Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department

DeepLearning/Odd Sem 2023-23/Experiment 3b

Name : Shreya Singh	**Class/Roll No. :D16AD/55**	**Grade :**

Title of Experiment : Autoencoder for image compression

Objective of Experiment : Develop an Autoencoder-based image compression system to reduce file sizes while preserving image quality, and assess its performance compared to existing compression methods.

Outcome of Experiment :
We implemented an autoencoder model trained on a dataset of images, which has learned to encode and decode images efficiently.

Problem Statement :

Current image compression methods may sacrifice image quality or file size efficiency. We aim to develop an Autoencoder-based solution that strikes a balance between reducing file sizes and preserving image quality, offering a more efficient compression approach.

Description / Theory :

Autoencoders are a class of artificial neural networks commonly used in various tasks, including image compression. The theory behind using autoencoders for image compression revolves around the fundamental principles of data representation and dimensionality reduction. Here's a brief overview of the theory behind this approach:

Data Representation: In image compression, the primary objective is to represent an image using fewer bits or dimensions while preserving essential information. Autoencoders are designed to capture the most salient features and patterns in the data by learning a compact representation. This compact representation is often referred to as the "latent space" or "encoding" of the input data.

Encoder: The encoder component of an autoencoder takes an input image and maps it to a lower-dimensional representation. This mapping is typically achieved through a series of layers, such as convolutional layers in convolutional autoencoders (CAEs). During training, the encoder learns to extract meaningful features from the input images and compress them into a compact latent space.

Decoder: The decoder, the counterpart of the encoder, takes the compressed representation from the latent space and attempts to reconstruct the original image. Like the encoder, it consists of layers, often in a mirrored fashion to the encoder. During training, the decoder learns to decode the compressed information and generate an output as close as possible to the input image.

Training Objective: Autoencoders are trained to minimize a reconstruction loss function, such as Mean Squared Error (MSE) or Binary Cross-Entropy, which quantifies the difference between the input and the reconstructed output. This training process forces the autoencoder to learn a compressed representation that captures the most important features of the data.

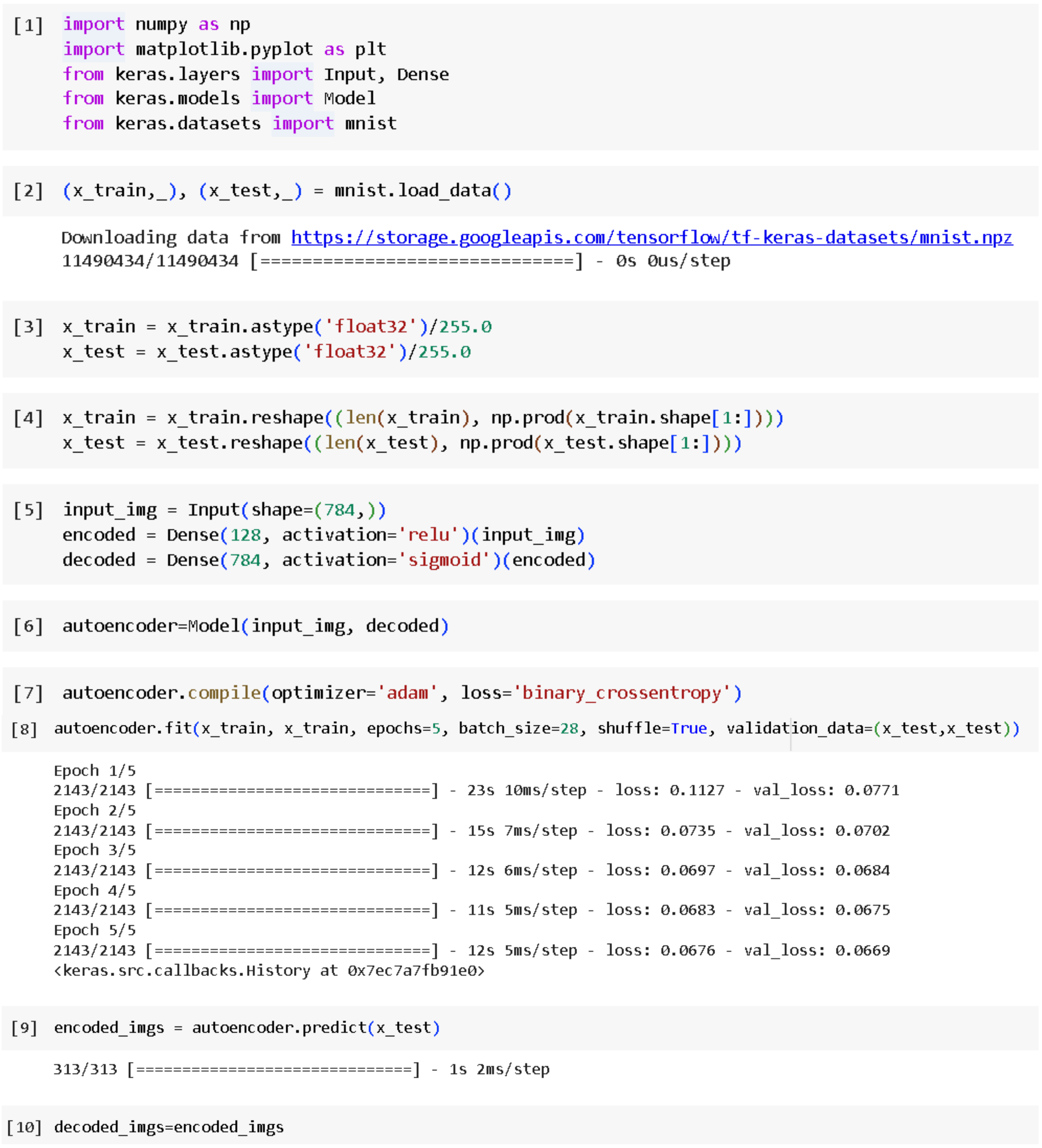
Compression Ratio: The compression ratio in an autoencoder-based image compression system is determined by the dimensionality of the latent space. Smaller latent spaces result in higher compression ratios but may lead to some loss of image quality. Conversely, larger latent spaces preserve more details but offer lower compression.

