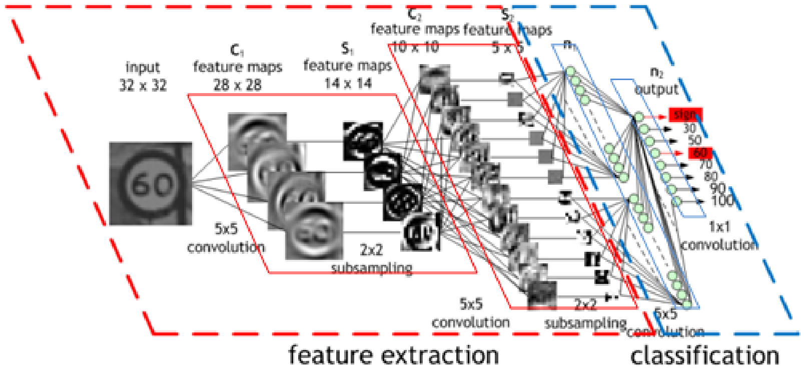
Xlearn Documentation

Release 0.1.1

Argonne National Laboratory

September 30, 2016

Contents



Convolutional Neural Networks for X-ray Science or xlearn is ...

Here is how to add a link to your documentation Docs and here is how to add a reference *[A1]*

CHAPTER 1

Features

- Example of how to write documentation

CHAPTER 2

Contribute

- Documentation: https://github.com/tomography/xlearn/tree/master/doc
- Issue Tracker: https://github.com/tomography/xlearn/docs/issues
- Source Code: https://github.com/tomography/xlearn/project

CHAPTER 3

Content

3.1 About

This section describes what the project is about Project.

3.2 Install

This section covers the basics of how to download and install xlearn.

Contents:

3.2.1 Installing from source

Clone the xlearn from GitHub repository:

```
git clone https://github.com/tomography/xlearn.git xlearn
```

then:

```
cd xlearn
python setup.py install
```

3.3 API reference

xlearn Modules:

3.3.1 `xlearn.transform`

Module containing model, predict and train routines

Functions:

model(dim_img, nb_filters, nb_conv)	the cnn model for image transformation
train(img_x, img_y, patch_size, patch_step, ...)	Function description.
predict(mdl, img, patch_size, patch_step, ...)	the cnn model for image transformation

xlearn.transform.**model** (*dim_img*, *nb_filters*, *nb_conv*)
the cnn model for image transformation

Parameters

- **dim_img** (*int*) – The input image dimension
- **nb_filters** (*int*) – Number of filters
- **nb_conv** (*int*) – The convolution weight dimension

Returns *mdl* – Description.

xlearn.transform.**train** (*img_x*, *img_y*, *patch_size*, *patch_step*, *dim_img*, *nb_filters*, *nb_conv*, *batch_size*, *nb_epoch*)
Function description.

Parameters

- **parameter_01** (*type*) – Description.
- **parameter_02** (*type*) – Description.
- **parameter_03** (*type*) – Description.

Returns *return_01* – Description.

xlearn.transform.**predict** (*mdl*, *img*, *patch_size*, *patch_step*, *batch_size*, *dim_img*)
the cnn model for image transformation

Parameters

- **img** (*array*) – The image need to be calculated
- **patch_size** (*(int, int)*) – The patches dimension
- **dim_img** (*int*) – The input image dimension

Returns *img_rec* – Description.

3.3.2 xlearn.classify

Module containing model, predict and train routines

Functions:

model(dim_img, nb_filters, nb_conv, nb_classes)	the cnn model for image transformation
train(x_train, y_train, x_test, y_test, ...)	Function description.

xlearn.classify.**model** (*dim_img*, *nb_filters*, *nb_conv*, *nb_classes*)
the cnn model for image transformation

Parameters

- **dim_img** (*int*) – The input image dimension

- **nb_filters** (*int*) – Number of filters
- **nb_conv** (*int*) – The convolution weight dimension

Returns *mdl* – Description.

xlearn.classify.**train**(*x_train*, *y_train*, *x_test*, *y_test*, *dim_img*, *nb_filters*, *nb_conv*, *batch_size*, *nb_epoch*, *nb_classes*)

Function description.

