\_boxes_on\_images)
Augment bounding boxes.

augment\_heatmaps(heatmaps[, parents, hooks])
Augment a heatmap.

augment\_image(image[, hooks])
Augment a single image.

augment\_images(images[, parents, hooks])
Augment multiple images.

augment\_keypoints(keypoints\_on\_images[, \ldots])
Augment image keypoints.

augment\_line\_strings(line\_strings\_on\_images)
Augment line strings.

augment\_polygons(polygons\_on\_images[, \ldots])
Augment polygons.

augment\_segmentation\_maps(segmaps[, \ldots])
Augment segmentation maps.

copy()
Create a shallow copy of this Augmenter instance.

copy\_random\_state(source[, recursive, \ldots])
Copy the random states from a source augmenter sequence.

copy\_random\_state\_inplace(source[, recursive, \ldots])
Copy the random states from a source augmenter sequence (inplace).

Continued on next page
Table 80 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([in])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters()

class imgaug.augmenters.geometric.ElasticTransformation(alpha=0,          sigma=0,
                                                         order=3,  cval=0,
                                                         mode='constant', polygon_recoverer='auto',
                                                         name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter to transform images by moving pixels locally around using displacement fields.

The augmenter has the parameters `alpha` and `sigma`. `alpha` controls the strength of the displacement: higher values mean that pixels are moved further. `sigma` controls the smoothness of the displacement: higher values lead to smoother patterns – as if the image was below water – while low values will cause individual pixels to be moved very differently from their neighbours, leading to noisy and pixelated images.

A relation of 10:1 seems to be good for `alpha` and `sigma`, e.g. `alpha=10` and `sigma=1` or `alpha=50`, `sigma=5`. For 128x128 a setting of `alpha=(0, 70.0), sigma=(4.0, 6.0)` may be a good choice and will lead to a water-like effect.

See

Simard, Steinkraus and Platt
Best Practices for Convolutional Neural Networks applied to Visual Document Analysis

(continues on next page)
for a detailed explanation.

dtype support:

<table>
<thead>
<tr>
<th>DType</th>
<th>Support</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>yes; fully tested (1)</td>
<td>(1) Always handled by <code>cv2</code></td>
</tr>
<tr>
<td>uint16</td>
<td>yes; tested (1)</td>
<td>(2) Always handled by <code>scipy</code></td>
</tr>
<tr>
<td>uint32</td>
<td>yes; tested (2)</td>
<td></td>
</tr>
<tr>
<td>uint64</td>
<td>limited; tested (3)</td>
<td>(3) Only supported for <code>order != 0</code>. Will fail for <code>order=0</code>.</td>
</tr>
<tr>
<td>int8</td>
<td>yes; tested (1) (4) (5)</td>
<td>(4) Mapped internally to <code>float64</code> when <code>order=1</code>.</td>
</tr>
<tr>
<td>int16</td>
<td>yes; tested (4) (6)</td>
<td>(5) Mapped internally to <code>int16</code> when <code>order&gt;=2</code>.</td>
</tr>
<tr>
<td>int32</td>
<td>yes; tested (4) (6)</td>
<td>(6) Handled by <code>cv2</code> when <code>order=0</code> or <code>order=1</code>, otherwise by <code>scipy</code>.</td>
</tr>
<tr>
<td>int64</td>
<td>limited; tested (3)</td>
<td></td>
</tr>
<tr>
<td>float16</td>
<td>yes; tested (1)</td>
<td></td>
</tr>
<tr>
<td>float32</td>
<td>yes; tested (1)</td>
<td></td>
</tr>
<tr>
<td>float64</td>
<td>yes; tested (1)</td>
<td></td>
</tr>
<tr>
<td>float128</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>bool</td>
<td>yes; tested (1) (7)</td>
<td>(7) Mapped internally to <code>float32</code>.</td>
</tr>
</tbody>
</table>

Parameters

- **alpha** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Strength of the distortion field. Higher values mean that pixels are moved further with respect to the distortion field’s direction. Set this to around 10 times the value of sigma for visible effects.
  
  - If number, then that value will be used for all images.
  
  - If tuple `(a, b)`, then a random value from range `a <= x <= b` will be sampled per image.
  
  - If a list, then for each image a random value will be sampled from that list.
  
  - If StochasticParameter, then that parameter will be used to sample a value per image.

- **sigma** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Standard deviation of the gaussian kernel used to smooth the distortion fields. Higher values (for 128x128 images around 5.0) lead to more water-like effects, while lower values (for 128x128 images around 1.0 and lower) lead to more noisy, pixelated images. Set this to around 1/10th of alpha for visible effects.
  
  - If number, then that value will be used for all images.
  
  - If tuple `(a, b)`, then a random value from range `a <= x <= b` will be sampled per image.
  
  - If a list, then for each image a random value will be sampled from that list.
  
  - If StochasticParameter, then that parameter will be used to sample a value per image.
• **order** (int or list of int or imgaug.ALL or imgaug.parameters.StochasticParameter, optional) – Interpolation order to use. Same meaning as in scipy.ndimage.map_coordinates() and may take any integer value in the range 0 to 5, where orders close to 0 are faster.
  
  – If a single int, then that order will be used for all images.
  
  – If a tuple \((a, b)\), then a random value from the range \(a <= x <= b\) is picked per image.
  
  – If a list, then for each image a random value will be sampled from that list.
  
  – If imgaug.ALL, then equivalent to list \([0, 1, 2, 3, 4, 5]\).
  
  – If StochasticParameter, then that parameter is queried per image to sample the order value to use.

• **cval** (number or tuple of number or list of number or imgaug.ALL or imgaug.parameters.StochasticParameter, optional) – The constant intensity value used to fill in new pixels. This value is only used if **mode** is set to “constant”. For standard uint8 images (value range 0-255), this value may also come from the range 0-255. It may be a float value, even for integer image dtypes.
  
  – If this is a single int or float, then that value will be used (e.g. 0 results in black pixels).
  
  – If a tuple \((a, b)\), then a random value from the range \(a <= x <= b\) is picked per image.
  
  – If a list, then a random value will be picked from that list per image.
  
  – If imgaug.ALL, a value from the discrete range \([0 .. 255]\) will be sampled per image.
  
  – If a StochasticParameter, a new value will be sampled from the parameter per image.

• **mode** (str or list of str or imgaug.ALL or imgaug.parameters.StochasticParameter, optional) – Parameter that defines the handling of newly created pixels. May take the same values as in scipy.ndimage.map_coordinates(), i.e. constant, nearest, reflect or wrap. The datatype of the parameter may be:
  
  – If a single string, then that mode will be used for all images.
  
  – If a list of strings, then per image a random mode will be picked from that list.
  
  – If imgaug.ALL, then a random mode from all possible modes will be picked.
  
  – If StochasticParameter, then the mode will be sampled from that parameter per image, i.e. it must return only the above mentioned strings.

• **polygon_recoverer** (‘auto’ or None or) – imgaug.imgaug._ConcavePolygonRecoverer, optional The class to use to repair invalid polygons. If "auto", a new instance of imgaug._ConcavePolygonRecoverer will be created. If None, no polygon recoverer will be used. If an object, then that object will be used and must provide a recover_from() method, similar to imgaug.imgaug._ConcavePolygonRecoverer.

• **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

• **deterministic** (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

• **random_state** (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
Examples

```python
>>> aug = iaa.ElasticTransformation(alpha=50.0, sigma=5.0)
apply elastic transformations with a strength/alpha of 50.0 and smoothness of 5.0 to all images.
```  
```python
>>> aug = iaa.ElasticTransformation(alpha=(0.0, 70.0), sigma=5.0)
apply elastic transformations with a strength/alpha that comes from the range 0.0 <= x <= 70.0 (randomly picked per image) and with a smoothness of 5.0.
```  
Methods

```python
__call__(*args, **kwargs)
```  
Alien for `imgaug.augmenters.meta.Augmenter.augment()`.

```python
augment([return_batch, hooks])
```  
Augment data.

```python
augment_batch(batch[, hooks])
```  
Augment a single batch.

```python
augment_batches(batches[, hooks, background])
```  
Augment multiple batches.

```python
augment_bounding_boxes(bounding_boxes_on_images)
```  
Augment bounding boxes.

```python
augment_heatmaps(heatmaps[, parents, hooks])
```  
Augment a heatmap.

```python
augment_image(image[, hooks])
```  
Augment a single image.

```python
augment_images(images[, parents, hooks])
```  
Augment multiple images.

```python
augment_keypoints(keypoints_on_images[, ...])
```  
Augment image keypoints.

```python
augment_line_strings(line_strings_on_images)
```  
Augment line strings.

```python
augment_polygons(polygons_on_images[, ...])
```  
Augment polygons.

```python
augment_segmentation_maps(segmaps[, ...])
```  
Augment segmentation maps.

```python
copy()
```  
Create a shallow copy of this Augmenter instance.

```python
copy_random_state(source[, recursive, ...])
```  
Copy the random states from a source augmenter sequence.

```python
copy_random_state_(source[, recursive, ...])
```  
Copy the random states from a source augmenter sequence (inplace).

```python
deepeopy()
```  
Create a deep copy of this Augmenter instance.

```python
draw_grid(images, rows, cols)
```  
Apply this augmenter to the given images and return a grid image of the results.

```python
find_augmenters(func[, parents, flat])
```  
Find augmenters that match a condition.

```python
find_augmenters_by_name(name[, regex, flat])
```  
Find augmenter(s) by name.

```python
find_augmenters_by_names(names[, regex, flat])
```  
Find augmenter(s) by names.

```python
get_all_children([flat])
```  
Returns all children of this augmenter as a list.

```python
get_children_lists()
```  
Get a list of lists of children of this augmenter.

```python
localize_random_state([recursive])
```  
Converts global random states to local ones.

```python
localize_random_state_(recursive)
```  
Converts global random states to local ones.

```python
map_coordinates(image, dx, dy[, order, ...])
```  
backend="scipy"

```python
pool([processes, maxtasksperchild, seed])
```  
Create a pool used for multicore augmentation from this augmenter.
Table 81 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove_augmenters(func, copy, noop_if_topmost)</td>
<td></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func, parents)</td>
<td></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic(In)</td>
<td></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

```python
generate_shift_maps
get_parameters
```

KEYPOINT_AUG_ALPHA_THRESH = 0.05
KEYPOINT_AUG_SIGMA_THRESH = 1.0
NB_NEIGHBOURING_KEYPOINTS = 3
NEIGHBOURING_KEYPOINTS_DISTANCE = 1.0

classmethod generate_shift_maps(shape, alpha, sigma, random_state)
classmethod get_parameters()
classmethod map_coordinates(image, dx, dy, order=1, cval=0, mode='constant')
    backend="scipy"
    order=0
        • uint8: yes
        • uint16: yes
        • uint32: yes
        • uint64: no (produces array filled with only 0)
        • int8: yes
        • int16: yes
        • int32: yes
        • int64: no (produces array filled with <min_value> when testing with <max_value>)
        • float16: yes
        • float32: yes
        • float64: yes
        • float128: no (causes: ‘data type no supported’)
        • bool: yes

    order=1 to 5
        • uint*, int*: yes (rather loose test, to avoid having to re-compute the interpolation)
        • float16 - float64: yes (rather loose test, to avoid having to re-compute the interpolation)
        • float128: no (causes: ‘data type no supported’)
```
- bool: yes
  backend="cv2"

**order=0**

- uint8: yes
- uint16: yes
- uint32: no (causes: src data type = 6 is not supported)
- uint64: no (silently converts to int32)
- int8: yes
- int16: yes
- int32: yes
- int64: no (silently converts to int32)
- float16: yes
- float32: yes
- float64: yes
- float128: no (causes: src data type = 13 is not supported)
- bool: no (causes: src data type = 0 is not supported)

**order=1**

- uint8: yes
- uint16: yes
- uint32: no (causes: src data type = 6 is not supported)
- float16: yes
- float32: yes
- float64: yes
- float128: no (causes: src data type = 13 is not supported)
- bool: no (causes: src data type = 0 is not supported)

**order=2 to 5:** as order=1, but int16 supported
class imgaug.augmenters.geometric.PerspectiveTransform(scale=0, cval=0,  
mode='constant',  
keep_size=True,  
name=None,  
deterministic=False,  
random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter that performs a random four point perspective transform.

Each of the four points is placed on the image using a random distance from its respective corner. The distance is sampled from a normal distribution. As a result, most transformations don’t change very much, while some “focus” on polygons far inside the image.

The results of this augmenter have some similarity with Crop.


dtype support:

if (keep_size=False):
    
    * `\'uint8\'`: yes; fully tested
    * `\'uint16\'`: yes; tested
    * `\'uint32\'`: no (1)
    * `\'uint64\'`: no (2)
    * `\'int8\'`: yes; tested (3)
    * `\'int16\'`: yes; tested
    * `\'int32\'`: no (2)
    * `\'int64\'`: no (2)
    * `\'float16\'`: yes; tested (4)
    * `\'float32\'`: yes; tested
    * `\'float64\'`: yes; tested
    * `\'float128\'`: no (1)
    * `\'bool\'`: yes; tested (4)

- (1) rejected by opencv
- (2) leads to opencv error: cv2.error: `\'OpenCV(3.4.4) (...)imgwarp. \ncpp:1805: error: (-215:Assertion failed) ifunc != 0 in function \'remap\'``.
- (3) mapped internally to `\'int16\'`.
- (4) mapped internally to `\'float32\'`.

if (keep_size=True):

    minimum of (
        `\'imgaug.augmenters.geometric.PerspectiveTransform(keep_size=False)\'',
        :func:`imgaug.imgaug.imresize_many_images`
    )

Parameters

- **scale** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Standard deviation of the normal distributions. These are used to sample the random distances of the subimage’s corners from the full image’s corners. The sampled values reflect percentage values (with respect to image height/width). Recommended values are in the range 0.0 to 0.1.

    – If a single number, then that value will always be used as the scale.
– If a tuple \((a, b)\) of numbers, then a random value will be picked from the interval \((a, b)\) (per image).

– If a list of values, a random one of the values will be picked per image.

– If a StochasticParameter, then that parameter will be queried to draw one value per image.

• **keep_size** (bool, optional) – Whether to resize image’s back to their original size after applying the perspective transform. If set to False, the resulting images may end up having different shapes and will always be a list, never an array.

• **cval** (number or tuple of number or list of number or imgaug.ALL or imgaug.parameters.StochasticParameter, optional) – The constant value used to fill up pixels in the result image that didn’t exist in the input image (e.g. when translating to the left, some new pixels are created at the right). Such a fill-up with a constant value only happens, when `mode` is “constant”. The expected value range is \([0, 255]\). It may be a float value.

  – If this is a single int or float, then that value will be used (e.g. 0 results in black pixels).

  – If a tuple \((a, b)\), then a random value from the range \(a \leq x \leq b\) is picked per image.

  – If a list, then a random value will be sampled from that list per image.

  – If imgaug.ALL, a value from the discrete range \([0 \ldots 255]\) will be sampled per image.

  – If a StochasticParameter, a new value will be sampled from the parameter per image.

• **mode** (int or str or list of str or list of int or imgaug.ALL or imgaug.parameters.StochasticParameter, optional) – optional Parameter that defines the handling of newly created pixels. Same meaning as in OpenCV’s border mode. Let abcdefgh be an image’s content and | be an image boundary, then:

  – `cv2.BORDER_REPLICATE`: aaaaaa|abcdefgh|hyyyyy

  – `cv2.BORDER_CONSTANT`: iiiiii|abcdefgh|iiiiiii, where i is the defined cval.

  – `replicate`: Same as `cv2.BORDER_REPLICATE`.

  – `constant`: Same as `cv2.BORDER_CONSTANT`.

The datatype of the parameter may be:

– If a single int, then it must be one of `cv2.BORDER_*`.

– If a single string, then it must be one of: `replicate`, `reflect`, `reflect_101`, `wrap`, `constant`.

– If a list of ints/strings, then per image a random mode will be picked from that list.

– If imgaug.ALL, then a random mode from all possible modes will be picked.

– If StochasticParameter, then the mode will be sampled from that parameter per image, i.e. it must return only the above mentioned strings.

• **name** (None or str, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

• **deterministic** (bool, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

• **random_state** (None or int or numpy.random.RandomState, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

396 Chapter 13. API
Examples

```python
>>> aug = iaa.PerspectiveTransform(scale=(0.01, 0.10))
```

Applies perspective transformations using a random scale between 0.01 and 0.1 per image, where the scale is roughly a measure of how far the perspective transform’s corner points may be distanced from the original image’s corner points.

Methods

```python
__call__(*args, **kwargs) Alias for imgaug.augmenters.meta.Augmenter.augment()

augment([return_batch, hooks]) Augment data.

augmentbatch(batch[, hooks]) Augment a single batch.

augmentbatches(batches[, hooks, background]) Augment multiple batches.

augmentbounding_boxes(bounding_boxes[, augment bounding boxes])

augment_heatmaps(heatmaps[, parents, hooks]) Augment a heatmap.

