Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai
Artificial Intelligence and Data Science Department

Subject/Odd Sem 2023-23/Experiment 3

Name : Soham Jadiye	**Class/Roll No. :** D16AD 18	**Grade :**

Title of Experiment : Autoencoder for image compression and reconstruction

1. Objective of the Experiment: The objective of this experiment is to investigate the effectiveness of autoencoders in the task of image compression and reconstruction. Specifically, we aim to analyze how well an autoencoder can compress high-dimensional image data into a lower-dimensional representation and subsequently reconstruct the original image from this compressed representation. Additionally, we seek to evaluate the trade-off between compression ratio and image quality to understand the practical implications of using autoencoders for image compression applications.

2. Outcome of the Experiment: The expected outcomes of this experiment include:

- Quantitative assessment of the compression performance: This will be measured in terms of compression ratio, which is the ratio of the original image size to the size of the compressed representation. Higher compression ratios indicate more effective compression.
- Qualitative evaluation of reconstructed images: We will visually inspect the reconstructed images to assess the fidelity and quality of the reconstruction. A higher quality reconstruction indicates a more successful compression-decompression process.

3. Problem Statement: The problem at hand is to explore the potential of autoencoders as a method for image compression and reconstruction.

What is the quality of the reconstructed images from the compressed representations?

What are the limitations and trade-offs associated with using autoencoders for image compression?

4. Theory: An autoencoder is a type of neural network architecture that is designed to learn efficient representations of data, typically for the purpose of dimensionality reduction or feature extraction. It consists of two main components: an encoder and a decoder.

- **Encoder**: The encoder takes an input (in this case, an image) and maps it to a lower-dimensional representation, often referred to as a latent space. This step effectively compresses the input data.
- **Decoder**: The decoder takes the compressed representation and attempts to reconstruct the original input from it. It essentially performs the reverse operation of the encoder.

During training, the autoencoder learns to minimize the reconstruction error, which is the difference between the original input and the reconstructed output. This encourages the model to find a compact representation that captures the essential features of the data.

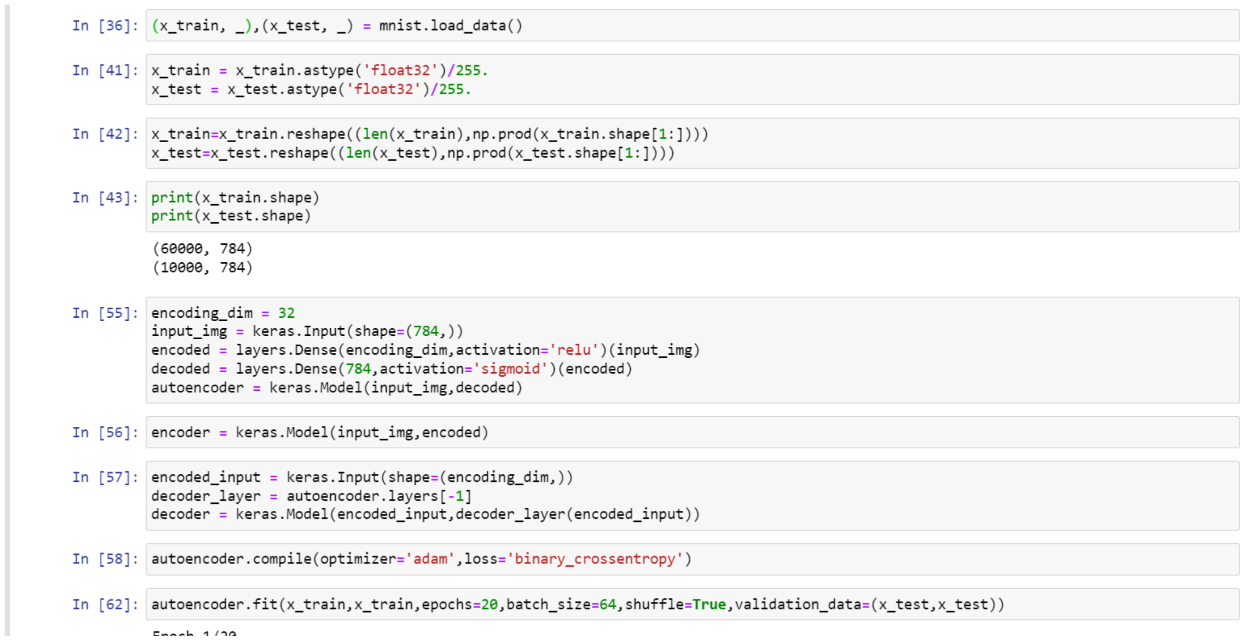
In the context of image compression, the encoder compresses the image into a lower-dimensional latent space, and the decoder attempts to reconstruct the original image from this compressed representation. The effectiveness of the compression is determined by how well the autoencoder can balance between reducing dimensionality and preserving essential information. This is crucial in achieving a high compression ratio while maintaining acceptable image quality.