Parameters

- **parameter_01** (*type*) – Description.
- **parameter_02** (*type*) – Description.
- **parameter_03** (*type*) – Description.

Returns *return_01* – Description.

3.3.3 xlearn.utils

Module containing utility routines

Functions:

nor_data(img)	Normalize the image
check_random_state(seed)	Turn seed into a np.random.RandomState instance If seed is None, return th
extract_patches(image, patch_size, step[, ...])	Reshape a 2D image into a collection of patches The resulting patches are a
reconstruct_patches(patches, image_size, step)	Reconstruct the image from all of its patches.
img_window(img, window_size)	Function Description
extract_3d(img, patch_size, step)	Function Description

xlearn.utils.**nor_data**(*img*)

Normalize the image

Parameters **img** (*array*) – The images need to be normalized

Returns *img* – Description.

xlearn.utils.**check_random_state**(*seed*)

Turn seed into a np.random.RandomState instance If seed is None, return the RandomState singleton used by np.random. If seed is an int, return a new RandomState instance seeded with seed. If seed is already a RandomState instance, return it. Otherwise raise ValueError.

Parameters **seed** (*type*) – Description.

xlearn.utils.**extract_patches**(*image*, *patch_size*, *step*, *max_patches=None*, *random_state=None*)

Reshape a 2D image into a collection of patches The resulting patches are allocated in a dedicated array.

Parameters

- **image** (*array, shape = (image_height, image_width) or*) – (image_height, image_width, n_channels) The original image data. For color images, the last dimension specifies the channel: a RGB image would have *n_channels=3*.
- **patch_size** (*tuple of ints (patch_height, patch_width)*) – the dimensions of one patch
- **step** (*number of pixels between two patches*)

- **max_patches** (*integer or float, optional default is None*) – The maximum number of patches to extract. If max_patches is a float between 0 and 1, it is taken to be a proportion of the total number of patches.
- **random_state** (*int or RandomState*) – Pseudo number generator state used for random sampling to use if *max_patches* is not None.

Returns patches (*array, shape = (n_patches, patch_height, patch_width) or*) – (n_patches, patch_height, patch_width, n_channels) The collection of patches extracted from the image, where *n_patches* is either *max_patches* or the total number of patches that can be extracted.

`xlearn.utils.`**`reconstruct_patches`**(*patches*, *image_size*, *step*)

Reconstruct the image from all of its patches. Patches are assumed to overlap and the image is constructed by filling in the patches from left to right, top to bottom, averaging the overlapping regions.

Parameters

- **patches** (*array, shape = (n_patches, patch_height, patch_width) or*) – (n_patches, patch_height, patch_width, n_channels) The complete set of patches. If the patches contain colour information, channels are indexed along the last dimension: RGB patches would have *n_channels=3*.
- **image_size** (*tuple of ints (image_height, image_width) or*) – (image_height, image_width, n_channels) the size of the image that will be reconstructed
- **step** (*number of pixels between two patches*)

Returns image (*array, shape = image_size*) – the reconstructed image

`xlearn.utils.`**`img_window`**(*img*, *window_size*)

Function Description

Parameters

- **img** (*define img*)
- **window_size** (*describe window_size*)

Returns img_wd (*describe img_wd*)

`xlearn.utils.`**`extract_3d`**(*img*, *patch_size*, *step*)

Function Description

Parameters

- **img** (*define img*)
- **patch_size** (*describe patch_size*)
- **step** (*describe step*)

Returns patches (*describe patches*)

3.4 Examples

This section contains Jupyter Notebooks and Python scripts examples for various tomoPy functions.

To run these examples in a notebooks install Jupyter or run the python scripts from here

3.4.1 Transform

Train

Here is an example on how to train a convolutional neural network to segment an image. The network is trained using one raw image and one that has been manually segmented.

Once the training is complete the network will be able to automatically segment a series of raw images.

You can download the python scritp `here` or the Jupyter notebook `here`

```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
import dxchange
```

Image data I/O in xlearn is supported by DXchange.

```
import matplotlib.pyplot as plt
```

matplotlib provide plotting of the result in this notebook.