augment_image(image[, hooks]) Augment a single image.

augment_images(images[, parents, hooks]) Augment multiple images.

augment_keypoints(keypoints_on_images[, ...]) Augment image keypoints.

augment_line_strings(line_strings_on_images) Augment line strings.

augment_polygons(polygons_on_images[, ...]) Augment polygons.

augment_segmentation_maps(segmaps[, ...]) Augment segmentation maps.

copy() Create a shallow copy of this Augmenter instance.

copy_random_state(source[, recursive, ...]) Copy the random states from a source augmenter sequence.

copy_random_state_(source[, recursive, ...]) Copy the random states from a source augmenter sequence (inplace).

deepecopy() Create a deep copy of this Augmenter instance.

draw_grid(images, rows, cols) Apply this augmenter to the given images and return a grid image of the results.

find_augmenters(func[, parents, flat]) Find augmenters that match a condition.

find_augmenters_by_name(name[, regex, flat]) Find augmenter(s) by name.

find_augmenters_by_names(names[, regex, flat]) Find augmenter(s) by names.

get_all_children([flat]) Returns all children of this augmenter as a list.

get_children_lists() Get a list of lists of children of this augmenter.

localize_random_state([recursive]) Converts global random states to local ones.

localize_random_state_([recursive]) Converts global random states to local ones.

pool([processes, maxtasksperchild, seed]) Create a pool used for multicore augmentation from this augmenter.

remove_augmenters(func[, noop_if_topmost]) Remove this augmenter or its children that match a condition.

remove_augmenters_inplace(func[, parents]) Remove in-place children of this augmenter that match a condition.
```

Continued on next page
Table 82 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_parameters()**

```python
get_parameters()
```

```python
class imgaug.augmenters.geometric.PiecewiseAffine(scale=0, nb_rows=4, nb_cols=4, order=1, cval=0, mode='constant', absolute_scale=False, polygon_recoverer=None, name=None, deterministic=False, random_state=None)
```

**Bases:** `imgaug.augmenters.meta.Augmenter`

Augmenter that places a regular grid of points on an image and randomly moves the neighbourhood of these point around via affine transformations. This leads to local distortions.

This is mostly a wrapper around scikit-image’s PiecewiseAffine. See also the Affine augmenter for a similar technique.

**dtype support:**

- `'uint8'`: yes; fully tested
- `'uint16'`: yes; tested (1)
- `'uint32'`: yes; tested (1) (2)
- `'uint64'`: no (3)
- `'int8'`: yes; tested (1)
- `'int16'`: yes; tested (1)
- `'int32'`: yes; tested (1) (2)
- `'int64'`: no (3)
- `'float16'`: yes; tested (1)
- `'float32'`: yes; tested (1)
- `'float64'`: yes; tested (1)
- `'float128'`: no (3)
- `'bool'`: yes; tested (1) (4)

- (1) Only tested with 'order' set to 0.
- (2) scikit-image converts internally to `'float64'`, which might introduce inaccuracies.
  Tests showed that these inaccuracies seemed to not be an issue.
- (3) results too inaccurate
- (4) mapped internally to `'float64'`

**Parameters**

- **scale** (`float` or `tuple of float` or `imgaug.parameters.StochasticParameter, optional`) – Each point on the regular grid is moved around via a normal distribution. This scale factor is equivalent to the normal distribution’s sigma. Note that the jitter (how far each point is moved in which direction) is multiplied by the height/width of the image if `absolute_scale=False` (default), so this scale can be the same for different sized images. Recommended values are in the range 0.01 to 0.05 (weak to strong augmentations).
- If a single float, then that value will always be used as the scale.
- If a tuple \((a, b)\) of floats, then a random value will be picked from the interval \((a, b)\) (per image).
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then that parameter will be queried to draw one value per image.

- **nb_rows** *(int or tuple of int or imgaug.parameters.StochasticParameter, optional)* – Number of rows of points that the regular grid should have. Must be at least 2. For large images, you might want to pick a higher value than 4. You might have to then adjust scale to lower values.
  - If a single int, then that value will always be used as the number of rows.
  - If a tuple \((a, b)\), then a value from the discrete interval \([a..b]\) will be sampled per image.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then that parameter will be queried to draw one value per image.

- **nb_cols** *(int or tuple of int or imgaug.parameters.StochasticParameter, optional)* – Number of columns. See **nb_rows**.

- **order** *(int or list of int or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* – See `imgaug.augmenters.geometric.Affine.__init__()`.  

- **cval** *(int or float or tuple of float or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* – See `imgaug.augmenters.geometric.Affine.__init__()`.  

- **mode** *(str or list of str or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)* – See `imgaug.augmenters.geometric.Affine.__init__()`.  

- **absolute_scale** *(bool, optional)* – Take `scale` as an absolute value rather than a relative value.

- **polygon_recoverer** *(‘auto’ or None or imgaug.imgaug._ConcavePolygonRecoverer, optional)* – The class to use to repair invalid polygons. If "auto", a new instance of `imgaug._ConcavePolygonRecoverer` will be created. If `None`, no polygon recoverer will be used. If an object, then that object will be used and must provide a `recover_from()` method, similar to `imgaug.imgaug._ConcavePolygonRecoverer`.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

**Examples**

```python
>>> aug = iaa.PiecewiseAffine(scale=(0.01, 0.05))
```

Puts a grid of points on each image and then randomly moves each point around by 1 to 5 percent (with respect to the image height/width). Pixels between these points will be moved accordingly.

```python
>>> aug = iaa.PiecewiseAffine(scale=(0.01, 0.05), nb_rows=8, nb_cols=8)
```
Same as the previous example, but uses a denser grid of 8x8 points (default is 4x4). This can be useful for large images.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, **kwargs)</td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch([batch[, hooks]])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches([batches[, hooks, background]])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes([bounding_boxes_on_images])</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps([heatmaps[, parents, hooks]])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image([image[, hooks]])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images([images[, parents, hooks]])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints([keypoints_on_images[, ...]])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings([line_strings_on_images])</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons([polygons_on_images[, ...]])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps([segmaps[, ...]])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state([source[, recursive, ...]])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters([func[, parents, flat]])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name([name[, regex, flat]])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names([names[, regex, flat]])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_(recursive)</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters([func[, copy, noop_if_topmost]])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace([func[, parents]])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 83 – continued from previous page

| to_deterministic([n]) | Converts this augmenter from a stochastic to a deterministic one. |

```
get_parameters()
```

```
class imgaug.augmenters.geometric.Rot90(k, keep_size=True, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter to rotate images clockwise by multiples of 90 degrees.

This could also be achieved using Affine, but Rot90 is significantly more efficient.

dtype support:

```python
if (keep_size=False):
    * `uint8`: yes; fully tested
    * `uint16`: yes; tested
    * `uint32`: yes; tested
    * `uint64`: yes; tested
    * `int8`: yes; tested
    * `int16`: yes; tested
    * `int32`: yes; tested
    * `int64`: yes; tested
    * `float16`: yes; tested
    * `float32`: yes; tested
    * `float64`: yes; tested
    * `float128`: yes; tested
    * `bool`: yes; tested

if (keep_size=True):
    minimum of (``imgaug.augmenters.geometric.Rot90(keep_size=False)``, :func:`imgaug.imresize_many_images`

Parameters

- **k** *(int or list or tuple of int or imgaug.ALL or imgaug.parameters.StochasticParameter, optional)*

  How often to rotate clockwise by 90 degrees.

  - If a single int, then that value will be used for all images.
  - If a tuple *(a, b)*, then a random value from the discrete range *a <= x <= b* is picked per image.
  - If a list, then for each image a random value will be sampled from that list.
  - If imgaug.ALL, then equivalent to list *[0, 1, 2, 3]*.
  - If StochasticParameter, then that parameter is queried per image to sample the value to use.
• **keep_size** *(bool, optional)* – After rotation by an odd-valued \( k \) (e.g. 1 or 3), the resulting image may have a different height/width than the original image. If this parameter is set to True, then the rotated image will be resized to the input image’s size. Note that this might also cause the augmented image to look distorted.

• **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

• **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

• **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

### Examples

```python
>>> aug = iaa.Rot90(1)
```

Rotates all images by 90 degrees. Resizes all images afterwards to keep the size that they had before augmentation. This may cause the images to look distorted.

```python
>>> aug = iaa.Rot90([1, 3])
```

Rotates all images by 90 or 270 degrees. Resizes all images afterwards to keep the size that they had before augmentation. This may cause the images to look distorted.

```python
>>> aug = iaa.Rot90((1, 3))
```

Rotates all images by 90, 180 or 270 degrees. Resizes all images afterwards to keep the size that they had before augmentation. This may cause the images to look distorted.

```python
>>> aug = iaa.Rot90((1, 3), keep_size=False)
```

Rotates all images by 90, 180 or 270 degrees. Does not resize to the original image size afterwards, i.e. each image’s size may change.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(*args, **kwargs)</code></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment_bounding_boxes(bounding_boxes_on_images)</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps(heatmaps[, parents, hooks])</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
<tr>
<td><code>augment_images(images[, parents, hooks])</code></td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td><code>augment_keypoints(keypoints_on_images[, ...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
</tbody>
</table>

Continued on next page
### get_parameters() (13.23)

**Augmenters that apply changes to images based on forms of segmentation.**

Do not import directly from this file, as the categorization is not final. Use instead

```python
from imgaug import augmenters as iaa
```

and then e.g.

```python
seq = iaa.Sequential(
    iaa.Superpixels(...)
)
```

List of augmenters:

```python
Table 84 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>augment_segmentation_maps(...)</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(...)</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_inplace(...)</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(...)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(...)</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(...)</code>, <code>find_augmenters_by_names(...)</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>get_all_children(...)</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state(...)</code>, <code>localize_random_state_inplace(...)</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool(...)</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(...)</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(...)</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed(...)</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(...)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic(...)</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>
```
• Superpixels

class imgaug.augmenters.segmentation.Superpixels(p_replace=0, n_segments=100, max_size=128, interpolation='linear', name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Completely or partially transform images to their superpixel representation.

This implementation uses skimage’s version of the SLIC algorithm.

dtype support:

```python
if (image size <= max_size):
    # dtype support
    * \`\`\`uint8\`\`\`: yes; fully tested
    * \`\`\`uint16\`\`\`: yes; tested
    * \`\`\`uint32\`\`\`: yes; tested
    * \`\`\`uint64\`\`\`: limited (1)
    * \`\`\`int8\`\`\`: yes; tested
    * \`\`\`int16\`\`\`: yes; tested
    * \`\`\`int32\`\`\`: yes; tested
    * \`\`\`int64\`\`\`: limited (1)
    * \`\`\`float16\`\`\`: no (2)
    * \`\`\`float32\`\`\`: no (2)
    * \`\`\`float64\`\`\`: no (3)
    * \`\`\`float128\`\`\`: no (2)
    * \`\`\`bool\`\`\`: yes; tested

- (1) Superpixel mean intensity replacement requires computing these means as \`\`\`float64\`\`\`
    This can cause inaccuracies for large integer values.
- (2) Error in scikit-image.
- (3) Loss of resolution in scikit-image.

if (image size > max_size):
    minimum of (
        \`\`\`imgaug.augmenters.segmentation.Superpixels(image size <= max_size)\`\`\`,
        \`\`\`imgaug.imgaug.imresize_many_images\`\`\`
    )
```

Parameters

• **p_replace** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Defines the probability of any superpixel area being replaced by the superpixel, i.e. by the average pixel color within its area:
  
  – A probability of 0 would mean, that no superpixel area is replaced by its average (image is not changed at all).
  
  – A probability of 0.5 would mean, that half of all superpixels are replaced by their average color.
  
  – A probability of 1.0 would mean, that all superpixels are replaced by their average color (resulting in a standard superpixel image).

Behaviour based on chosen datatypes for this parameter:

- If number, then that number will always be used.
- If tuple \((a, b)\), then a random probability will be sampled from the interval \([a, b]\) per image.
- If a list, then a random value will be sampled from that list per image.
- If this parameter is a StochasticParameter, it is expected to return values between 0 and 1. Values \(\geq 0.5\) will be interpreted as the command to replace a superpixel region with its mean. Recommended to be some form of \(\text{Binomial}(...)\).

- **\text{n\_segments}** (\text{int or tuple of int or list of int or imgaug\_parameters\_StochasticParameter, optional}) –

Target number of superpixels to generate. Lower numbers are faster.
- If a single int, then that value will always be used as the number of segments.
- If a tuple \((a, b)\), then a value from the discrete interval \([a..b]\) will be sampled per image.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then that parameter will be queried to draw one value per image.

- **\text{max\_size}** (\text{int or None, optional}) – Maximum image size at which the superpixels are generated. If the width or height of an image exceeds this value, it will be downscaled so that the longest side matches \text{max\_size}. Though, the final output (superpixel) image has the same size as the input image. This is done to speed up the superpixel algorithm. Use None to apply no downsampling.

- **\text{interpolation}** (\text{int or str, optional}) – Interpolation method to use during downsampling when \text{max\_size} is exceeded. Valid methods are the same as in \text{imgaug.imgaug.imresize\_single\_image}().

- **\text{name}** (\text{None or str, optional}) – See \text{imgaug.augmenters.meta.Augmenter.__init__}().

- **\text{deterministic}** (\text{bool, optional}) – See \text{imgaug.augmenters.meta.Augmenter.__init__}().

- **\text{random\_state}** (\text{None or int or numpy\_random\_RandomState, optional}) – See \text{imgaug.augmenters.meta.Augmenter.__init__}().

**Examples**

```python
>>> aug = iaa.Superpixels(p_replace=1.0, n_segments=64)
```

generates ~64 superpixels per image and replaces all of them with their average color (standard superpixel image).

```python
>>> aug = iaa.Superpixels(p_replace=0.5, n_segments=64)
```

generates always ~64 superpixels per image and replaces half of them with their average color, while the other half are left unchanged (i.e. they still show the input image’s content).

```python
>>> aug = iaa.Superpixels(p_replace=(0.25, 1.0), n_segments=(16, 128))
```

generates between ~16 and ~128 superpixels per image and replaces 25 to 100 percent of them with their average color.
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, **kwargs)</td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code>.</td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([in])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>
get_parameters()

13.24 \texttt{imgaug.augmenters.size}

Augmenters that somehow change the size of the images.

Do not import directly from this file, as the categorization is not final. Use instead

\begin{verbatim}
from imgaug import augmenters as iaa
\end{verbatim}

and then e.g.

\begin{verbatim}
seq = iaa.Sequential([
    iaa.Resize({"height": 32, "width": 64})
    iaa.Crop((0, 20))
])
\end{verbatim}

List of augmenters:

- Resize
- CropAndPad
- Crop
- Pad
- PadToFixedSize
- CropToFixedSize
- KeepSizeByResize

\texttt{imgaug.augmenters.size.Crop}(px=None, percent=None, keep_size=True, sample_independently=True, name=None, deterministic=False, random_state=None)

Augmenter that crops/cuts away pixels at the sides of the image.

That allows to cut out subimages from given (full) input images. The number of pixels to cut off may be defined in absolute values or percent of the image sizes.

dtype support:

See ```imgaug.augmenters.size.CropAndPad```.

**Parameters**

- \textbf{px} (None or int or \texttt{imgaug.parameters.StochasticParameter or tuple, optional}) – The number of pixels to crop away (cut off) on each side of the image. Either this or the parameter \texttt{percent} may be set, not both at the same time.
  - If None, then pixel-based cropping will not be used.
  - If int, then that exact number of pixels will always be cropped.
  - If \texttt{StochasticParameter}, then that parameter will be used for each image. Four samples will be drawn per image (top, right, bottom, left).
  - If a tuple of two ints with values \texttt{a} and \texttt{b}, then each side will be cropped by a random amount in the range \texttt{a} \leq x \leq \texttt{b}. \texttt{x} is sampled per image side.
If a tuple of four entries, then the entries represent top, right, bottom, left. Each entry may
be a single integer (always crop by exactly that value), a tuple of two ints a and b (crop
by an amount a <= x <= b), a list of ints (crop by a random value that is contained in
the list) or a StochasticParameter (sample the amount to crop from that parameter).

* percent (None or int or float or imgaug.parameters.StochasticParameter or tuple, optional)
  
The number of pixels to crop away (cut off) on each side of the image given *in percent* of
the image height/width. E.g. if this is set to 0.1, the augmenter will always crop away 10
percent of the image’s height at the top, 10 percent of the width on the right, 10 percent of
the height at the bottom and 10 percent of the width on the left. Either this or the parameter
px may be set, not both at the same time.

  - If None, then percent-based cropping will not be used.
  - If int, then expected to be 0 (no cropping).
  - If float, then that percentage will always be cropped away.
  - If StochasticParameter, then that parameter will be used for each image. Four samples
    will be drawn per image (top, right, bottom, left).
  - If a tuple of two floats with values a and b, then each side will be cropped by a random
    percentage in the range a <= x <= b. x is sampled per image side.
  - If a tuple of four entries, then the entries represent top, right, bottom, left. Each entry may
    be a single float (always crop by exactly that percent value), a tuple of two floats a and
    b (crop by a percentage a <= x <= b), a list of floats (crop by a random value that is
    contained in the list) or a StochasticParameter (sample the percentage to crop from that
    parameter).

* keep_size (bool, optional) – After cropping, the result image has a different height/width
  than the input image. If this parameter is set to True, then the cropped image will be resized
to the input image’s size, i.e. the image size is then not changed by the augmenter.

* sample_independently (bool, optional) – If False AND the values for px/percent result in
  exactly one probability distribution for the amount to crop, only one single value will be
  sampled from that probability distribution and used for all sides. I.e. the crop amount then
  is the same for all sides.

* name (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

* deterministic (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

* random_state (None or int or numpy.random.RandomState, optional) – See imgaug.
augmenters.meta.Augmenter.__init__().

**Examples**

```python
>>> aug = iaa.Crop(px=(0, 10))
crops each side by a random value from the range 0px to 10px (the value is sampled per side).

