Xlearn Documentation

Release 0.2.0

Argonne National Laboratory

Jul 10, 2018

Contents

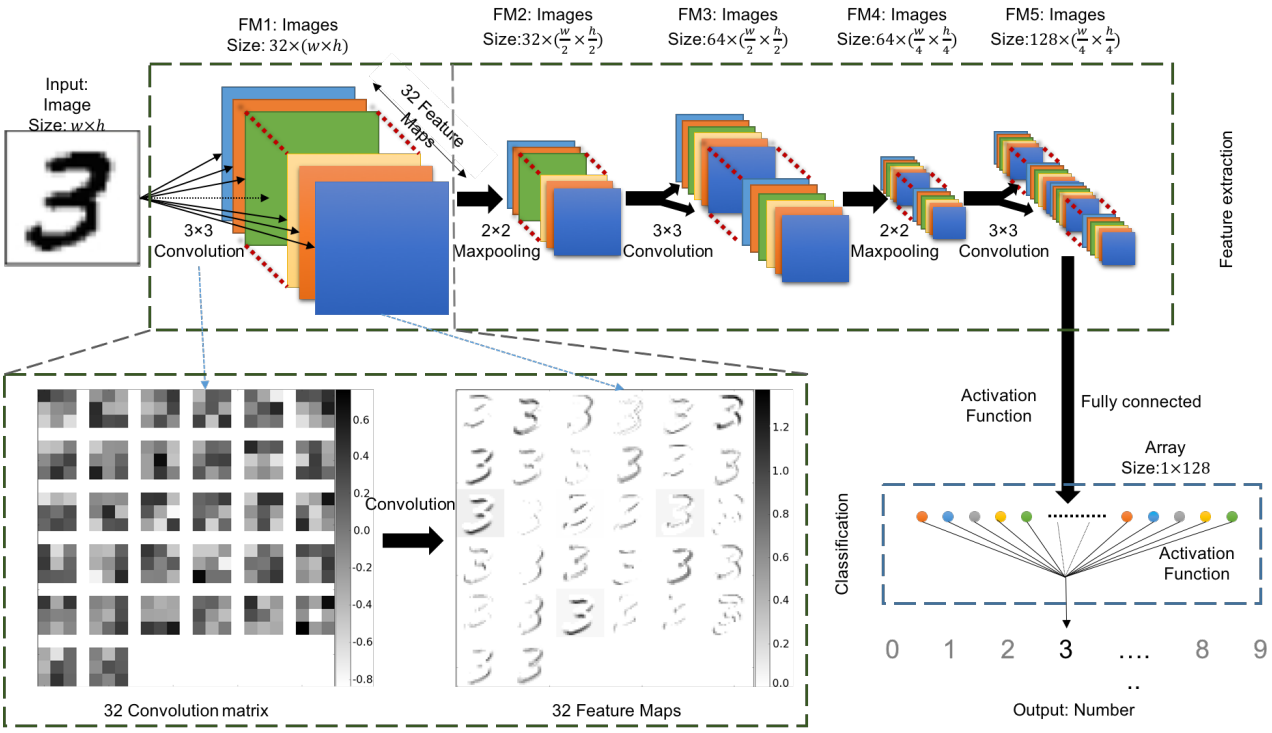
1 Features 3

2 Contribute 5

3 Content 7

Bibliography 25

Python Module Index 31



Xlearn is an open-source Python package implementing Convolutional Neural Networks (CNN) for X-ray Science.

CHAPTER 1

Features

- Correction of instrument and beam instability artifacts
- Low-dose image enhancement
- Feature extraction, segmentation
- Super-resolution X-ray microscopy

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/tree/master/xlearn

CHAPTER 3

Content

3.1 About

Xlearn implements Deep Neural Network to X-ray science imaging problems including:

- Tomography rotation axis calibration *[A3]*.
- Low-dose image enhancement *[A2]*.
- Feature extraction, segmentation *[A1]*.
- Super-resolution X-ray microscopy (will update later).
- Solving inverse problems with deep learning, such as tomography reconstruction and phase retrieval (will update late).

3.2 Install

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

3.2.1 Installing from source

To install xlearn follow these steps:

1. Install anaconda
2. Install tensorflow. Please install the tensorflow-gpu version with pip. Before the tensorflow-gpu installation, make sure that the cuda drivers and cudnn are correctly installed to your OS.
3. Install the Xlearn toolbox: 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.segmentation

Module containing model_choose, seg_train and seg_predict routines

Functions:

seg_train(img_x, img_y[, patch_size, ...])	Function description.
seg_predict(img, wpath, spath[, patch_size, ...])	Function description

xlearn.segmentation.**model_choose**(*ih*, *iw*, *nb_conv*, *size_conv*, *nb_down*, *nb_gpu*)

xlearn.segmentation.**seg_train**(*img_x*, *img_y*, *patch_size=32*, *patch_step=1*, *nb_conv=32*, *size_conv=3*, *batch_size=1000*, *nb_epoch=20*, *nb_down=2*, *nb_gpu=1*)

Function description.

Parameters

- **img_x** (*array, 2D or 3D*) – Training input of the model. It is the raw image for the segmentation.

- **img_y** (*array, 2D or 3D*) – Training output of the model. It is the corresponding segmentation of the training input.
- **patch_size** (*int*) – The size of the small patches extracted from the input images. This size should be big enough to cover the features of the segmentation object.
- **patch_step** (*int*) – The pixel steps between neighbour patches. Larger steps leads faster speed, but less quality. I recommend 1 unless you need quick test of the algorithm.
- **nb_conv** (*int*) – Number of the covolutional kernals for the first layer. This number doubles after each downsampling layer.
- **size_conv** (*int*) – Size of the convolutional kernals.
- **batch_size** (*int*) – Batch size for the training. Bigger size leads faster speed. However, it is restricted by the memory size of the GPU. If the user got the memory error, please decrease the batch size.
- **nb_epoch** (*int*) – Number of the epoches for the training. It can be understand as the number of iterations during the training. Please define this number as the actual convergence for different data.
- **nb_down** (*int*) – Number of the downsampling for the images in the model.
- **nb_gpu** (*int*) – Number of GPUs you want to use for the training.