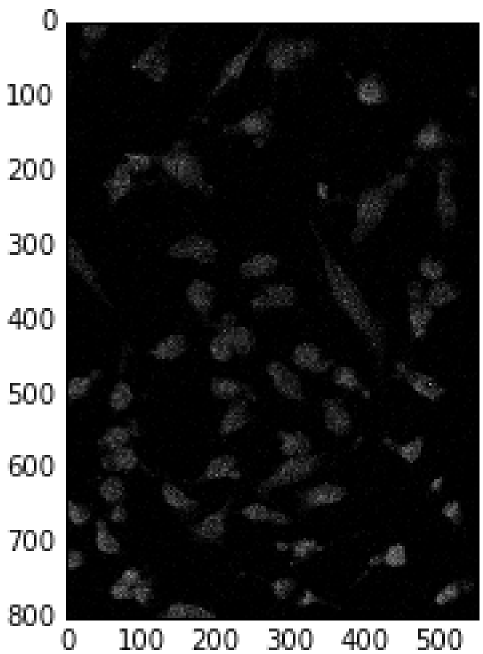
Install xlearn then:

```
from xlearn.transform import train
from xlearn.transform import model
```

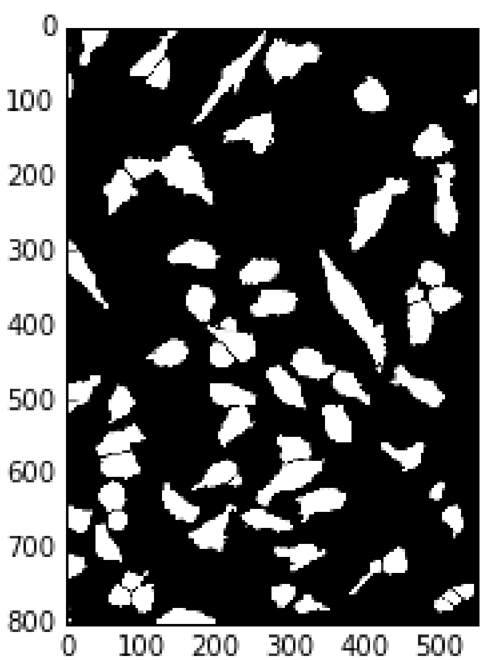
```
batch_size = 800
nb_epoch = 10
dim_img = 20
nb_filters = 32
nb_conv = 3
patch_step = 4
patch_size = (dim_img, dim_img)
```

```
img_x = dxchange.read_tiff('../../test/test_data/training_input.tiff')
img_y = dxchange.read_tiff('../../test/test_data/training_output.tiff')
```

```
plt.imshow(img_x, cmap='Greys_r')
plt.show()
```




```
plt.imshow(img_y, cmap='Greys_r')
plt.show()
```




```
mdl = train(img_x, img_y, patch_size, patch_step, dim_img, nb_filters, nb_conv, batch_size, nb_epoch)
mdl.save_weights('training_weights.h5')
```

```
Epoch 1/10
26068/26068 [==============================] - 39s - loss: 0.4458
Epoch 2/10
26068/26068 [==============================] - 39s - loss: 0.2074
```

```
Epoch 3/10
26068/26068 [==============================] - 39s - loss: 0.1607
Epoch 4/10
26068/26068 [==============================] - 39s - loss: 0.1428
Epoch 5/10
26068/26068 [==============================] - 39s - loss: 0.1321
Epoch 6/10
26068/26068 [==============================] - 39s - loss: 0.1258
Epoch 7/10
26068/26068 [==============================] - 39s - loss: 0.1244
Epoch 8/10
26068/26068 [==============================] - 39s - loss: 0.1169
Epoch 9/10
26068/26068 [==============================] - 39s - loss: 0.1135
Epoch 10/10
26068/26068 [==============================] - 39s - loss: 0.1106
```

Predict

Here is an example on how to use an already trained convolutional neural network to automatically segment a series of raw images.

You can download the python scritp `here` or the Jupyter notebook `here`

```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
import dxchange
```

Image data I/O in xlearn is supported by DXchange.

```
import matplotlib.pyplot as plt
```

matplotlib provide plotting of the result in this notebook.

