```
#BE_CSE_60_Shreya Suvarna
#Experiment 05
#Implementation of autoencoder model for Image Compression.

import numpy as np
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import matplotlib.pyplot as plt


# this is the size of our encoded representations
encoding_dim = 32 # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
# configure our model to use a per-pixel binary crossentropy loss, and the Adadelta optimizer:
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')


(x_train, _), (x_test, _) = mnist.load_data()
# normalize all values between 0 and 1 and we will flatten the 28x28 images into vectors of size 784.
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
print (x_train.shape)
print (x_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
(60000, 784)
(10000, 784)
```

```
autoencoder.fit(x_train, x_train,
epochs=50,
batch_size=256,
shuffle=True,
validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

```
Epoch 1/50
235/235 [==============================] - 5s 17ms/step - loss: 0.6939 - val_loss: 0.6939
Epoch 2/50
235/235 [==============================] - 3s 11ms/step - loss: 0.6938 - val_loss: 0.6937
Epoch 3/50
235/235 [==============================] - 3s 11ms/step - loss: 0.6936 - val_loss: 0.6935
Epoch 4/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6935 - val_loss: 0.6934
Epoch 5/50
235/235 [==============================] - 4s 15ms/step - loss: 0.6933 - val_loss: 0.6932
Epoch 6/50
235/235 [==============================] - 3s 13ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 7/50
235/235 [==============================] - 3s 13ms/step - loss: 0.6930 - val_loss: 0.6929
Epoch 8/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6928 - val_loss: 0.6927
Epoch 9/50
235/235 [==============================] - 4s 16ms/step - loss: 0.6927 - val_loss: 0.6926
Epoch 10/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6925 - val_loss: 0.6924
```

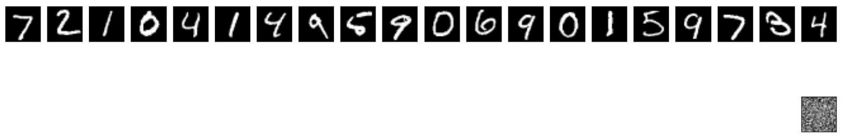
```
Epoch 11/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6924 - val_loss: 0.6923
Epoch 12/50
235/235 [==============================] - 3s 11ms/step - loss: 0.6922 - val_loss: 0.6921
Epoch 13/50
235/235 [==============================] - 4s 16ms/step - loss: 0.6921 - val_loss: 0.6920
Epoch 14/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6919 - val_loss: 0.6918
Epoch 15/50
235/235 [==============================] - 3s 11ms/step - loss: 0.6917 - val_loss: 0.6916
Epoch 16/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6916 - val_loss: 0.6915
Epoch 17/50
235/235 [==============================] - 4s 18ms/step - loss: 0.6914 - val_loss: 0.6913
Epoch 18/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6913 - val_loss: 0.6912
Epoch 19/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6911 - val_loss: 0.6910
Epoch 20/50
235/235 [==============================] - 3s 13ms/step - loss: 0.6910 - val_loss: 0.6908
Epoch 21/50
235/235 [==============================] - 4s 15ms/step - loss: 0.6908 - val_loss: 0.6907
Epoch 22/50
235/235 [==============================] - 3s 13ms/step - loss: 0.6906 - val_loss: 0.6905
Epoch 23/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6905 - val_loss: 0.6904
Epoch 24/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6903 - val_loss: 0.6902
Epoch 25/50
235/235 [==============================] - 3s 11ms/step - loss: 0.6901 - val_loss: 0.6900
Epoch 26/50
235/235 [==============================] - 4s 15ms/step - loss: 0.6900 - val_loss: 0.6898
Epoch 27/50
235/235 [==============================] - 3s 13ms/step - loss: 0.6898 - val_loss: 0.6897
Epoch 28/50
235/235 [==============================] - 3s 14ms/step - loss: 0.6896 - val_loss: 0.6895
Epoch 29/50
235/235 [==============================] - 3s 12ms/step - loss: 0.6894 - val_loss: 0.6893
```

```
n = 20 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):

# display original
 ax = plt.subplot(2, n, i + 1)
 plt.imshow(x_test[i].reshape(28, 28))
 plt.gray()
 ax.get_xaxis().set_visible(False)
 ax.get_yaxis().set_visible(False)



# display reconstruction
ax = plt.subplot(2, n, i + 1 + n)
plt.imshow(decoded_imgs[i].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.show()
```



✓ 0s completed at 9:36 AM