Experiment 2

AIM: Implementing DNN.

Code:

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import numpy as np

# 1. Data Preparation

# Load the dataset

(x\_train, y\_train), (x\_test, y\_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 vectors of size 784 (28\*28)

x\_train = x\_train.reshape((x\_train.shape[0], 28 \* 28))

x\_test = x\_test.reshape((x\_test.shape[0], 28 \* 28))

# One-hot encode the labels

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

# 2. Model Definition

# Define the model

model = Sequential()

# Add input layer and first hidden layer with 512 neurons and ReLU activation

model.add(Dense(512, activation='relu', input\_shape=(28 \* 28,)))

# Add second hidden layer with 512 neurons and ReLU activation

model.add(Dense(512, activation='relu'))

# Add third hidden layer with 256 neurons and ReLU activation

model.add(Dense(256, activation='relu'))

# Add fourth hidden layer with 128 neurons and ReLU activation

model.add(Dense(128, activation='relu'))

# Add fifth hidden layer with 64 neurons and ReLU activation

model.add(Dense(64, activation='relu'))

# Add the output layer with 10 neurons (one for each class) and softmax activation

model.add(Dense(10, activation='softmax'))

# 3. Compilation

# Compile the model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# 4. Training

# Train the model

model.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_split=0.2)

# 5. Evaluation

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_accuracy}')

# 6. Prediction

# Predict on the test set

predictions = model.predict(x\_test)

# Example: print the predicted and true labels for the first test image

print(f'Predicted label: {np.argmax(predictions[0])}, True label: {np.argmax(y\_test[0])}')