>>> aug = iaa.Crop(px=(0, 10), sample_independently=False)
samples one value v from the discrete range [0..10] and crops all sides by v pixels.
```
>>> aug = iaa.Crop(px={0, 10}, keep_size=False)

crops each side by a random value from the range 0px to 10px (the value is sampled per side). After cropping, the images are NOT resized to their original size (i.e. the images may end up having different heights/widths).

>>> aug = iaa.Crop(px=((0, 10), (0, 5), (0, 10), (0, 5)))

crops the top and bottom by a random value from the range 0px to 10px and the left and right by a random value in the range 0px to 5px.

>>> aug = iaa.Crop(percent=(0, 0.1))

crops each side by a random value from the range 0 percent to 10 percent. (Percent with respect to the side’s size, e.g. for the top side it uses the image’s height.)

>>> aug = iaa.Crop(percent=((0.05, 0.1), [0.05, 0.1], [0.05, 0.1], [0.05, 0.1]))

crops each side by either 5 percent or 10 percent.

class imgaug.augmenters.size.CropAndPad(px=None, percent=None, pad_mode='constant', pad_cval=0, keep_size=True, sample_independently=True, name=None, deterministic=False, random_state=None)

Bases: imgaug.augmenters.meta.Augmenter

Augmenter that crops/pads images by defined amounts in pixels or percent (relative to input image size). Cropping removes pixels at the sides (i.e. extracts a subimage from a given full image). Padding adds pixels to the sides (e.g. black pixels).

dtype support:

if (keep_size=False):
    * `'uint8``: yes; fully tested
    * `'uint16`": yes; tested
    * `'uint32`": yes; tested
    * `'uint64`": yes; tested
    * `'int8`": yes; tested
    * `'int16`": yes; tested
    * `'int32`": yes; tested
    * `'int64`": yes; tested
    * `'float16`": yes; tested
    * `'float32`": yes; tested
    * `'float64`": yes; tested
    * `'float128`": yes; tested
    * `'bool`": yes; tested

if (keep_size=True):
    minimum of (``imgaug.augmenters.size.CropAndPad(keep_size=False)``, :func:`imgaug.imgaug.imresize_many_images`)

Parameters

- `px (None or int or imgaug.parameters.StochasticParameter or tuple, optional)` – The number of pixels to crop (negative values) or pad (positive values) on each side of the image.
Either this or the parameter `percent` may be set, not both at the same time.

- If None, then pixel-based cropping/padding will not be used.
- If int, then that exact number of pixels will always be cropped/padded.
- If StochasticParameter, then that parameter will be used for each image. Four samples will be drawn per image (top, right, bottom, left), unless `sample_independently` is set to False, as then only one value will be sampled per image and used for all sides.
- If a tuple of two ints with values \(a\) and \(b\), then each side will be cropped/padded by a random amount in the range \(a \leq x \leq b\). \(x\) is sampled per image side. If however `sample_independently` is set to False, only one value will be sampled per image and used for all sides.
- If a tuple of four entries, then the entries represent top, right, bottom, left. Each entry may be a single integer (always crop/pad by exactly that value), a tuple of two ints \(a\) and \(b\) (crop/pad by an amount \(a \leq x \leq b\)), a list of ints (crop/pad by a random value that is contained in the list) or a StochasticParameter (sample the amount to crop/pad from that parameter).

**percent** *(None or int or float or imgaug.parameters.StochasticParameter or tuple, optional)*

- The number of pixels to crop (negative values) or pad (positive values) on each side of the image given in percent of the image height/width. E.g. if this is set to 0.1, the augmenter will always crop away 10 percent of the image’s height at the top, 10 percent of the width on the right, 10 percent of the height at the bottom and 10 percent of the width on the left. Either this or the parameter `px` may be set, not both at the same time.
- If None, then percent-based cropping/padding will not be used.
- If int, then expected to be 0 (no cropping/padding).
- If float, then that percentage will always be cropped/padded.
- If StochasticParameter, then that parameter will be used for each image. Four samples will be drawn per image (top, right, bottom, left). If however `sample_independently` is set to False, only one value will be sampled per image and used for all sides.
- If a tuple of two floats with values \(a\) and \(b\), then each side will be cropped/padded by a random percentage in the range \(a \leq x \leq b\). \(x\) is sampled per image side. If however `sample_independently` is set to False, only one value will be sampled per image and used for all sides.
- If a tuple of four entries, then the entries represent top, right, bottom, left. Each entry may be a single float (always crop/pad by exactly that percent value), a tuple of two floats \(a\) and \(b\) (crop/pad by a percentage \(a \leq x \leq b\)), a list of floats (crop by a random value that is contained in the list) or a StochasticParameter (sample the percentage to crop/pad from that parameter).

**pad_mode** *(imgaug.ALL or str or list of str or imgaug.parameters.StochasticParameter, optional)* – Padding mode to use. The available modes match the numpy padding modes, i.e. constant, edge, linear_ramp, maximum, median, minimum, reflect, symmetric, wrap. The modes constant and linear_ramp use extra values, which are provided by `pad_cval` when necessary. See `imgaug.imgaug.pad()` for more details.

- If `imgaug.ALL`, then a random mode from all available modes will be sampled per image.
- If a string, it will be used as the pad mode for all images.
- If a list of strings, a random one of these will be sampled per image and used as the mode.
- If StochasticParameter, a random mode will be sampled from this parameter per image.

- **pad_cval** *(number or tuple of number list of number or imgaug.parameters.StochasticParameter, optional)* – The constant value to use if the pad mode is constant or the end value to use if the mode is linear_ramp. See `imgaug.imgaug.pad()` for more details.
  - If number, then that value will be used.
  - If a tuple of two numbers and at least one of them is a float, then a random number will be sampled from the continuous range \(a \leq x \leq b\) and used as the value. If both numbers are integers, the range is discrete.
  - If a list of number, then a random value will be chosen from the elements of the list and used as the value.
  - If StochasticParameter, a random value will be sampled from that parameter per image.

- **keep_size** *(bool, optional)* – After cropping and padding, the result image will usually have a different height/width compared to the original input image. If this parameter is set to True, then the cropped/padded image will be resized to the input image’s size, i.e. the augmenter’s output shape is always identical to the input shape.

- **sample_independently** *(bool, optional)* – If False AND the values for px/percent result in exactly one probability distribution for the amount to crop/pad, only one single value will be sampled from that probability distribution and used for all sides. I.e. the crop/pad amount then is the same for all sides.

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.

### Examples

```python
>>> aug = iaa.CropAndPad(px=(-10, 0))
crops each side by a random value from the range -10px to 0px (the value is sampled per side).

>>> aug = iaa.CropAndPad(px=(0, 10))
pads each side by a random value from the range 0px to 10px (the values are sampled per side). The padding happens by zero-padding (i.e. adds black pixels).

>>> aug = iaa.CropAndPad(px=(0, 10), pad_mode="edge")
pads each side by a random value from the range 0px to 10px (the values are sampled per side). The padding uses the edge mode from numpy’s pad function.

>>> aug = iaa.CropAndPad(px=(0, 10), pad_mode=["constant", "edge"])
pads each side by a random value from the range 0px to 10px (the values are sampled per side). The padding uses randomly either the constant or edge mode from numpy’s pad function.
```
pads each side by a random value from the range 0px to 10px (the values are sampled per side). It uses a random mode for numpy’s pad function. If the mode is constant or linear_ramp, it samples a random value \( v \) from the range \([0, 255]\) and uses that as the constant value (mode=constant) or end value (mode=linear_ramp).

```python
>>> aug = iaa.CropAndPad(px=(0, 10), sample_independently=False)
```
samples one value \( v \) from the discrete range \([0..10]\) and pads all sides by \( v \) pixels.

```python
>>> aug = iaa.CropAndPad(px=(0, 10), keep_size=False)
```
pads each side by a random value from the range 0px to 10px (the value is sampled per side). After padding, the images are NOT resized to their original size (i.e. the images may end up having different heights/widths).

```python
>>> aug = iaa.CropAndPad(px=((0, 10), (0, 5), (0, 10), (0, 5)))
```
pads the top and bottom by a random value from the range 0px to 10px and the left and right by a random value in the range 0px to 5px.

```python
>>> aug = iaa.CropAndPad(percent=(0, 0.1))
```
pads each side by a random value from the range 0 percent to 10 percent. (Percent with respect to the side’s size, e.g. for the top side it uses the image’s height.)

```python
>>> aug = iaa.CropAndPad(percent=((0.05, 0.1), [0.05, 0.1], [0.05, 0.1], [0.05, 0.1]))
```
pads each side by either 5 percent or 10 percent.

```python
>>> aug = iaa.CropAndPad(px=(-10, 10))
```
samples per side and image a value \( v \) from the discrete range \([-10..10]\) and either crops (negative value) or pads (positive value) the side by \( v \) pixels.

**Methods**

- `__call__(*args, **kwargs)`
- `augment([return_batch, hooks])`
  - Augment data.
- `augment_batch(batch[, hooks])`
  - Augment a single batch.
- `augment_batches(batches[, hooks, background])`
  - Augment multiple batches.
- `augment_bounding_boxes(bounding_boxes_on_images)`
  - Augment bounding boxes.
- `augment_heatmaps(heatmaps[, parents, hooks])`
  - Augment a heatmap.
- `augment_image(image[, hooks])`
  - Augment a single image.
- `augment_images(images[, parents, hooks])`
  - Augment multiple images.
- `augment_keypoints(keypoints_on_images[, ...])`
  - Augment image keypoints.
- `augment_line_strings(line_strings_on_images)`
  - Augment line strings.
- `augment_polygons(polygons_on_images[, ...])`
  - Augment polygons.

Continued on next page

412 Chapter 13. API
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>augment_segmentation_maps</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(sequence, recursive, ...)</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_inplace</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func, parents, flat)</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name, regex, flat)</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names, regex, flat)</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children(flat)</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state(recursive)</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_inplace(recursive)</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool(processes, maxtasksperchild, seed)</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func, copy, parents)</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func, parents)</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed(random_state, deterministic_too)</td>
<td>Rseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic()</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

`get_parameters()`

```python
class imgaug.augmenters.size.CropToFixedSize(width, height, position='uniform', name=None, deterministic=False, random_state=None)
```

Augmenter that crops down to a fixed maximum width/height.

If images are already at the maximum width/height or are smaller, they will not be cropped. Note: This also means that images will not be padded if they are below the required width/height.

The augmenter randomly decides per image how to distribute the required cropping amounts over the image axis. E.g. if 2px have to be cropped on the left or right to reach the required width, the augmenter will sometimes remove 2px from the left and 0px from the right, sometimes remove 2px from the right and 0px from the left and sometimes remove 1px from both sides. Set `position` to `center` to prevent that.

**dtype support:**

```python
get_parameters()
```
``uint8``: yes; fully tested
``uint16``: yes; tested
``uint32``: yes; tested
``uint64``: yes; tested
``int8``: yes; tested
``int16``: yes; tested
``int32``: yes; tested
``int64``: yes; tested
``float16``: yes; tested
``float32``: yes; tested
``float64``: yes; tested
``float128``: yes; tested
``bool``: yes; tested

Parameters

- **width** *(int)* – Fixed width of new images.
- **height** *(int)* – Fixed height of new images.
- **position** *({'uniform', 'normal', 'center', 'left-top', 'left-center', 'left-bottom', 'center-top', 'center-center', 'center-bottom', 'right-top', 'right-center', 'right-bottom'} or tuple of float or StochasticParameter or tuple of StochasticParameter, optional)* – Sets the center point of the cropping, which determines how the required cropping amounts are distributed to each side. For a tuple `(a, b)`, both `a` and `b` are expected to be in range `[0.0, 1.0]` and describe the fraction of cropping applied to the left/right (low/high values for `a`) and the fraction of cropping applied to the top/bottom (low/high values for `b`). A cropping position at `(0.5, 0.5)` would be the center of the image and distribute the cropping equally over all sides. A cropping position at `(1.0, 0.0)` would be the right-top and would apply 100% of the required cropping to the right and top sides of the image.
  - If string `uniform` then the share of cropping is randomly and uniformly distributed over each side. Equivalent to `(Uniform(0.0, 1.0), Uniform(0.0, 1.0))`.
  - If string `normal` then the share of cropping is distributed based on a normal distribution, leading to a focus on the center of the images. Equivalent to `(Clip(Normal(0.5, 0.45/2), 0, 1), Clip(Normal(0.5, 0.45/2), 0, 1))`.
  - If string `center` then center point of the cropping is identical to the image center. Equivalent to `(0.5, 0.5)`.
  - If a string matching regex `^(left|center|right)-(top|center|bottom)$`, e.g. `left-top` or `center-bottom` then sets the center point of the cropping to the X-Y position matching that description.
  - If a tuple of float, then expected to have exactly two entries between 0.0 and 1.0, which will always be used as the combination the position matching (x, y) form.
  - If a StochasticParameter, then that parameter will be queries once per call to `augment_*()` to get `Nx2` center positions matching (x, y) form.
  - If a tuple of StochasticParameter, then expected to have exactly two entries that will both be queries per call to `augment_*()`, each for (`N`, ) values, to get the center positions. First parameter is used for x, second for y.
- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter._init__()`.
• **random_state** *(None or int or numpy.random.RandomState, optional) – See `imgaug.augmenters.meta.Augmenter.__init__()*.}

**Examples**

```python
>>> aug = iaa.CropToFixedSize(width=100, height=100)
```

For sides larger than 100 pixels, crops to 100 pixels. Does nothing for the other sides. The cropping amounts are randomly (and uniformly) distributed over the sides of the image.

```python
>>> aug = iaa.CropToFixedSize(width=100, height=100, position="center")
```

For sides larger than 100 pixels, crops to 100 pixels. Does nothing for the other sides. The cropping amounts are always equally distributed over the left/right sides of the image (and analogously for top/bottom).

