Numpy Data Preprocessing

September 18, 2018

```
In [14]: def imcrop_tosquare(img):
             """Make any image a square image.

             Parameters
             ----------
             img : np.ndarray
                 Input image to crop, assumed at least 2d.

             Returns
             -------
             crop : np.ndarray
                 Cropped image.
             """
             if img.shape[0] > img.shape[1]:
                 extra = (img.shape[0] - img.shape[1])
                 if extra % 2 == 0:
                     crop = img[extra // 2:-extra // 2, :]
                 else:
                     crop = img[max(0, extra // 2 + 1):min(-1, -(extra // 2)), :]
             elif img.shape[1] > img.shape[0]:
                 extra = (img.shape[1] - img.shape[0])
                 if extra % 2 == 0:
                     crop = img[:, extra // 2:-extra // 2]
                 else:
                     crop = img[:, max(0, extra // 2 + 1):min(-1, -(extra // 2))]
             else:
                 crop = img
             return crop
```

https://stackoverflow.com/questions/31621414/share-data-between-ipython-notebooks

```
In [22]: def imcrop(img, amt):
             if amt <= 0 or amt >= 1:
                 return img
             row_i = int(img.shape[0] * amt) // 2
             col_i = int(img.shape[1] * amt) // 2
             return img[row_i:-row_i, col_i:-col_i]
```

```
In [12]: import numpy as np
         from matplotlib import pylab as plt

         testfilename='D:\\Users\\svekhande\\sino_1021.raw'
         B = np.fromfile(testfilename, dtype='float32', sep="")


         B=np.reshape(B,[689,2048])
         print(B.shape)
```

```
(689, 2048)
```

```
In [15]: square = imcrop_tosquare(B)
```

https://github.com/pkmital/CADL/tree/master/session-1

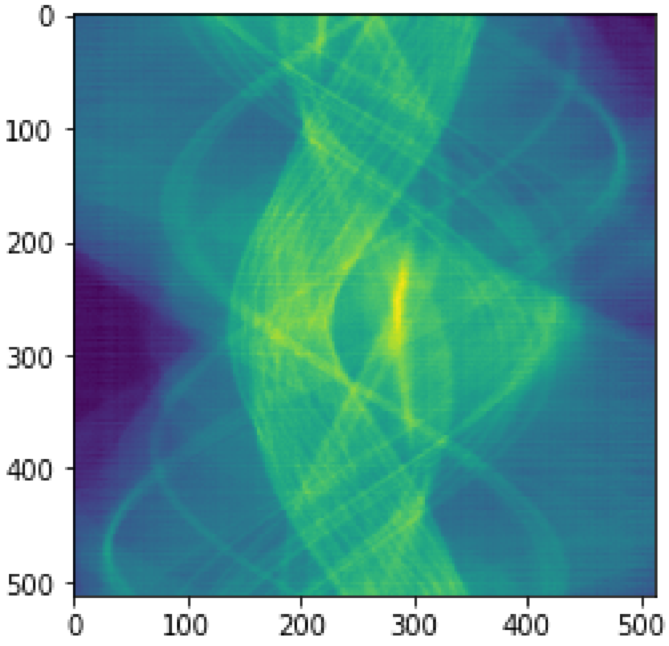
```
In [46]: print(square.shape)
         %store square
```

```
(689, 689)
Stored 'square' (ndarray)
```

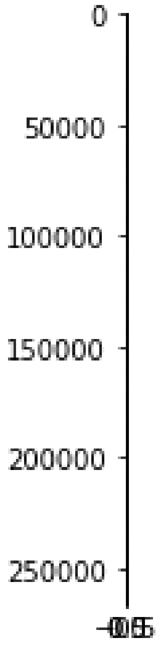
```
In [47]: from scipy.misc import imresize

         crop = imcrop(square, 0.2)
         rsz = imresize(crop, (512, 512))
         plt.imshow(rsz)
         %store rsz
```

```
Stored 'rsz' (ndarray)
```

```
C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_launcher.py:4:
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``skimage.transform.resize`` instead.
  after removing the cwd from sys.path.
