**Experiment: -9**

**AIM:** Implementing Autoencoder

Code:

import numpy as np

import tensorflow as tf

from tensorflow.keras.layers import Input, Dense

from tensorflow.keras.models import Model

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

# Load MNIST dataset

(x\_train, \_), (x\_test, \_) = mnist.load\_data()

# Normalize the images to the range [0, 1]

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

# Flatten the images to 1D (28\*28)

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:])))

# Define the size of the encoded representation

encoding\_dim = 32

# 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))

# Compile the autoencoder

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

# Train the autoencoder

autoencoder.fit(x\_train, x\_train,

epochs=50,

batch\_size=256,

shuffle=True,

validation\_data=(x\_test, x\_test))

# Encode and decode some digits

encoded\_imgs = encoder.predict(x\_test)

decoded\_imgs = decoder.predict(encoded\_imgs)

# Display the original and reconstructed images

n = 10 # Number of digits to 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()