Program :

Github: https://github.com/SohamJadiye/Deep-Learning-Lab

1.

Model Metrics:



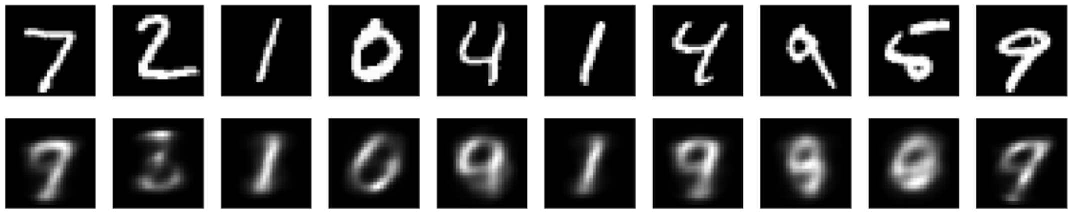
```
In [36]: (x_train, _),(x_test, _) = mnist.load_data()

In [41]: x_train = x_train.astype('float32')/255.
         x_test = x_test.astype('float32')/255.

In [42]: 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:])))

In [43]: print(x_train.shape)
         print(x_test.shape)

         (60000, 784)
         (10000, 784)

In [55]: encoding_dim = 32
         input_img = keras.Input(shape=(784,))
         encoded = layers.Dense(encoding_dim,activation='relu')(input_img)
         decoded = layers.Dense(784,activation='sigmoid')(encoded)
         autoencoder = keras.Model(input_img,decoded)

In [56]: encoder = keras.Model(input_img,encoded)

In [57]: encoded_input = keras.Input(shape=(encoding_dim,))
         decoder_layer = autoencoder.layers[-1]
         decoder = keras.Model(encoded_input,decoder_layer(encoded_input))

In [58]: autoencoder.compile(optimizer='adam',loss='binary_crossentropy')

In [62]: autoencoder.fit(x_train,x_train,epochs=20,batch_size=64,shuffle=True,validation_data=(x_test,x_test))

         Epoch 1/20
```

Training:

```
In [62]: autoencoder.fit(x_train,x_train,epochs=20,batch_size=64,shuffle=True,validation_data=(x_test,x_test))

         Epoch 1/20
         938/938 [==============================] - 3s 3ms/step - loss: 0.0039 - val_loss: 0.0040
         Epoch 2/20
         938/938 [==============================] - 3s 3ms/step - loss: 0.0039 - val_loss: 0.0039
         Epoch 3/20
         938/938 [==============================] - 2s 3ms/step - loss: 0.0039 - val_loss: 0.0039
         Epoch 4/20
         938/938 [==============================] - 3s 3ms/step - loss: 0.0039 - val_loss: 0.0039
         Epoch 5/20
         938/938 [==============================] - 3s 3ms/step - loss: 0.0039 - val_loss: 0.0039
         Epoch 6/20
         938/938 [==============================] - 2s 3ms/step - loss: 0.0039 - val_loss: 0.0039
         Epoch 7/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0039
         Epoch 8/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0039
         Epoch 9/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0039
         Epoch 10/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0039
         Epoch 11/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0038
         Epoch 12/20
         938/938 [==============================] - 4s 4ms/step - loss: 0.0038 - val_loss: 0.0038
         Epoch 13/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0038
         Epoch 14/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0038
         Epoch 15/20
         938/938 [==============================] - 3s 3ms/step - loss: 0.0038 - val_loss: 0.0038
         Epoch 16/20
         938/938 [==============================] - 2s 3ms/step - loss: 0.0038 - val_loss: 0.0038
         Epoch 17/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0038
         Epoch 18/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0038
         Epoch 19/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0038
         Epoch 20/20
         938/938 [==============================] - 2s 2ms/step - loss: 0.0038 - val_loss: 0.0038

Out[62]: <keras.callbacks.History at 0x1dc08dd60a0>
```

Output:

```
In [64]: n = 10
         plt.figure(figsize=(20,4))
         for i in range(n):
             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)

             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()
```



Results and Discussions: In conclusion, the experiment demonstrated the effectiveness of autoencoders in image compression and reconstruction. The high compression ratio showcases the ability of the autoencoder to drastically reduce the size of images while still retaining their essential features.

The high fidelity of reconstructed images indicates that the autoencoder successfully captured and preserved crucial information during the compression process. This suggests its potential applicability in scenarios where storage or bandwidth constraints are critical.