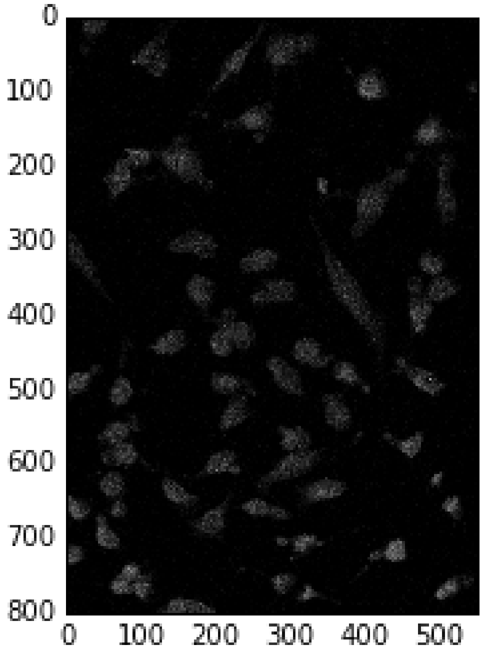
Install xlearn then:

```
from xlearn.transform import model
from xlearn.transform import predict
```

```
batch_size = 800
nb_epoch = 40
dim_img = 20
nb_filters = 32
nb_conv = 3
patch_step = 4

patch_size = (dim_img, dim_img)
```

```
mdl = model(dim_img, nb_filters, nb_conv)
mdl.load_weights('training_weights.h5')
```

```
fname = '../../test/test_data/predict_test.tiff'
img_test = dxchange.read_tiff(fname)
plt.imshow(img_test, cmap='Greys_r')
plt.show()
```

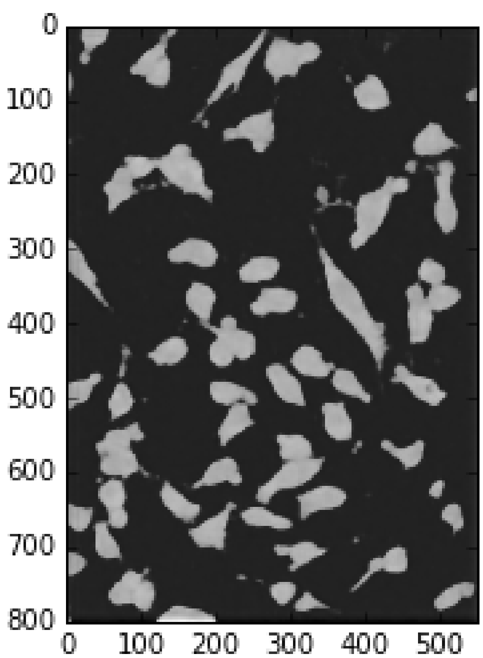



```
fname_save = '../../test/test_data/predict_test_result'
```

```
img_rec = predict(mdl, img_test, patch_size, patch_step, batch_size, dim_img)
```

```
dxchange.write_tiff(img_rec, fname_save, dtype='float32')
```

```
plt.imshow(img_rec, cmap='Greys_r')
plt.show()
```



3.4.2 Classify

Train

Here is an example on how to train a convolutional neural network to identify a tomographic reconstructed image that has the best center.

The network is trained using one image off center and the best centered reconstruction. Once the training is complete the network will be able to evaulate a series of reconstructed images with different rotation center and select the one with the best center.

You can download the python scritp `here` or the Jupyter notebook `here`

To run this example please download the test data from the classify_train folder at url

```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
import dxchange
import numpy as np
from xlearn.utils import nor_data
from xlearn.utils import extract_3d
from xlearn.utils import img_window
from xlearn.classify import train
```

```
Using Theano backend.
Using gpu device 0: Tesla M2050 (CNMeM is disabled, cuDNN not available)
```

```
np.random.seed(1337)
dim_img = 128
patch_size = (dim_img, dim_img)
batch_size = 50
nb_classes = 2
nb_epoch = 12
```