```python
>>> aug = iaa.Sequential([  
>>>    iaa.PadToFixedSize(width=100, height=100),  
>>>    iaa.CropToFixedSize(width=100, height=100)
>>> ])
```

pads to 100x100 pixel for smaller images, and crops to 100x100 pixel for larger images. The output images have fixed size, 100x100 pixel.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(args, **kwargs)</code></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code>.</td>
</tr>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment_bounding_boxes(bounding_boxes_on_images)</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps(heatmaps[, parents, hooks])</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
<tr>
<td><code>augment_images(images[, parents, hooks])</code></td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td><code>augment_keypoints(keypoints_on_images[, ...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
<tr>
<td><code>augment_segmentation_maps(segs[, ...])</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_inplace(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
</tbody>
</table>
Table 87 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>find_augmenters_by_name(name[, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([n])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_parameters()**

```python
get_parameters()
```

```python
class imgaug.augmenters.size.KeepSizeByResize(children, interpolation='cubic', interpolation_heatmaps='SAME_AS_IMAGES', name=None, deterministic=False, random_state=None)
```

**Bases:** `imgaug.augmenters.meta.Augmenter`

Augmenter that resizes images before/after augmentation so that they retain their original height and width. This can e.g. be placed after a cropping operation. Some augmenters have a `keep_size` parameter that does mostly the same if set to True, though this augmenter offers control over the interpolation mode.

**dtype support:**

```python
See :func:`imgaug.imgaug.imresize_many_images`.
```

**Parameters**

- **children** (Augmenter or list of imgaug.augmenters.meta.Augmenter or None, optional) – One or more augmenters to apply to images. These augmenters may change the image size.

- **interpolation** (KeepSizeByResize.NO_RESIZE or {'nearest', 'linear', 'area', 'cubic'}) or [cv2.INTER_NEAREST, cv2.INTER_LINEAR, cv2.INTER_AREA, cv2.INTER_CUBIC] or list of str or list of int or StochasticParameter, optional) – The interpolation mode to use when resizing images. Can take any value that `imgaug.imresize.imresize_single_image()` accepts, e.g. cubic.
  
  - If this is KeepSizeByResize.NO_RESIZE then images will not be resized.
  
  - If this is a single string, it is expected to have one of the following values: nearest, linear, area, cubic.
- If this is a single integer, it is expected to have a value identical to one of:
cv2.INTER_NEAREST, cv2.INTER_LINEAR, cv2.INTER_AREA, cv2.
INTER_CUBIC.

- If this is a list of strings or ints, it is expected that each string/int is one of the above
mentioned valid ones. A random one of these values will be sampled per image.

- If this is a StochasticParameter, it will be queried once per call to
__augment_images() and must return N strings or ints (matching the above
mentioned ones) for N images.

• interpolation_heatmaps (KeepSizeByResize.SAME_AS_IMAGES or KeepSizeByRe-
size.NO_RESIZE or {'nearest', 'linear', 'area', 'cubic'} or {cv2.INTER_NEAREST,
cv2.INTER_LINEAR, cv2.INTER_AREA, cv2.INTER_CUBIC} or list of str or list of int or
StochasticParameter, optional) – The interpolation mode to use when resizing heatmaps.
Meaning and valid values are similar to interpolation. This parameter may also take
the value KeepSizeByResize.SAME_AS_IMAGES, which will lead to copying the
interpolation modes used for the corresponding images. The value may also be returned on
a per-image basis if interpolation_heatmaps is provided as a StochasticParameter or may
be one possible value if it is provided as a list of strings.

• name (None or str, optional) – See imgaug.augmenters.meta.Augmenter.
__init__().

• deterministic (bool, optional) – See imgaug.augmenters.meta.Augmenter.
__init__().

• random_state (None or int or numpy.random.RandomState, optional) – See imgaug.
augmenters.meta.Augmenter.__init__().

Methods

__call__(*args, **kwargs) Alias for imgaug.augmenters.meta.Augmenter.augment().

augment([return_batch, hooks]) Augment data.

augment_batch(batch[, hooks]) Augment a single batch.

augment_batches(batches[, hooks, background]) Augment multiple batches.

augment_bounding_boxes(bounding_boxes_on_images) Augment bounding boxes.

augment_heatmaps(heatmaps[, parents, hooks]) Augment a heatmap.

augment_image(image[, hooks]) Augment a single image.

augment_images(images[, parents, hooks]) Augment multiple images.

augment_keypoints(keypoints_on_images[, ...]) Augment image keypoints.

augment_line_strings(line_strings_on_images) Augment line strings.

augment_polygons(polygons_on_images[, ...]) Augment polygons.

augment_segmentation_maps(segmaps[, ...]) Augment segmentation maps.

copy() Create a shallow copy of this Augmenter instance.

copy_random_state(source[, recursive, ...]) Copy the random states from a source augmenter se-
quence.

copy_random_state_(source[, recursive, ...]) Copy the random states from a source augmenter se-
quence (inplace).

Continued on next page
Table 88 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state()([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

---

**get_parameters()**

**get_parameters()**

**get_parameters()**

**get_parameters()**

**get_parameters()**

**get_parameters()**

**get_parameters()**

**get_parameters()**

**get_parameters()**
Augmenter that pads images, i.e. adds columns/rows to them.

dtype support:

See `imgaug.augmenters.size.CropAndPad`.

Parameters

- **px** *(None or int or imgaug.parameters.StochasticParameter or tuple, optional)* – The number of pixels to pad on each side of the image. Either this or the parameter *percent* may be set, not both at the same time.
  - If None, then pixel-based padding will not be used.
  - If int, then that exact number of pixels will always be padded.
  - If StochasticParameter, then that parameter will be used for each image. Four samples will be drawn per image (top, right, bottom, left).
  - If a tuple of two ints with values a and b, then each side will be padded by a random amount in the range $a \leq x \leq b$. $x$ is sampled per image side.
  - If a tuple of four entries, then the entries represent top, right, bottom, left. Each entry may be a single integer (always pad by exactly that value), a tuple of two ints a and b (pad by an amount $a \leq x \leq b$), a list of ints (pad by a random value that is contained in the list) or a StochasticParameter (sample the amount to pad from that parameter).

- **percent** *(None or int or float or imgaug.parameters.StochasticParameter or tuple, optional)* – The number of pixels to pad on each side of the image given in percent of the image height/width. E.g. if this is set to 0.1, the augmenter will always add 10 percent of the image’s height to the top, 10 percent of the width to the right, 10 percent of the height at the bottom and 10 percent of the width to the left. Either this or the parameter *px* may be set, not both at the same time.
  - If None, then percent-based padding will not be used.
  - If int, then expected to be 0 (no padding).
  - If float, then that percentage will always be padded.
  - If StochasticParameter, then that parameter will be used for each image. Four samples will be drawn per image (top, right, bottom, left).
  - If a tuple of two floats with values a and b, then each side will be padded by a random percentage in the range $a \leq x \leq b$. $x$ is sampled per image side.
  - If a tuple of four entries, then the entries represent top, right, bottom, left. Each entry may be a single float (always pad by exactly that percent value), a tuple of two floats a and b (pad by a percentage $a \leq x \leq b$), a list of floats (pad by a random value that is contained in the list) or a StochasticParameter (sample the percentage to pad from that parameter).

- **pad_mode** *(imgaug.ALL or str or list of str or imgaug.parameters.StochasticParameter, optional)* – Padding mode to use. The available modes match the numpy padding modes, i.e. constant, edge, linear_ramp, maximum, median, minimum, reflect, symmetric, wrap. The modes constant and linear_ramp use extra values, which are provided by *pad_cval* when necessary. See *imgaug.imgaug.pad()* for more details.
- If `imgaug.ALL`, then a random mode from all available modes will be sampled per image.
- If a string, it will be used as the pad mode for all images.
- If a list of strings, a random one of these will be sampled per image and used as the mode.
- If a StochasticParameter, a random mode will be sampled from this parameter per image.

- `pad_cval` *(number or tuple of number, list of number or `imgaug.parameters.StochasticParameter`, optional)* – The constant value to use if the pad mode is constant or the end value to use if the mode is `linear_ramp`. See `imgaug.imgaug.pad()` for more details.
  - If a number, then that value will be used.
  - If a tuple of two numbers and at least one of them is a float, then a random number will be sampled from the continuous range \(a \leq x \leq b\) and used as the value. If both numbers are integers, the range is discrete.
  - If a list of numbers, then a random value will be chosen from the elements of the list and used as the value.
  - If a StochasticParameter, a random value will be sampled from that parameter per image.

- `keep_size` *(bool, optional)* – After padding, the result image will usually have a different height/width compared to the original input image. If this parameter is set to True, then the padded image will be resized to the input image’s size, i.e. the augmenter’s output shape is always identical to the input shape.

- `sample_independently` *(bool, optional)* – If False AND the values for `px/percent` result in exactly one probability distribution for the amount to pad, only one single value will be sampled from that probability distribution and used for all sides. I.e. the pad amount then is the same for all sides.

- `name` *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`.  


- `random_state` *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()`. 

### Examples

```python
>>> aug = iaa.Pad(px=(0, 10))
```

pads each side by a random value from the range 0px to 10px (the value is sampled per side). The added rows/columns are filled with black pixels.

```python
>>> aug = iaa.Pad(px=(0, 10), sample_independently=False)
```

samples one value \(v\) from the discrete range \([0..10]\) and pads all sides by \(v\) pixels.

```python
>>> aug = iaa.Pad(px=(0, 10), keep_size=False)
```

pads each side by a random value from the range 0px to 10px (the value is sampled per side). After padding, the images are NOT resized to their original size (i.e. the images may end up having different heights/widths).
```python
>>> aug = iaa.Pad(px=((0, 10), (0, 5), (0, 10), (0, 5)))
```
pads the top and bottom by a random value from the range 0px to 10px and the left and right by a random value in the range 0px to 5px.

```python
>>> aug = iaa.Pad(percent=(0, 0.1))
```
pads each side by a random value from the range 0 percent to 10 percent. (Percent with respect to the side’s size, e.g. for the top side it uses the image’s height.)

```python
>>> aug = iaa.Pad(percent=((0.05, 0.1), (0.05, 0.1), (0.05, 0.1), (0.05, 0.1)))
```
pads each side by either 5 percent or 10 percent.

```python
>>> aug = iaa.Pad(px=(0, 10), pad_mode="edge")
```
pads each side by a random value from the range 0px to 10px (the values are sampled per side). The padding uses the edge mode from numpy’s pad function.

```python
>>> aug = iaa.Pad(px=(0, 10), pad_mode=["constant", "edge"])
```
pads each side by a random value from the range 0px to 10px (the values are sampled per side). The padding uses randomly either the constant or edge mode from numpy’s pad function.

```python
>>> aug = iaa.Pad(px=(0, 10), pad_mode=ia.ALL, pad_cval=(0, 255))
```
pads each side by a random value from the range 0px to 10px (the values are sampled per side). It uses a random mode for numpy’s pad function. If the mode is constant or linear_ramp, it samples a random value \(v\) from the range \([0, 255]\) and uses that as the constant value (mode=constant) or end value (mode=linear_ramp).

```python
class imgaug.augmenters.size.PadToFixedSize(width, height, pad_mode='constant',
pad_cval=0, position='uniform',
name=None, deterministic=False, random_state=None)
```
Bases: `imgaug.augmenters.meta.Augmenter`

Pad images to minimum width/height.

If images are already at the minimum width/height or are larger, they will not be padded. Note: This also means that images will not be cropped if they exceed the required width/height.

The augmenter randomly decides per image how to distribute the required padding amounts over the image axis. E.g. if 2px have to be padded on the left or right to reach the required width, the augmenter will sometimes add 2px to the left and 0px to the right, sometimes add 2px to the right and 0px to the left and sometimes add 1px to both sides. Set `position` to `center` to prevent that.

dtype support:

See :func:`imgaug.imgaug.pad`.

Parameters

- **width** (*int*) – Minimum width of new images.
- **height** (*int*) – Minimum height of new images.
- **pad_mode** (*imgaug.ALL or str or list of str or imgaug.parameters.StochasticParameter, optional*) – See `imgaug.augmenters.size.CropAndPad.__init__()`.  

13.24. `imgaug.augmenters.size` 421
• **pad_cval** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional) – See imgaug.augmenters.size.CropAndPad.__init__().

• **position** ('uniform', 'normal', 'center', 'left-top', 'left-center', 'left-bottom', 'center-top', 'center-center', 'center-bottom', 'right-top', 'right-center', 'right-bottom') or tuple of float or StochasticParameter or tuple of StochasticParameter, optional – Sets the center point of the padding, which determines how the required padding amounts are distributed to each side. For a tuple \((a, b)\), both \(a\) and \(b\) are expected to be in range \([0.0, 1.0]\) and describe the fraction of padding applied to the left/right (low/high values for \(a\)) and the fraction of padding applied to the top/bottom (low/high values for \(b\)). A padding position at \((0.5, 0.5)\) would be the center of the image and distribute the padding equally to all sides. A padding position at \((0.0, 1.0)\) would be the left-bottom and would apply 100% of the required padding to the bottom and left sides of the image so that the bottom left corner becomes more and more the new image center (depending on how much is padded).

  – If string `uniform` then the share of padding is randomly and uniformly distributed over each side. Equivalent to \((\text{Uniform}(0.0, 1.0), \text{Uniform}(0.0, 1.0))\).

  – If string `normal` then the share of padding is distributed based on a normal distribution, leading to a focus on the center of the images. Equivalent to \((\text{Clip}(\text{Normal}(0.5, 0.45/2), 0, 1), \text{Clip}(\text{Normal}(0.5, 0.45/2), 0, 1))\).

  – If string `center` then center point of the padding is identical to the image center. Equivalent to \((0.5, 0.5)\).

  – If a string matching regex `^(left|center|right)-(top|center|bottom)$`, e.g. `left-top` or `center-bottom` then sets the center point of the padding to the X-Y position matching that description.

  – If a tuple of float, then expected to have exactly two entries between 0.0 and 1.0, which will always be used as the combination the position matching \((x, y)\) form.

  – If a StochasticParameter, then that parameter will be queries once per call to augment_*() to get \(N\times2\) center positions matching \((x, y)\) form.

  – If a tuple of StochasticParameter, then expected to have exactly two entries that will both be queries per call to augment_*(), each for \((N,)\) values, to get the center positions. First parameter is used for \(x\), second for \(y\).

• **name** (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

• **deterministic** (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

• **random_state** (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> aug = iaa.PadToFixedSize(width=100, height=100)
```

For edges smaller than 100 pixels, pads to 100 pixels. Does nothing for the other edges. The padding is randomly (uniformly) distributed over the sides, so that e.g. sometimes most of the required padding is applied to the left, sometimes to the right (analogous top/bottom).

```python
>>> aug = iaa.PadToFixedSize(width=100, height=100, position="center")
```
For edges smaller than 100 pixels, pads to 100 pixels. Does nothing for the other edges. The padding is always equally distributed over the left/right and top/bottom sides.

```python
>>> aug = iaa.Sequential([
>>>     iaa.PadToFixedSize(width=100, height=100),
>>>     iaa.CropToFixedSize(width=100, height=100)
>>> ])
```

Pads to 100x100 pixel for smaller images, and crops to 100x100 pixel for larger images. The output images have fixed size, 100x100 pixel.

**Methods**

- `__call__(*args, **kwargs)` Alias for `imgaug.augmenters.meta.Augmenter.augment()`.
- `augment([return_batch, hooks])` Augment data.
- `augment_batch(batch[, hooks])` Augment a single batch.
- `augment_batches(batches[, hooks, background])` Augment multiple batches.
- `augment_bounding_boxes(bounding_boxes_on_images)` Augment bounding boxes.
- `augment_heatmaps(heatmaps[, parents, hooks])` Augment a heatmap.
- `augment_image(image[, hooks])` Augment a single image.
- `augment_images(images[, parents, hooks])` Augment multiple images.
- `augment_keypoints(keypoints_on_images[, ...])` Augment image keypoints.
- `augment_line_strings(line_strings_on_images)` Augment line strings.
- `augment_polygons(polygons_on_images[, ...])` Augment polygons.
- `augment_segmentation_maps(segmaps[, ...])` Augment segmentation maps.
- `copy()` Create a shallow copy of this Augmenter instance.
- `copy_random_state(source[, recursive, ...])` Copy the random states from a source augmenter sequence.
- `copy_random_state_(source[, recursive, ...])` Copy the random states from a source augmenter sequence (inplace).
- `deepcopy()` Create a deep copy of this Augmenter instance.
- `draw_grid(images, rows, cols)` Apply this augmenter to the given images and return a grid image of the results.
- `find_augmenters(func[, parents, flat])` Find augmenters that match a condition.
- `find_augmenters_by_name(name[, regex, flat])` Find augmenter(s) by name.
- `find_augmenters_by_names(names[, regex, flat])` Find augmenter(s) by names.
- `get_all_children([flat])` Returns all children of this augmenter as a list.
- `get_children_lists()` Get a list of lists of children of this augmenter.
- `localize_random_state([recursive])` Converts global random states to local ones.
- `localize_random_state_([recursive])` Converts global random states to local ones.
- `pool([processes, maxtasksperchild, seed])` Create a pool used for multicore augmentation from this augmenter.
- `remove_augmenters(func[, copy, noop_if_topmost])` Remove this augmenter or its children that match a condition.

Continued on next page
Table 89 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>remove_augmenters_inplace(func, parents)</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([n])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**get_parameters()**

```python
class imgaug.augmenters.size.Resize(size, interpolation='cubic', name=None, deterministic=False, random_state=None)
```

Augmenter that resizes images to specified heights and widths.

dtype support:

See :func:`imgaug.imgaug.imresize_many_images`.

**Parameters**

- `size` (‘keep’ or int or float or tuple of int or tuple of float or list of int or list of float or imgaug.parameters.StochasticParameter or dict) –
  The new size of the images.
  - If this has the string value “keep”, the original height and width values will be kept (image is not resized).
  - If this is an integer, this value will always be used as the new height and width of the images.
  - If this is a float v, then per image the image’s height H and width W will be changed to \( H \times v \) and \( W \times v \).
  - If this is a tuple, it is expected to have two entries \((a, b)\). If at least one of these are floats, a value will be sampled from range \([a, b]\) and used as the float value to resize the image (see above). If both are integers, a value will be sampled from the discrete range \([a..b]\) and used as the integer value to resize the image (see above).
  - If this is a list, a random value from the list will be picked to resize the image. All values in the list must be integers or floats (no mixture is possible).
  - If this is a StochasticParameter, then this parameter will first be queried once per image. The resulting value will be used for both height and width.
  - If this is a dictionary, it may contain the keys “height” and “width”. Each key may have the same datatypes as above and describes the scaling on x and y-axis. Both axis are sampled independently. Additionally, one of the keys may have the value “keep-aspect-ratio”, which means that the respective side of the image will be resized so that the original aspect ratio is kept. This is useful when only resizing one image size by a pixel value (e.g. resize images to a height of 64 pixels and resize the width so that the overall aspect ratio is maintained).
• **interpolation** *(imgaug.ALL or int or str or list of int or list of str or imgaug.parameters.StochasticParameter, optional)* –

Interpolation to use.