Returns *mdl* – The trained CNN model for segmenation. The model can be saved for future segmentations.

xlearn.segmentation.**seg_predict**(*img*, *wpath*, *spath*, *patch_size=32*, *patch_step=1*, *nb_conv=32*, *size_conv=3*, *batch_size=1000*, *nb_down=2*, *nb_gpu=1*)

Function description

Parameters

- **img** (*array*) – The images need to be segmented.
- **wpath** (*string*) – The path where the trained weights of the model can be read.
- **spath** (*string*) – The path to save the segmented images.
- **patch_size** (*int*) – The size of the small patches extracted from the input images. This size should be big enough to cover the features of the segmentation object.
- **patch_step** (*int*) – The pixel steps between neighbour patches. Larger steps leads faster speed, but less quality. I recommend 1 unless you need quick test of the algorithm.
- **nb_conv** (*int*) – Number of the covolutional kernals for the first layer. This number doubles after each downsampling layer.
- **size_conv** (*int*) – Size of the convolutional kernals.
- **batch_size** (*int*) – Batch size for the training. Bigger size leads faster speed. However, it is restricted by the memory size of the GPU. If the user got the memory error, please decrease the batch size.
- **nb_epoch** (*int*) – Number of the epoches for the training. It can be understand as the number of iterations during the training. Please define this number as the actual convergence for different data.
- **nb_down** (*int*) – Number of the downsampling for the images in the model.
- **nb_gpu** (*int*) – Number of GPUs you want to use for the training.

Returns *save the segmented images to the spath.*

3.3.4 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 the RandomState singleton used by np.random.
extract_patches(image, patch_size, step[, ...])	Reshape a 2D image into a collection of patches The resulting patches are allocated in a dedicated array.
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.**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.**expimg**(*img*)

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*)

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.**img_window**(*img*, *window_size*)
Function Description

Parameters

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

Returns **img_wd** (*describe img_wd*)

xlearn.utils.**mlog**(*img*)

xlearn.utils.**nor_data**(*img*)
Normalize the image

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

Returns *img* – Description.

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.**rescale_intensity**(*img*)

3.4 Introduction

Machine learning is a robust solution to mimic human's estimations and predictions for complex data problems *[B46]*. Convolutional neural network (CNN) *[B27]* is an efficient and universal algorithm in the family of machine learning for image processing. It processes the feature of image pattern, rather than the value of each pixel as with classical methods. Its accuracy and efficiency for image recognition and classification have been proved from various applications *[B36]*, *[B26]*, *[B14]*, *[B33]*. The specially designed architecture of CNN also works for supervised transformation of image style *[B15]*.

Convolutional neural network (CNN) *[B27]* is a branch of Artificial Neural Network (ANN), which is also known as Multilayer Perceptron (MLP). As it allows the computational model to learn representations of data with multiple levels of abstraction, it also belongs to the popular families of "Deep learning" *[B29]*. As the same idea of ANN and "Deep learning", CNN is the process to build a function f between the input data X and output data Y ($f: X -> Y$). The function f does not, like the traditional formulas, represent the data relations with simple mathematical operators and symbols. It is a composition of multiple layers of weights (W) with activation functions (K). The CNNs we present for this article are supervised learning, which we fit the W for specific input and output data ($X -> Y$). After that, this f can be used to predict the future data with the same rule of the fitting data.

3.4.1 Transform

Inspired from the CNN classification model, we developed an image transformation model of CNN. The CNN classification model extracts the features of input image to build a link between the input image and output label. Our image transformation model extracts the features of the input image to build the link between the input image and output image, which has close feature of the input, but with different style.

The architecture of the transformation model is as shown in Figure 2.

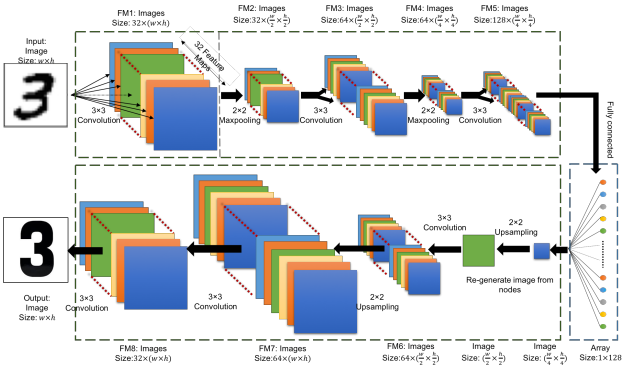


Figure 2: The architecture of the CNN transformation model using for this article. We use the transformation from handwriting number 3 to standard shape of 3 as the example.

Half architecture of the transformation model is the same as the classification model. The difference is that we do not connect the nodes of the fully connected layer to a single number. Instead, we transform it to an image with the same image size of the previous convolutional layer. After that, we keep using the upsampling layer and convolutional to convert the feature maps back to a single image. We use this image as the output data and fit the whole network from the input image. Once the network is trained between specific input and output image, we can transform the future images as the same rule of the training data.

3.4.2 Classify

CNN was originally developed for image classifications. Its basic and most popular applications are hand-writing recognition and human face recognition. In these cases, CNN plays the role as a fitting function between the input images and the output labels. The process to fit the CNN model is so-called "train". The iterations during the "train" are called "epochs". Typically a Stochastic Gradient Descent (SGD) is used for training. Once the CNN is trained for a specific data model, we can use it as the function to estimate the label of an unknown image containing features that are close to the training data. This step is called "predict".

