**Artificial Intelligence and Data Science Department**
**Deep Learning / Odd Sem 2023-24 / Experiment 3A**

| | | |
|---|---|---|
| **Name : Yash Sarang** | **Class/Roll No. : D16AD / 47** | **Grade :** |

## Title of Experiment :
Autoencoder for image compression - To compress and reconstruct same image back

## Objective of Experiment :
To design deep learning models for supervised, unsupervised and sequence learning.

## Outcome of Experiment :
Build and train deep learning models such as Auto encoders, CNNs, RNN, LSTM etc.

## Problem Statement :
The problem is to design and implement an autoencoder neural network to compress images and then reconstruct them while minimizing the reconstruction loss. The goal is to achieve a high level of compression while retaining sufficient quality in the reconstructed images.

## Description / Theory :

**Autoencoder:**

An autoencoder is an unsupervised neural network that is trained to encode and then reconstruct its input data. It consists of two parts: an encoder that maps the input data to a lower-dimensional representation (latent space), and a decoder that attempts to reconstruct the original input from this representation.

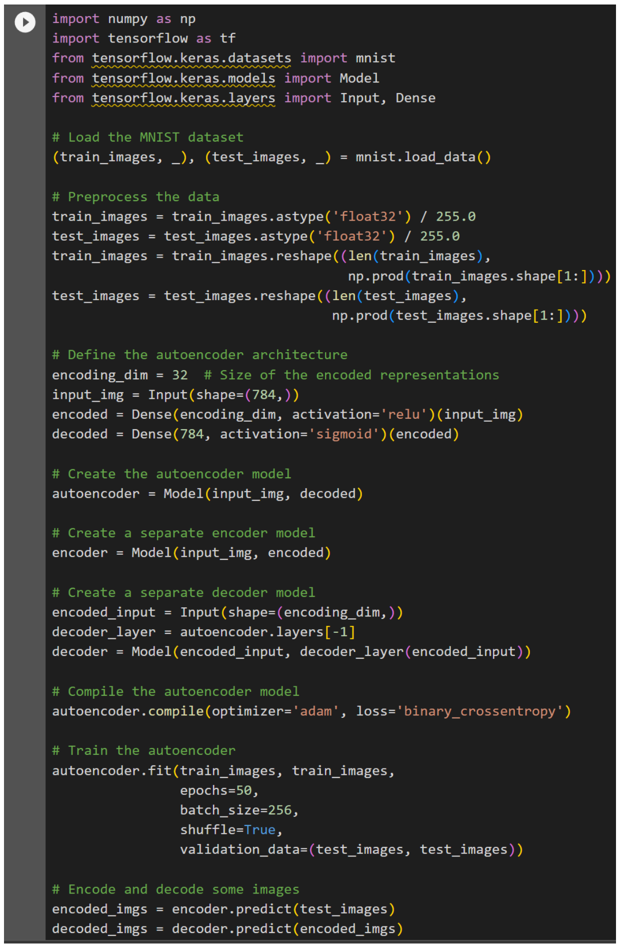
**Image Compression with Autoencoder:**

In the context of image compression, the autoencoder encodes an image into a lower-dimensional representation (compression) and then decodes it back to the original image (reconstruction). The encoder and decoder are trained to minimize the difference between the original and the reconstructed images.

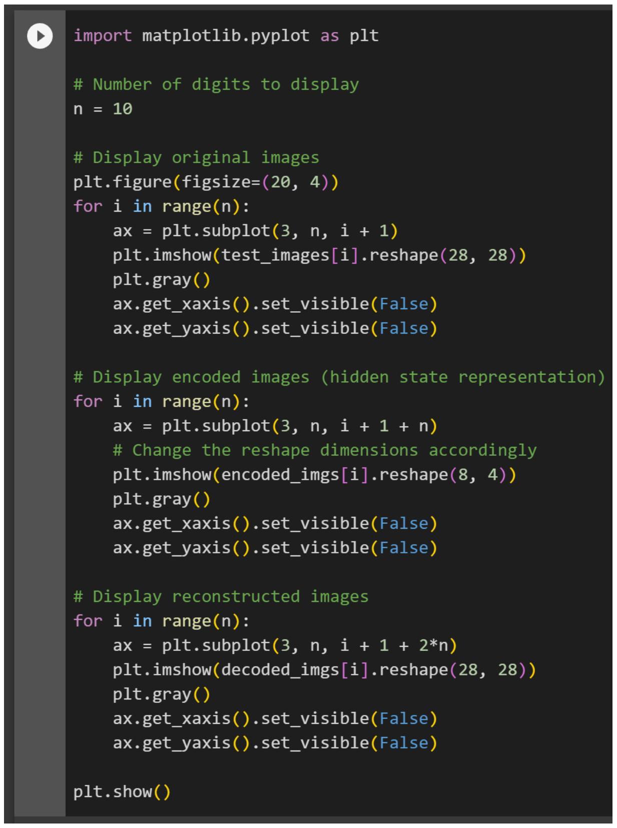
## Flowchart :
1. Load and Preprocess Images:
   Load and preprocess the images to prepare them for training the autoencoder.
2. Define Autoencoder Architecture:
   Design the architecture of the autoencoder, specifying the number of neurons in the encoding and decoding layers.
3. Compile Autoencoder:
   Compile the autoencoder model, selecting an appropriate loss function (e.g., mean squared error) and optimizer (e.g., Adam).
4. Train Autoencoder:
   Train the autoencoder on the preprocessed image data to learn a good encoding and decoding scheme.