- If imgaug.ALL, then a random interpolation from nearest, linear, area or cubic will be picked (per image).
- If int, then this interpolation will always be used. Expected to be any of the following: cv2.INTER_NEAREST, cv2.INTER_LINEAR, cv2.INTER_AREA, cv2.INTER_CUBIC
- If string, then this interpolation will always be used. Expected to be any of the following: nearest, linear, area, cubic
- If list of ints/strings, then a random one of the values will be picked per image as the interpolation. If a StochasticParameter, then this parameter will be queried per image and is expected to return an integer or string.

• **name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

• **deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

• **random_state** *(None or int or numpy.random.RandomState, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

### Examples

```python
>>> aug = iaa.Resize(32)
```
resizes all images to 32x32 pixels.

```python
>>> aug = iaa.Resize(0.5)
```
resizes all images to 50 percent of their original size.

```python
>>> aug = iaa.Resize((16, 22))
```
resizes all images to a random height and width within the discrete range 16<=x<=22.

```python
>>> aug = iaa.Resize((0.5, 0.75))
```
resizes all image’s height and width to H*v and W*v, where v is randomly sampled from the range 0.5<=x<=0.75.

```python
>>> aug = iaa.Resize([16, 32, 64])
```
resizes all images either to 16x16, 32x32 or 64x64 pixels.

```python
>>> aug = iaa.Resize("height": 32))
```
resizes all images to a height of 32 pixels and keeps the original width.

```python
>>> aug = iaa.Resize({"height": 32, "width": 48})
```
resizes all images to a height of 32 pixels and a width of 48.
resizes all images to a height of 32 pixels and resizes the x-axis (width) so that the aspect ratio is maintained.

resizes all images to a height of $H \times v$, where $H$ is the original height and $v$ is a random value sampled from the range $0.5 \leq v \leq 0.75$. The width/x-axis of each image is resized to either 16 or 32 or 64 pixels.

resizes all images to 32x32 pixels. Randomly uses either linear or cubic interpolation.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, **kwargs)</td>
<td>Alias for imgaug.augmenters.meta.Augmenter.augment().</td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_batches(batches[, hooks, background])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_images)</td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
<tr>
<td>augment_polygons(polygons_on_images[, ...])</td>
<td>Augment polygons.</td>
</tr>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children(flat)</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 90 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic([in])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters()

imgaug.augmenters.size.Scale(*args, **kwargs)

Deprecated. Use Resize instead. Resize has the exactly same interface as Scale.

13.25 imgaug.augmenters.weather

Augmenters that create weather effects.

Do not import directly from this file, as the categorization is not final. Use instead:

```python
from imgaug import augmenters as iaa
```

and then e.g.:

```python
seq = iaa.Sequential([iaa.Snowflakes()])
```

List of augmenters:

- FastSnowyLandscape
- Clouds
- Fog
- CloudLayer
- Snowflakes
- SnowflakesLayer

```python
class imgaug.augmenters.weather.CloudLayer(intensity_mean, intensity_freq_exponent, intensity_coarse_scale, alpha_min, alpha_multiplier, alpha_size_px_max, alpha_freq_exponent, sparsity, density_multiplier, name=None, deterministic=False, random_state=None)
```

Augmenter to add a single layer of clouds to an image.

dtype support:
Parameters

- **intensity_mean** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* —
  
  Mean intensity of the clouds (i.e. mean color). Recommended to be around \((190, 255)\).
  
  - If a number, then that value will be used for all images.
  
  - If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
  
  - If a list, then a random value will be sampled from that list per image.
  
  - If a StochasticParameter, then a value will be sampled per image from that parameter.

- **intensity_freq_exponent** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* — Exponent of the frequency noise used to add fine intensity to the mean intensity. Recommended to be somewhere around \((-2.5, -1.5)\). See imgaug.parameters.FrequencyNoise.__init__() for details.

- **intensity_coarse_scale** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* — Standard deviation of the gaussian distribution used to add more localized intensity to the mean intensity. Sampled in low resolution space, i.e. affects final intensity on a coarse level. Recommended to be around \((0, 10)\).
  
  - If a number, then that value will be used for all images.
  
  - If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
  
  - If a list, then a random value will be sampled from that list per image.
  
  - If a StochasticParameter, then a value will be sampled per image from that parameter.

- **alpha_min** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* — Minimum alpha when blending cloud noise with the image. High values will lead to clouds being “everywhere”. Recommended to usually be at around 0.0 for clouds and >0 for fog.
  
  - If a number, then that value will be used for all images.
- If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then a value will be sampled per image from that parameter.

**alpha_multiplier** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* – Multiplier for the sampled alpha values. High values will lead to denser clouds wherever they are visible. Recommended to be at around \((0.3, 1.0)\). Note that this parameter currently overlaps with density_multiplier, which is applied a bit later to the alpha mask.

- If a number, then that value will be used for all images.
- If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then a value will be sampled per image from that parameter.

**alpha_size_px_max** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* – Controls the image size at which the alpha mask is sampled. Lower values will lead to coarser alpha masks and hence larger clouds (and empty areas). See imgaug.parameters.FrequencyNoise.__init__() for details.

**alpha_freq_exponent** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* – Exponent of the frequency noise used to sample the alpha mask. Similarly to alpha_size_max_px, lower values will lead to coarser alpha patterns. Recommended to be somewhere around \((-4.0, -1.5)\). See imgaug.parameters.FrequencyNoise.__init__() for details.

**sparsity** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* – Exponent applied late to the alpha mask. Lower values will lead to coarser cloud patterns, higher values to finer patterns. Recommended to be somewhere around 1.0. Do not deviate far from that values, otherwise the alpha mask might get weird patterns with sudden fall-offs to zero that look very unnatural.

- If a number, then that value will be used for all images.
- If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then a value will be sampled per image from that parameter.

**density_multiplier** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter)* – Late multiplier for the alpha mask, similar to alpha_multiplier. Set this higher to get “denser” clouds wherever they are visible. Recommended to be around \((0.5, 1.5)\).

- If a number, then that value will be used for all images.
- If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then a value will be sampled per image from that parameter.

**name** *(None or str, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().

**deterministic** *(bool, optional)* – See imgaug.augmenters.meta.Augmenter.__init__().
- **random_state** *(None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().*

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(*args, **kwargs)</code></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment bounding boxes(bounding_boxes_on_images)</code>, <code>augment heatmaps(heatmaps[, parents, hooks])</code>, <code>augment image(image[, hooks])</code>, <code>augment images(images[, parents, hooks])</code>, <code>augment keypoints(keypoints_on_images[, ...])</code>, <code>augment line strings(line_strings_on_images)</code>, <code>augment polygons(polygons_on_images[, ...])</code>, <code>augment segmentation maps(segmaps[, ...])</code></td>
<td>Augment bounding boxes, a heatmap, a single image, multiple images, keypoints, line strings, polygons, segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_(recursive)</code></td>
<td>Converts global random states to local ones (inplace).</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
</tbody>
</table>
Table 91 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>to_deterministic</strong>([n])</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**draw_on_image**(*image*, *random_state*)

**generate_maps**(*image*, *random_state*)

**get_parameters**()

`imgaug.augmenters.weather.Clouds`(*name=None*, *deterministic=False*, *random_state=None*)

Augmenter to draw clouds in images.

This is a wrapper around CloudLayer. It executes 1 to 2 layers per image, leading to varying densities and frequency patterns of clouds.

This augmenter seems to be fairly robust w.r.t. the image size. Tested with 96x128, 192x256 and 960x1280.

dtype support:

* `\'uint8\'`: yes; tested
* `\'uint16\'`: no (1)
* `\'uint32\'`: no (1)
* `\'uint64\'`: no (1)
* `\'int8\'`: no (1)
* `\'int16\'`: no (1)
* `\'int32\'`: no (1)
* `\'int64\'`: no (1)
* `\'float16\'`: no (1)
* `\'float32\'`: no (1)
* `\'float64\'`: no (1)
* `\'float128\'`: no (1)
* `\'bool\'`: no (1)

- (1) Parameters of this augmenter are optimized for the value range of uint8. While other dtypes may be accepted, they will lead to images augmented in ways inappropriate for the respective dtype.

**Parameters**

- **name** *(None or str, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()``.
- **deterministic** *(bool, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()``.
- **random_state** *(None or int or numpy.random.RandomState, optional)* – See `imgaug.augmenters.meta.Augmenter.__init__()``.

**Examples**

```python
>>> aug = iaa.Clouds()
```
Creates an augmenter that adds clouds to images.

```python
class imgaug.augmenters.weather.FastSnowyLandscape:
    lightness_threshold=(100, 255),
    lightness_multiplier=(1.0, 4.0),
    from_colorspace='RGB',
    name=None, deterministic=False,
    random_state=None)
```

Bases: `imgaug.augmenters.meta.Augmenter`

Augmenter to convert non-snowy landscapes to snowy ones.

This expects to get an image that roughly shows a landscape.

This is based on the method proposed by https://medium.freecodecamp.org/image-augmentation-make-it-rain-make-it-snow-how-to-modify-a-photo-with-machine-learning-163c0cb3843f?gi=bca4a13e634c

dtype support:

- \`\`uint8`\`: yes; fully tested
- \`\`uint16`\`: no (1)
- \`\`uint32`\`: no (1)
- \`\`uint64`\`: no (1)
- \`\`int8`\`: no (1)
- \`\`int16`\`: no (1)
- \`\`int32`\`: no (1)
- \`\`int64`\`: no (1)
- \`\`float16`\`: no (1)
- \`\`float32`\`: no (1)
- \`\`float64`\`: no (1)
- \`\`float128`\`: no (1)
- \`\`bool`\`: no (1)

- (1) This augmenter is based on a colorspace conversion to HLS. Hence, only RGB or uint8 inputs are sensible.

Parameters

- **lightness_threshold** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – All pixels with lightness in HLS colorspace below this value will have their lightness increased by `lightness_multiplier`.
  - If an int, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value from the discrete range \([a .. b]\) will be used.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then a value will be sampled per image from that parameter.

- **lightness_multiplier** *(number or tuple of number or list of number or imgaug.parameters.StochasticParameter, optional)* – Multiplier for pixel’s lightness value in HLS colorspace. Affects all pixels selected via `lightness_threshold`.
  - If a number, then that value will be used for all images.
  - If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
  - If a list, then a random value will be sampled from that list per image.
– If a StochasticParameter, then a value will be sampled per image from that parameter.

• `from_colorspace (str, optional)` – The source colorspace of the input images. See `imgaug.augmenters.color.ChangeColorspace.__init__()`.  
• `name (None or str, optional)` – See `imgaug.augmenters.meta.Augmenter.__init__()`.  
• `deterministic (bool, optional)` – See `imgaug.augmenters.meta.Augmenter.__init__()`.  
• `random_state (None or int or numpy.random.RandomState, optional)` – See `imgaug.augmenters.meta.Augmenter.__init__()`.  

### Examples

```python
g aug = iaa.FastSnowyLandscape(lightness_threshold=140, lightness_multiplier=2.5)
```

Search for all pixels in the image with a lightness value in HLS colorspace of less than 140 and increase their lightness by a factor of 2.5. This is the configuration proposed in the original article (see link above).

```python
g aug = iaa.FastSnowyLandscape(lightness_threshold=[128, 200], lightness_multiplier=(1.5, 3.5))
```

Search for all pixels in the image with a lightness value in HLS colorspace of less than 128 or less than 200 (one of these values is picked per image) and multiply their lightness by a factor of $x$ with $x$ being sampled from $\text{uniform}(1.5, 3.5)$ (once per image).

```python
g aug = iaa.FastSnowyLandscape(lightness_threshold=(100, 255), lightness_multiplier=(1.0, 4.0))
```

Similar to above, but the lightness threshold is sampled from $\text{uniform}(100, 255)$ (per image) and the multiplier from $\text{uniform}(1.0, 4.0)$ (per image). This seems to produce good and varied results.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__(*args, **kwargs)</code></td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code>.</td>
</tr>
<tr>
<td><code>augment([return_batch, hooks])</code></td>
<td>Augment data.</td>
</tr>
<tr>
<td><code>augment_batch(batch[, hooks])</code></td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td><code>augment_batches(batches[, hooks, background])</code></td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td><code>augment_bounding_boxes(bounding_boxes_on_images)</code></td>
<td>Augment bounding boxes.</td>
</tr>
<tr>
<td><code>augment_heatmaps(heatmaps[, parents, hooks])</code></td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td><code>augment_image(image[, hooks])</code></td>
<td>Augment a single image.</td>
</tr>
<tr>
<td><code>augment_images(images[, parents, hooks])</code></td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td><code>augment_keypoints(keypoints_on_images[, ...])</code></td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td><code>augment_line_strings(line_strings_on_images)</code></td>
<td>Augment line strings.</td>
</tr>
<tr>
<td><code>augment_polygons(polygons_on_images[, ...])</code></td>
<td>Augment polygons.</td>
</tr>
</tbody>
</table>
Table 92 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>augment_segmentation_maps(segmaps[, ...])</td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td>copy()</td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td>copy_random_state(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td>copy_random_state_(source[, recursive, ...])</td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td>deepcopy()</td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td>draw_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td>find_augmenters(func[, parents, flat])</td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td>find_augmenters_by_name(name[, regex, flat])</td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td>find_augmenters_by_names(names[, regex, flat])</td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td>get_all_children([flat])</td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td>get_children_lists()</td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td>localize_random_state([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>localize_random_state_([recursive])</td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td>pool([processes, maxtasksperchild, seed])</td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td>remove_augmenters(func[, copy, noop_if_topmost])</td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td>remove_augmenters_inplace(func[, parents])</td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td>reseed([random_state, deterministic_too])</td>
<td>Recess this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td>show_grid(images, rows, cols)</td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td>to_deterministic(In)</td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

get_parameters()

imgaug.augmenters.weather.Fog(name=None, deterministic=False, random_state=None)
Augmenter to draw fog in images.

This is a wrapper around CloudLayer. It executes a single layer per image with a configuration leading to fairly dense clouds with low-frequency patterns.

This augmenter seems to be fairly robust w.r.t. the image size. Tested with 96x128, 192x256 and 960x1280.

dtype support:

* `'uint8'`: yes; tested
* `'uint16'`: no (1)
* `'uint32'`: no (1)
* `'uint64'`: no (1)
* `'int8'`: no (1)
* `'int16'`: no (1)
* `'int32'`: no (1)

(continues on next page)
Parameters

- name (None or str, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
- deterministic (bool, optional) – See imgaug.augmenters.meta.Augmenter.__init__().
- random_state (None or int or numpy.random.RandomState, optional) – See imgaug.augmenters.meta.Augmenter.__init__().

Examples

```python
>>> aug = iaa.Fog()
```

Creates an augmenter that adds fog to images.

* imgaug.augmenters.weather.Snowflakes (density=(0.005, 0.075), density_uniformity=(0.3, 0.9), flake_size=(0.2, 0.7), flake_size_uniformity=(0.4, 0.8), angle=(-30, 30), speed=(0.007, 0.03), name=None, deterministic=False, random_state=None)

Augmenter to add falling snowflakes to images.

This is a wrapper around SnowflakesLayer. It executes 1 to 3 layers per image.

dtype support:

* `uint8`: yes; tested
* `uint16`: no (1)
* `uint32`: no (1)
* `uint64`: no (1)
* `int8`: no (1)
* `int16`: no (1)
* `int32`: no (1)
* `int64`: no (1)
* `float16`: no (1)
* `float32`: no (1)
* `float64`: no (1)
* `float128`: no (1)
* `bool`: no (1)

- (1) Parameters of this augmenter are optimized for the value range of uint8. While other dtypes may be accepted, they will lead to images augmented in ways inappropriate for the respective dtype.
• density (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Density of

Valid value range is (0.0, 1.0). Recommended to be around (0.01, 0.075).

– If a number, then that value will be used for all images.
– If a tuple (a, b), then a value from the continuous range [a, b] will be used.
– If a list, then a random value will be sampled from that list per image.
– If a StochasticParameter, then a value will be sampled per image from that parameter.

– density_uniformity (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Size uniformity of the snowflakes. Higher values denote more similarly sized snowflakes. Valid value range is (0.0, 1.0). Recommended to be around 0.5.

* If a number, then that value will be used for all images.
* If a tuple (a, b), then a value from the continuous range [a, b] will be used.
* If a list, then a random value will be sampled from that list per image.
* If a StochasticParameter, then a value will be sampled per image from that parameter.

– flake_size (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Size of the snowflakes. This parameter controls the resolution at which snowflakes are sampled. Higher values mean that the resolution is closer to the input image’s resolution and hence each sampled snowflake will be smaller (because of the smaller pixel size).

Valid value range is [0.0, 1.0). Recommended values:
* On 96x128 a value of (0.1, 0.4) worked well.
* On 192x256 a value of (0.2, 0.7) worked well.
* On 960x1280 a value of (0.7, 0.95) worked well.

