In [ ]:

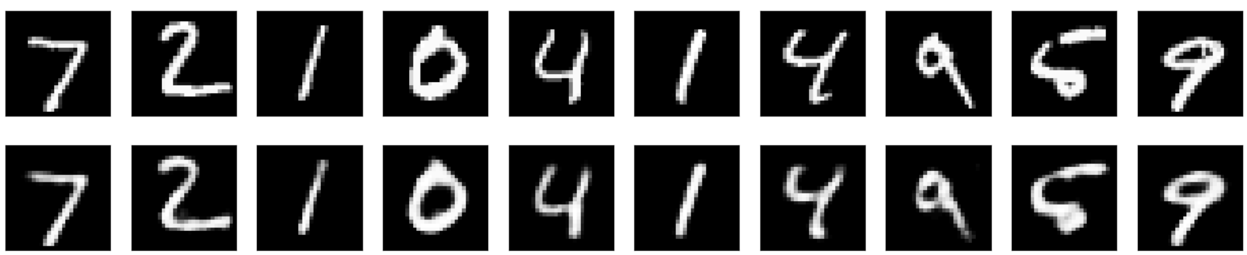
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam

(x_train, _), (x_test, _) = mnist.load_data()
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:])))

input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(), loss='binary_crossentropy')

history = autoencoder.fit(x_train, x_train,
                          epochs=50,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(x_test, x_test),
                          verbose=1)

decoded_imgs = autoencoder.predict(x_test)

plt.figure(figsize=(6,4))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

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()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datase
ts/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Epoch 1/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 8s 24ms/step - loss: 0.3364 - val_loss: 0.1626
Epoch 2/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - loss: 0.1534 - val_loss: 0.1330
Epoch 3/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 24ms/step - loss: 0.1313 - val_loss: 0.1229
Epoch 4/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.1227 - val_loss: 0.1174
Epoch 5/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.1173 - val_loss: 0.1130
Epoch 6/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - loss: 0.1131 - val_loss: 0.1095
Epoch 7/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - loss: 0.1102 - val_loss: 0.1073
Epoch 8/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.1073 - val_loss: 0.1044
Epoch 9/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 21ms/step - loss: 0.1053 - val_loss: 0.1021
Epoch 10/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.1030 - val_loss: 0.1005
Epoch 11/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - loss: 0.1014 - val_loss: 0.0993
Epoch 12/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0999 - val_loss: 0.0976
Epoch 13/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0984 - val_loss: 0.0964
Epoch 14/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - loss: 0.0968 - val_loss: 0.0950
Epoch 15/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 9s 19ms/step - loss: 0.0957 - val_loss: 0.0941
Epoch 16/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - loss: 0.0944 - val_loss: 0.0934
Epoch 17/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.0938 - val_loss: 0.0926
Epoch 18/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.0931 - val_loss: 0.0917
Epoch 19/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.0924 - val_loss: 0.0910
Epoch 20/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0917 - val_loss: 0.0906
Epoch 21/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - loss: 0.0914 - val_loss: 0.0902
Epoch 22/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 19ms/step - loss: 0.0909 - val_loss: 0.0897
Epoch 23/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.0904 - val_loss: 0.0894
Epoch 24/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 24ms/step - loss: 0.0899 - val_loss: 0.0889
Epoch 25/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.0892 - val_loss: 0.0884
Epoch 26/50
```

```
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0890 - val_loss: 0.0883
Epoch 27/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - loss: 0.0887 - val_loss: 0.0886
Epoch 28/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0886 - val_loss: 0.0877
Epoch 29/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.0884 - val_loss: 0.0876
Epoch 30/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - loss: 0.0877 - val_loss: 0.0873
Epoch 31/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.0877 - val_loss: 0.0869
Epoch 32/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 21ms/step - loss: 0.0874 - val_loss: 0.0870
Epoch 33/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.0874 - val_loss: 0.0864
Epoch 34/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0870 - val_loss: 0.0864
Epoch 35/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 23ms/step - loss: 0.0870 - val_loss: 0.0862
Epoch 36/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.0866 - val_loss: 0.0860
Epoch 37/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0861 - val_loss: 0.0858
Epoch 38/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - loss: 0.0862 - val_loss: 0.0856
Epoch 39/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0861 - val_loss: 0.0855
Epoch 40/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.0859 - val_loss: 0.0855
Epoch 41/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 8s 33ms/step - loss: 0.0855 - val_loss: 0.0851
Epoch 42/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 10s 30ms/step - loss: 0.0857 - val_loss: 0.0850
Epoch 43/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 8s 19ms/step - loss: 0.0853 - val_loss: 0.0850
Epoch 44/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - loss: 0.0851 - val_loss: 0.0848
Epoch 45/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0850 - val_loss: 0.0846
Epoch 46/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.0850 - val_loss: 0.0844
Epoch 47/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - loss: 0.0850 - val_loss: 0.0843
Epoch 48/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0845 - val_loss: 0.0842
Epoch 49/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 20ms/step - loss: 0.0845 - val_loss: 0.0843
Epoch 50/50
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step - loss: 0.0845 - val_loss: 0.0841
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```