```



```
In [34]: plt.imshow(rsz, interpolation='nearest')
         rsz=np.reshape(rsz,[262144,])
```



```
In [48]: mean_img = np.mean(rsz, axis=2)
         print(mean_img.shape)
         plt.imshow(mean_img, cmap='gray')
```

```
        ---------------------------------------------------------------------------

        IndexError                                Traceback (most recent call last)

        <ipython-input-48-44cbd031d909> in <module>()
    ----> 1 mean_img = np.mean(rsz, axis=2)
          2 print(mean_img.shape)
          3 plt.imshow(mean_img, cmap='gray')


        C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\numpy\core\fromnu
       2955
       2956     return _methods._mean(a, axis=axis, dtype=dtype,
    -> 2957                           out=out, **kwargs)
       2958
       2959


        C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\numpy\core\_metho
         55
         56     is_float16_result = False
    ---> 57     rcount = _count_reduce_items(arr, axis)
         58     # Make this warning show up first
         59     if rcount == 0:


        C:\Users\svekhand\AppData\Local\Continuum\anaconda3\lib\site-packages\numpy\core\_metho
         48     items = 1
         49     for ax in axis:
    ---> 50         items *= arr.shape[ax]
         51     return items
         52


        IndexError: tuple index out of range
```

```
In [42]: from __future__ import print_function
         import tensorflow as tf
         import matplotlib.pyplot as plt
         from scipy.misc import imresize
         import numpy as np
         from skimage.io import imread
```

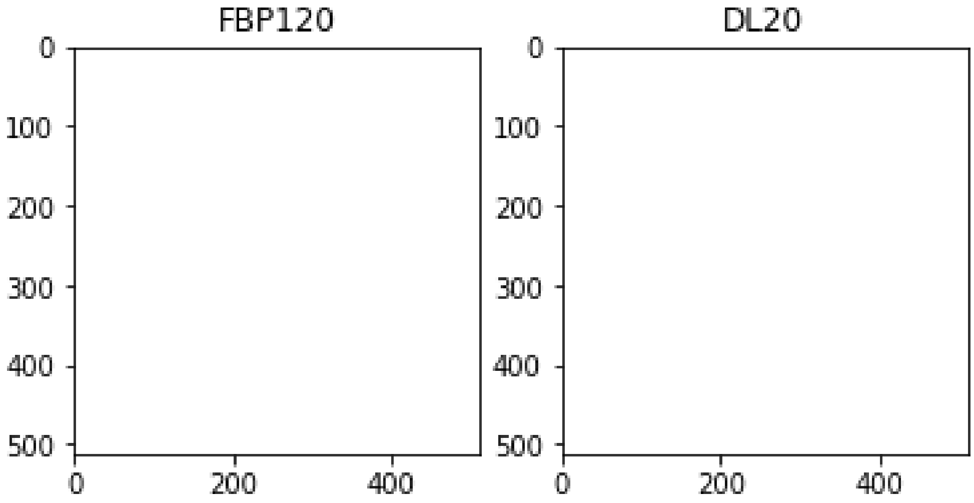
```
import math
import csv
import random
import time
import scipy.io
from functools import reduce
import os
import psutil
import time
from sklearn import metrics
from sklearn.metrics.cluster import normalized_mutual_info_score
from scipy import signal
from scipy import ndimage

tf.reset_default_graph()
def memory():
    pid = os.getpid()
    py = psutil.Process(pid)
    memoryUse = py.memory_info()[0] / 2. ** 30  # memory use in GB...I think
    print('memory use:', memoryUse)

#####################################################################################
def BN(img):
    batch_mean, batch_var = tf.nn.moments(img, [0, 1, 2], name='moments')
    img = tf.nn.batch_normalization(img, batch_mean, batch_var, 0, 1, 1e-3)
    return img

def weight_variable(shape):
    with tf.name_scope('Weight'):
        initial = tf.truncated_normal(shape, stddev=0.01, name='W')
    return tf.Variable(initial)


def bias_variable(shape):
    with tf.name_scope('Bias'):
        initial = tf.constant(0.1, shape=shape, name='b')
    return tf.Variable(initial)


def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')


def max_pool_2x1(x):
    return tf.nn.max_pool(x, ksize=[1, 1, 2, 1], strides=[1, 1, 2, 1], padding='SAME'


def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME'


def max_pool(x, n):
    return tf.nn.max_pool(x, ksize=[1, n, n, 1], strides=[1, n, n, 1], padding='VALID


def build_unpool(source, kernel_shape):
    input_shape = source.get_shape().as_list()
