**Question**

Implement classification on the Iris Dataset using Multi Layer Perceptron.

**Source Code**

import tensorflow as tf

from tensorflow import keras

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import OneHotEncoder

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from matplotlib import pyplot as plt

iris = load\_iris()

X = iris.data

y = iris.target.reshape(-1, 1)

encoder = OneHotEncoder(sparse\_output=False)

y = encoder.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

def build\_model(hidden\_units=10):

model = keras.Sequential()

model.add(keras.layers.Dense(hidden\_units, input\_shape=(4,), activation='relu'))

model.add(keras.layers.Dense(hidden\_units, activation='relu'))

model.add(keras.layers.Dense(3, activation='softmax'))

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

return model

def train\_model(model, epochs=100, batch\_size=8, validation\_split=0.2):

early\_stopping = keras.callbacks.EarlyStopping(

monitor='val\_loss',

patience=10,

restore\_best\_weights=True

)

reduce\_lr = keras.callbacks.ReduceLROnPlateau(

monitor='val\_loss',

factor=0.1,

patience=5

)

best\_model = keras.callbacks.ModelCheckpoint(

'model.keras',

save\_best\_only=True

)

hist = model.fit(

X\_train, y\_train,

epochs=epochs,

batch\_size=batch\_size,

validation\_split=validation\_split,

callbacks=[early\_stopping, reduce\_lr, best\_model]

)

return hist

def evaluate(hist, model, X\_test, y\_test, model\_name='Multi Layer Perceptron'):

y\_pred = model.predict(X\_test)

y\_pred\_classes = tf.argmax(y\_pred, axis=1)

y\_test\_classes = tf.argmax(y\_test, axis=1)

accuracy = accuracy\_score(y\_test\_classes, y\_pred\_classes)

precision = precision\_score(y\_test\_classes, y\_pred\_classes, average='weighted')

recall = recall\_score(y\_test\_classes, y\_pred\_classes, average='weighted')

f1 = f1\_score(y\_test\_classes, y\_pred\_classes, average='weighted')

print(f'Model: {model\_name}')

print(f'Accuracy: {accuracy:.4f}')

print(f'Precision: {precision:.4f}')

print(f'Recall: {recall:.4f}')

print(f'F1 Score: {f1:.4f}')

print()

plt.plot(hist.history['accuracy'], label='Train Accuracy')

plt.plot(hist.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(loc='upper left')

plt.show()

plt.plot(hist.history['loss'], label='Train Loss')

plt.plot(hist.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(loc='upper left')

plt.show()

def main():

model = build\_model(hidden\_units=10)

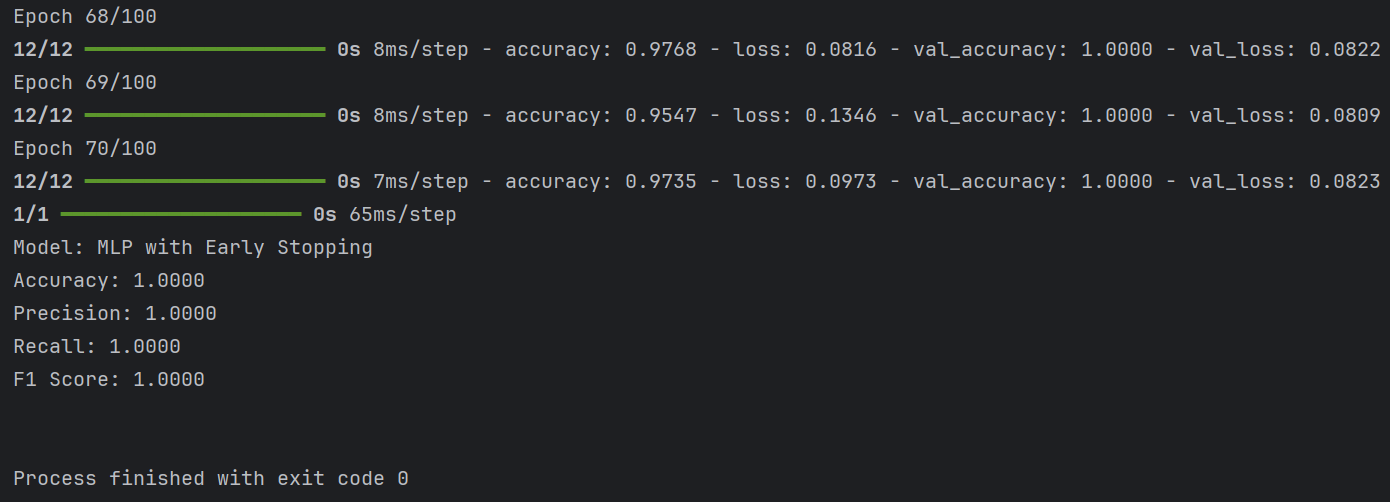
hist = train\_model(model, epochs=100, batch\_size=8, validation\_split=0.2)

