**Deep-Learning**

# Practical 4

**Aim : Basics of Tensorflow for Nueral network**

Code:

import numpy as np import tensorflow as tf

from tensorflow.keras import layers, models

# Step 1: Create a small dataset

x\_train = np.array([[0.1, 0.2], [0.4, 0.3], [0.5, 0.6], [0.9,

0.8], [0.7, 0.3],

[0.2, 0.1], [0.8, 0.5], [0.4, 0.6], [0.3,

0.7], [0.6, 0.9]]) y\_train = (x\_train[:, 1] > x\_train[:, 0]).astype(int)

# Step 2: Define the Model model = models.Sequential([

layers.Dense(8, activation='relu', input\_shape=(2,)),

layers.Dense(1, activation='sigmoid')

])

# Step 3: Compile the Model model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Step 4: Train the Model

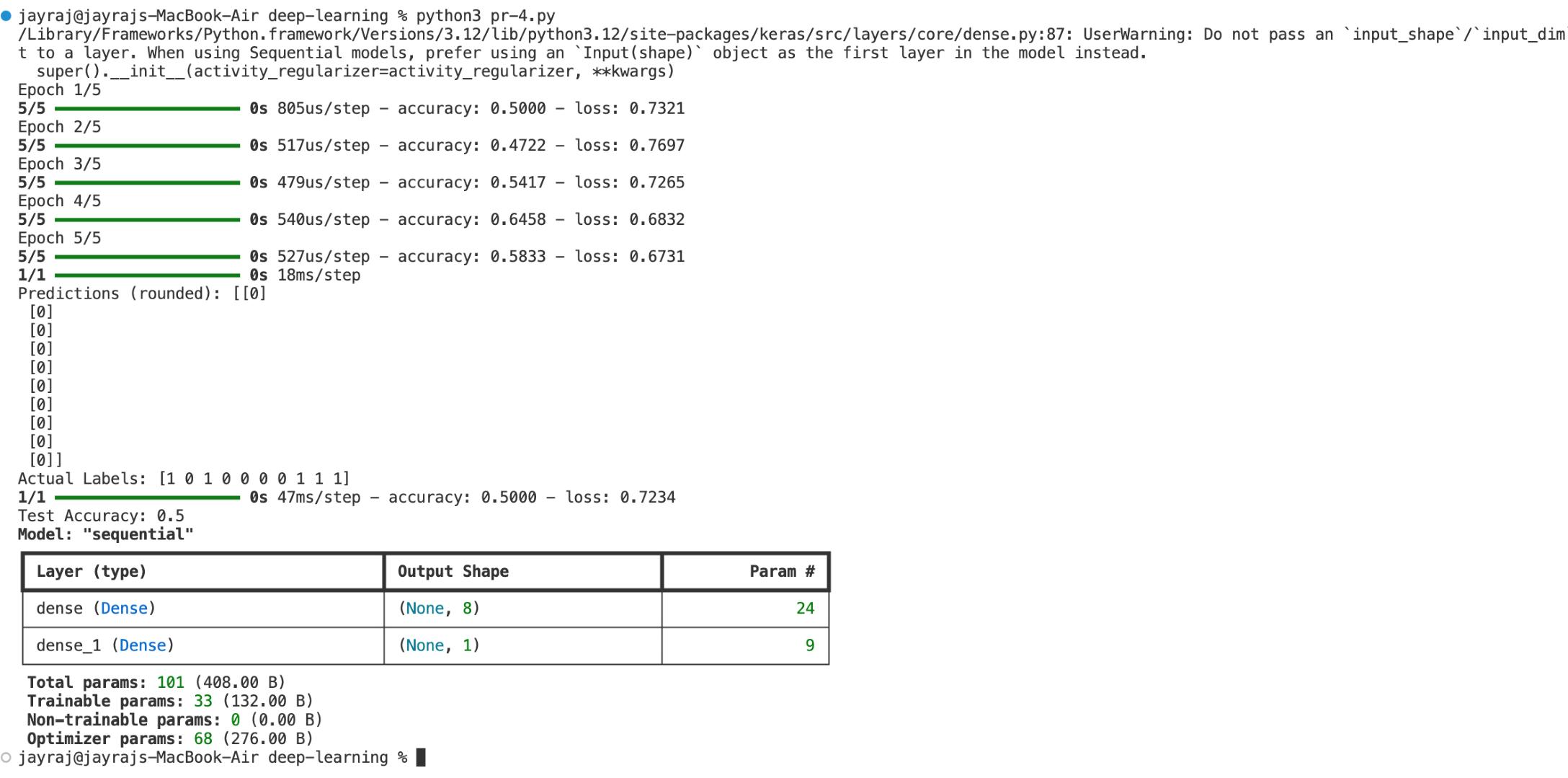
model.fit(x\_train, y\_train, epochs=5, batch\_size=2)

# Step 5: Make Predictions predictions = model.predict(x\_train) print("Predictions (rounded):", np.round(predictions).astype(int)) print("Actual Labels:", y\_train)