number of convolutional filters to use

```
nb_filters = 32
```

size of pooling area for max pooling

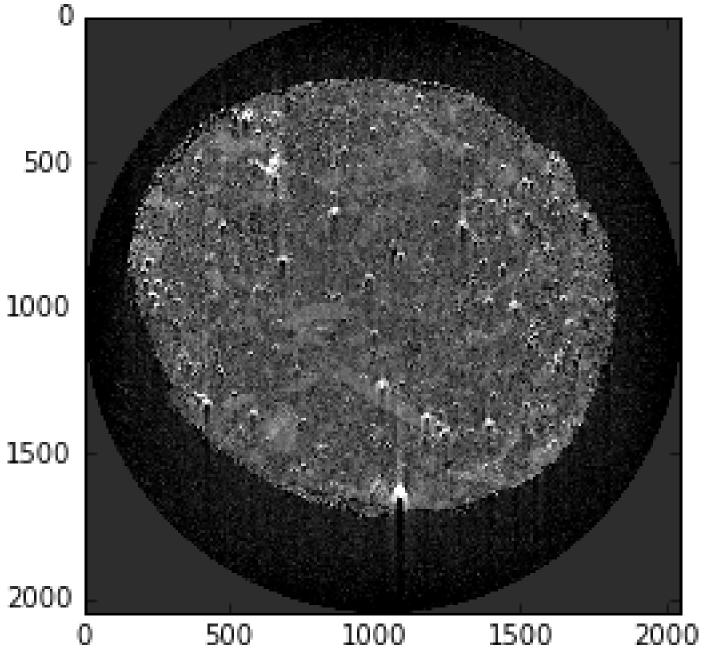
```
nb_pool = 2
```

convolution kernel size

```
nb_conv = 3
```

```
fname = '../../test/test_data/1038.tiff'
img_x = dxchange.read_tiff(fname)
```

```
plt.imshow(img_x, cmap='Greys_r')
plt.clim(-0.0005,0.0028)
plt.show()
```




```
ind_uncenter1 = range(1038, 1047)
ind_uncenter2 = range(1049, 1057)
uncenter1 = dxchange.read_tiff_stack(fname, ind=ind_uncenter1, digit=4)
uncenter2 = dxchange.read_tiff_stack(fname, ind=ind_uncenter2, digit=4)
uncenter = np.concatenate((uncenter1, uncenter2), axis=0)
uncenter = nor_data(uncenter)
```