Allowed datatypes:
* If a number, then that value will be used for all images.
* If a tuple (a, b), then a value from the continuous range [a, b] will be used.
* If a list, then a random value will be sampled from that list per image.
* If a StochasticParameter, then a value will be sampled per image from that parameter.

– flake_size_uniformity (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Controls the size uniformity of the snowflakes. Higher values mean that the snowflakes are more similarly sized. Valid value range is (0.0, 1.0). Recommended to be around 0.5.

* If a number, then that value will be used for all images.
* If a tuple (a, b), then a value from the continuous range [a, b] will be used.
* If a list, then a random value will be sampled from that list per image.
* If a StochasticParameter, then a value will be sampled per image from that parameter.
- **angle** *(number or tuple of number or list of number or im-
gaug.parameters.StochasticParameter)* – Angle in degrees of motion blur ap-
plied to the snowflakes, where 0.0 is motion blur that points straight upwards.
Recommended to be around (-30, 30). See also imgaug.augmenters.
blur.MotionBlur.__init__().
  * If a number, then that value will be used for all images.
  * If a tuple (a, b), then a value from the continuous range [a, b] will be used.
  * If a list, then a random value will be sampled from that list per image.
  * If a StochasticParameter, then a value will be sampled per image from that parameter.

- **speed** *(number or tuple of number or list of number or im-
gaug.parameters.StochasticParameter)* – Perceived falling speed of the snowflakes.
This parameter controls the motion blur’s kernel size. It follows roughly the form
kernel_size = image_size * speed. Hence, Values around 1.0 denote
that the motion blur should “stretch” each snowflake over the whole image.
Valid value range is (0.0, 1.0). Recommended values:
  * On 96x128 a value of (0.01, 0.05) worked well.
  * On 192x256 a value of (0.007, 0.03) worked well.
  * On 960x1280 a value of (0.001, 0.03) worked well.

Allowed datatypes:
  * If a number, then that value will be used for all images.
  * If a tuple (a, b), then a value from the continuous range [a, b] will be used.
  * If a list, then a random value will be sampled from that list per image.
  * If a StochasticParameter, then a value will be sampled per image from that parameter.

name [None or str, optional] See imgaug.augmenters.meta.Augmenter.__init__().
deterministic [bool, optional] See imgaug.augmenters.meta.Augmenter.__init__().
random_state [None or int or numpy.random.RandomState, optional] See imgaug.augmenters.meta.
Augmenter.__init__().

**Examples**

```python
>>> aug = iaa.Snowflakes(flake_size=(0.1, 0.4), speed=(0.01, 0.05))
```

Adds snowflakes to small images (around 96x128).

```python
>>> aug = iaa.Snowflakes(flake_size=(0.2, 0.7), speed=(0.007, 0.03))
```

Adds snowflakes to medium-sized images (around 192x256).

```python
>>> aug = iaa.Snowflakes(flake_size=(0.7, 0.95), speed=(0.001, 0.03))
```

Adds snowflakes to large images (around 960x1280).
class `imgaug.augmenters.weather.SnowflakesLayer`(`density`, `density_uniformity`, `flake_size`, `flake_size_uniformity`, `angle`, `speed`, `blur_sigma_fraction`, `blur_sigma_limits`=(0.5, 3.75), `name`=None, `deterministic`=False, `random_state`=None)

Bases: `imgaug.augmenters.meta.Augmenter`

Augmenter to add a single layer of falling snowflakes to images.

dtype support:

* `"uint8```: yes; indirectly tested (1)
* `"uint16```: no
* `"uint32```: no
* `"uint64```: no
* `"int8```: no
* `"int16```: no
* `"int32```: no
* `"int64```: no
* `"float16```: no
* `"float32```: no
* `"float64```: no
* `"float128```: no
* `"bool```: no

- (1) indirectly tested via tests for `"Snowflakes```

**Parameters**

- **density (number or tuple of number or list of number or imgaug.parameters.StochasticParameter)** – Density of the snowflake layer, as a probability of each pixel in low resolution space to be a snowflake. Valid value range is (0.0, 1.0). Recommended to be around (0.01, 0.075).
  - If a number, then that value will be used for all images.
  - If a tuple (a, b), then a value from the continuous range [a, b] will be used.
  - If a list, then a random value will be sampled from that list per image.
  - If a StochasticParameter, then a value will be sampled per image from that parameter.

- **density_uniformity (number or tuple of number or list of number or imgaug.parameters.StochasticParameter)** – Size uniformity of the snowflakes. Higher values denote more similarly sized snowflakes. Valid value range is (0.0, 1.0). Recommended to be around 0.5.
  * If a number, then that value will be used for all images.
  * If a tuple (a, b), then a value from the continuous range [a, b] will be used.
  * If a list, then a random value will be sampled from that list per image.
  * If a StochasticParameter, then a value will be sampled per image from that parameter.
- **flake_size** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Size of the snowflakes. This parameter controls the resolution at which snowflakes are sampled. Higher values mean that the resolution is closer to the input image’s resolution and hence each sampled snowflake will be smaller (because of the smaller pixel size).

Valid value range is \([0.0, 1.0)\). Recommended values:

* On 96x128 a value of \((0.1, 0.4)\) worked well.
* On 192x256 a value of \((0.2, 0.7)\) worked well.
* On 960x1280 a value of \((0.7, 0.95)\) worked well.

Allowed datatypes:

* If a number, then that value will be used for all images.
* If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
* If a list, then a random value will be sampled from that list per image.
* If a StochasticParameter, then a value will be sampled per image from that parameter.

- **flake_size_uniformity** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Controls the size uniformity of the snowflakes. Higher values mean that the snowflakes are more similarly sized.

Valid value range is \((0.0, 1.0)\). Recommended to be around 0.5.

* If a number, then that value will be used for all images.
* If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
* If a list, then a random value will be sampled from that list per image.
* If a StochasticParameter, then a value will be sampled per image from that parameter.

- **angle** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Angle in degrees of motion blur applied to the snowflakes, where 0.0 is motion blur that points straight upwards. Recommended to be around \((-30, 30)\). See also imgaug.augmenters.blur.MotionBlur.__init__().

* If a number, then that value will be used for all images.
* If a tuple \((a, b)\), then a value from the continuous range \([a, b]\) will be used.
* If a list, then a random value will be sampled from that list per image.
* If a StochasticParameter, then a value will be sampled per image from that parameter.

- **speed** (number or tuple of number or list of number or imgaug.parameters.StochasticParameter) – Perceived falling speed of the snowflakes. This parameter controls the motion blur’s kernel size. It follows roughly the form \(\text{kernel\_size} = \text{image\_size} \times \text{speed}\). Hence, Values around 1.0 denote that the motion blur should “stretch” each snowflake over the whole image.
Valid value range is $(0.0, 1.0)$. Recommended values:

- On 96x128 a value of $(0.01, 0.05)$ worked well.
- On 192x256 a value of $(0.007, 0.03)$ worked well.
- On 960x1280 a value of $(0.001, 0.03)$ worked well.

Allowed datatypes:

- If a number, then that value will be used for all images.
- If a tuple $(a, b)$, then a value from the continuous range $[a, b]$ will be used.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then a value will be sampled per image from that parameter.

**blur_sigma_fraction** [number or tuple of number or list of number or imgaug.parameters.StochasticParameter]

Standard deviation (as a fraction of the image size) of gaussian blur applied to the snowflakes. Valid value range is $(0.0, 1.0)$. Recommended to be around $(0.0001, 0.001)$. May still require tinkering based on image size.

- If a number, then that value will be used for all images.
- If a tuple $(a, b)$, then a value from the continuous range $[a, b]$ will be used.
- If a list, then a random value will be sampled from that list per image.
- If a StochasticParameter, then a value will be sampled per image from that parameter.

**blur_sigma_limits** [tuple of float, optional]

Controls allows min and max values of `blur_sigma_fraction` after(!) multiplication with the image size. First value is the minimum, second value is the maximum. Values outside of that range will be clipped to be within that range. This prevents extreme values for very small or large images.

**name** [None or str, optional] See `imgaug.augmenters.meta.Augmenter.__init__()`.  

**deterministic** [bool, optional] See `imgaug.augmenters.meta.Augmenter.__init__()`.  

**random_state** [None or int or numpy.random.RandomState, optional] See `imgaug.augmenters.meta.Augmenter.__init__()`.  

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(*args, **kwargs)</td>
<td>Alias for <code>imgaug.augmenters.meta.Augmenter.augment()</code></td>
</tr>
<tr>
<td>augment([return_batch, hooks])</td>
<td>Augment data.</td>
</tr>
<tr>
<td>augment_batch(batch[, hooks])</td>
<td>Augment a single batch.</td>
</tr>
<tr>
<td>augment_bounding_boxes(bounding_boxes_on_image[, augment_bounding_boxes])</td>
<td>Augment multiple batches.</td>
</tr>
<tr>
<td>augment_heatmaps(heatmaps[, parents, hooks])</td>
<td>Augment a heatmap.</td>
</tr>
<tr>
<td>augment_image(image[, hooks])</td>
<td>Augment a single image.</td>
</tr>
<tr>
<td>augment_images(images[, parents, hooks])</td>
<td>Augment multiple images.</td>
</tr>
<tr>
<td>augment_keypoints(keypoints_on_images[, ...])</td>
<td>Augment image keypoints.</td>
</tr>
<tr>
<td>augment_line_strings(line_strings_on_images)</td>
<td>Augment line strings.</td>
</tr>
</tbody>
</table>
Table 93 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>augment_polygons()</code></td>
<td>Augment polygons.</td>
</tr>
<tr>
<td><code>augment_segmentation_maps()</code></td>
<td>Augment segmentation maps.</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Create a shallow copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>copy_random_state(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence.</td>
</tr>
<tr>
<td><code>copy_random_state_(source[, recursive, ...])</code></td>
<td>Copy the random states from a source augmenter sequence (inplace).</td>
</tr>
<tr>
<td><code>deepcopy()</code></td>
<td>Create a deep copy of this Augmenter instance.</td>
</tr>
<tr>
<td><code>draw_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and return a grid image of the results.</td>
</tr>
<tr>
<td><code>find_augmenters(func[, parents, flat])</code></td>
<td>Find augmenters that match a condition.</td>
</tr>
<tr>
<td><code>find_augmenters_by_name(name[, regex, flat])</code></td>
<td>Find augmenter(s) by name.</td>
</tr>
<tr>
<td><code>find_augmenters_by_names(names[, regex, flat])</code></td>
<td>Find augmenter(s) by names.</td>
</tr>
<tr>
<td><code>get_all_children([flat])</code></td>
<td>Returns all children of this augmenter as a list.</td>
</tr>
<tr>
<td><code>get_children_lists()</code></td>
<td>Get a list of lists of children of this augmenter.</td>
</tr>
<tr>
<td><code>localize_random_state([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>localize_random_state_([recursive])</code></td>
<td>Converts global random states to local ones.</td>
</tr>
<tr>
<td><code>pool([processes, maxtasksperchild, seed])</code></td>
<td>Create a pool used for multicore augmentation from this augmenter.</td>
</tr>
<tr>
<td><code>remove_augmenters(func[, copy, noop_if_topmost])</code></td>
<td>Remove this augmenter or its children that match a condition.</td>
</tr>
<tr>
<td><code>remove_augmenters_inplace(func[, parents])</code></td>
<td>Remove in-place children of this augmenter that match a condition.</td>
</tr>
<tr>
<td><code>reseed([random_state, deterministic_too])</code></td>
<td>Reseed this augmenter and all of its children (if it has any).</td>
</tr>
<tr>
<td><code>show_grid(images, rows, cols)</code></td>
<td>Apply this augmenter to the given images and show/plot the results as a grid of images.</td>
</tr>
<tr>
<td><code>to_deterministic([in])</code></td>
<td>Converts this augmenter from a stochastic to a deterministic one.</td>
</tr>
</tbody>
</table>

**draw_on_image** *(image, random_state)*

**get_parameters** ()

See modindex for API.
CHAPTER 14

Indices and tables

- genindex
- modindex
- search
imgaug.augmentables.batches, 185
imgaug.augmentables.bbs, 189
imgaug.augmentables.heatmaps, 199
imgaug.augmentables.kps, 203
imgaug.augmentables.lines, 211
imgaug.augmentables.normalization, 225
imgaug.augmentables.polys, 226
imgaug.augmentables.segmaps, 237
imgaug.augmentables.utils, 242
imgaug.augmenters.arithmetic, 282
imgaug.augmenters.blend, 313
imgaug.augmenters.blur, 326
imgaug.augmenters.color, 339
imgaug.augmenters.contrast, 347
imgaug.augmenters.convolutional, 364
imgaug.augmenters.flip, 370
imgaug.augmenters.geometric, 375
imgaug.augmenters.meta, 243
imgaug.augmenters.segmentation, 403
imgaug.augmenters.size, 407
imgaug.augmenters.weather, 427
imgaug.dtypes, 184
imgaug.imgaug, 141
imgaug.multicore, 181
imgaug.parameters, 158
Index

A

Absolute (class in imgaug.parameters), 158
Add (class in imgaug.augmenters.arithmetic), 282
Add (class in imgaug.parameters), 158
add() (imgaug.augmenters.meta.Sequential method), 272
add() (imgaug.augmenters.meta.SomeOf method), 275
AddElementwise (class in imgaug.augmenters.arithmetic), 285
AdditiveGaussianNoise() (in module imgaug.augmenters.arithmetic), 287
AdditiveLaplaceNoise() (in module imgaug.augmenters.arithmetic), 288
AdditivePoissonNoise() (in module imgaug.augmenters.arithmetic), 289
AddToHueAndSaturation (class in imgaug.augmenters.color), 339
adjust_contrast_gamma() (in module imgaug.augmenters.contrast), 360
adjust_contrast_linear() (in module imgaug.augmenters.contrast), 361
adjust_contrast_log() (in module imgaug.augmenters.contrast), 362
adjust_contrast_sigmoid() (in module imgaug.augmenters.contrast), 363
Affine (class in imgaug.augmenters.geometric), 375
AffineCv2 (class in imgaug.augmenters.geometric), 383
all_finished() (imgaug.multicore.BackgroundAugmenter method), 181
all_finished() (imgaug.multicore.BatchLoader method), 182
AllChannelsCLAHE (class in imgaug.augmenters.contrast), 348
AllChannelsHistogramEqualization (class in imgaug.augmenters.contrast), 350
ALLOW_DTYPES_CUSTOM_MINMAX (imgaug.augmenters.arithmetic.Invert attribute), 301
almost_equals() (imgaug.augmentables.lines.LineString method), 212
almost_equals() (imgaug.augmentables.polys.Polygon method), 228
Alpha (class in imgaug.augmenters.blend), 313
AlphaElementwise (class in imgaug.augmenters.blend), 317
angle_between_vectors() (in module imgaug.imgaug), 144
area (imgaug.augmentables.bbs.BoundingBox attribute), 190
area (imgaug.augmentables.polys.Polygon attribute), 228
AssertLambda() (in module imgaug.augmenters.meta), 243
AssertShape() (in module imgaug.augmenters.meta), 244
augment() (imgaug.augmenters.meta.Augmenter method), 247
augment_batch() (imgaug.augmenters.meta.Augmenter method), 250
augment_batches() (imgaug.augmenters.meta.Augmenter method), 250
augment_bounding_boxes() (imgaug.augmenters.meta.Augmenter method), 251
augment_heatmaps() (imgaug.augmenters.meta.Augmenter method), 251
augment_image() (imgaug.augmenters.meta.Augmenter method), 252
augment_images() (imgaug.augmenters.meta.Augmenter method), 252
augment_keypoints() (imgaug.augmenters.meta.Augmenter method), 252
augment_line_strings() (imgaug.augmenters.meta.Augmenter method), 253
augment_polygons() (imgaug.augmenters.meta.Augmenter method), 254
augment_segmentation_maps() (imgaug.augmenters.meta.Augmenter method), 255
Augmenter (class in imgaug.augmenters.meta), 246
AverageBlur (class in imgaug.augmenters.blur), 326
avg_pool() (imgaug.augmentables.heatmaps.HeatmapsOnImage method), 199
avg_pool() (in module imgaug.imgaug), 144