    return tf.image.resize_images(source, [input_shape[1] * kernel_shape[1], input_sha
#####################################################################################
def rmse(predictions, targets):
    return np.sqrt(np.mean(np.square(predictions - targets)))

def ssim(img1, img2, cs_map=False):
    """Return the Structural Similarity Map corresponding to input images img1
    and img2 (images are assumed to be uint8)
    This function attempts to mimic precisely the functionality of ssim.m a
    MATLAB provided by the author's of SSIM
    https://ece.uwaterloo.ca/~z70wang/research/ssim/ssim_index.m
    """
    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)
    size = 11
    sigma = 1.5
    x, y = np.mgrid[-size // 2 + 1:size // 2 + 1, -size // 2 + 1:size // 2 + 1]
    g = np.exp(-((x ** 2 + y ** 2) / (2.0 * sigma ** 2)))
    window = g / np.sum(g)
    K1 = 0.01
    K2 = 0.03
    L = 0.082  # bitdepth of image
    C1 = (K1 * L) ** 2
    C2 = (K2 * L) ** 2
    mu1 = signal.fftconvolve(window, img1, mode='valid')
    mu2 = signal.fftconvolve(window, img2, mode='valid')
    mu1_sq = mu1 * mu1
    mu2_sq = mu2 * mu2
    mu1_mu2 = mu1 * mu2
    sigma1_sq = signal.fftconvolve(window, img1 * img1, mode='valid') - mu1_sq
    sigma2_sq = signal.fftconvolve(window, img2 * img2, mode='valid') - mu2_sq
    sigma12 = signal.fftconvolve(window, img1 * img2, mode='valid') - mu1_mu2
    if cs_map:
        ssim_map = (((2 * mu1_mu2 + C1) * (2 * sigma12 + C2)) / ((mu1_sq + mu2_sq + C
                                                                 (sigma1_sq + sigma2_s
                    (2.0 * sigma12 + C2) / (sigma1_sq + sigma2_sq + C2))
    else:
        ssim_map = ((2 * mu1_mu2 + C1) * (2 * sigma12 + C2)) / ((mu1_sq + mu2_sq + C1
                                                                (sigma1_sq + sigma2_s
    return np.mean(ssim_map)
#####################################################################################
def DenseNet(input, growth_rate=16, nb_filter=16, filter_wh=5):
    shape = input.get_shape().as_list()
    with tf.name_scope('layer1'):
        input = BN(input)
        input = tf.nn.relu(input)

        w1_1 = weight_variable([1, 1, shape[3], nb_filter * 4])
        b1_1 = bias_variable([nb_filter * 4])
        c1_1 = tf.nn.conv2d(input, w1_1, strides=[1, 1, 1, 1], padding='SAME') + b1_1
        ##

        c1_1 = BN(c1_1)
        c1_1 = tf.nn.relu(c1_1)

        w1 = weight_variable([filter_wh, filter_wh, nb_filter * 4, nb_filter])
        b1 = bias_variable([nb_filter])
        c1 = tf.nn.conv2d(c1_1, w1, strides=[1, 1, 1, 1], padding='SAME') + b1

    h_concat1 = tf.concat([input, c1], 3)

    with tf.name_scope('layer2'):
        h_concat1 = BN(h_concat1)
        h_concat1 = tf.nn.relu(h_concat1)

        w2_1 = weight_variable([1, 1, shape[3] + nb_filter, nb_filter * 4])
        b2_1 = bias_variable([nb_filter * 4])
        c2_1 = tf.nn.conv2d(h_concat1, w2_1, strides=[1, 1, 1, 1], padding='SAME') +
        ##

        c2_1 = BN(c2_1)
        c2_1 = tf.nn.relu(c2_1)

        w2 = weight_variable([filter_wh, filter_wh, nb_filter * 4, nb_filter])
