**Module 17**

**Name: Upadhyay Sachin Naresh**

**Batch id: AIWDEOB 30082021**

**Topic: Autoencoders and Types of autoencoders**

Write a code to denoise an image using autoencoders.You Have to use MNIST dataset and use salt and pepper noise to add noise to the image.This noisy image has to be denoised using autoencoders.

### Image De noising Auto encoder

### With Salt and Pepper Noise

Encoder: - a function f that compresses the input into a latent-space representation. f(x) = h

De coder: - a function g that reconstruct the input from the latent space representation. g (h) ~ x.

Importing dataset

from keras.datasets import mnist

Importing required libraries

import keras

from keras.models import Model

from keras.optimizers import Adadelta

from keras.layers import Input, Conv2D, MaxPool2D, UpSampling2D

from keras.callbacks import EarlyStopping

import numpy as np

import matplotlib.pyplot as plt

Preprocessing of Dataset And Adding Salt and Pepper Noise to our dataset

(train\_X, train\_y), (test\_X, test\_y) = mnist.load\_data()

Convert values from 0 to 255 into range 0 to 1.

train\_X = train\_X.astype('float32') / 255.

test\_X = test\_X.astype('float32') / 255.

train\_X = np.reshape(train\_X, (len(train\_X), 28, 28, 1))

test\_X = np.reshape(test\_X, (len(test\_X), 28, 28, 1))

noise\_factor = 0.5

#np.random.normal => random means to obtain random samples

#normal means normal or gaussian distribution,

#i.e. random sample from gaussian distribution

train\_X\_noisy = train\_X + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=train\_X.shape)

test\_X\_noisy = test\_X + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=test\_X.shape)

# to make values in the range of 0 to 1,

#if values < 0 then they will be equal to 0 and

#if values > 1 then they will be equal to 1.

train\_X\_noisy = np.clip(train\_X\_noisy, 0., 1.)

test\_X\_noisy = np.clip(test\_X\_noisy, 0., 1.)

#Defining our Image denoising autoencoder using keras

Input\_img = Input(shape=(28, 28, 1))

Encoding architecture

x1 = Conv2D(64, (3, 3), activation='relu', padding='same')(Input\_img)

x1 = MaxPool2D( (2, 2), padding='same')(x1)

x2 = Conv2D(32, (3, 3), activation='relu', padding='same')(x1)

x2 = MaxPool2D( (2, 2), padding='same')(x2)

x3 = Conv2D(16, (3, 3), activation='relu', padding='same')(x2)

encoded    = MaxPool2D( (2, 2), padding='same')(x3)

Decoding architecture

x3 = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)

x3 = UpSampling2D((2, 2))(x3)

x2 = Conv2D(32, (3, 3), activation='relu', padding='same')(x3)

x2 = UpSampling2D((2, 2))(x2)

x1 = Conv2D(64, (3, 3), activation='relu')(x2)

x1 = UpSampling2D((2, 2))(x1)

decoded   = Conv2D(1, (3, 3), padding='same')(x1)

autoencoder = Model(Input\_img, decoded)

autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy')

Structure of our Autoencoder

autoencoder.summary()

Performing Early Stopping and then fitting training and testing data to our autoencoder

from keras.callbacks import EarlyStopping

early\_stopper = EarlyStopping(monitor='val\_loss', min\_delta=0, patience=10, verbose=1, mode='auto')

Training our autoencoder and validating it on validation data

a\_e = autoencoder.fit(train\_X\_noisy, train\_X,epochs=1000,batch\_size=128,shuffle=True,validation\_data=(test\_X\_noisy, test\_X),callbacks=[early\_stopper])

Saving model

autoencoder.save('autoenc\_1000\_epoch.h5')

Making Predictions

Predict the reconstructed images for the original images...

pred = autoencoder.predict(test\_X\_noisy)

Visualizing original image

plt.figure(figsize=(10,10))

for i in range(5):

    plt.subplot(1, 5, i+1)

    plt.xticks([]) # to remove x-axis  the [] empty list indicates this

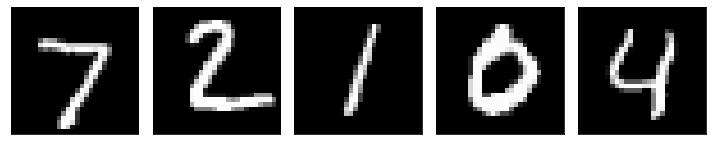
    plt.yticks([]) # to remove y-axis

    plt.grid(False) # to remove grid

    plt.imshow(test\_X[i].reshape(28, 28), cmap='gray') #display the image

plt.tight\_layout() # to have a proper space in the subplots

plt.show()



Visualizing noise image

plt.figure(figsize=(10,10))

for i in range(5):

    plt.subplot(1, 5, i+1)

    plt.xticks([]) # to remove x-axis  the [] empty list indicates this

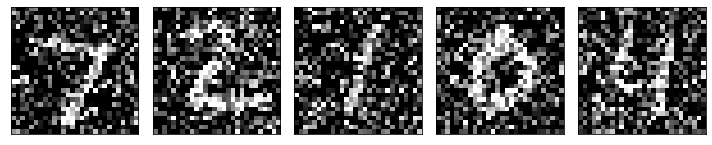
    plt.yticks([]) # to remove y-axis

    plt.grid(False) # to remove grid

    plt.imshow(test\_X\_noisy[i].reshape(28, 28), cmap='gray') #display the image

plt.tight\_layout() # to have a proper space in the subplots

plt.show()



Visualize reconstructed images(output of autoencoder)

plt.figure(figsize=(10,10))

for i in range(5):

    plt.subplot(1, 5, i+1)

    plt.xticks([]) # to remove x-axis  the [] empty list indicates this

    plt.yticks([]) # to remove y-axis

    plt.grid(False) # to remove grid

    plt.imshow(pred[i].reshape(28, 28), cmap='gray') #display the image

plt.tight\_layout() # to have a proper space in the subplots

plt.show()