BackgroundAugmenter (class in imgaug.multicore), 181
BackgroundAugmenter() (in module imgaug.imgaug), 141
Batch (class in imgaug.augmentables.batches), 185
Batch() (in module imgaug.imgaug), 141
BatchLoader (class in imgaug.multicore), 182
BatchLoader() (in module imgaug.imgaug), 141
Beta (class in imgaug.parameters), 159
BGR (imgaug.augmenters.color.ChangeColorspace attribute), 343
BGR (imgaug.augmenters.contrast.CLAHE attribute), 355
BGR (imgaug.augmenters.contrast.HistogramEqualization attribute), 358
BilateralBlur (class in imgaug.augmenters.blur), 329
Binomial (class in imgaug.parameters), 160
blend_alpha() (in module imgaug.augmenters.blend), 325
blur_gaussian_() (in module imgaug.augmenters.blur), 337
both_np_float_if_one_is_float() (in module imgaug.parameters), 180
bounding_boxes (imgaug.augmentables.batches.Batch attribute), 186
BoundingBox (class in imgaug.augmentables.bbs), 189
BoundingBox() (in module imgaug.imgaug), 141
BoundingBoxesOnImage (class in imgaug.augmentables.bbs), 195
BoundingBoxesOnImage() (in module imgaug.imgaug), 141

caller_name() (in module imgaug.imgaug), 145
center_x (imgaug.augmentables.bbs.BoundingBox attribute), 190
center_y (imgaug.augmentables.bbs.BoundingBox attribute), 190
change_first_point_by_coords() (imgaug.augmentables.polys.Polygon method), 228
change_first_point_by_index() (imgaug.augmentables.polys.Polygon method), 228
change_normalization() (imgaug.augmentables.heatmaps.HeatmapsOnImage class method), 200
ChangeColorspace (class in imgaug.augmenters.color), 341
ChannelShuffle (class in imgaug.augmenters.meta), 262
ChiSquare (class in imgaug.parameters), 161
Choice (class in imgaug.parameters), 161
CIE (imgaug.augmenters.color.ChangeColorspace attribute), 343
CLAHE (class in imgaug.augmenters.contrast), 351
Clip (class in imgaug.parameters), 162
clip_() (in module imgaug.dtypes), 184
clip_augmented_image() (in module imgaug.augmenters.meta), 281
clip_augmented_image_() (in module imgaug.augmenters.meta), 281
clip_augmented_images() (in module imgaug.augmenters.meta), 281
clip_augmented_images_() (in module imgaug.augmenters.meta), 281
clip_out_of_image() (imgaug.augmentables.bbs.BoundingBox method), 190
clip_out_of_image() (imgaug.augmentables.bbs.BoundingBoxesOnImage method), 196
clip_out_of_image() (imgaug.augmentables.lines.LineString method), 213
clip_out_of_image() (imgaug.augmentables.lines.LineStringsOnImage method), 222
clip_out_of_image() (imgaug.augmentables.polys.Polygon method), 229
clip_out_of_image() (imgaug.augmentables.polys.PolygonsOnImage method), 235
clip_to_dtype_value_range_() (in module imgaug.dtypes), 184
deecopy() *(imgaug.augmentables.bbs.BoundingBox method), 191* 

deeopy() *(imgaug.augmentables.bbs.BoundingBoxesOnImage method), 196* 

deeopy() *(imgaug.augmentables.heatsmaps.HeatmapsOnImage method), 200* 

deeopy() *(imgaug.augmentables.kps.Keypoint method), 204* 

deeopy() *(imgaug.augmentables.kps.KeypointsOnImage method), 207* 

deeopy() *(imgaug.augmentables.lines.LineString method), 214* 

deeopy() *(imgaug.augmentables.lines.LineStringsOnImage method), 223* 

deeopy() *(imgaug.augmentables.polys.Polygon method), 229* 

deeopy() *(imgaug.augmentables.polys.PolygonsOnImage method), 235* 

deeopy() *(imgaug.augmentables.segmaps.SegmentationMapOnImage method), 238* 

deeopy() *(imgaug.augmenters.meta.Augmenter method), 256* 

deeopy() *(imgaug.parameters.StochasticParameter method), 177* 

```
DEFAULT_SEGMENT_COLORS *(imgaug.augmentables.segmaps.SegmentationMapOnImage attribute), 238
```

deprecated *(class in imgaug.imgaug), 146* 

```
DeprecationWarning, 141
```

derive_random_state() *(in module imgaug.imgaug), 147* 

derive_random_states() *(in module imgaug.imgaug), 147* 

```
Deterministic *(class in imgaug.parameters), 162
```

```
DirectedEdgeDetect() *(in module imgaug.augmenters.convolutional), 162
```

```
DiscreteUniform *(class in imgaug.parameters), 163
```

```
Discretize *(class in imgaug.parameters), 164
```

```
Divide *(class in imgaug.parameters), 164
```

do_assert() *(in module imgaug.imgaug), 147* 

draw() *(imgaug.augmentables.heatmaps.HeatmapsOnImage method), 200* 

draw() *(imgaug.augmentables.segmaps.SegmentationMapOnImage method), 239* 

draw_distribution_graph() *(imgaug.parameters.StochasticParameter method), 177* 

draw_distributions_grid() *(in module imgaug.parameters), 180* 

draw_grid() *(imgaug.augmenters.meta.Augmenter method), 256* 

draw_grid() *(in module imgaug.imgaug), 147* 

draw_heatmap_array() *(imgaug.augmentables.lines.LineString method), 191* 

draw_lines_heatmap_array() *(imgaug.augmentables.lines.LineString method), 215* 

draw_lines_on_image() *(imgaug.augmentables.lines.LineString method), 215* 

draw_mask() *(imgaug.augmentables.lines.LineString method), 216* 

draw_on_image() *(imgaug.augmentables.bbs.BoundingBox method), 191* 

draw_on_image() *(imgaug.augmentables.bbs.BoundingBoxesOnImage method), 197* 

draw_on_image() *(imgaug.augmentables.heatsmaps.HeatmapsOnImage method), 200* 

draw_on_image() *(imgaug.augmentables.kps.Keypoint method), 204* 

draw_on_image() *(imgaug.augmentables.kps.KeypointsOnImage method), 207* 

draw_on_image() *(imgaug.augmentables.lines.LineString method), 223* 

draw_on_image() *(imgaug.augmentables.lines.LineStringsOnImage method), 223* 

draw_on_image() *(imgaug.augmentables.polys.Polygon method), 229* 

draw_on_image() *(imgaug.augmentables.polys.PolygonsOnImage method), 235* 

draw_on_image() *(imgaug.augmentables.segmaps.SegmentationMapOnImage method), 238* 

draw_on_image() *(imgaug.augmenters.weather.CloudLayer method), 431* 

draw_on_image() *(imgaug.augmenters.weather.CloudLayer method), 431* 

draw_on_image() *(imgaug.augmenters.weather.SnowflakesLayer method), 441* 

draw_points_heatmap_array() *(imgaug.augmentables.array.LineString method), 217* 

draw_sample() *(imgaug.parameters.StochasticParameter method), 177*
from_shapely()
  (imgaug.augmentables.polys.MultiPolygon static method), 226
from_shapely()
  (imgaug.augmentables.polys.Polygon static method), 231
from_uint8()  (imgaug.augmentables.heatmaps.HeatmapsOnImage static method), 201
from_xy_array()
  (imgaug.augmentables.kps.KeypointsOnImage class method), 209
from_xy_arrays()
  (imgaug.augmentables.lines.LineStringsOnImage class method), 224
from_xyxy_array()
  (imgaug.augmentables.bbs.BoundingBoxesOnImage class method), 197
FromLowerResolution (class in imgaug.parameters), 167
G
GammaContrast()  (in module imgaug.augmenters.contrast), 355
gate_dtypes()  (in module imgaug.dtypes), 185
GaussianBlur (class in imgaug.augmenters.blur), 331
generate_maps()
  (imgaug.augmenters.weather.CloudLayer method), 431
generate_shift_maps()
  (imgaug.augmenters.geometric.ElasticTransformation class method), 393
generate_similar_points_manhattan()  (imgaug.augmentables.kps.Keypoint method), 205
get_all_children()
  (imgaug.augmenters.meta.Augmenter method), 258
get_arr()  (imgaug.augmentables.heatmaps.HeatmapsOnImage method), 202
get_arr_int()
  (imgaug.augmentables.segmaps.SegmentationMapOnImage method), 240
get_batch()  (imgaug.multicore.BackgroundAugmenter method), 181
get_children_lists()
  (imgaug.augmenters.meta.SomeOf method), 272
get_children_lists()
  (imgaug.augmenters.meta.Sometimes method), 276
get_children_lists()
  (imgaug.augmenters.meta.WithChannels method), 280
get_children_lists()
  (imgaug.augmenters.size.KeepSizeByResize method), 418
get_coords_array()
  (imgaug.augmentables.kps.KeypointsOnImage method), 209
get_minimal_dtype()  (in module imgaug.dtypes), 185
get_minimal_dtype_by_value_range()  (in module imgaug.dtypes), 185
get_minimal_dtype_for_values()  (in module imgaug.dtypes), 185
get_parameters()
  (imgaug.augmenters.arithmetic.Add method), 285
get_parameters()
  (imgaug.augmenters.arithmetic.AddElementwise method), 287
get_parameters()
  (imgaug.augmenters.arithmetic.Invert method), 301
get_parameters()
  (imgaug.augmenters.arithmetic.JpegCompression method), 303
get_parameters()
  (imgaug.augmenters.arithmetic.Multiply method), 306
get_parameters()
  (imgaug.augmenters.arithmetic.MultiplyElementwise method), 308
get_parameters()
  (imgaug.augmenters.arithmetic.ReplaceElementwise method), 311
get_parameters()
  (imgaug.augmenters.blend.Alpha method), 317
get_parameters()
  (imgaug.augmenters.blur.AverageBlur method), 329
get_parameters()
  (imgaug.augmenters.blur.BilateralBlur method), 331
get_parameters()
  (imgaug.augmenters.blur.GaussianBlur method), 333
get_parameters() (imgaug.augmenters.blur.MedianBlur method), get_parameters() (imgaug.augmenters.meta.Lambda method), 264
get_parameters() (imgaug.augmenters.color.AddToHueAndSaturation method), 341
get_parameters() (imgaug.augmenters.color.ChangeColorspace method), 343
get_parameters() (imgaug.augmenters.color.WithColorspace method), 347
get_parameters() (imgaug.augmenters.contrast.AllChannelsCLAHE method), 350
get_parameters() (imgaug.augmenters.contrast.AllChannelsHistogramEqualization method), 351
get_parameters() (imgaug.augmenters.contrast.CLAHE method), 355
get_parameters() (imgaug.augmenters.contrast.HistogramEqualization method), 358
get_parameters() (imgaug.augmenters.convolutional.Convolve method), 367
get_parameters() (imgaug.augmenters.flip.Flip method), 373
get_parameters() (imgaug.augmenters.flip.Flipud method), 374
get_parameters() (imgaug.augmenters.geometric.Affine method), 383
get_parameters() (imgaug.augmenters.geometric.AffineCv2 method), 389
get_parameters() (imgaug.augmenters.geometric.ElasticTransformation method), 393
get_parameters() (imgaug.augmenters.geometric.PerspectiveTransform method), 398
get_parameters() (imgaug.augmenters.geometric.PiecewiseAffine method), 401
get_parameters() (imgaug.augmenters.geometric.Rot90 method), 403
get_parameters() (imgaug.augmenters.meta.Augmenter method), 258
get_parameters() (imgaug.augmenters.meta.ChannelShuffle method), 267
get_parameters() (imgaug.augmenters.meta.Noop method), 269
get_parameters() (imgaug.augmenters.meta.Sequential method), 273
get_parameters() (imgaug.augmenters.meta.SomeOf method), 276
get_parameters() (imgaug.augmenters.meta.Sometimes method), 278
get_parameters() (imgaug.augmenters.meta.WithChannels method), 281
get_parameters() (imgaug.augmenters.size.CropAndPad method), 413
get_parameters() (imgaug.augmenters.size.CropToFixedSize method), 416
get_parameters() (imgaug.augmenters.size.KeepSizeByResize method), 418
get_parameters() (imgaug.augmenters.size.PadToFixedSize method), 424
get_parameters() (imgaug.augmenters.size.Resize method), 427
get_parameters() (imgaug.augmenters.weather.CloudLayer method), 431
get_parameters() (imgaug.augmenters.weather.FastSnowyLandscape method), 434
get_parameters() (imgaug.augmenters.weather.SnowflakesLayer method), 441
get_pointwise_inside_image_mask() (imgaug.augmentables.lines.LineString method), 218
get_value_range_of_dtype() (in module imgaug.dtypes), 185
GRAY (imgaug.augmenters.color.ChangeColorspace attribute), 343
Grayscale() (in module imgaug.augmenters.color), 344
H

handle_children_list() (in module imgaug.augmenters.meta), 281
handle_continuous_param() (in module imgaug.parameters), 180
handle_discrete_kernel_size_param() (in module imgaug.parameters), 181
handle_discrete_param() (in module imgaug.parameters), 181
handle_probability_param() (in module imgaug.parameters), 181
heatmaps (imgaug.augmentables.batches.Batch attribute), 186
HeatmapsOnImage (class in imgaug.augmentables.heatmaps), 199
HeatmapsOnImage() (in module imgaug.imgaug), 141
height (imgaug.augmentables.bbs.BoundingBox attribute), 193
height (imgaug.augmentables.bbs.BoundingBoxesOnImage attribute), 197
height (imgaug.augmentables.kps.KeypointsOnImage attribute), 209
height (imgaug.augmentables.lines.LineString attribute), 218
height (imgaug.augmentables.polys.Polygon attribute), 232
HistogramEqualization (class in imgaug.augmenters.contrast), 356
HLS (imgaug.augmenters.color.ChangeColorspace attribute), 343
HLS (imgaug.augmenters.contrast.CLAHE attribute), 355
HLS (imgaug.augmenters.contrast.HistogramEqualization attribute), 358
HooksHeatmaps (class in imgaug.imgaug), 141
HooksImages (class in imgaug.imgaug), 142
HooksKeypoints (class in imgaug.imgaug), 143
HorizontalFlip() (in module imgaug.augmenters.flip), 374
HSV (imgaug.augmenters.color.ChangeColorspace attribute), 343
HSV (imgaug.augmenters.color.CLAHE attribute), 355
HSV (imgaug.augmenters.contrast.HistogramEqualization attribute), 358

I

dimage (imgaug.augmentables.batches.Batch attribute), 186
imap_batches() (imgaug.multicore.Pool method), 183
imap_batches_unordered() (imgaug.multicore.Pool method), 183
imresize_many_images() (in module imgaug.imgaug), 149
imresize_single_image() (in module imgaug.imgaug), 150
InColorspace() (in module imgaug.augmenters.color), 345
increase_array_resolutions_() (in module imgaug.dtypes), 185
interpolate_point_pair() (in module imgaug.augmentables.utils), 242
interpolate_points() (in module imgaug.augmentables.utils), 242
interpolate_points_by_max_distance() (in module imgaug.augmentables.utils), 242
intersection() (imgaug.augmentables.bbs.BoundingBox attribute method), 193
Invert (class in imgaug.augmenters.arithmetic), 298
invert() (imgaug.augmentables.heatmaps.HeatmapsOnImage method), 202
invert_normalize_heatmaps() (module imgaug.augmentables.normalization), 225
invert_normalize_images() (module imgaug.augmentables.normalization), 225
invert_normalize_keypoints() (module imgaug.augmentables.normalization), 225
invert_normalize_line_strings() (module imgaug.augmentables.normalization), 225
invert_normalize_polygons() (module imgaug.augmentables.normalization), 225
invert_normalize_segmentation_maps() (module imgaug.augmentables.normalization), 225
invert_reduce_to_nonempty() (module imgaug.augmenters.meta), 225
iou() (module imgaug.augmentables.bbs.BoundingBox method), 193
is_activated() (module imgaug.imgaug.HooksImages method), 143
is_callable() (module imgaug.imgaug), 151
is_fully_within_image() (module imgaug.augmentables.bbs.BoundingBox method), 193
is_fully_within_image() (module imgaug.augmentables.lines.LineString method), 218
is_fully_within_image() (module imgaug.augmentables.polys.Polygon method), 232
is_generator() (module imgaug.imgaug), 151
is_integer_array() (module imgaug.imgaug), 151
is_out_of_image() (module imgaug.augmentables.bbs.BoundingBox method), 193
is_out_of_image() (module imgaug.augmentables.lines.LineString method), 219
is_out_of_image() (module imgaug.augmentables.polys.Polygon method), 232
is_partly_within_image() (module imgaug.augmentables.bbs.BoundingBox method), 194
is_partly_within_image() (module imgaug.augmentables.lines.LineString method), 219
is_partly_within_image() (module imgaug.augmentables.polys.Polygon method), 232
is_propagating() (module imgaug.imgaug.HooksImages method), 143
is_single_bool() (module imgaug.imgaug), 152
is_single_float() (module imgaug.imgaug), 152
is_single_integer() (module imgaug.imgaug), 152
is_single_number() (module imgaug.imgaug), 152
is_string() (module imgaug.imgaug), 152
is_valid (module imgaug.augmentables.polys.Polygon attribute), 232
IterativeNoiseAggregator (class in module imgaug.parameters), 168
J
join() (module imgaug.multicore.Pool method), 184
JpegCompression (class in module imgaug.augmenters.arithmetic), 301
K
KeepSizeByResize (class in module imgaug.augmenters.size), 416
Keypoint (class in module imgaug.augmentables.kps), 203
Keypoint() (module imgaug.imgaug), 144
KEYPOINT_AUG_ALPHA_THRESH (module imgaug.augmenters.geometric.ElasticTransformation attribute), 393
KEYPOINT_AUG_SIGMA_THRESH (module imgaug.augmenters.geometric.ElasticTransformation attribute), 393
keypoints (module imgaug.augmentables.batches.Batch attribute), 186
KeypointsOnImage (class in module imgaug.augmentables.kps), 206
KeypointsOnImage() (module imgaug.imgaug), 144
L
Lab (module imgaug.augmenters.color.ChangeColorspace attribute), 343
Lab (module imgaug.augmenters.contrast.CLAHE attribute), 355
Lab (module imgaug.augmenters.contrast.HistogramEqualization attribute), 358
Lambda (class in module imgaug.augmenters.meta), 264
Laplace (class in module imgaug.parameters), 169
length (module imgaug.augmentables.lines.LineString attribute), 219
LinearContrast() (module imgaug.augmenters.contrast), 358
LineString (class in module imgaug.augmentables.lines), 211
LineStringsOnImage (class in module imgaug.augmentables.lines), 221
localize_random_state()  (im-
  gaug.augmenters.meta.Augmenter method), 258
localize_random_state_()  (im-
  gaug.augmenters.meta.Augmenter method), 258
LogContrast()  (in module im-
  gaug.augmenters.contrast), 359
Luv  (imgaug.augmenters.color.ChangeColorspace attribute), 343
map_batches()  (imgaug.multicore.Pool method), 184
map_batches_async()  (imgaug.multicore.Pool method), 184
map_coordinates()  (im-
  gaug.augmenters.geometric.ElasticTransformation class method), 393
max_pool()  (imgaug.augmentables.heatmaps.HeatmapsOnImage method), 202
max_pool()  (in module imgaug.imgaug), 152
MedianBlur  (class in imgaug.augmenters.blur), 333
MotionBlur()  (in module imgaug.augmenters.blur), 335
Multiply  (class in imgaug.augmenters.arithmetic), 303
Multiply  (class in imgaug.parameters), 170
MultiplyElementwise  (class in im-
  gaug.augmenters.arithmetic), 306
MultiPolygon  (class in imgaug.augmentables.polys), 226
MultiPolygon()  (in module imgaug.imgaug), 144