        b2 = bias_variable([nb_filter])
        c2 = tf.nn.conv2d(c2_1, w2, strides=[1, 1, 1, 1], padding='SAME') + b2

    h_concat2 = tf.concat([input, c1, c2], 3)

    with tf.name_scope('layer3'):
        h_concat2 = BN(h_concat2)
        h_concat2 = tf.nn.relu(h_concat2)

        w3_1 = weight_variable([1, 1, shape[3] + nb_filter + nb_filter, nb_filter * 4
        b3_1 = bias_variable([nb_filter * 4])
        c3_1 = tf.nn.conv2d(h_concat2, w3_1, strides=[1, 1, 1, 1], padding='SAME') +
        ##
        c3_1 = BN(c3_1)
        c3_1 = tf.nn.relu(c3_1)

        w3 = weight_variable([filter_wh, filter_wh, nb_filter * 4, nb_filter])
        b3 = bias_variable([nb_filter])
        c3 = tf.nn.conv2d(c3_1, w3, strides=[1, 1, 1, 1], padding='SAME') + b3

    h_concat3 = tf.concat([input, c1, c2, c3], 3)

    with tf.name_scope('layer4'):
        h_concat3 = BN(h_concat3)
        h_concat3 = tf.nn.relu(h_concat3)

        w4_1 = weight_variable([1, 1, shape[3] + nb_filter + nb_filter + nb_filter, nb
        b4_1 = bias_variable([nb_filter * 4])
        c4_1 = tf.nn.conv2d(h_concat3, w4_1, strides=[1, 1, 1, 1], padding='SAME') + b
        ##

        c4_1 = BN(c4_1)
        c4_1 = tf.nn.relu(c4_1)

        w4 = weight_variable([filter_wh, filter_wh, nb_filter * 4, nb_filter])
        b4 = bias_variable([nb_filter])
        c4 = tf.nn.conv2d(c4_1, w4, strides=[1, 1, 1, 1], padding='SAME') + b4

    return tf.concat([input, c1, c2, c3, c4], 3)

#####################################################################################
if __name__ == '__main__':

    row=512
    column = 512
    size = row * column
    ##
    batch = 1
    test_num = 1
    #########################################################################
    with tf.name_scope('inputs'):
        xs = tf.placeholder(tf.float32, [batch, size], name='xs')
        ys = tf.placeholder(tf.float32, [batch, size], name='ys')
        step = tf.placeholder(tf.float32, name='prob')
        keep_prob = tf.placeholder(tf.float32, name='prob')

    input_image = tf.reshape(xs, [batch, row, column, 1])
    output_image = tf.reshape(ys, [batch, row, column, 1])
    #################################################################################
    nb_filter = 16
```

```
W_conv1 = weight_variable([7, 7, 1, nb_filter])
b_conv1 = bias_variable([nb_filter])
h_conv1 = (tf.nn.conv2d(input_image, W_conv1, strides=[1, 1, 1, 1], padding='SAME

h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1],
                         padding='SAME')  # 128*128*(nb_filter)

D1 = DenseNet(h_pool1, growth_rate=16, nb_filter=nb_filter, filter_wh=5)  # 128*1

D1 = BN(D1)
D1 = tf.nn.relu(D1)
W_conv1_T = weight_variable([1, 1, nb_filter + nb_filter * 4, nb_filter])
b_conv1_T = bias_variable([nb_filter])
h_conv1_T = (
    tf.nn.conv2d(D1, W_conv1_T, strides=[1, 1, 1, 1], padding='SAME') + b_conv1_T)

h_pool1_T = tf.nn.max_pool(h_conv1_T, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
                           padding='SAME')  # 64*64*(nb_filter)

##
D2 = DenseNet(h_pool1_T, growth_rate=16, nb_filter=nb_filter, filter_wh=5)  # 64*
D2 = BN(D2)
