Dl -1st

import pandas as pd

import tensorflow as tf

import keras

df=pd.read\_csv('1\_boston\_housing.csv')

print(df.head())

print(df.columns)

from sklearn.preprocessing import StandardScaler

X=df.drop('MEDV',axis=1)

Y=df['MEDV']

scaler = StandardScaler()

scaler.fit\_transform(X)

print(X[:5])

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

# Split the data into training and testing sets

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.3, random\_state=42)

print('Training set shape:', X\_train.shape, Y\_train.shape)

print('Testing set shape:', X\_test.shape, Y\_test.shape)

from keras.models import Sequential

from keras.layers import Dense, Dropout

print(model.summary())

from keras.losses import MeanSquaredError

from keras.optimizers import Adam

model.compile(loss='mean\_squared\_error', optimizer=Adam(), metrics=['mean\_absolute\_error'])

from keras.callbacks import EarlyStopping

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5)

from keras.callbacks import EarlyStopping

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

history = model.fit(X\_train, Y\_train, validation\_split=0.2, epochs=100, batch\_size=32, callbacks=[early\_stopping])

import matplotlib.pyplot as plt

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

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

plt.title('Model Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

mae = model.evaluate (X\_test, Y\_test)

# Print the mean absolute error

print('Mean Absolute Error:', mae)

predictions = model.predict(X\_test)

print("Predicted house prices:")

for i in range(5): # Print predictions for the first 5 houses

print(f"Prediction {i+1}: ${predictions[i][0]:.2f}")

dL-2nd

import numpy as np

import tensorflow as tf

from keras.datasets import imdb

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Sequential

from keras.layers import Embedding, Bidirectional, LSTM, Dense

# Load the IMDB dataset

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data()

max\_len = 250

x\_train= pad\_sequences(x\_train, maxlen=max\_len)

x\_test = pad\_sequences(x\_test, maxlen=max\_len)

# Define the deep neural network architecture

model = Sequential()

model.add(Embedding(input\_dim=10000, output\_dim=128, input\_length=max\_len)) model.add(Bidirectional(LSTM(64, return\_sequences=True)))

model.add(Bidirectional(LSTM(32)))

model.add(Dense(1, activation='sigmoid'))

# Compile the model

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

print(x\_train.min(), x\_train.max())

dl-3rd

import tensorflow as tf

from tensorflow import keras

import numpy as np

import matplotlib.pyplot as plt

import keras

fashion\_mnist = keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

# Normalize the images

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

# Define the model

model = keras.Sequential([

keras.layers.Conv2D(64, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

keras.layers.MaxPooling2D(2, 2),

keras.layers.Flatten(),

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

keras.layers.Dense(10, activation='softmax')

])

# Compile the model with the correct loss function

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

#train the model

model.fit(train\_images, train\_labels, epochs=10)

#evaluate the model

test\_loss, test\_acc=model.evaluate(test\_images, test\_labels)

print('Test accuracy', test\_acc)

#make predictions

predictions= model.predict(test\_images)

predicted\_labels= np.argmax(predictions, axis=1)

import matplotlib.pyplot as plt

num\_rows = 5

num\_cols = 5

num\_images = num\_rows \* num\_cols

plt.figure(figsize=(2 \* 2 \* num\_cols, 2 \* num\_rows))

for i in range(num\_images):

plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 1)

plt.imshow(test\_images[i], cmap='gray')

plt.axis('off')

plt.subplot(num\_rows, 2 \* num\_cols, 2 \* i + 2)

# Replace `predictions` with your actual prediction array

plt.bar(range(10), predictions[i])

plt.xticks(range(10))

plt.ylim([0, 1])

plt.tight\_layout()

plt.title(f"Predicted label: {predictions[i]}")

plt.show()