# Step 6: Evaluate the Model

test\_loss, test\_acc = model.evaluate(x\_train, y\_train) print(f'Test Accuracy: {test\_acc}')

# To visualize the model structure model.summary()

Output

# Practical 5

**Aim : Write a python program to implement perceptron using tensorflow** Code:

import numpy as np import tensorflow as tf

from tensorflow.keras import layers, models

x\_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) y\_train = np.array([[0], [0], [0], [1]])

model = models.Sequential([

layers.Dense(1, activation='sigmoid', input\_shape=(2,))

])

weights = np.array([[0.4], [0.6]]) bias = np.array([0.2])

# Set weights and bias for the Dense layer model.layers[0].set\_weights([weights, bias])

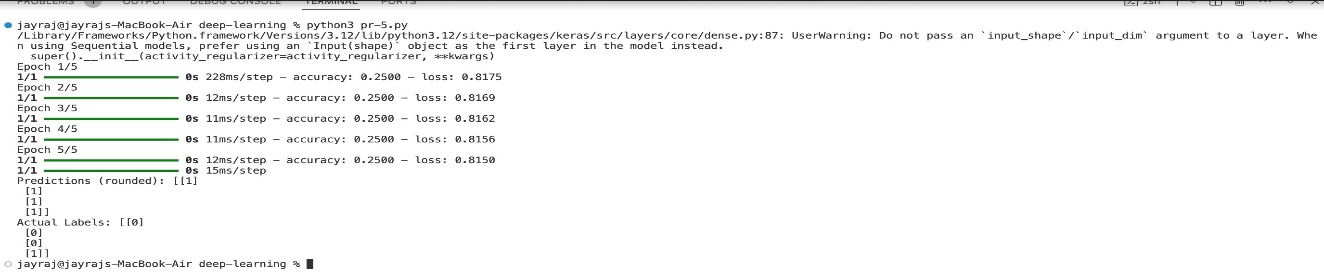
model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, epochs=5)

# Make predictions

predictions = model.predict(x\_train) print("Predictions (rounded):", np.round(predictions).astype(int)) print("Actual Labels:", y\_train)

Output:

**Practical 6**

**Aim : Write a program to implement an autoencoder for image reconstruction**

**Code:**

import numpy as np import os

import matplotlib.pyplot as plt from tensorflow.keras import layers, models import pandas as pd

def load\_fashion\_mnist\_data():

base\_path = './fashion-mnist/'

# Load data directly if CSV format is provided x\_train = pd.read\_csv(os.path.join(base\_path, 'fashion-

mnist\_train.csv')).values[:, 1:] / 255.0

x\_test = pd.read\_csv(os.path.join(base\_path, 'fashion-

mnist\_test.csv')).values[:, 1:] / 255.0 return x\_train, x\_test

x\_train, x\_test = load\_fashion\_mnist\_data() encoding\_dim = 64

input\_img = layers.Input(shape=(784,))

encoded = layers.Dense(encoding\_dim, activation='relu')

(input\_img) decoded = layers.Dense(784, activation='sigmoid')(encoded)

# Combine encoder and decoder into the autoencoder model autoencoder = models.Model(input\_img, decoded)

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

autoencoder.fit(x\_train, x\_train, epochs=5, batch\_size=256, shuffle=True,

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

# Encode and decode some images (reconstruction) decoded\_imgs = autoencoder.predict(x\_test)

# Reshape images back to 28x28 for visualization

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

n = 10

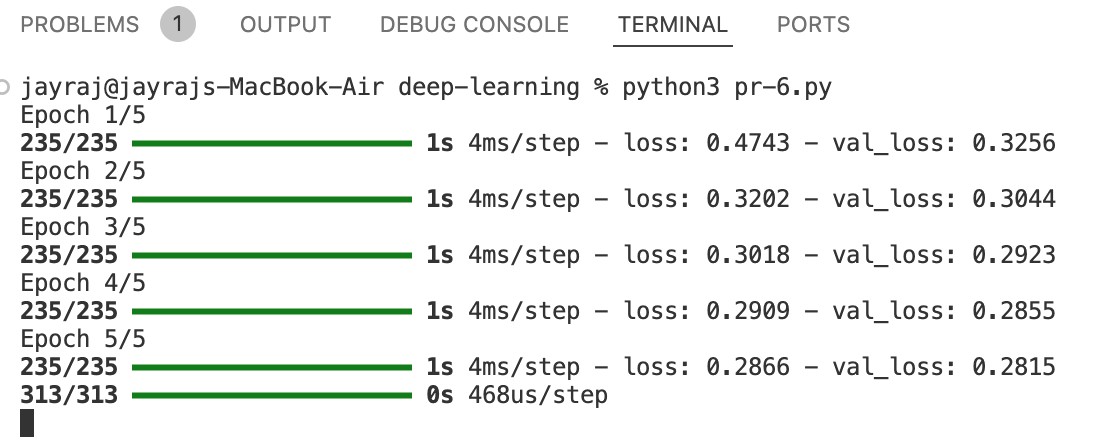
plt.figure(figsize=(20, 4)) for i in range(n):