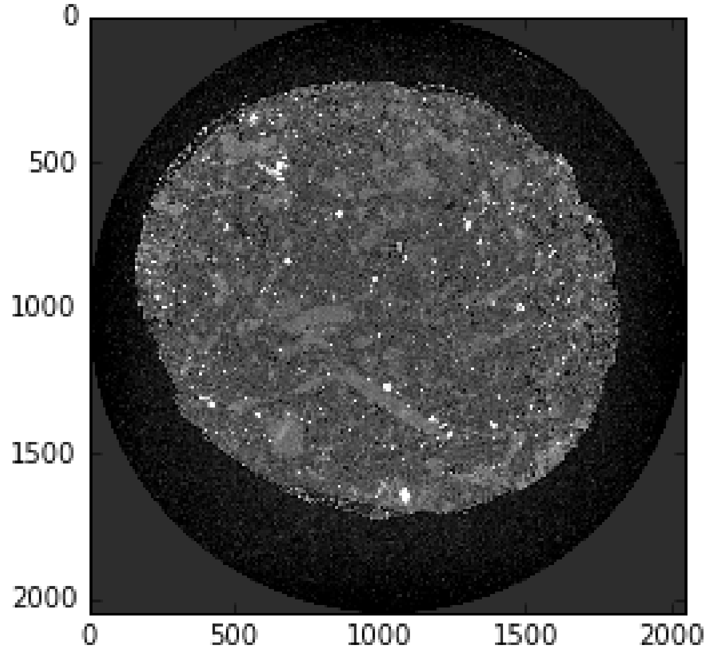
```
uncenter = img_window(uncenter[:, 360:1460, 440:1440], 200)
```

```
uncenter_patches = extract_3d(uncenter, patch_size, 1)
```

```
np.random.shuffle(uncenter_patches)
```

```
center_img = dxchange.read_tiff('../../test/test_data/1048.tiff')
```

```
plt.imshow(center_img, cmap='Greys_r')
plt.clim(-0.0005,0.0028)
plt.show()
```




```
center_img = nor_data(center_img)
```

```
center_img = img_window(center_img[360:1460, 440:1440], 400)
center_patches = extract_3d(center_img, patch_size, 1)
np.random.shuffle(center_patches)
```

```
x_train = np.concatenate((uncenter_patches[0:50000], center_patches[0:50000]), axis=0)
x_test = np.concatenate((uncenter_patches[50000:60000], center_patches[50000:60000]), axis=0)
x_train = x_train.reshape(x_train.shape[0], 1, dim_img, dim_img)
x_test = x_test.reshape(x_test.shape[0], 1, dim_img, dim_img)
y_train = np.zeros(100000)
y_train[50000:99999] = 1
y_test = np.zeros(20000)
y_test[10000:19999] = 1
```

```
model = train(x_train, y_train, x_test, y_test, dim_img, nb_filters, nb_conv, batch_size, nb_epoch, n
```

```
(100000, 1, 128, 128) (100000, 2) (20000, 1, 128, 128) (20000, 2)
Train on 100000 samples, validate on 20000 samples
Epoch 1/12
100000/100000 [==============================] - 836s - loss: 0.1251 - acc: 0.9604 - val_loss: 0.0726
Epoch 2/12
100000/100000 [==============================] - 835s - loss: 0.0085 - acc: 0.9977 - val_loss: 0.1675
Epoch 3/12
100000/100000 [==============================] - 835s - loss: 0.0045 - acc: 0.9989 - val_loss: 0.0155
Epoch 4/12
100000/100000 [==============================] - 832s - loss: 0.0034 - acc: 0.9990 - val_loss: 0.0090
Epoch 5/12
100000/100000 [==============================] - 834s - loss: 0.0018 - acc: 0.9995 - val_loss: 0.1212
Epoch 6/12
100000/100000 [==============================] - 835s - loss: 9.9921e-04 - acc: 0.9998 - val_loss: 0
Epoch 7/12
100000/100000 [==============================] - 835s - loss: 5.3466e-04 - acc: 0.9999 - val_loss: 6
Epoch 8/12
```

```
100000/100000 [==============================] - 836s - loss: 7.6305e-04 - acc: 0.9998 - val_loss: 0
Epoch 9/12
100000/100000 [==============================] - 833s - loss: 3.9566e-04 - acc: 0.9999 - val_loss: 8
Epoch 10/12
100000/100000 [==============================] - 835s - loss: 4.5675e-04 - acc: 0.9999 - val_loss: 8
Epoch 11/12
100000/100000 [==============================] - 833s - loss: 3.1511e-04 - acc: 1.0000 - val_loss: 8
Epoch 12/12
100000/100000 [==============================] - 833s - loss: 2.0671e-04 - acc: 1.0000 - val_loss: 8
```

```
Test score: 0.000806061122949
Test accuracy: 0.99995
```

```
model.save_weights('classify_training_weights.h5')
```

Evaluate

Here is an example on how to use an already trained convolutional neural network to evaluate and select the best image according to the training received. In this example the network has been trained to select the best rotation axis centered reconstruction. The test consists of asking the network to select the best centered images coming from a similar sample collected on a different tomographic beamline.

You can download the python scritp `here` or the Jupyter notebook `here`

To run this example please download the test data from the classify_evaluate folder at url

```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
import dxchange
import numpy as np
from xlearn.utils import nor_data
from xlearn.utils import extract_3d
from xlearn.utils import img_window
from xlearn.classify import model
import matplotlib.pyplot as plt
import time
import glob
```

```
Using Theano backend.
Using gpu device 0: Tesla M2050 (CNMeM is disabled, cuDNN not available)
```

```
np.random.seed(1337)

dim_img = 128
patch_size = (dim_img, dim_img)
batch_size = 50
nb_classes = 2
nb_epoch = 12
```