NB_NEIGHBOURING_KEYPOINTS  (im-
  gaug.augmenters.geometric.ElasticTransformation attribute), 393
Negative()  (in module imgaug.parameters), 171
NEIGHBOURING_KEYPOINTS_DISTANCE  (im-
  gaug.augmenters.geometric.ElasticTransformation attribute), 393
new_random_state()  (in module imgaug.imgaug), 153
NO_RESIZE  (imgaug.augmenters.size.KeepSizeByResize attribute), 418
Noop  (class in imgaug.augmenters.meta), 267
Normal  (class in imgaug.parameters), 171
normalize_bounding_boxes()  (in module imgaug.augmentables.normalization), 225
normalize_heatmaps()  (in module imgaug.augmentables.normalization), 225
normalize_images()  (in module imgaug.augmentables.normalization), 225
normalize_keypoints()  (in module imgaug.augmentables.normalization), 225
normalize_line_strings()  (in module imgaug.augmentables.normalization), 226
normalize_polygons()  (in module imgaug.augmentables.normalization), 226
normalize_segmentation_maps()  (in module imgaug.augmentables.normalization), 226
normalize_shape()  (in module imgaug.augmentables.utils), 242

O

on()  (imgaug.augmentables.bbs.BoundingBoxesOnImage method), 197
on()  (imgaug.augmentables.kps.KeypointsOnImage method), 209
on()  (imgaug.augmentables.lines.LineStringsOnImage method), 224
OneOf()  (in module imgaug.augmenters.meta), 269

P

pad()  (imgaug.augmentables.heatmaps.HeatmapsOnImage method), 202
pad()  (imgaug.augmentables.segmaps.SegmentationMapOnImage method), 240
Pad()  (in module imgaug.augmenters.size), 418
pad()  (in module imgaug.imgaug), 153
pad_to_aspect_ratio()  (im-
  gaug.augmentables.heatmaps.HeatmapsOnImage method), 203
pad_to_aspect_ratio()  (im-
  gaug.augmentables.segmaps.SegmentationMapOnImage method), 241
pad_to_aspect_ratio()  (in module imgaug.imgaug), 154
PadToFixedSize  (class in imgaug.augmenters.size), 421
Pepper()  (in module imgaug.augmenters.arithmetic), 308
PerspectiveTransform  (class in imgaug.augmenters.geometric), 394
PiecewiseAffine  (class in imgaug.augmenters.geometric), 398
Poisson  (class in imgaug.parameters), 172
Polygon  (class in imgaug.augmentables.polys), 226
Polygon()  (in module imgaug.imgaug), 144
PolygonsOnImage  (class in imgaug.augmentables.polys), 234
PolygonsOnImage()  (in module imgaug.imgaug), 144
Pool  (class in imgaug.multicore), 182
pool  (imgaug.multicore.Pool attribute), 184
pool() (imgaug.augmenters.meta.Augmenter method), 259
pool() (in module imgaug.imgaug), 154
Positive() (in module imgaug.parameters), 172
postprocess() (imgaug.imgaug.HooksImages method), 143
Power (class in imgaug.parameters), 173
preprocess() (imgaug.imgaug.HooksImages method), 143
project() (imgaug.augmentables.bbs.BoundingBox method), 194
project() (imgaug.augmentables.kps.Keypoint method), 205
project() (imgaug.augmentables.lines.LineString method), 219
project() (imgaug.augmentables.polys.Polygon method), 233
project_coords() (in module imgaug.augmentables.utils), 242
promote_array_dtypes_() (in module imgaug.dtypes), 185
Quokka() (in module imgaug.imgaug), 155
QuokkaBoundingBoxes() (in module imgaug.imgaug), 156
QuokkaHeatmap() (in module imgaug.imgaug), 156
QuokkaKeypoints() (in module imgaug.imgaug), 156
QuokkaPolygons() (in module imgaug.imgaug), 157
QuokkaSegmentationMap() (in module imgaug.imgaug), 157
QuokkaSquare() (in module imgaug.imgaug), 157
ReplaceElementwise (class in imgaug.augmenters.arithmetic), 309
reseed() (imgaug.augmenters.meta.Augmenter method), 261
Resize (class in imgaug.augmenters.size), 424
Resize() (imgaug.augmentables.heatmaps.HeatmapsOnImage method), 203
Resize() (imgaug.augmentables.segmaps.SegmentationMapOnImage method), 241
restore_dtypes_and_merge() (in module imgaug.augmenters.normalization), 226
restore_dtypes_() (in module imgaug.dtypes), 185
RGB (imgaug.augmenters.color.ChangeColorspace attribute), 343
RGB (imgaug.augmenters.contrast.CLAHE attribute), 355
RGB (imgaug.augmenters.contrast.HistogramEqualization attribute), 358
Rot90 (class in imgaug.augmenters.geometric), 401
Salt() (in module imgaug.augmenters.arithmetic), 312
SaltAndPepper() (in module imgaug.augmenters.arithmetic), 312
SAME_AS_IMAGES (imgaug.augmenters.size.KeepSizeByResize attribute), 418
Scale() (imgaug.augmentables.heatmaps.HeatmapsOnImage method), 203
Scale() (imgaug.augmentables.segmaps.SegmentationMapOnImage method), 242
Scale() (in module imgaug.augmenters.size), 427
seed() (in module imgaug.imgaug), 157
segmentation_maps (imgaug.augmenters.batches.Batch attribute), 186
SegmentationMapOnImage (class in imgaug.augmenters.segmaps), 237
SegmentationMapOnImage() (in module imgaug.imgaug), 144
Sequential (class in imgaug.augmenters.meta), 270
Sharpen() (in module imgaug.augmenters.convolutional), 369
Shift() (imgaug.augmentables.bbs.BoundingBox method), 194
Shift() (imgaug.augmentables.bbs.BoundingBoxesOnImage method), 198
Shift() (imgaug.augmentables.kps.Keypoint method), 205
Shift() (imgaug.augmentables.kps.KeypointsOnImage method), 209
Shift() (imgaug.augmentables.lines.LineString method), 219
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>shift()</code></td>
<td>(imgaug.augmentables.lines.LineStringsOnImage method), 224</td>
</tr>
<tr>
<td><code>shift()</code></td>
<td>(imgaug.augmentables.polys.Polygon method), 233</td>
</tr>
<tr>
<td><code>shift()</code></td>
<td>(imgaug.augmentables.polys.PolygonsOnImage method), 237</td>
</tr>
<tr>
<td><code>show_distributions_grid()</code></td>
<td>(in module imgaug.parameters), 181</td>
</tr>
<tr>
<td><code>show_grid()</code></td>
<td>(imgaug.augmenters.meta.Augmenter method), 261</td>
</tr>
<tr>
<td><code>show_grid()</code></td>
<td>(in module imgaug.imgaug), 157</td>
</tr>
<tr>
<td><code>shuffle_channels()</code></td>
<td>(in module imgaug.augmenters.meta), 281</td>
</tr>
<tr>
<td><code>Sigmoid</code></td>
<td>(class in imgaug.parameters), 174</td>
</tr>
<tr>
<td><code>SigmoidContrast()</code></td>
<td>(in module imgaug.augmenters.contrast), 360</td>
</tr>
<tr>
<td><code>SimplexNoise</code></td>
<td>(class in imgaug.parameters), 175</td>
</tr>
<tr>
<td><code>SimplexNoiseAlpha()</code></td>
<td>(in module imgaug.augmenters.blend), 322</td>
</tr>
<tr>
<td><code>Snowflakes()</code></td>
<td>(in module imgaug.augmenters.weather), 435</td>
</tr>
<tr>
<td><code>SnowflakesLayer</code></td>
<td>(class in imgaug.augmenters.weather), 437</td>
</tr>
<tr>
<td><code>SomeOf</code></td>
<td>(class in imgaug.augmenters.meta), 273</td>
</tr>
<tr>
<td><code>Sometimes</code></td>
<td>(class in imgaug.augmenters.meta), 276</td>
</tr>
<tr>
<td><code>StochasticParameter</code></td>
<td>(class in imgaug.parameters), 176</td>
</tr>
<tr>
<td><code>subdivide()</code></td>
<td>(imgaug.augmentables.lines.LineString method), 220</td>
</tr>
<tr>
<td><code>Subtract</code></td>
<td>(class in imgaug.augmenters.lines.LineString method), 221</td>
</tr>
<tr>
<td><code>Superpixels</code></td>
<td>(class in imgaug.augmenters.segmentation), 404</td>
</tr>
<tr>
<td><code>terminate()</code></td>
<td>(imgaug.multicore.BackgroundAugmenter method), 182</td>
</tr>
<tr>
<td><code>terminate()</code></td>
<td>(imgaug.multicore.BatchLoader method), 182</td>
</tr>
<tr>
<td><code>terminate()</code></td>
<td>(imgaug.multicore.Pool method), 184</td>
</tr>
<tr>
<td><code>to_bounding_box()</code></td>
<td>(imgaug.augmentables.lines.LineString method), 220</td>
</tr>
<tr>
<td><code>to_boundings_box()</code></td>
<td>(imgaug.augmentables.lines.LineString method), 220</td>
</tr>
<tr>
<td><code>to_bounding_box()</code></td>
<td>(imgaug.augmentables.polys.Polygon method), 233</td>
</tr>
<tr>
<td><code>to_deterministic()</code></td>
<td>(imgaug.augmenters.meta.Augmenter method), 262</td>
</tr>
<tr>
<td><code>to_distance_maps()</code></td>
<td>(imgaug.augmentables.kps.KeypointsOnImage method), 210</td>
</tr>
<tr>
<td><code>to_heatmap()</code></td>
<td>(imgaug.augmentables.lines.LineString method), 220</td>
</tr>
<tr>
<td><code>to_heatmaps()</code></td>
<td>(imgaug.augmentables.segmaps.SegmentationMapOnImage method), 242</td>
</tr>
<tr>
<td><code>to_keypoint_image()</code></td>
<td>(imgaug.augmentables.bbs.BoundingBox method), 194</td>
</tr>
<tr>
<td><code>to_keypoints()</code></td>
<td>(imgaug.augmentables.bbs.BoundingBox method), 194</td>
</tr>
<tr>
<td><code>to_keypoints()</code></td>
<td>(imgaug.augmentables.lines.LineString method), 221</td>
</tr>
<tr>
<td><code>to_keypoints()</code></td>
<td>(imgaug.augmentables.polys.Polygon method), 233</td>
</tr>
<tr>
<td><code>to_line_string()</code></td>
<td>(imgaug.augmentables.polys.Polygon method), 233</td>
</tr>
<tr>
<td><code>to_normalized_batch()</code></td>
<td>(imgaug.augmentables.batches.UnnormalizedBatch method), 188</td>
</tr>
<tr>
<td><code>to_polygon()</code></td>
<td>(imgaug.augmentables.lines.LineString method), 221</td>
</tr>
<tr>
<td><code>to_segmentation_map()</code></td>
<td>(imgaug.augmentables.lines.LineString method), 221</td>
</tr>
<tr>
<td><code>to_shapely_line_string()</code></td>
<td>(imgaug.augmentables.polys.Polygon method), 233</td>
</tr>
<tr>
<td><code>to_shapely_polygon()</code></td>
<td>(imgaug.augmentables.polys.Polygon method), 234</td>
</tr>
<tr>
<td><code>to_uint8()</code></td>
<td>(imgaug.augmentables.heatmaps.HeatmapsOnImage method), 203</td>
</tr>
<tr>
<td><code>to_xy_array()</code></td>
<td>(imgaug.augmentables.kps.KeypointsOnImage method), 210</td>
</tr>
<tr>
<td><code>to_xy_arrays()</code></td>
<td>(imgaug.augmentables.lines.LineStringsOnImage method), 225</td>
</tr>
<tr>
<td><code>to_xyxy_array()</code></td>
<td>(imgaug.augmentables.bbs.BoundingBoxesOnImage method), 198</td>
</tr>
<tr>
<td><code>TruncatedNormal</code></td>
<td>(class in imgaug.parameters), 178</td>
</tr>
<tr>
<td><code>Uniform</code></td>
<td>(class in imgaug.parameters), 179</td>
</tr>
<tr>
<td><code>union()</code></td>
<td>(imgaug.augmentables.bbs.BoundingBox method), 195</td>
</tr>
<tr>
<td><code>UnnormalizedBatch</code></td>
<td>(class in imgaug.augmentables.batches), 186</td>
</tr>
</tbody>
</table>

**Index**

458
gaug.augmenters.geometric.Affine attribute), \texttt{yy\_int} (imgaug.augmentables.lines.LineString attribute), 221
\texttt{yy\_int} (imgaug.augmentables.polys.Polygon attribute), 234
VerticalFlip() \texttt{(in module imgaug.augmenters.flip)}, 374
warn_deprecated() \texttt{(in module imgaug.imgaug)}, 158
\texttt{Weibull} \texttt{(class in imgaug.parameters)}, 180
\texttt{width} \texttt{(imgaug.augmentables.bbs.BoundingBox attribute)}, 195
\texttt{width} \texttt{(imgaug.augmentables.bbs.BoundingBoxesOnImage attribute)}, 198
\texttt{width} \texttt{(imgaug.augmentables.kps.KeypointsOnImage attribute)}, 210
\texttt{width} \texttt{(imgaug.augmentables.lines.LineString attribute)}, 221
\texttt{width} \texttt{(imgaug.augmentables.polys.Polygon attribute)}, 234
\texttt{WithChannels} \texttt{(class in imgaug.augmenters.meta)}, 278
\texttt{WithColorspace} \texttt{(class in imgaug.augmenters.color)}, 345
\texttt{X}
\texttt{x\_int} \texttt{(imgaug.augmentables.bbs.BoundingBox attribute)}, 195
\texttt{x\_int} \texttt{(imgaug.augmentables.bbs.BoundingBox attribute)}, 195
\texttt{x\_int} \texttt{(imgaug.augmentables.kps.Keypoint attribute)}, 206
\texttt{xx} \texttt{(imgaug.augmentables.lines.LineString attribute)}, 221
\texttt{xx} \texttt{(imgaug.augmentables.polys.Polygon attribute)}, 234
\texttt{xx\_int} \texttt{(imgaug.augmentables.lines.LineString attribute)}, 221
\texttt{xx\_int} \texttt{(imgaug.augmentables.polys.Polygon attribute)}, 234
\texttt{Y}
\texttt{y\_int} \texttt{(imgaug.augmentables.bbs.BoundingBox attribute)}, 195
\texttt{y\_int} \texttt{(imgaug.augmentables.bbs.BoundingBox attribute)}, 195
\texttt{y\_int} \texttt{(imgaug.augmentables.kps.Keypoint attribute)}, 206
\texttt{YCrCb} \texttt{(imgaug.augmenters.color.ChangeColorspace attribute)}, 343
\texttt{yy} \texttt{(imgaug.augmentables.lines.LineString attribute)}, 221
\texttt{yy} \texttt{(imgaug.augmentables.polys.Polygon attribute)}, 234