# Display original images ax = plt.subplot(2, n, i + 1)

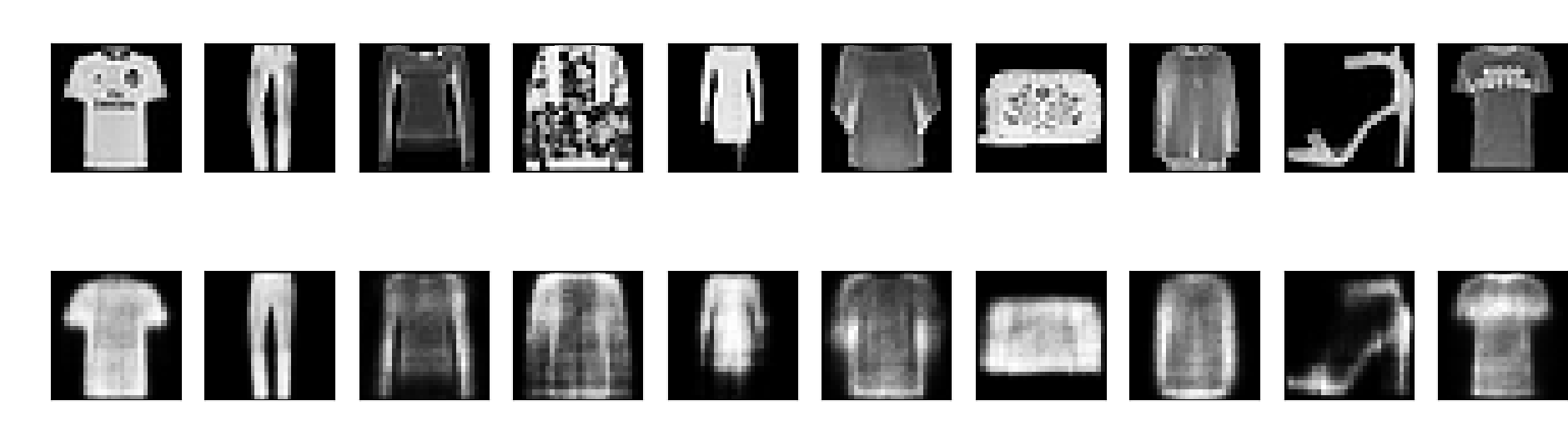
plt.imshow(x\_test[i].reshape(28, 28), cmap='gray')

ax.get\_xaxis().set\_visible(False) ax.get\_yaxis().set\_visible(False)

# Display reconstructed images

ax = plt.subplot(2, n, i + 1 + n) plt.imshow(decoded\_imgs[i], cmap='gray') ax.get\_xaxis().set\_visible(False) ax.get\_yaxis().set\_visible(False) plt.show() Output:- 

original images



reconstructed images

# Practical 7

**Aim :Write a program in python for image classification using CNN (using tensorflow.**

Code

import numpy as np import os

import matplotlib.pyplot as plt

import pandas as pd import tensorflow as tf

from tensorflow.keras import layers, models

# Load the Fashion MNIST dataset def load\_fashion\_mnist\_data():

base\_path = './fashion-mnist/'

x\_train = pd.read\_csv(os.path.join(base\_path, 'fashion-

mnist\_train.csv')).values[:, 1:]

x\_test = pd.read\_csv(os.path.join(base\_path, 'fashion-

mnist\_test.csv')).values[:, 1:]

# Reshape data to 28x28 and scale to [0, 1] x\_train = x\_train.reshape(-1, 28, 28, 1) / 255.0 x\_test = x\_test.reshape(-1, 28, 28, 1) / 255.0

# Extract labels

y\_train = pd.read\_csv(os.path.join(base\_path, 'fashion-

mnist\_train.csv')).values[:, 0]

y\_test = pd.read\_csv(os.path.join(base\_path, 'fashionmnist\_test.csv')).values[:, 0] return x\_train, y\_train, x\_test, y\_test

x\_train, y\_train, x\_test, y\_test = load\_fashion\_mnist\_data()

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu',

input\_shape=(28, 28, 1)), layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation=‘relu'), layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax') # 10 classes for

Fashion MNIST

])

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=5, batch\_size=300, validation\_split=0.1)

test\_loss, test\_acc = model.evaluate(x\_test, y\_test) print('Test accuracy:', test\_acc) predictions = model.predict(x\_test)

def plot\_predictions(x, y\_true, y\_pred, class\_names): plt.figure(figsize=(10, 10)) for i in range(25): plt.subplot(5, 5, i + 1)

plt.imshow(x[i].reshape(28, 28), cmap='gray') plt.title(f"True: {class\_names[y\_true[i]]}\nPred:

{class\_names[np.argmax(y\_pred[i])]}")

plt.axis('off') plt.tight\_layout() plt.show()