The process to prepare the training data decides the computing model. The more training data we prepare, the better the prediction results will be. Normally the number of training data should be at least on the magnitude of 10^4 for a reasonable prediction. This procedure is always considered difficult, because most of the steps for this task have to be done manually. in some cases, like solving a general image classification problems for nature images, this can be an overwhelming task and explains why machine learning techniques are yet not widely applied. However, for the image classification problems of synchrotron imaging, the image features are normally restricted to some specific aspect and therefore, we only need to use few images, or, in some cases, as discussed this in Section 3, even use the simulated images, to train the CNN model.

There is not a standard architecture of CNN for image classification. After we tested different architectures and parameter to consider their performances and stability, we choose to use the CNN architecture as shown in Fiure 1:

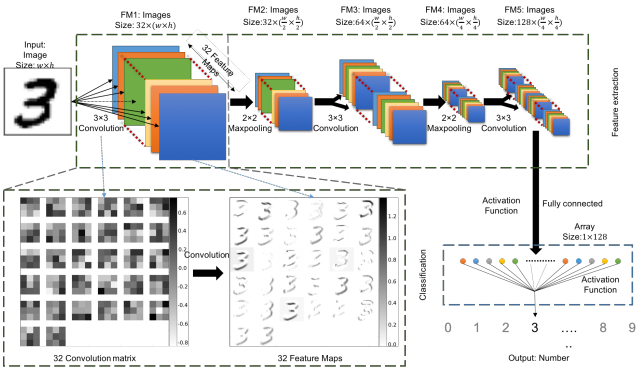


Figure 1: The architecture of the CNN classification model using for this article. We use the classification of handwriting number 3 as the example. This diagram shows how the handwriting image has been classified as its related number.

It includes 3 convolutional layers and 2 maxpooling layers. The first convolutional layer includes 32 convolution weights to extract 32 feature maps from the inputs. The image size reduces to half at each maxpooling layer. The number of convolution weights and feature maps doubles after each maxpooling layer. The final layer of the feature maps are fully connected to data nodes with the activation function. These nodes are connected again with another activation function and become a single value. We fit this value to be the label that defined in the training data.

3.5 Examples

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

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

3.5.1 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`

```
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.

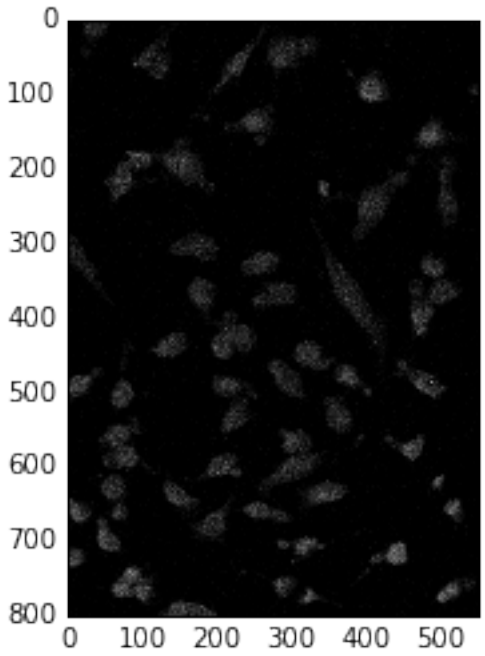
Install xlearn then:

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

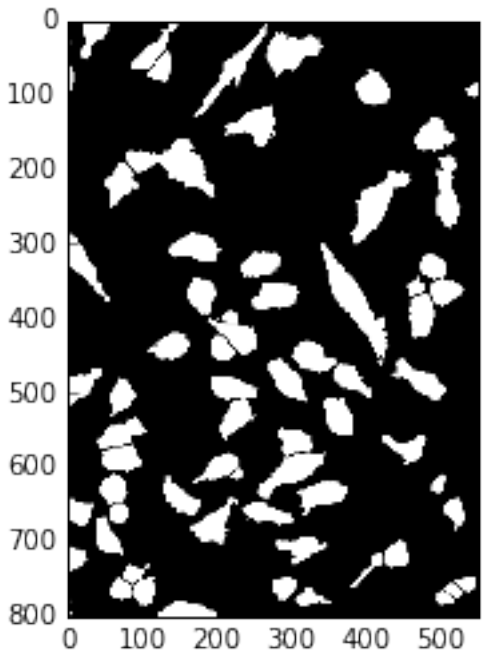
```
batch_size = 800
nb_epoch = 10
dim_img = 20
nb_filters = 32
nb_conv = 3
patch_step = 4
patch_size = (dim_img, dim_img)
```

```
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`

```
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.

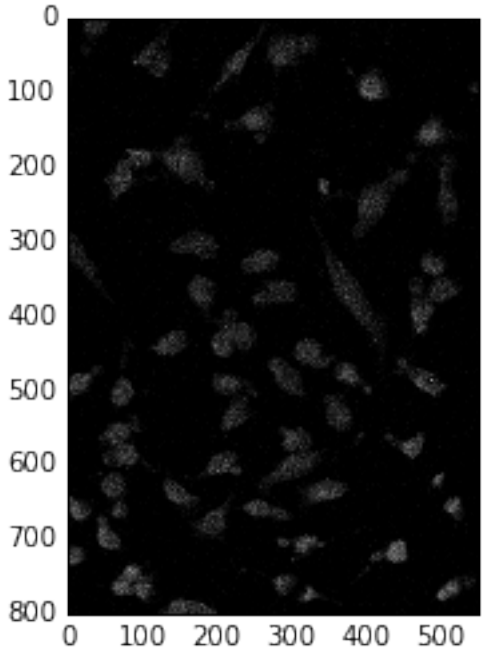
Install xlearn then:

```
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()
```

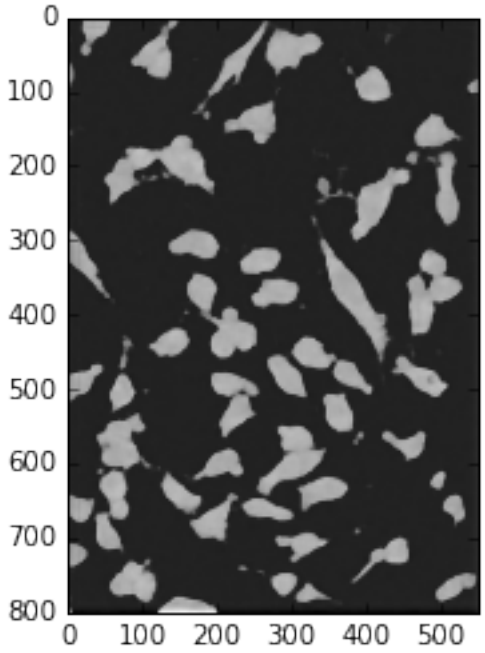



