import tensorflow as tf

import matplotlib.pyplot as plt

import numpy as np

import os,fnmatch

import seaborn as sns ; sns.set()

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import cv2

%matplotlib inline

print(tf.\_\_version\_\_)

!wget http://peipa.essex.ac.uk/pix/mias/all-mias.tar.gz

!wget https://mynotebook.labarchives.com/attachments/My45MDAwMDAwMDAwMDAwMDA0fDg1NzYyLzMvRW50cnlQYXJ0LzQxNTk4ODc0Nzh8OS44OTk5OTk5OTk5OTk5OTk=/1/original?sf324=343

!mkdir Images

!tar -C Images -zxvf all-mias.tar.gz

#optional (first time)

def load\_data():

    files = os.listdir("Images")

    files.remove("Info.txt")

    files.remove("Licence.txt")

    files.remove("README")

    os.mkdir("Images/train")

    os.mkdir("Images/test")

    for i in range(len(files)):

      x = cv2.imread("Images/"+files[i])

      if(i<round(0.6\*len(files))):

        cv2.imwrite("Images/train/"+str(i)+".png",x)

      else:

         cv2.imwrite("Images/test/"+str(i)+".png",x)

load\_data()

class DataGen:

    def \_\_init\_\_(self,path,batch\_size,target\_size):

        self.path = path

        self.batch\_size = batch\_size

        self.target\_size = target\_size

    def getImage(self,imPath):

        img = cv2.imread(imPath,0)

        return img

    def rescale(self,img):

        x = img/255.0

        return x

    def resize(self,img):

        x = cv2.resize(img,self.target\_size,interpolation=cv2.INTER\_CUBIC)

        x = np.reshape(x,(64,64,1))

        return x

    def addNoise(self,img,noise\_factor,mu=0,sigma=1):

        img+= noise\_factor\*np.random.normal(mu,sigma,size=img.shape)

img = np.clip(img,0.,1.)

        return img

def poisson\_noise(self,img):

        img+= numpy.random.poisson(img).astype(float)

        img = np.clip(img,0.,1.)

        return img

    def generate(self):

        import os

        files = os.listdir(self.path)

        while(True):

            batch\_paths  = np.random.choice(a = files,

            size = self.batch\_size)

            batch\_input  = []

            batch\_output = []

          # Read in each input, perform preprocessing and get labels

            for input\_path in batch\_paths:

                img = self.getImage(self.path+input\_path)

                img = self.rescale(img)

                img = img.astype(np.float)

                output = self.resize(img)

                img = self.addNoise(img,0.1) # second argument is p  in pa per for gaussian noise

                img = self.resize(img)

                batch\_input += [ img ]

                batch\_output += [ output ]

          # Return a tuple of (input, output) to feed the network

            batch\_x = np.array( batch\_input )

            batch\_y = np.array( batch\_output )

         yield batch\_x, batch\_y

# (Important) split all images in folder "Images" in 60-40 percent as 60% images in "Images/train" folder and 40% in "Images/test" folder

train = DataGen("Images/train/",16,(64,64))

train\_gen = train.generate() # training set generator

test = DataGen("Images/test/",10,(64,64))

test\_gen = test.generate() # test set generator

class callback(tf.keras.callbacks.Callback):

    def on\_epoch\_end(self, epoch, logs={}):

        if logs.get('loss')<0.23 and logs.get('val\_loss')<0.23:

            print("\n Stopping training .....\n")

            self.model.stop\_training=True

myCallback = callback()

autoencoder = tf.keras.models.Sequential([

tf.keras.layers.Conv2D(64,(3,3),input\_shape=(64,64,1),activation="relu ",padding="same",data\_format="channels\_last"),

    tf.keras.layers.MaxPool2D(padding="same"),

    tf.keras.layers.Conv2D(64,(3,3),activation="relu",padding="same"),

    tf.keras.layers.MaxPool2D(padding="same"),

    tf.keras.layers.Conv2D(64,(3,3),activation="relu",padding="same"),

    tf.keras.layers.UpSampling2D(),

    tf.keras.layers.Conv2D(64,(3,3),activation="relu",padding="same"),

    tf.keras.layers.UpSampling2D(),

    tf.keras.layers.Conv2D(1,(3,3),activation="sigmoid",padding="same")

    ])

autoencoder.summary()

autoencoder.compile(optimizer='adam', loss="binary\_crossentropy",metrics = ['accuracy'])

# let it run upto 10 epochs

hist = autoencoder.fit\_generator(train\_gen,epochs=10,steps\_per\_epoch=16,validation\_data=test\_gen,validation\_steps=10,callbacks=[myCallback])

plt.ylabel("Loss")

plt.xlabel("epochs")

plt.plot(hist.history["loss"],label="Training Loss")

plt.plot(hist.history["val\_loss"],label="Validation Loss")

plt.legend()

plt.show()

from mpl\_toolkits.axes\_grid1 import ImageGrid

r,t = next(test\_gen) # sample

z = autoencoder.predict(r) # predicted result

z.shape

img1 = r[0][:,:,0]

img2 = r[1][:,:,0]

img3 = r[2][:,:,0]

img4 = r[3][:,:,0]

img5 = r[4][:,:,0]

den\_img1 = z[0][:,:,0]\*255.

den\_img2 = z[1][:,:,0]\*255.

den\_img3 = z[2][:,:,0]\*255.

den\_img4 = z[3][:,:,0]\*255.

den\_img5 = z[4][:,:,0]\*255.

fig = plt.figure(figsize=(20., 10.))

grid = ImageGrid(fig, 111,  # similar to subplot(111)

                 nrows\_ncols=(2, 5),  # creates 2x5 grid of axes

                 axes\_pad=0.1,  # pad between axes in inch.)

for ax, im in zip(grid, [img1,img2,img3,img4,img5,den\_img1,den\_img2,den\_img3,den\_img4,den\_img5]):

# Iterating over the grid returns the Axes.

ax.imshow(im,cmap="gray")

plt.show()

from skimage.measure import compare\_ssim as ssim

def get\_ssim\_result(originalSet,noisySet):

    ssim\_sum = 0

    originalSet = originalSet.reshape(originalSet.shape[0],64, 64, 1)

    noisySet = noisySet.reshape(noisySet.shape[0],64, 64, 1)

for i in range(originalSet.shape[0]):

        ssim\_sum += ssim(originalSet[i], noisySet[i],data\_range=originalSet[i].max() - noisySet[i].min(), multichannel=True)

 return 1.0\*ssim\_sum/originalSet.shape[0]

get\_ssim\_result(r, z)