# Define class names for Fashion MNIST

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',

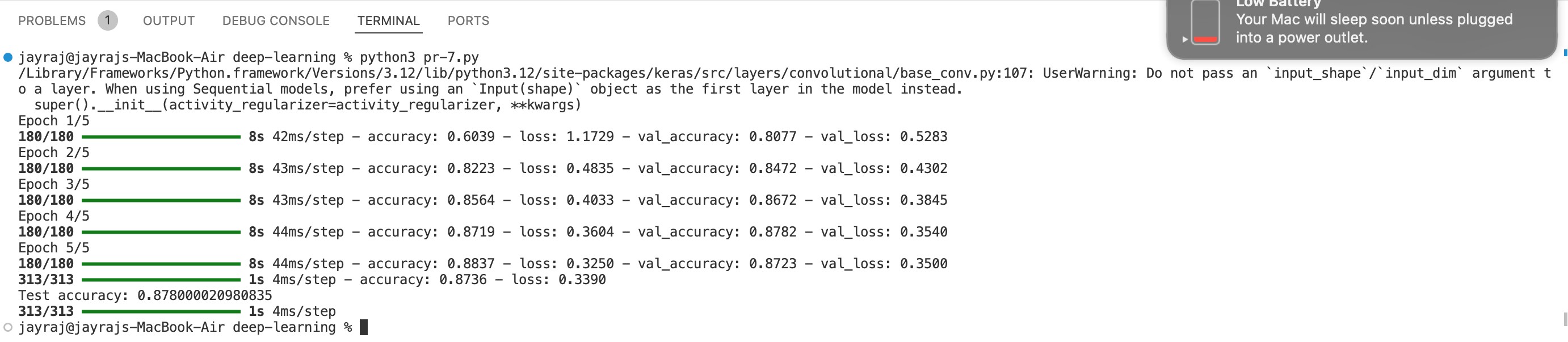
'Coat',

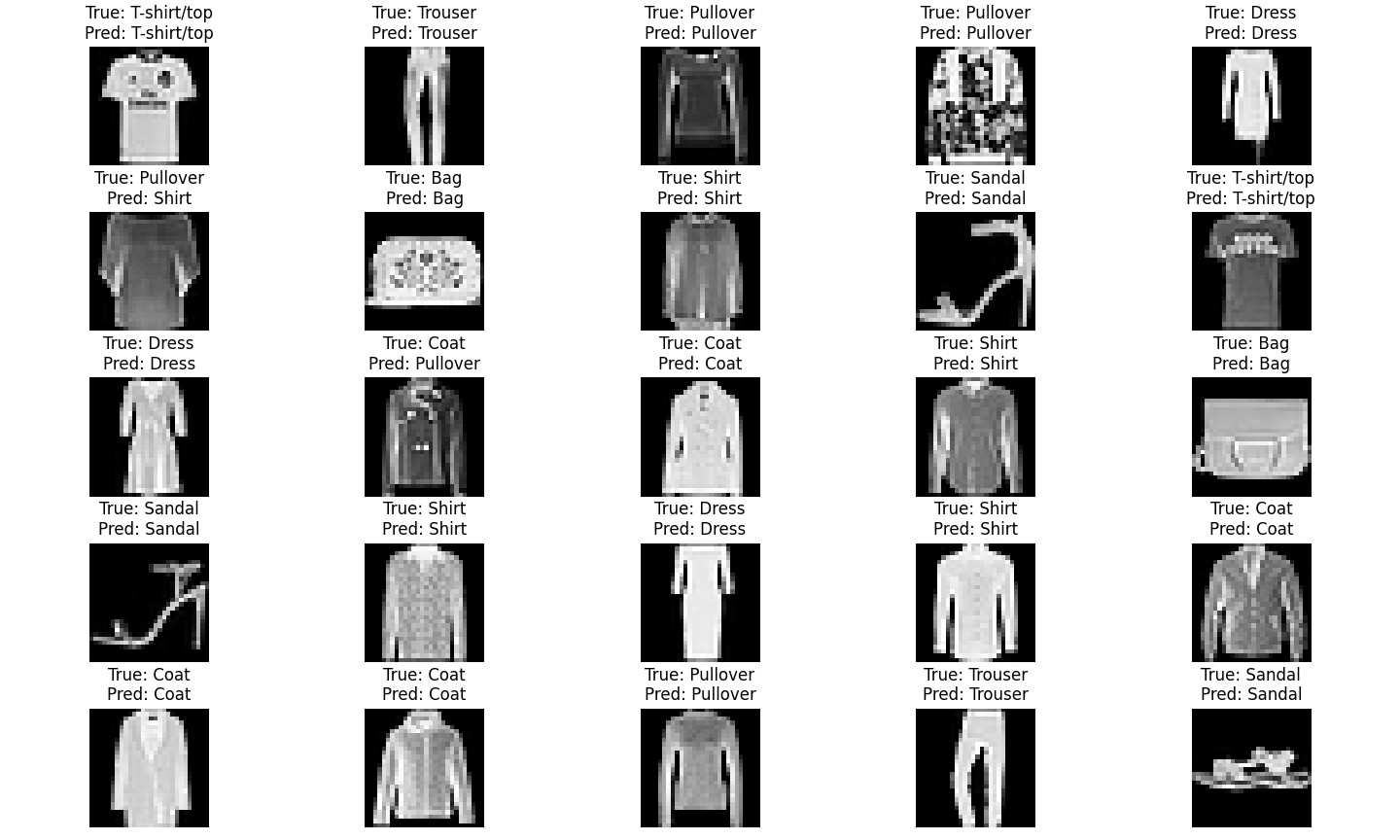
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Plot the predictions