```
fname_save = '../../test/test_data/predict_test_result'
```

```
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()
```



3.5.2 Rotation Center

An experienced beam line scientist can easily distinguish the well-centered and off-centered reconstructions directly by eyes without any mathematical calculation. This is always the most accurate way to evaluate the results for a final step. However, this approach is not applicable for large data sets because it costs too much effort. Here we use, instead, the classification model of CNN to mimic this process of the human's brain. Thus an automatic routine to compute the tomographic rotation axis is developed.

The rotation axis problem can be considered as an image classification problem, because there are significant different features between a well-center and an off-centered reconstructions. Once the trained CNN can accurately recognize the well-centered reconstruction from the off-centered reconstructions, we can use it as the principle to automatically finding the correct rotation center.

We use here the same classification model described in the CNN introduction, which requires two steps to evaluate the data: train the CNN model, and predict the data with trained model. We developed a special method to prepare the training data sets and to process the prediction.

In the procedure of preparing the training data, we reconstruct a slice of tomographic image with different rotation center. A group of reconstruction results are obtained. During the training phase, we select the well-centered reconstruction by eyes, and label the rest as off-centered reconstruction. We extract overlapped patches from the well-centered image and label these patches as 1, and also from the off-centered images and label them as 0.

A patch is a square window (s_p X s_p) from the image. The patches are overlapped one by one. The distance of the

center between two neighbor patches is the patch steps (*n_s*). The patch number is

$$N_p = \frac{1}{n_s^2}(h - s_p) \cdot (w - s_p)$$

for a image with height (*h*) and width (*w*). There are two reasons for using small patches instead of the whole image:

- We can generate enough train data only from one single image.
- The overlapped patches provide multiple evaluations of the same feature of the image.

Once we have extracted the patches from the well-centered and off-centered images, we select specific number (*P_{train}*) of patches with their labels (*Y_l*) from both of these groups. We use the patches as input *X_{train}* and the labels as output to train the CNN model. The trained CNN classification model is now capable to distinguish the well-centered or off-centered patches.

The prediction procedure evaluates the *Y_l* of the patches from the reconstructions of different rotation axis. We first do tomographic reconstruction with different rotation axis. For each reconstructed images, we extract specific number of patches. The size of the patches should be the same as the training data. The number of the patches can be roughly in the hundreds. We use these patches as the input data for trained CNN and evaluate their label. If the value of the label is close to 1, it means the feature of the patch is close to well-centered. If it is 0, it is off-centered. We computer the summation (*S_l*) of the labels for the patches from one reconstruction. The reconstruction with the maximum *S_l* is the one well-centered as our evaluation model.

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

```
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

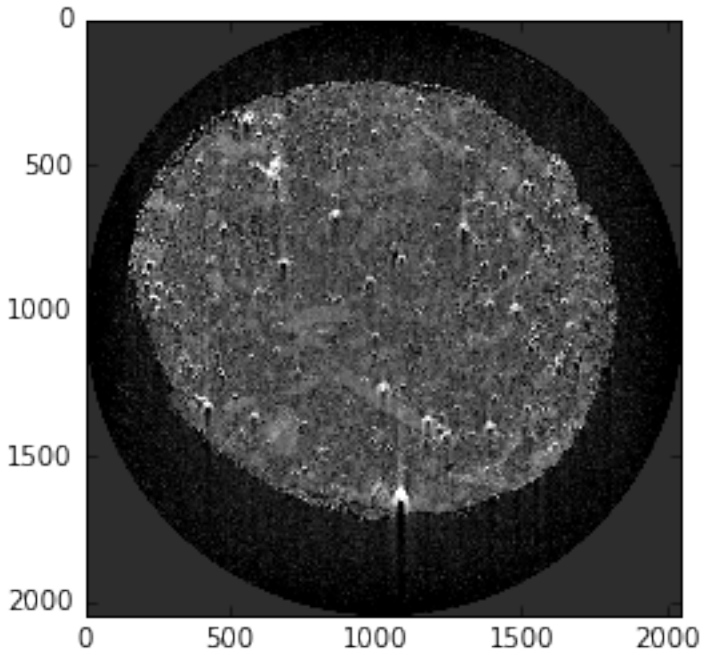
```
nb_pool = 2
```

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)
```

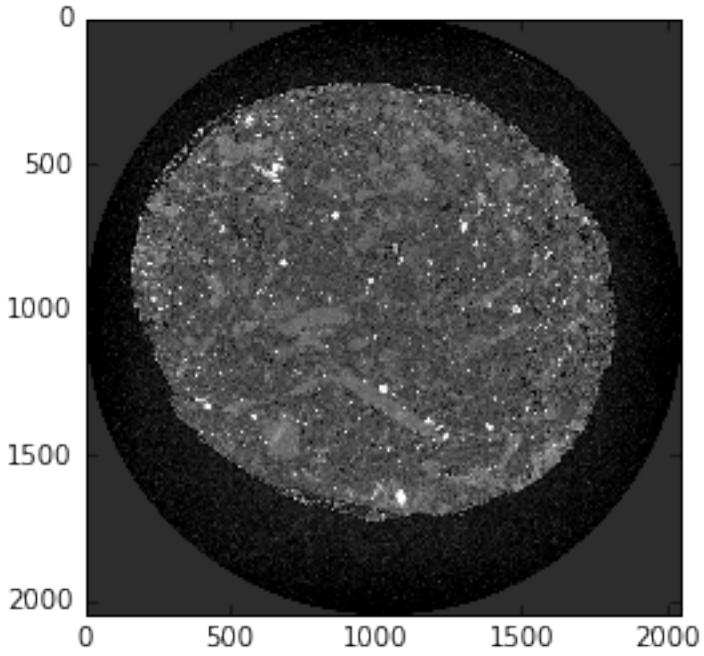
```
uncenter = img_window(uncenter[:, 360:1460, 440:1440], 200)
```