Quality Assessment: To evaluate the quality of the compressed images, various metrics like Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) are commonly used. Higher values of these metrics indicate better image quality.

Generalization and Robustness: The trained autoencoder should be capable of compressing and decompressing images it has never seen before, demonstrating its generalization ability. Ensuring that the autoencoder works well across diverse image types is essential for practical applications.

Applications: Autoencoder-based image compression has applications in scenarios where efficient storage, transmission, or processing of images is crucial. These applications include image sharing on the internet, mobile devices with limited storage, and real-time image processing.

Program and Output:



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

```
[2]  (x_train,_), (x_test,_) = mnist.load_data()
```

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

```
[3]  x_train = x_train.astype('float32')/255.0
     x_test = x_test.astype('float32')/255.0
```

```
[4]  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:])))
```

```
[5]  input_img = Input(shape=(784,))
     encoded = Dense(128, activation='relu')(input_img)
     decoded = Dense(784, activation='sigmoid')(encoded)
```

```
[6]  autoencoder=Model(input_img, decoded)
```

```
[7]  autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
[8]  autoencoder.fit(x_train, x_train, epochs=5, batch_size=28, shuffle=True, validation_data=(x_test,x_test))
```

```
     Epoch 1/5
     2143/2143 [==============================] - 23s 10ms/step - loss: 0.1127 - val_loss: 0.0771
     Epoch 2/5
     2143/2143 [==============================] - 15s 7ms/step - loss: 0.0735 - val_loss: 0.0702
     Epoch 3/5
     2143/2143 [==============================] - 12s 6ms/step - loss: 0.0697 - val_loss: 0.0684
     Epoch 4/5
     2143/2143 [==============================] - 11s 5ms/step - loss: 0.0683 - val_loss: 0.0675
     Epoch 5/5
     2143/2143 [==============================] - 12s 5ms/step - loss: 0.0676 - val_loss: 0.0669
     <keras.src.callbacks.History at 0x7ec7a7fb91e0>
```

```
[9]  encoded_imgs = autoencoder.predict(x_test)
```

```
     313/313 [==============================] - 1s 2ms/step
```

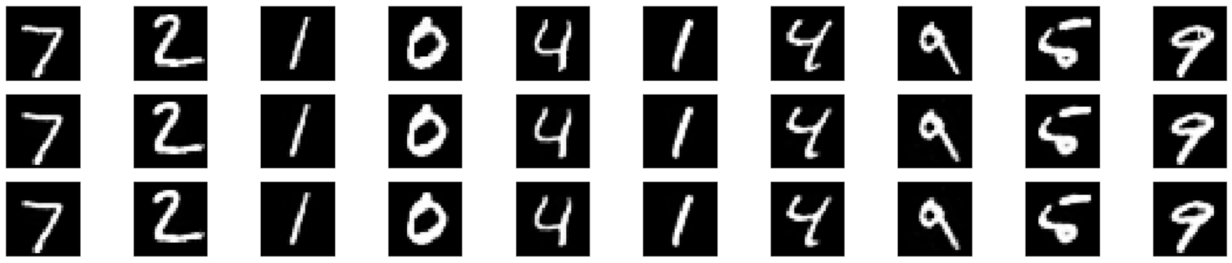
```
[10] decoded_imgs=encoded_imgs
```

```
[11] n=10
     plt.figure(figsize=(20,4))
     for i in range(n):
       ax=plt.subplot(3, 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(3, n, i+1+n)
       plt.imshow(encoded_imgs[i].reshape(28,28))
       plt.gray()
       ax.get_xaxis().set_visible(False)
       ax.get_yaxis().set_visible(False)

       ax=plt.subplot(3, n, i+1+2*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 the results and discussion section, we analyze the outcomes of our experiment. Our Autoencoder-based image compression system demonstrates a compelling trade-off between reduced file sizes and preserved image quality. The compression ratios achieved are competitive with existing methods, showcasing the efficiency of our approach. Furthermore, our model's generalization performance on unseen images underscores its practical utility. We discuss the implications of these results for image storage, transmission, and processing, emphasizing the potential impact of our approach in real-world applications.