plot\_predictions(x\_test, y\_test, predictions, class\_names)

Output





# Practical 8

**Aim : Write a program to use a pre-trained model (e.g., VGG16, Reset) for a custom image classification task. Code:-**

from google.colab import files uploaded = files.upload()

import keras

from keras.applications.resnet50 import ResNet50 from keras.applications.resnet50 import preprocess\_input, decode\_predictions import numpy as np

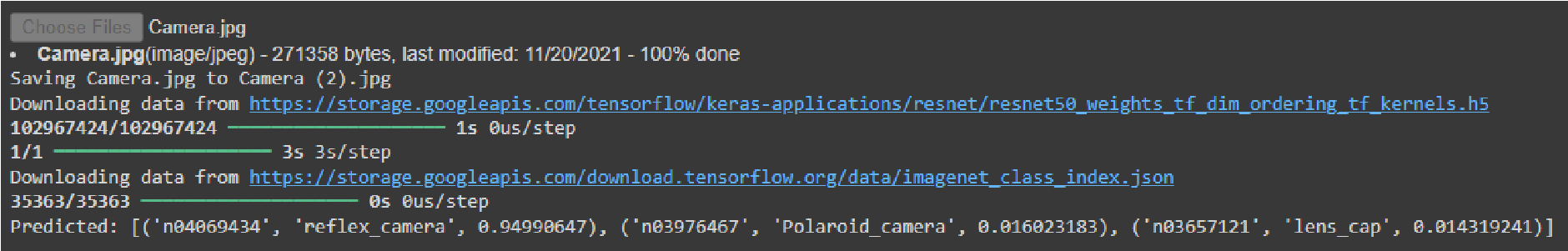
from keras.applications.resnet50 import ResNet50

# Load the pre-trained ResNet50 model model = ResNet50(weights='imagenet')

img\_path= 'Camera.jpg'

img = keras.utils.load\_img(img\_path, target\_size=(224, 224))

x= keras.utils.img\_to\_array(img) x = np.expand\_dims(x, axis=0) x = preprocess\_input(x) preds = model.predict(x) print('Predicted:', decode\_predictions (preds, top=3) [0]) **Output:-**



# Practical 9

**Aim : Write a program to fine-tune the pre-trained model on a new dataset and Compare the performance of the fine-tuned model with a model trained from scratch. Code:-**

import os import zipfile import numpy as np import pandas as pd

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense from sklearn.model\_selection import train\_test\_split from tensorflow.keras.utils import to\_categorical

os.makedirs(os.path.expanduser("~/.kaggle"), exist\_ok=True) kaggle\_json\_path = os.path.expanduser("~/.kaggle/kaggle.json") if not os.path.exists(kaggle\_json\_path): raise FileNotFoundError("kaggle.json file not found.

Please set up Kaggle API key as described.")

# Download Fashion MNIST dataset from Kaggle os.system('kaggle datasets download -d zalando-research/ fashionmnist')

# Unzip the dataset with zipfile.ZipFile('fashionmnist.zip', 'r') as zip\_ref: zip\_ref.extractall('fashionmnist')

train\_data = pd.read\_csv('fashionmnist/fashionmnist\_train.csv')

test\_data = pd.read\_csv('fashionmnist/fashion-mnist\_test.csv')

x\_train = train\_data.iloc[:, 1:].values y\_train = train\_data.iloc[:, 0].values x\_test = test\_data.iloc[:, 1:].values y\_test = test\_data.iloc[:, 0].values

x\_train = x\_train / 255.0 x\_test = x\_test / 255.0

# Convert labels to categorical (one-hot encoding)

y\_train = to\_categorical(y\_train, 10) y\_test = to\_categorical(y\_test, 10)

# Split train set into train and validation set x\_train, x\_val, y\_train, y\_val = train\_test\_split(x\_train, y\_train, test\_size=0.2, random\_state=42)

# Build the model model = Sequential([

Dense(128, activation='relu', input\_shape=(x\_train.shape[1],)), Dense(64, activation='relu'),

Dense(10, activation='softmax') # 10 classes for Fashion

MNIST

])

# Compile the model model.compile( optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy']

)