```
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, nb_classes)
```

```
(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 - val_acc: 0.9704
Epoch 2/12
100000/100000 [==============================] - 835s - loss: 0.0085 - acc: 0.9977 -␣
↪val_loss: 0.1675 - val_acc: 0.9311
Epoch 3/12
100000/100000 [==============================] - 835s - loss: 0.0045 - acc: 0.9989 -␣
↪val_loss: 0.0155 - val_acc: 0.9949
Epoch 4/12
100000/100000 [==============================] - 832s - loss: 0.0034 - acc: 0.9990 -␣
↪val_loss: 0.0090 - val_acc: 0.9976
```

(continues on next page)

(continued from previous page)

```
Epoch 5/12
100000/100000 [==============================] - 834s - loss: 0.0018 - acc: 0.9995 -
↪val_loss: 0.1212 - val_acc: 0.9512
Epoch 6/12
100000/100000 [==============================] - 835s - loss: 9.9921e-04 - acc: 0.
↪9998 - val_loss: 0.0033 - val_acc: 0.9991
Epoch 7/12
100000/100000 [==============================] - 835s - loss: 5.3466e-04 - acc: 0.
↪9999 - val_loss: 6.5040e-04 - val_acc: 1.0000
Epoch 8/12
100000/100000 [==============================] - 836s - loss: 7.6305e-04 - acc: 0.
↪9998 - val_loss: 0.0016 - val_acc: 0.9997
Epoch 9/12
100000/100000 [==============================] - 833s - loss: 3.9566e-04 - acc: 0.
↪9999 - val_loss: 8.2169e-04 - val_acc: 1.0000
Epoch 10/12
100000/100000 [==============================] - 835s - loss: 4.5675e-04 - acc: 0.
↪9999 - val_loss: 8.0605e-04 - val_acc: 1.0000
Epoch 11/12
100000/100000 [==============================] - 833s - loss: 3.1511e-04 - acc: 1.
↪0000 - val_loss: 8.0620e-04 - val_acc: 1.0000
Epoch 12/12
100000/100000 [==============================] - 833s - loss: 2.0671e-04 - acc: 1.
↪0000 - val_loss: 8.0606e-04 - val_acc: 1.0000
```

```
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

```
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
```

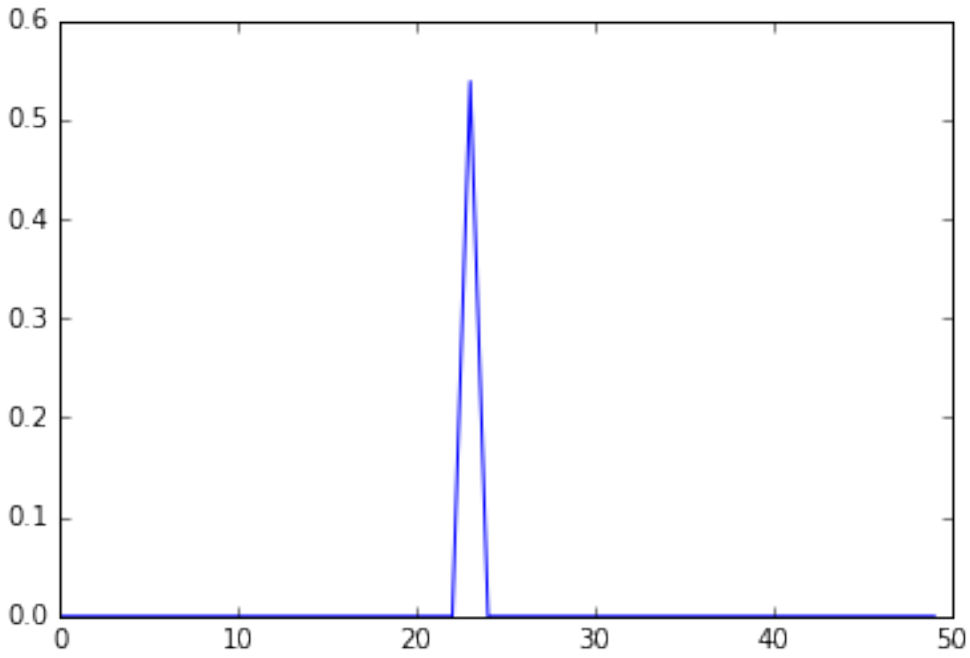
```
fnames = glob.glob('../../test/test_data/*.tiff')
fnames = np.sort(fnames)
```

```
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')
```



3.6 Credits

3.6.1 Citations

We kindly request that you cite the following article *[A2]*, *[A3]*, and *[A1]* if you use Xlearn.

3.6.2 References

Bibliography

[A1] C. Shashank Kaira, Xiaogang Yang, Vincent De Andrade, Francesco De Carlo, William Scullin, Doga Gursoy, and Nikhilesh Chawla. Automated correlative segmentation of large transmission x-ray microscopy (txm) tomograms using deep learning. *Materials Characterization*, 142:203–210, 2018. URL: https://www.sciencedirect.com/science/article/pii/S1044580318301906, doi:https://doi.org/10.1016/j.matchar.2018.05.053.

[A2] Xiaogang Yang, Vincent De Andrade, William Scullin, Eva L. Dyer, Narayanan Kasthuri, Francesco De Carlo, and Doğa Gürsoy. Low-dose x-ray tomography through a deep convolutional neural network. *Scientific Reports*, 8(1):2575, 2018. URL: https://doi.org/10.1038/s41598-018-19426-7https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5803233/pdf/41598_2018_Article_19426.pdf, doi:10.1038/s41598-018-19426-7.