Training vs Validation Loss
Training Loss
Validation Loss
Loss
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0
10
20
30
40
50
Epochs



Exp-10: Perform Compression on MNIST Dataset Using Autoencoder.

Aim:
To implement an Autoencoder neural network that compresses and reconstructs handwritten digit images from the MNIST dataset, demonstrating data dimensionality reduction and unsupervised feature learning.

Description
An Autoencoder is an unsupervised neural network architecture that learns to compress data into a lower-dimensional latent space and then reconstruct it as accurately as possible. The model consists of two main components:

→ Encoder: This part compresses the input image into a latent-space representation by progressively reducing its dimensionality.

→ Decoder: This part reconstructs the original input from the compressed latent code, aiming to minimize the reconstruction loss.

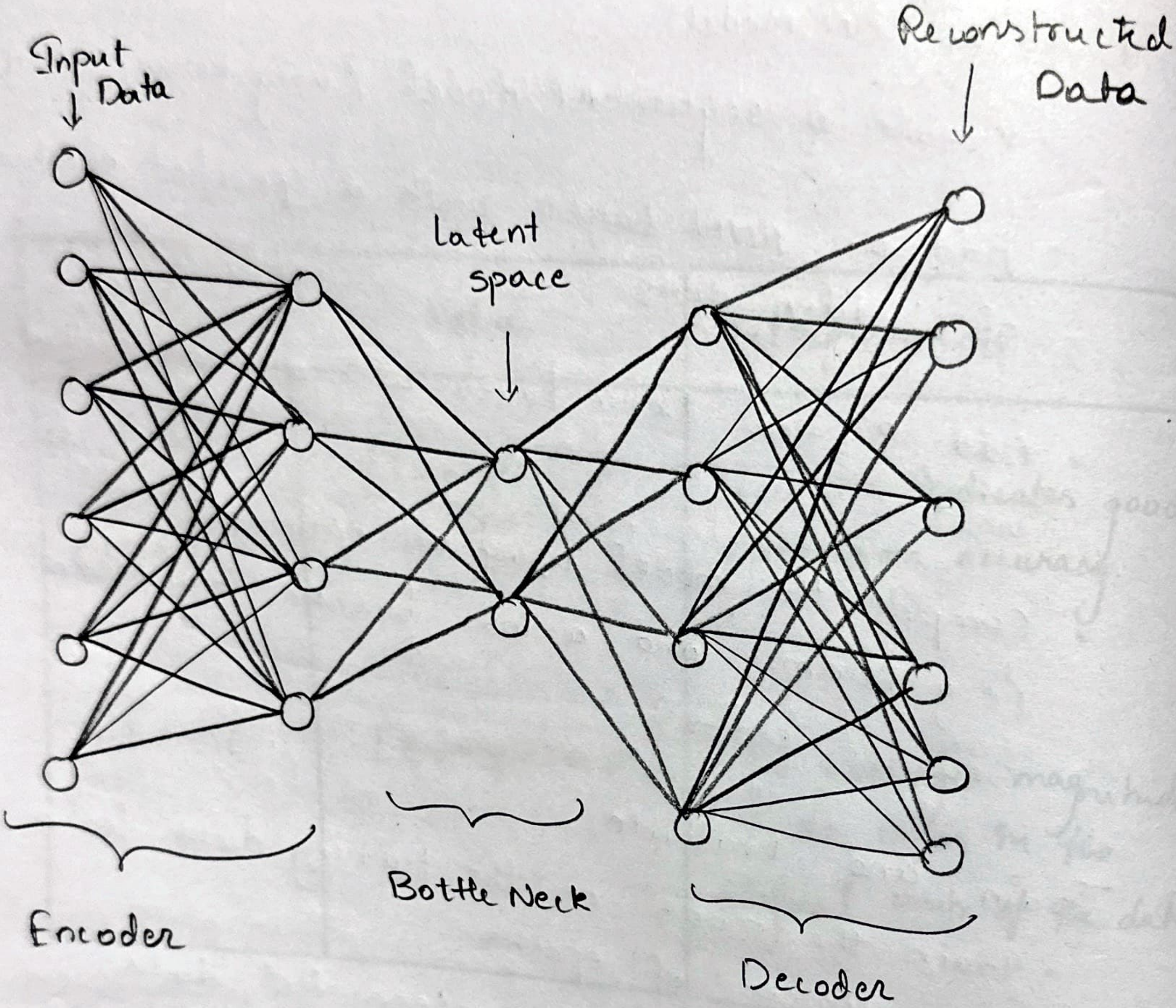
Mathematically, the autoencoder learns function f and g such that