D2 = tf.nn.relu(D2)

W_conv2_T = weight_variable([1, 1, nb_filter + nb_filter * 4, nb_filter])
b_conv2_T = bias_variable([nb_filter])
h_conv2_T = (
    tf.nn.conv2d(D2, W_conv2_T, strides=[1, 1, 1, 1], padding='SAME') + b_conv2_T)

h_pool2_T = tf.nn.max_pool(h_conv2_T, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
                           padding='SAME')  # 32*32*(nb_filter)

##
D3 = DenseNet(h_pool2_T, growth_rate=16, nb_filter=nb_filter, filter_wh=5)  # 32*
D3 = BN(D3)
D3 = tf.nn.relu(D3)
W_conv3_T = weight_variable([1, 1, nb_filter + nb_filter * 4, nb_filter])
b_conv3_T = bias_variable([nb_filter])
h_conv3_T = (
    tf.nn.conv2d(D3, W_conv3_T, strides=[1, 1, 1, 1], padding='SAME') + b_conv3_T)

h_pool3_T = tf.nn.max_pool(h_conv3_T, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
                           padding='SAME')  # 16*16*(nb_filter)

##
D4 = DenseNet(h_pool3_T, growth_rate=16, nb_filter=nb_filter, filter_wh=5)  # 16*
D4 = BN(D4)
```

```
D4 = tf.nn.relu(D4)
W_conv4_T = weight_variable([1, 1, nb_filter + nb_filter * 4, nb_filter])
b_conv4_T = bias_variable([nb_filter])
h_conv4_T = (
    tf.nn.conv2d(D4, W_conv4_T, strides=[1, 1, 1, 1], padding='SAME') + b_conv4_T

##

W_conv40 = weight_variable([5, 5, 2 * nb_filter, 2 * nb_filter])
b_conv40 = bias_variable([2 * nb_filter])
h_conv40 = tf.nn.relu(
    tf.nn.conv2d_transpose(tf.concat([build_unpool(h_conv4_T, [1, 2, 2, 1]), h_co
                           [batch, 64, 64, 2 * nb_filter], strides=[1, 1, 1, 1],
                           padding='SAME') + b_conv40)  # 32*32*40
batch_mean, batch_var = tf.nn.moments(h_conv40, [0, 1, 2], name='moments')
h_conv40 = tf.nn.batch_normalization(h_conv40, batch_mean, batch_var, 0, 1, 1e-3)

W_conv40_T = weight_variable([1, 1, nb_filter, (2 * nb_filter)])
b_conv40_T = bias_variable([nb_filter])
h_conv40_T = tf.nn.relu(
    tf.nn.conv2d_transpose(h_conv40, W_conv40_T, [batch, 64, 64, nb_filter], stri
                           padding='SAME') + b_conv40_T)  # 32*32*40
batch_mean, batch_var = tf.nn.moments(h_conv40_T, [0, 1, 2], name='moments')
h_conv40_T = tf.nn.batch_normalization(h_conv40_T, batch_mean, batch_var, 0, 1, 1

##
W_conv5 = weight_variable([5, 5, 2 * nb_filter, 2 * nb_filter])
b_conv5 = bias_variable([2 * nb_filter])
h_conv5 = tf.nn.relu(
    tf.nn.conv2d_transpose(tf.concat([build_unpool(h_conv40_T, [1, 2, 2, 1]), h_c
                           [batch, 128, 128, 2 * nb_filter], strides=[1, 1, 1, 1]
                           padding='SAME') + b_conv5)  # 64*64*20
batch_mean, batch_var = tf.nn.moments(h_conv5, [0, 1, 2], name='moments')
h_conv5 = tf.nn.batch_normalization(h_conv5, batch_mean, batch_var, 0, 1, 1e-3)

W_conv5_T = weight_variable([1, 1, nb_filter, 2 * nb_filter])
b_conv5_T = bias_variable([nb_filter])
h_conv5_T = tf.nn.relu(
    tf.nn.conv2d_transpose(h_conv5, W_conv5_T, [batch, 128, 128, nb_filter], stri
                           padding='SAME') + b_conv5_T)  # 64*64*20
batch_mean, batch_var = tf.nn.moments(h_conv5_T, [0, 1, 2], name='moments')
h_conv5_T = tf.nn.batch_normalization(h_conv5_T, batch_mean, batch_var, 0, 1, 1e-