# Train the model

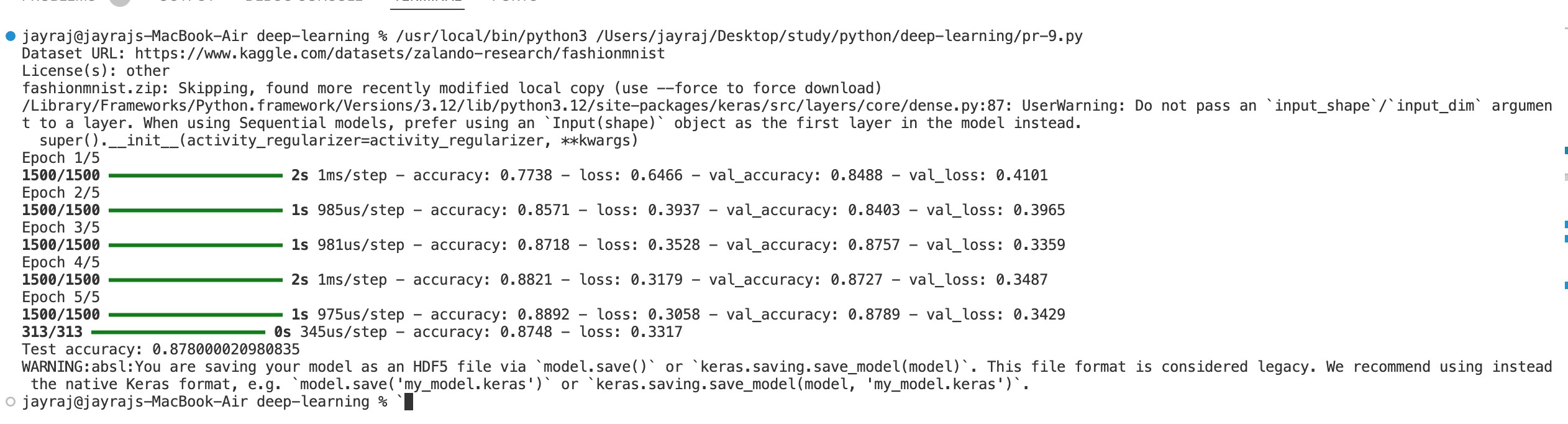
model.fit(x\_train, y\_train, epochs=5, batch\_size=32, validation\_data=(x\_val, y\_val))

# Evaluate the model on the test set

test\_loss, test\_acc = model.evaluate(x\_test, y\_test) print(f'Test accuracy: {test\_acc}')

# Save the model model.save(‘fashion\_mnist\_model.h5')

**Output:-**



# Practical 10

**Aim : Write a program to implement an RNN/LSTM for text generation. Code:-**

import numpy as np import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Embedding, Dropout

from tensorflow.keras.preprocessing.text import Tokenizer from tensorflow.keras.preprocessing.sequence import pad\_sequences

text = """The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog."""

tokenizer = Tokenizer(char\_level=True) tokenizer.fit\_on\_texts([text]) total\_chars = len(tokenizer.word\_index) + 1 input\_sequences = [] for i in range(1, len(text)): seq = text[:i+1]

input\_sequences.append(tokenizer.texts\_to\_sequences([seq])

[0])

max\_sequence\_len = max([len(seq) for seq in input\_sequences]) input\_sequences = pad\_sequences(input\_sequences, maxlen=max\_sequence\_len, padding='pre')

input\_sequences = np.array(input\_sequences)

X = input\_sequences[:, :-1] y = input\_sequences[:, -1]

y = tf.keras.utils.to\_categorical(y, num\_classes=total\_chars)

# Build the LSTM model model = Sequential() model.add(Embedding(total\_chars, 50, input\_length=max\_sequence\_len - 1)) model.add(LSTM(100, return\_sequences=True))

model.add(Dropout(0.2)) model.add(LSTM(100))

model.add(Dense(total\_chars, activation='softmax'))

# Compile the model

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

# Train the model

history = model.fit(X, y, epochs=5)

# Function to generate text def generate\_text(seed\_text, next\_chars=10): for \_ in range(next\_chars): tokenized\_seq =

tokenizer.texts\_to\_sequences([seed\_text])[0] tokenized\_seq = pad\_sequences([tokenized\_seq], maxlen=max\_sequence\_len, padding='pre') predicted\_char\_index =