number of convolutional filters to use

```
nb_filters = 32
```

size of pooling area for max pooling

```
nb_pool = 2
```

convolution kernel size

```
nb_conv = 3
```

Please download the test data from the classify_evaluate folder at

http://tinyurl.com/APS-xlearn

and put them in the test_data folder

```
nb_evl = 100
```

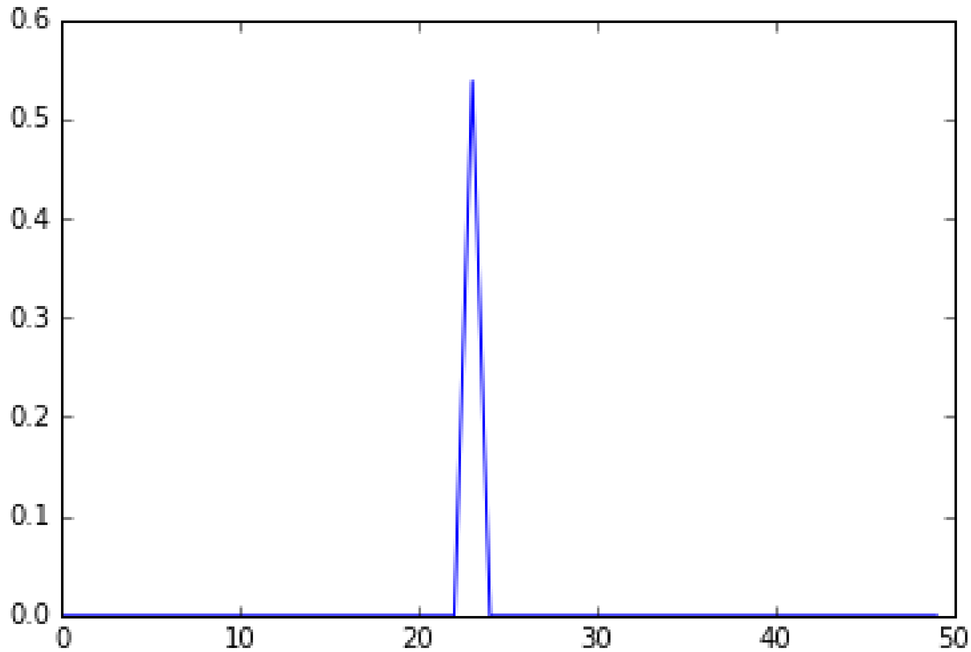
```
fnames = glob.glob('../../test/test_data/*.tiff')
fnames = np.sort(fnames)
```

```
mdl = model(dim_img, nb_filters, nb_conv, nb_classes)

mdl.load_weights('classify_training_weights.h5')

Y_score = np.zeros((len(fnames)))
```

```
for i in range(len(fnames)):
    img = dxchange.read_tiff(fnames[i])
    img = nor_data(img)
    X_evl = np.zeros((nb_evl, dim_img, dim_img))

    for j in range(nb_evl):
        X_evl[j] = img_window(img[360:1460, 440:1440], dim_img)
    X_evl = X_evl.reshape(X_evl.shape[0], 1, dim_img, dim_img)
    Y_evl = mdl.predict(X_evl, batch_size=batch_size)
    Y_score[i] = sum(np.dot(Y_evl, [0, 1]))
```

```
ind_max = np.argmax(Y_score)
print('The well-centered reconstruction is:', fnames[ind_max])
plt.plot(Y_score)
plt.show()
```

```
('The well-centered reconstruction is:', '../../test/test_data/1023.00.tiff')
```



3.5 Credits

3.5.1 Citations

We kindly request that you cite the following article *[A1]* if you use project.

3.5.2 References

[A1] De Carlo F, Gursoy D, Marone F, Rivers M, Parkinson YD, Khan F, Schwarz N, Vine DJ, Vogt S, Gleber SC, Narayanan S, Newville M, Lanzirotti T, Sun Y, Hong YP, and Jacobsen C. Scientific data exchange: a schema for hdf5-based storage of raw and analyzed data. *Journal of Synchrotron Radiation*, 21(6):1224–1230, 2014.

[B1] De Carlo F, Gursoy D, Marone F, Rivers M, Parkinson YD, Khan F, Schwarz N, Vine DJ, Vogt S, Gleber SC, Narayanan S, Newville M, Lanzirotti T, Sun Y, Hong YP, and Jacobsen C. Scientific data exchange: a schema for hdf5-based storage of raw and analyzed data. *Journal of Synchrotron Radiation*, 21(6):1224–1230, 2014.

Python Module Index

x

C

E

I

M

N

P

R

T

X