##
W_conv6 = weight_variable([5, 5, 2 * nb_filter, 2 * nb_filter])
b_conv6 = bias_variable([2 * nb_filter])
h_conv6 = tf.nn.relu(
    tf.nn.conv2d_transpose(tf.concat([build_unpool(h_conv5_T, [1, 2, 2, 1]), h_co
                           [batch, 256, 256, 2 * nb_filter], strides=[1, 1, 1, 1]
                           padding='SAME') + b_conv6)
batch_mean, batch_var = tf.nn.moments(h_conv6, [0, 1, 2], name='moments')
h_conv6 = tf.nn.batch_normalization(h_conv6, batch_mean, batch_var, 0, 1, 1e-3)

W_conv6_T = weight_variable([1, 1, nb_filter, 2 * nb_filter])
b_conv6_T = bias_variable([nb_filter])
h_conv6_T = tf.nn.relu(
    tf.nn.conv2d_transpose(h_conv6, W_conv6_T, [batch, 256, 256, nb_filter], stri
                           padding='SAME') + b_conv6_T)  # 64*64*20
batch_mean, batch_var = tf.nn.moments(h_conv6_T, [0, 1, 2], name='moments')
h_conv6_T = tf.nn.batch_normalization(h_conv6_T, batch_mean, batch_var, 0, 1, 1e-

W_conv7 = weight_variable([5, 5, 2 * nb_filter, 2 * nb_filter])
b_conv7 = bias_variable([2 * nb_filter])
h_conv7 = tf.nn.relu(
    tf.nn.conv2d_transpose(tf.concat([build_unpool(h_conv6_T, [1, 2, 2, 1]), h_co
                           [batch, 512, 512, 2 * nb_filter], strides=[1, 1, 1, 1]
                           padding='SAME') + b_conv7)


W_conv8 = weight_variable([1, 1, 1, 2 * nb_filter])
b_conv8 = bias_variable([1])
h_conv8 = tf.nn.relu(tf.nn.conv2d_transpose(h_conv7, W_conv8, [batch, 512, 512, 1
                                 padding='SAME') + b_conv8)
output_net2 = tf.reshape(h_conv8, [batch, size])
#################################################################################

saver = tf.train.Saver()
init = tf.global_variables_initializer()
fig = plt.figure()
plt.ion()
plt.show()

test_img120 = np.zeros((test_num, row * column), dtype=np.float32)
for i in range(batch):
    #test_img120[i] = np.reshape(np.transpose(scipy.io.loadmat('FBP120_512_57_19_
    test_img120[i]=rsz


#################################################################################
with tf.Session() as sess:
    sess.run(init)
    savename = "D:\\Users\\svekhande\\downloads\\DD_Net_" + str(30) + ".ckpt"
    saver.restore(sess,  savename)

    start = time.clock()
    DL_120 = np.reshape(np.transpose(sess.run(h_conv8, feed_dict={xs: test_img120
                        [row, column])
    elapsed = (time.clock() - start)
    print("Time used:", elapsed)
    plt.subplot(121)

    plt.imshow(np.transpose(np.reshape(test_img120[0], [row, column])), interpola
               clim=(0, 0.029), cmap="gray")
    plt.title('FBP120')

    plt.subplot(122)
    plt.imshow(DL_120, interpolation="nearest", clim=(0., 0.029), cmap="gray")
    plt.title('DL20')
    plt.pause(10)

print('end')
```

```
<Figure size 432x288 with 0 Axes>
```

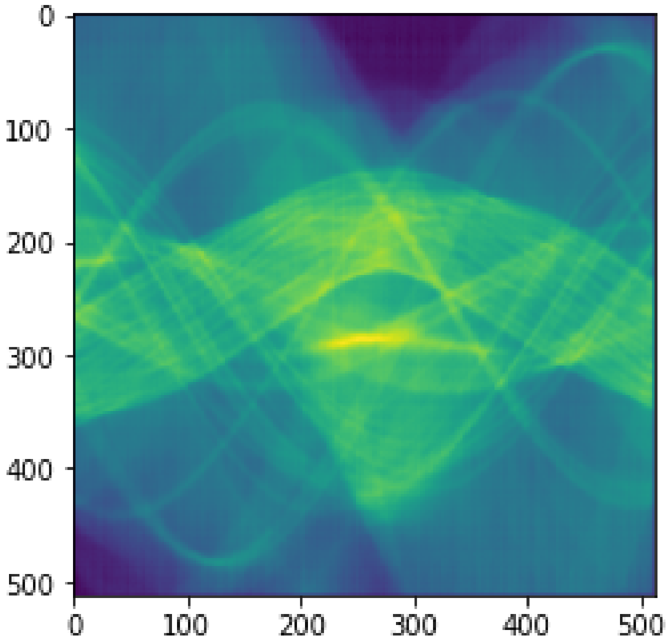
```
INFO:tensorflow:Restoring parameters from D:\Users\svekhande\downloads\DD_Net_30.ckpt
Time used: 2.581150533248092
```



```
end
```

```
In [45]: DL120_=np.reshape(DL_120,[512,512])
         plt.imshow(DL120_,interpolation="nearest")
```

```
Out[45]: <matplotlib.image.AxesImage at 0x1ed2aeec7b8>
```


0
100
200
300
400
500
0
100
200
300
400
500