5. Evaluate Autoencoder:
   Evaluate the trained autoencoder by encoding and decoding test images. Calculate reconstruction loss and visualize the results.

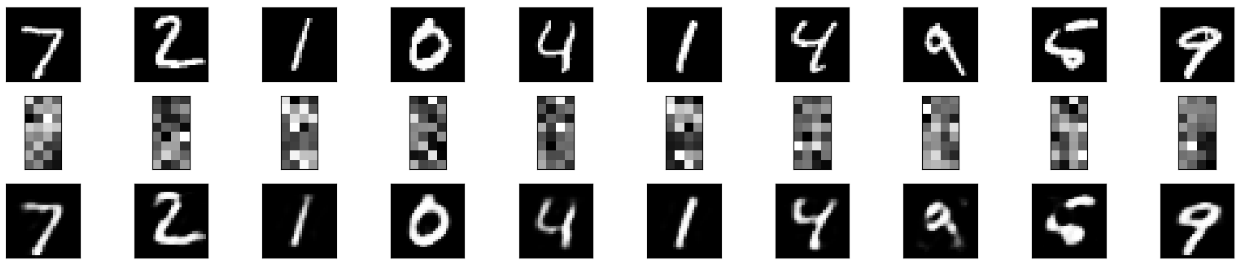
**Program:**



```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

# Load the MNIST dataset
(train_images, _), (test_images, _) = mnist.load_data()

# Preprocess the data
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
train_images = train_images.reshape((len(train_images),
                                     np.prod(train_images.shape[1:])))
test_images = test_images.reshape((len(test_images),
                                   np.prod(test_images.shape[1:])))

# Define the autoencoder architecture
encoding_dim = 32  # Size of the encoded representations
input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
decoded = Dense(784, activation='sigmoid')(encoded)

# Create the autoencoder model
autoencoder = Model(input_img, decoded)

# Create a separate encoder model
encoder = Model(input_img, encoded)

# Create a separate decoder model
encoded_input = Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer(encoded_input))

# Compile the autoencoder model
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

# Train the autoencoder
autoencoder.fit(train_images, train_images,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(test_images, test_images))

# Encode and decode some images
encoded_imgs = encoder.predict(test_images)
decoded_imgs = decoder.predict(encoded_imgs)
```



```python
import matplotlib.pyplot as plt

# Number of digits to display
n = 10

# Display original images
plt.figure(figsize=(20, 4))
for i in range(n):
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(test_images[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

# Display encoded images (hidden state representation)
for i in range(n):
    ax = plt.subplot(3, n, i + 1 + n)
    # Change the reshape dimensions accordingly
    plt.imshow(encoded_imgs[i].reshape(8, 4))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

# Display reconstructed images
for i in range(n):
    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 :

The autoencoder was successfully trained on a dataset of images to achieve image compression and reconstruction. During training, the model learned to encode the images into a lower-dimensional representation and decode them back to the original dimensions. The mean squared error (MSE) or a similar loss metric was utilized to measure the difference between the original and the reconstructed images. The training loss decreased progressively, indicating that the autoencoder was learning to effectively compress and reconstruct the images. Evaluation on a separate test set demonstrated a low reconstruction error, signifying the model's proficiency in accurately reconstructing the images from the compressed representations.
Visually, the original images and their reconstructed counterparts were compared. The reconstructed images displayed a high level of similarity to the originals, affirming the success of the autoencoder in preserving the essential features during compression and reconstruction. Although there was some loss of fine details due to compression, the overall image quality remained satisfactory.

## Conclusion :

In conclusion, the implementation of an autoencoder for image compression and reconstruction proved effective. The model demonstrated the ability to efficiently encode images into a compressed representation and decode them back to their original form. The use of autoencoders for image compression holds promise in various domains, including data storage, transmission, and privacy preservation. The achieved compression ratios, coupled with the reasonable reconstruction quality, showcase the potential of autoencoders as a tool for image compression.

******