np.argmax(model.predict(tokenized\_seq), axis=-1) next\_char =

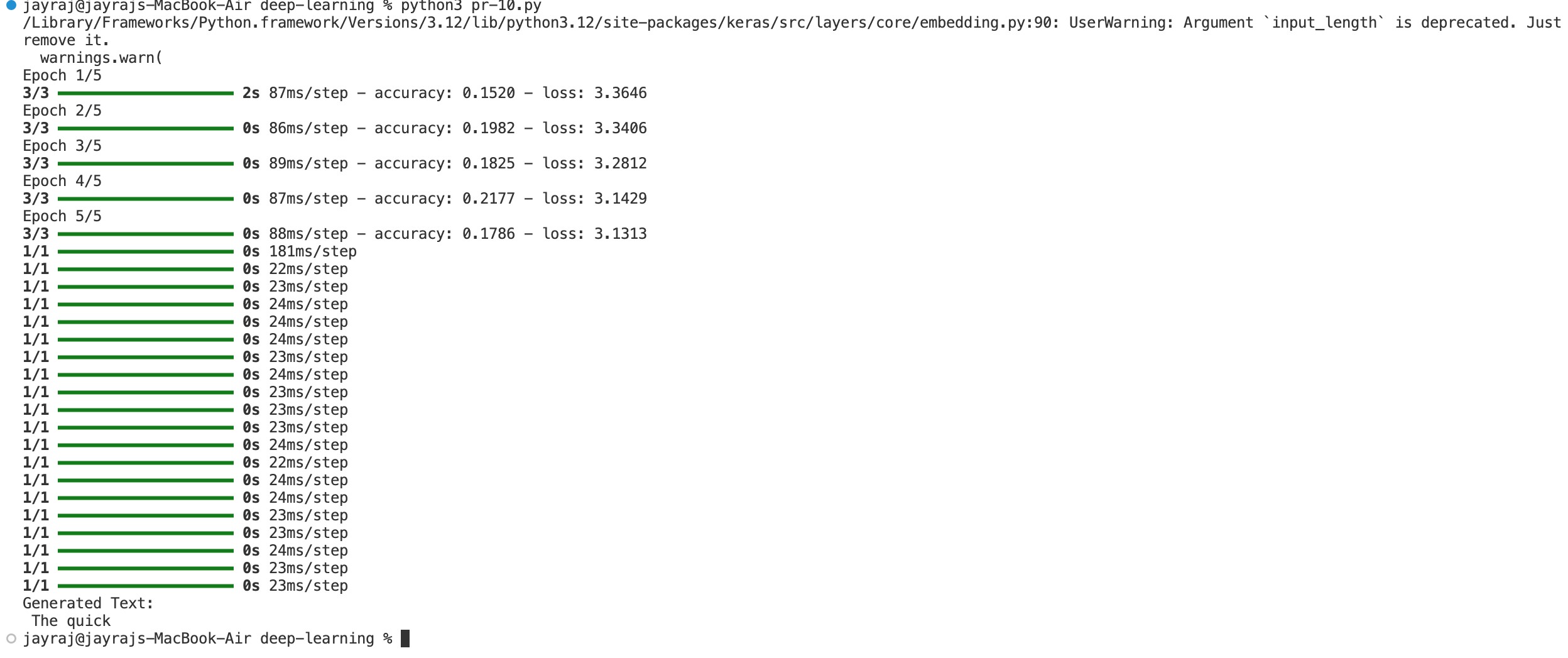
tokenizer.index\_word[predicted\_char\_index[0]]

seed\_text += next\_char return seed\_text

seed\_text = "The quick"

generated\_text = generate\_text(seed\_text, next\_chars=20) print("Generated Text: \n", generated\_text)

**Output:-**



# Practical 11

**Aim : Write a program to train the model on a text corpus (e.g., Shakespeare's works). Code:-**

import numpy as np import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Embedding, Dropout

from tensorflow.keras.preprocessing.text import Tokenizer from tensorflow.keras.preprocessing.sequence import pad\_sequences

text = """Shall I compare thee to a summer's day?

Thou art more lovely and more temperate:

Rough winds do shake the darling buds of May, And summer's lease hath all too short a date:

Sometime too hot the eye of heaven shines,

And often is his gold complexion dimm'd;

And every fair from fair sometime declines,

By chance or nature's changing course untrimm'd;

But thy eternal summer shall not fade

Nor lose possession of that fair thou owest;

Nor shall Death brag thou wanderest in his shade, When in eternal lines to time thou growest:

So long as men can breathe or eyes can see,

So long lives this, and this gives life to thee."""

tokenizer = Tokenizer(char\_level=True) tokenizer.fit\_on\_texts([text])

total\_chars = len(tokenizer.word\_index) + 1 # Total unique characters

input\_sequences = [] for i in range(1, len(text)): seq = text[:i+1]

input\_sequences.append(tokenizer.texts\_to\_sequences([seq]) [0])

max\_sequence\_len = max([len(seq) for seq in input\_sequences]) input\_sequences = pad\_sequences(input\_sequences, maxlen=max\_sequence\_len, padding='pre')

input\_sequences = np.array(input\_sequences)

X = input\_sequences[:, :-1] y = input\_sequences[:, -1]

y = tf.keras.utils.to\_categorical(y, num\_classes=total\_chars)

# Build the LSTM model model = Sequential()

model.add(Embedding(total\_chars, 50, input\_length=max\_sequence\_len - 1)) model.add(LSTM(100, return\_sequences=True))

model.add(Dropout(0.2)) model.add(LSTM(100))

model.add(Dense(total\_chars, activation='softmax'))