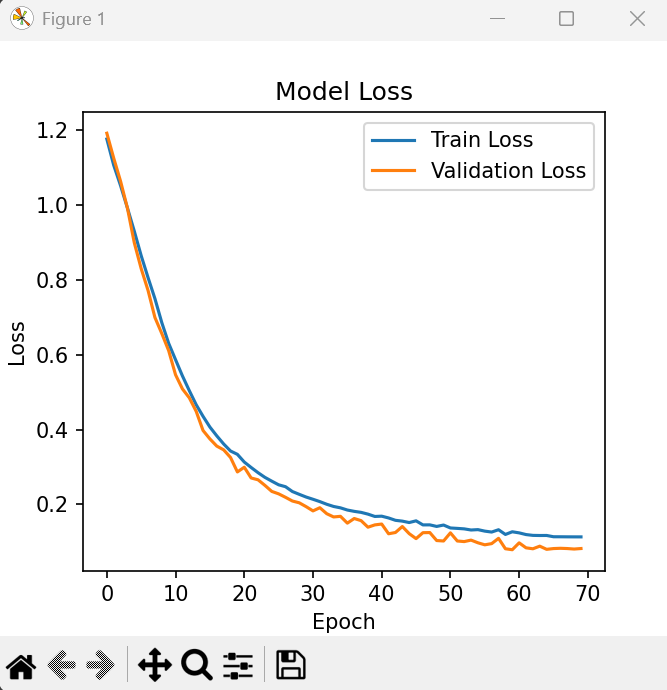
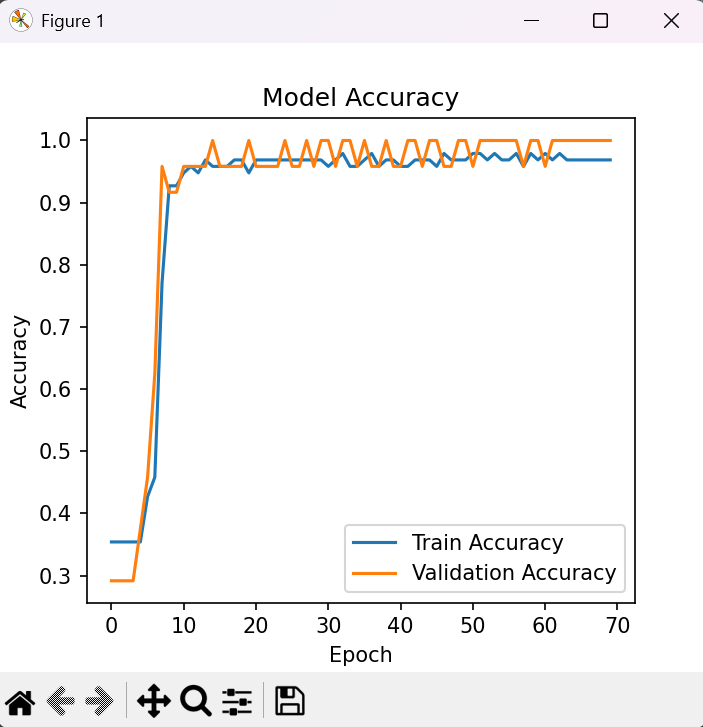
evaluate(hist, model, X\_test, y\_test, model\_name='MLP with Early Stopping')

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Output**

*Terminal*

 Plot training and validation accuracy *Plot training and validation loss*