[A3] Xiaogang Yang, Francesco De Carlo, Charudatta Phatak, and Doga Gursoy. A convolutional neural network approach to calibrating the rotation axis for x-ray computed tomography. *Journal of Synchrotron Radiation*, 2017. URL: https://doi.org/10.1107/S1600577516020117http://journals.iucr.org/s/issues/2017/02/00/vv5155/vv5155.pdf, doi:doi:10.1107/S1600577516020117.

[B1] S. G. Azevedo, D. J. Schneberk, J. P. Fitch, and H. E. Martz. Calculation of the rotational centers in computed-tomography sinograms. *Ieee Transactions on Nuclear Science*, 37(4):1525–1540, 1990. doi:10.1109/23.55866.

[B2] I. A. Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, 2000. doi:http://dx.doi.org/10.1016/S0167-7012(00)00201-3.

[B3] Amir Beck and Marc Teboulle. Fast gradient-based algorithms for constrained total variation image denoising and deblurring problems. *Image Processing, IEEE Transactions on*, 18(11):2419–2434, 2009.

[B4] Tekin Biçer, Doga Gürsoy, Rajkumar Kettimuthu, Francesco De Carlo, Gagan Agrawal, and Ian T. Foster. Rapid tomographic image reconstruction via large-scale parallelization. In Jesper Larsson Trff, Sascha Hunold, and Francesco Versaci, editors, *Euro-Par 2015: Parallel Processing*, volume 9233 of Lecture Notes in Computer Science, pages 289–302. Springer Berlin Heidelberg, 2015.

[B5] Folkert Bleichrodt, Tristan Leeuwen, Willem Jan Palenstijn, Wim Aarle, Jan Sijbers, and K. Joost Batenburg. Easy implementation of advanced tomography algorithms using the astra toolbox with spot operators. *Numerical Algorithms*, 71(3):673–697, 2015. doi:10.1007/s11075-015-0016-4.

[B6] Mirko Boin and Astrid Haibel. Compensation of ring artefacts in synchrotron tomographic images. *Optics Express*, 14(25):12071–12075, 2006. doi:10.1364/OE.14.012071.

[B7] Ji-Ho Chang, J. M. M. Anderson, and J. R. Votaw. Regularized image reconstruction algorithms for positron emission tomography. *IEEE Transactions on Medical Imaging*, 23(9):1165–1175, September 2004. doi:10.1109/TMI.2004.831224.

[B8] Francesco De Carlo, Doğa Gürsoy, Federica Marone, Mark Rivers, Dilworth Y. Parkinson, Faisal Khan, Nicholas Schwarz, David J. Vine, Stefan Vogt, Sophie-Charlotte Gleber, Suresh Narayanan, Matt Newville, Tony Lanzirotti, Yue Sun, Young Pyo Hong, and Chris Jacobsen. Scientific data exchange: a schema for HDF5-based storage of raw and analyzed data. *Journal of Synchrotron Radiation*, 21(6):1224–1230, Nov 2014. doi:10.1107/S160057751401604X.

[B9] Alexander H Delaney and Yoram Bresler. Globally convergent edge-preserving regularized reconstruction: an application to limited-angle tomography. *Image Processing, IEEE Transactions on*, 7(2):204–221, 1998.

[B10] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[B11] Tilman Donath, Felix Beckmann, and Andreas Schreyer. Automated determination of the center of rotation in tomography data. *J. Opt. Soc. Am. A*, 23(5):1048–1057, 2006. doi:10.1364/JOSAA.23.001048.

[B12] Betsy A. Dowd, Graham H. Campbell, Robert B. Marr, Vivek V. Nagarkar, Sameer V. Tipnis, Lisa Axe, and D. P. Siddons. Developments in synchrotron x-ray computed microtomography at the national synchrotron light source. In *Developments in X-Ray Tomography II*, volume 3772, 224–236. 1999. doi:10.1117/12.363725.

[B13] Daniel J Duke, Andrew B Swantek, Nicolas M Sovis, F Zak Tilocco, Christopher F Powell, Alan L Kastengren, Doğa Gürsoy, and Tekin Biçer. Time-resolved x-ray tomography of gasoline direct injection sprays. *SAE International Journal of Engines*, 9(1):151–161, 2015.

[B14] C. Garcia and M. Delakis. Convolutional face finder: a neural architecture for fast and robust face detection. *Ieee Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1408–1423, 2004. doi:10.1109/tpami.2004.97.

[B15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2414–2423. June 2016.

[B16] J. W. Gibbs, K. A. Mohan, E. B. Gulsoy, A. J. Shahani, X. Xiao, C. A. Bouman, M. De Graef, and P. W. Voorhees. The three-dimensional morphology of growing dendrites. *Scientific Reports*, 5:11824, 2015.

[B17] Jens Gregor and Thomas Benson. Computational analysis and improvement of sirt. *Medical Imaging, IEEE Transactions on*, 27(7):918–924, 2008.

[B18] Doga Gürsoy, Francesco De Carlo, Xianghui Xiao, and Chris Jacobsen. Tomopy: a framework for the analysis of synchrotron tomographic data. *Journal of synchrotron radiation*, 21(5):1188–1193, 2014.

[B19] Dogaa Gürsoy, Tekin Biçer, Jonathan D Almer, Raj Kettimuthu, Stuart R Stock, and Francesco De Carlo. Maximum a posteriori estimation of crystallographic phases in x-ray diffraction tomography. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 373(2043):20140392, 2015.

[B20] Doğa Gürsoy, Tekin Biçer, Antonio Lanzirotti, Matthew G Newville, and Francesco De Carlo. Hyperspectral image reconstruction for x-ray fluorescence tomography. *Optics express*, 23(7):9014–9023, 2015.

[B21] P. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems*. Society for Industrial and Applied Mathematics, 1998. arXiv:http://epubs.siam.org/doi/pdf/10.1137/1.9780898719697, doi:10.1137/1.9780898719697.