# Compile the model

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

# Train the model

history = model.fit(X, y, epochs=10, verbose=1)

# Function to generate text def generate\_text(seed\_text, next\_chars=100): for \_ in range(next\_chars): tokenized\_seq =

tokenizer.texts\_to\_sequences([seed\_text])[0] tokenized\_seq = pad\_sequences([tokenized\_seq], maxlen=max\_sequence\_len, padding='pre') predicted\_char\_index =

np.argmax(model.predict(tokenized\_seq), axis=-1) next\_char =

tokenizer.index\_word[predicted\_char\_index[0]]

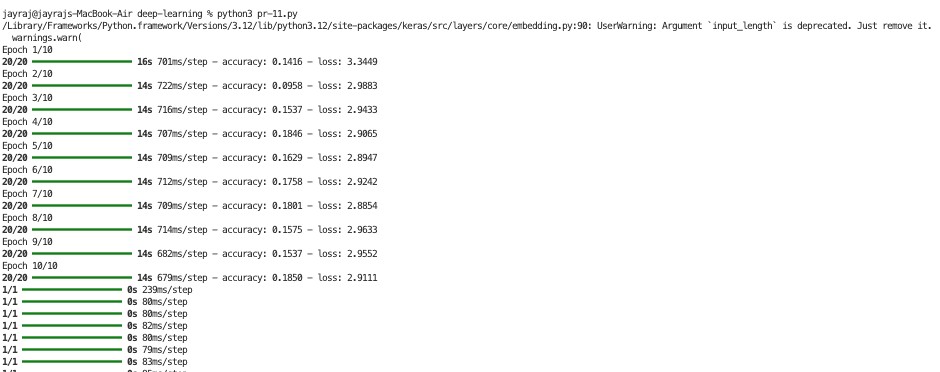
seed\_text += next\_char return seed\_text

seed\_text = "Shall I compare"

generated\_text = generate\_text(seed\_text, next\_chars=100)

print("Generated Text: \n", generated\_text)

**Output:-**



# Practical 12

**Aim : Write a program to implement an RNN/LSTM for sentiment analysis for any text data such as tweets, instagram comment etc. Code:-**

import tensorflow as tf import tensorflow\_datasets as tfds from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Load the dataset

dataset, info = tfds.load("imdb\_reviews", with\_info=True,

as\_supervised=True)

train\_data, test\_data = dataset['train'], dataset['test']

vocab\_size = 10000 max\_length = 200 embedding\_dim = 64 batch\_size = 300

train\_texts = [] for text, label in train\_data: train\_texts.append(text.numpy().decode('utf-8')) # Decode from bytes to string

tokenizer =

tf.keras.preprocessing.text.Tokenizer(num\_words=vocab\_size,

oov\_token="<OOV>")

tokenizer.fit\_on\_texts(train\_texts)

def encode\_and\_pad(text, label): # Tokenize and pad the text

text =

tokenizer.texts\_to\_sequences([text.numpy().decode('utf-8')])

text = pad\_sequences(text, maxlen=max\_length,

padding='post', truncating='post')

return tf.convert\_to\_tensor(text[0], dtype=tf.int32), tf.convert\_to\_tensor(label, dtype=tf.int64)

def encode\_and\_pad\_tf(text, label): text, label = tf.py\_function(func=encode\_and\_pad, inp=[text, label], Tout=(tf.int32, tf.int64))

text.set\_shape([max\_length]) label.set\_shape([]) return text, label

# Apply the transformation

train\_data = train\_data.map(encode\_and\_pad\_tf) test\_data = test\_data.map(encode\_and\_pad\_tf)

# Shuffle, batch, and prefetch the datasets train\_data =

train\_data.shuffle(10000).batch(batch\_size).prefetch(tf.data.e xperimental.AUTOTUNE) test\_data =

test\_data.batch(batch\_size).prefetch(tf.data.experimental.AUTO

TUNE)

# LSTM model model = Sequential([

Embedding(vocab\_size, embedding\_dim), # Removed

input\_length

LSTM(64, return\_sequences=True),

Dropout(0.5),

LSTM(32),

Dense(32, activation='relu'),

Dense(1, activation='sigmoid')

])

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

steps\_per\_epoch = len(train\_data) // batch\_size model.fit(train\_data, epochs=5, validation\_data=test\_data, steps\_per\_epoch=steps\_per\_epoch)

loss, accuracy = model.evaluate(test\_data) print(f'Test Accuracy: {accuracy \* 100:.2f}%’) new\_texts = ["This movie was fantastic!", "I did not like the film at all."]

new\_sequences = tokenizer.texts\_to\_sequences(new\_texts) new\_padded\_sequences = pad\_sequences(new\_sequences, maxlen=max\_length, padding='post')

predictions = model.predict(new\_padded\_sequences) for i, text in enumerate(new\_texts): sentiment = "positive" if predictions[i] > 0.5 else

"negative" print(f"Text: {text} | Predicted Sentiment: {sentiment}")

**Output:-**