$$x' = g(f(x))$$

where x is the input and x' is the reconstructed output.

The loss function (MSE) is minimized to make

$x' \approx x$



Autoencoder Architecture

Procedure

1.) Load and normalize MNIST data.

2.) Flatten each 28x28 image into a 784-dimensional vector.

3.) Define an encoder that reduces data to a small latent dimension (e.g 32)

4.) Define a decoder that reconstructs the input from the latent vector.

5.) Compile the model using Adam optimizer and binary-crossentropy loss.

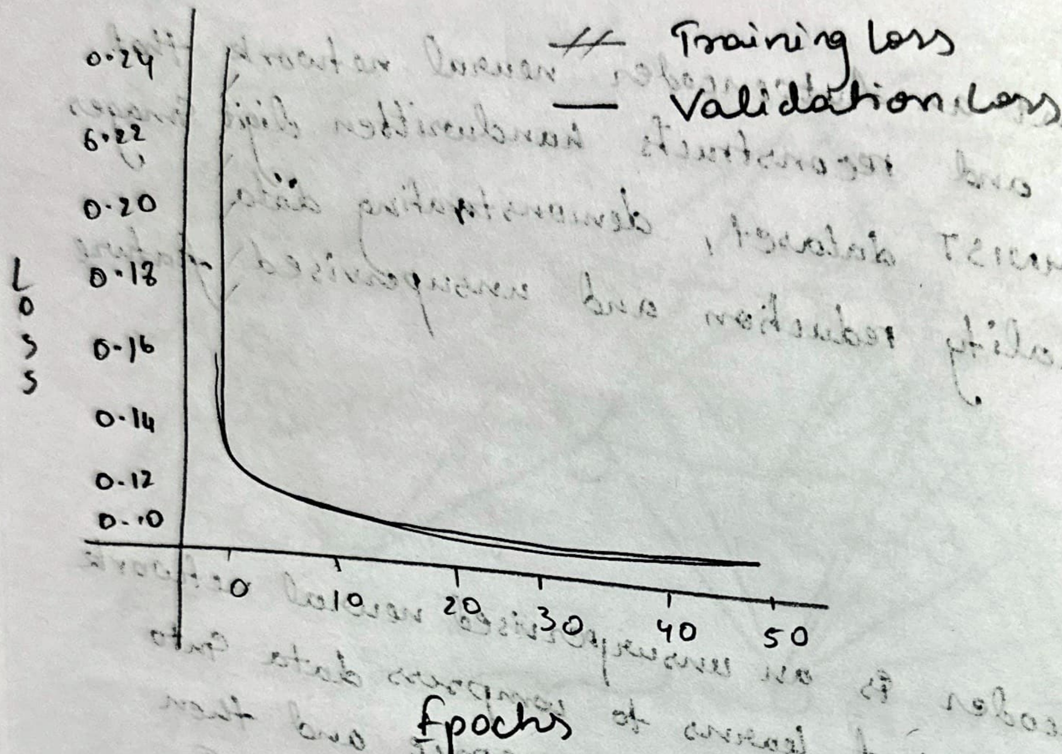
6.) Train for 50 epochs and visualize original vs reconstructed images.

Pseudocode

Load MNIST dataset
Normalize and Flatten images
Build encoder (784 → 128 → 64 → 32)
Build decoder (32 → 64 → 128 → 784)
Compile autoencoder
Train on MNIST data
Display original and reconstructed images

Observation:
Reconstructed images are slightly blurred but preserve main digit shapes.

Result:
Autoencoder successfully compressed and reconstructed MNIST images, proving its ability to perform unsupervis[ed] feature extraction and dimensionality reduction.



Output:

Epoch [1/5], Loss: 164.3796

Epoch [2/5], Loss: 121.5539

Epoch [3/5], Loss: 114.5693

Epoch [4/5], Loss: 111.5922

Epoch [5/5], Loss: 109.8722