[B22] H. M. Hudson and R. S. Larkin. Accelerated image reconstruction using ordered subsets of projection data. *IEEE Transactions on Medical Imaging*, 13(4):601–609, December 1994. doi:10.1109/42.363108.

[B23] Sijbers Jan and Postnov Andrei. Reduction of ring artefacts in high resolution micro-ct reconstructions. *Physics in Medicine and Biology*, 49(14):N247, 2004.

[B24] Avinash C.. Kak and Malcolm Slaney. *Principles of computerized tomographic imaging*. Society for Industrial and Applied Mathematics, 2001.

[B25] Waruntorn Kanitpanyacharoen, Dilworth Y. Parkinson, Francesco De Carlo, Federica Marone, Marco Stampanoni, Rajmund Mokso, Alastair MacDowell, and Hans-Rudolf Wenk. A comparative study of x-ray tomographic microscopy?on shales at different synchrotron facilities: als, aps and sls. *J Synchrotron Radiat*, 20(Pt 1):172–180, Jan 2013. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3943535/, doi:10.1107/S0909049512044354.

[B26] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: a convolutional neural-network approach. *Ieee Transactions on Neural Networks*, 8(1):98–113, 1997. doi:10.1109/72.554195.

[B27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the Ieee*, 86(11):2278–2324, 1998. doi:10.1109/5.726791.

[B28] Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, 253–256. 2010. doi:10.1109/ISCAS.2010.5537907.

[B29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi:10.1038/nature14539.

[B30] Goran Lovric, Sébastien F Barré, Johannes C Schittny, Matthias Roth-Kleiner, Marco Stampanoni, and Rajmund Mokso. Dose optimization approach to fast x-ray microtomography of the lung alveoli. *Journal of applied crystallography*, 46(4):856–860, 2013.

[B31] GS Manuel, ST Thurman, and JR Fienup. Efficient subpixel image registration algorithms. *Optics Letters*, 33(2):156–158, 2008.

[B32] F Marone and M Stampanoni. Regridding reconstruction algorithm for real-time tomographic imaging. *Journal of synchrotron radiation*, 19(6):1029–1037, 2012.

[B33] M. Matsugu, K. Mori, Y. Mitari, and Y. Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5-6):555–559, 2003. doi:10.1016/s0893-6080(03)00115-1.

[B34] J. C. E. Mertens, J. J. Williams, and Nikhilesh Chawla. A method for zinger artifact reduction in high-energy x-ray computed tomography. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 800:82–92, 2015. doi:10.1016/j.nima.2015.08.012.

[B35] E. X. Miqueles, J. Rinkel, F. O'Dowd, and J. S. V. Bermúdez. Generalized Titarenko's algorithm for ring artefacts reduction. *Journal of Synchrotron Radiation*, 21(6):1333–1346, 2014. doi:10.1107/S1600577514016919.

[B36] Piotr Mirowski, Deepak Madhavan, Yann LeCun, and Ruben Kuzniecky. Classification of patterns of eeg synchronization for seizure prediction. *Clinical Neurophysiology*, 120(11):1927–1940, 2009. doi:10.1016/j.clinph.2009.09.002.

[B37] R Mokso, F Marone, S Irvine, M Nyvlt, D Schwyn, K Mader, G K Taylor, H G Krapp, M Skeren, and M Stampanoni. Advantages of phase retrieval for fast x-ray tomographic microscopy. *Journal of Physics D: Applied Physics*, 46(49):494004, 2013.

[B38] Julian Moosmann, Alexey Ershov, Venera Altapova, Tilo Baumbach, Maneeshi S. Prasad, Carole LaBonne, Xianghui Xiao, Jubin Kashef, and Ralf Hofmann. X-ray phase-contrast in vivo microtomography probes new aspects of xenopus gastrulation. *Nature*, 497(7449):374–377, 2013. doi:10.1007/s10853-015-9355-8.

[B39] Beat Münch, Pavel Trtik, Federica Marone, and Marco Stampanoni. Stripe and ring artifact removal with combined wavelet—Fourier filtering. *Optics express*, 17(10):8567–8591, 2009.

[B40] David Paganin, S. C. Mayo, Tim E Gureyev, Peter R Miller, and Steve W Wilkins. Simultaneous phase and amplitude extraction from a single defocused image of a homogeneous object. *Journal of microscopy*, 206(1):33–40, 2002.

[B41] W. J. Palenstijn, K. J. Batenburg, and J. Sijbers. Performance improvements for iterative electron tomography reconstruction using graphics processing units (GPUs). *Journal of structural biology*, 176(2):250–253, 2011.

[B42] Willem Jan Palenstijn, Jeroen Bédorf, and K Joost Batenburg. A distributed sirt implementation for the astra toolbox. In *Proceedings of the 13\textsuperscript th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, 166–169. 2015.

[B43] P. Paleo and A. Mirone. Ring artifacts correction in compressed sensing tomographic reconstruction. *Journal of Synchrotron Radiation*, 22:1268–1278, 2015. doi:10.1107/s1600577515010176.

[B44] Brian M. Patterson, Nikolaus L. Cordes, Kevin Henderson, Jason J. Williams, Tyler Stannard, Sudhanshu S. Singh, Angel Rodriguez Ovejero, Xianghui Xiao, Mathew Robinson, and Nikhilesh Chawla. In situ x-ray synchrotron tomographic imaging during the compression of hyper-elastic polymeric materials. *Journal of Materials Science*, 51(1):171–187, 2016. doi:10.1007/s10853-015-9355-8.

[B45] S Peetermans and E H Lehmann. Simultaneous neutron transmission and diffraction contrast tomography as a non-destructive 3D method for bulk single crystal quality investigations. *Journal of Applied Physics*, 114(12):124905, 2013.

[B46] F. Pereira, T. Mitchell, and M. Botvinick. Machine learning classifiers and fmri: a tutorial overview. *Neuroimage*, 45(1):S199–S209, 2009. doi:10.1016/j.neuroimage.2008.11.007.

