PRACTICAL-4

PROBLEM STATEMENT:

4.Use Autoencoder to implement anomaly detection. Build the model by using:

a. Import required libraries

b. Upload / access the dataset

c. Encoder converts it into latent representation

d. Decoder networks convert it back to the original input

e. Compile the models with Optimizer, Loss, and Evaluation Metrics

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CODE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# a. IMPORT REQUIRED LIBRARIES

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers, models

import matplotlib.pyplot as plt

import random

# b. LOAD THE DATASET (MNIST)

# Using MNIST digits dataset, which has images of digits as normal data

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

# Normalize and reshape data

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

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

x\_train = x\_train.reshape((len(x\_train), 28, 28, 1))

x\_test = x\_test.reshape((len(x\_test), 28, 28, 1))

# c. BUILD THE ENCODER

input\_img = tf.keras.Input(shape=(28, 28, 1))

x = layers.Flatten()(input\_img)

x = layers.Dense(128, activation='relu')(x)

latent = layers.Dense(64, activation='relu')(x) # Latent representation

# d. BUILD THE DECODER

x = layers.Dense(128, activation='relu')(latent)

x = layers.Dense(28 \* 28, activation='sigmoid')(x)

output\_img = layers.Reshape((28, 28, 1))(x)

# Combine Encoder and Decoder into Autoencoder Model

autoencoder = models.Model(input\_img, output\_img)

# e. COMPILE THE MODEL

autoencoder.compile(optimizer='adam', loss='mse', metrics=['accuracy'])

# TRAIN THE MODEL

history = autoencoder.fit(x\_train, x\_train, epochs=10, batch\_size=64, validation\_split=0.1)

# EVALUATE THE MODEL FOR ANOMALY DETECTION

reconstructions = autoencoder.predict(x\_test)

reconstruction\_error = np.mean(np.square(x\_test - reconstructions), axis=(1, 2, 3))

# Set an anomaly threshold (mean + 2 std deviations)

threshold = np.mean(reconstruction\_error) + 2 \* np.std(reconstruction\_error)

anomalies = reconstruction\_error > threshold

# f. VISUALIZE RESULTS

# Plot a histogram of reconstruction errors

plt.hist(reconstruction\_error, bins=50)

plt.axvline(threshold, color='r', linestyle='--', label='Anomaly threshold')

plt.xlabel("Reconstruction Error")

plt.ylabel("Number of Samples")

plt.legend()

plt.show()

# Show random normal and anomalous samples

normal\_samples = x\_test[~anomalies]

anomalous\_samples = x\_test[anomalies]

print("Number of anomalies detected:", len(anomalous\_samples))

# Display a random normal sample

n = random.randint(0, len(normal\_samples) - 1)

plt.imshow(normal\_samples[n].reshape(28, 28), cmap='gray')

plt.title("Random Normal Sample")

plt.show()

# Display a random anomaly sample, if any

if len(anomalous\_samples) > 0:

n = random.randint(0, len(anomalous\_samples) - 1)

plt.imshow(anomalous\_samples[n].reshape(28, 28), cmap='gray')

plt.title("Random Anomalous Sample")

plt.show()

else:

print("No anomalies detected in the test dataset.")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*