[B47] C Phatak and D Gürsoy. Iterative reconstruction of magnetic induction using lorentz transmission electron tomography. *Ultramicroscopy*, 150:54–64, 2015.

[B48] L. Plantagie, W. van Aarle, J. Sijbers, and K. J. Batenburg. Filtered backprojection using algebraic filters; application to biomedical micro-CT data. In *Biomedical Imaging (ISBI), 2015 IEEE 12\textsuperscript th International Symposium on*, 1596–1599. IEEE, 2015.

[B49] Sabrina Rashid, Soo Yeol Lee, and Md Kamrul Hasan. An improved method for the removal of ring artifacts in high resolution ct imaging. *EURASIP Journal on Advances in Signal Processing*, 2012(1):1–18, 2012. doi:10.1186/1687-6180-2012-93.

[B50] Carsten Raven. Numerical removal of ring artifacts in microtomography. *Review of Scientific Instruments*, 69(8):2978–2980, 1998. doi:doi:http://dx.doi.org/10.1063/1.1149043.

[B51] Peter Reischig, Andrew King, Laura Nervo, Nicola Vigano, Yoann Guilhem, Willem Jan Palenstijn, K Joost Batenburg, Michael Preuss, and Wolfgang Ludwig. Advances in x-ray diffraction contrast tomography: flexibility in the setup geometry and application to multiphase materials. *Journal of Applied Crystallography*, 46(2):297–311, 2013.

[B52] T Roelandts, K J Batenburg, E Biermans, C Kübel, S Bals, and J Sijbers. Accurate segmentation of dense nanoparticles by partially discrete electron tomography. *Ultramicroscopy*, 114:96–105, 2012.

[B53] Anne Sakdinawat and David Attwood. Nanoscale x-ray imaging. *Nature photonics*, 4(12):840–848, 2010.

[B54] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, 2016. URL: http://arxiv.org/abs/1605.06211.

[B55] Valeriy Titarenko, Robert Bradley, Christopher Martin, Philip J. Withers, and Sofya Titarenko. Regularization methods for inverse problems in x-ray tomography. In *Proc. SPIE 7804, Developments in X-Ray Tomography VII*, volume 7804, 78040Z–10. 2010.

[B56] Wim van Aarle, Willem Jan Palenstijn, Jan De Beenhouwer, Thomas Altantzis, Sara Bals, K Joost Batenburg, and Jan Sijbers. The astra toolbox: a platform for advanced algorithm development in electron tomography. *Ultramicroscopy*, 157:35–47, 2015.

[B57] G. Van Eyndhoven, K. J. Batenburg, D. Kazantsev, V. Van Nieuwenhove, P. D. Lee, K. J. Dobson, and J. Sijbers. An iterative CT reconstruction algorithm for fast fluid flow imaging. *Image Processing, IEEE Transactions on*, 24(11):4446–4458, November 2015. doi:10.1109/TIP.2015.2466113.

[B58] Nghia T. Vo, Michael Drakopoulos, Robert C. Atwood, and Christina Reinhard. Reliable method for calculating the center of rotation in parallel-beam tomography. *Optics Express*, 22(16):19078–19086, 2014. URL: http://www.opticsexpress.org/abstract.cfm?URI=oe-22-16-19078, doi:10.1364/OE.22.019078.

[B59] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Ieee Transactions on Image Processing*, 13(4):600–612, 2004. doi:10.1109/tip.2003.819861.

[B60] Fang Xu and Klaus Mueller. A comparative study of popular interpolation and integration methods for use in computed tomography. In *Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium on*, 1252–1255. IEEE, 2006.

[B61] Yimeng Yang, Feifei Yang, Ferdinand F. Hingerl, Xianghui Xiao, Yijin Liu, Ziyu Wu, Sally M. Benson, Michael F. Toney, Joy C. Andrews, and Piero Pianetta. Registration of the rotation axis in x-ray tomography. *Journal of Synchrotron Radiation*, 22(2):452–457, 2015. doi:doi:10.1107/S160057751402726X.

[B62] Kyriakou Yiannis, Prell Daniel, and A. Kalender Willi. Ring artifact correction for high-resolution micro ct. *Physics in Medicine and Biology*, 54(17):N385, 2009.

[B63] Wei Zhouping, Wiebe Sheldon, and Chapman Dean. Ring artifacts removal from synchrotron ct image slices. *Journal of Instrumentation*, 8(06):C06006, 2013.

[B64] Leqing Zhu, Dadong Wang, and Huiyan Wang. An improved method for the removal of ring artifacts in synchrotron radiation images by using gpgpu computing with compute unified device architecture. *Concurrency and Computation: Practice and Experience*, 26(18):2880–2892, 2014. doi:10.1002/cpe.3183.

Python Module Index

x

xlearn, 18
xlearn.classify, 9
xlearn.segmentation, 9
xlearn.transform, 8
xlearn.utils, 11

Index

C

check_random_state() (in module xlearn.utils), 11

E

expimg() (in module xlearn.utils), 11
extract_3d() (in module xlearn.utils), 11
extract_patches() (in module xlearn.utils), 11

I

img_window() (in module xlearn.utils), 12

M

mlog() (in module xlearn.utils), 12
model() (in module xlearn.classify), 9
model() (in module xlearn.transform), 8
model_choose() (in module xlearn.segmentation), 9

N

nor_data() (in module xlearn.utils), 12

P

predict() (in module xlearn.transform), 8

R

reconstruct_patches() (in module xlearn.utils), 12
rescale_intensity() (in module xlearn.utils), 12

S

seg_predict() (in module xlearn.segmentation), 10
seg_train() (in module xlearn.segmentation), 9

T

train() (in module xlearn.classify), 9
train() (in module xlearn.transform), 8

X

xlearn (module), 12, 18, 24
xlearn.classify (module), 9
xlearn.segmentation (module), 9
xlearn.transform (module), 8
xlearn.utils (module), 11