UNIVERSITY OF MUMBAI

**DEPARTMENT OF COMPUTER SCIENCE**



M.Sc. Computer Science – Semester IV Advanced Deep Learning

JOURNAL 2023-2024

Seat No.



UNIVERSITY OF MUMBAI

**DEPARTMENT OF COMPUTER SCIENCE**

**CERTIFICATE**

This is to certify that the work entered in this journal was done in the University Department of Computer Science laboratory by

Mr. **Tanmay Nishant Gharat** Seat No. for the course of M.Sc. Computer Science - Semester IV (CBCS) (Revised) during the academic year 2023- 2024 in a satisfactory manner.

**Subject In-charge Head of Department**

**External Examiner**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No** | **Practical** | **Page No** | **Date** | **Sign** |
| 1 | Implement Feed-forward Neural Network and train the network with different optimizers and compare the results. | 1 |  |  |
| 2 | Write a Program to implement regularization to preventthe model from overfitting | 3 |  |  |
| 3 | Implement deep learning for recognizing classes for datasets like CIFAR-10 images for previously unseen images and assign them to one of the 10 classes. | 5 |  |  |
| 4 | Implement deep learning for the Prediction of theautoencoder from the test data (e.g. MNIST data set) | 7 |  |  |
| 5 | Implement Convolutional Neural Network for DigitRecognition on the MNIST Dataset | 9 |  |  |
| 6 | Write a program to implement Transfer Learning on the suitable dataset (e.g. classify the cats versus dogs datasetfrom Kaggle). | 11 |  |  |
| 7 | Write a program to implement a simple form of a recurrent neural network.   1. E.g. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day 2. LSTM for sentiment analysis on datasets like UMICHSI650 for similar. | 13 |  |  |
| 8 | Write a program for object detection using pre- trainedmodels to use object detection. | 15 |  |  |

# Practical 1

**Aim : Implement Feed-forward Neural Network and train the network with different optimizers and compare the results**

# !pip install tensorflow matplotlib import tensorflow as tf

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import SGD, Adam, RMSprop from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

# Load and preprocess MNIST dataset

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

x\_train, x\_test = x\_train.reshape(-1, 784) / 255.0, x\_test.reshape(-1, 784) / 255.0 y\_train, y\_test = tf.keras.utils.to\_categorical(y\_train), tf.keras.utils.to\_categorical(y\_test)

# Define a function to build the model def build\_model():

return Sequential([

Dense(128, activation='relu', input\_shape=(784,)), Dense(64, activation='relu'),

Dense(10, activation='softmax')

])

# Optimizers to test

optimizers = {'SGD': SGD(), 'Adam': Adam(), 'RMSprop': RMSprop()} results = {}

# Train and evaluate model for each optimizer for name, optimizer in optimizers.items():

model = build\_model()

model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy']) history = model.fit(x\_train, y\_train, epochs=10, validation\_data=(x\_test, y\_test), verbose=0) results[name] = history.history['val\_accuracy']

# Plot validation accuracy plt.figure(figsize=(10, 5))

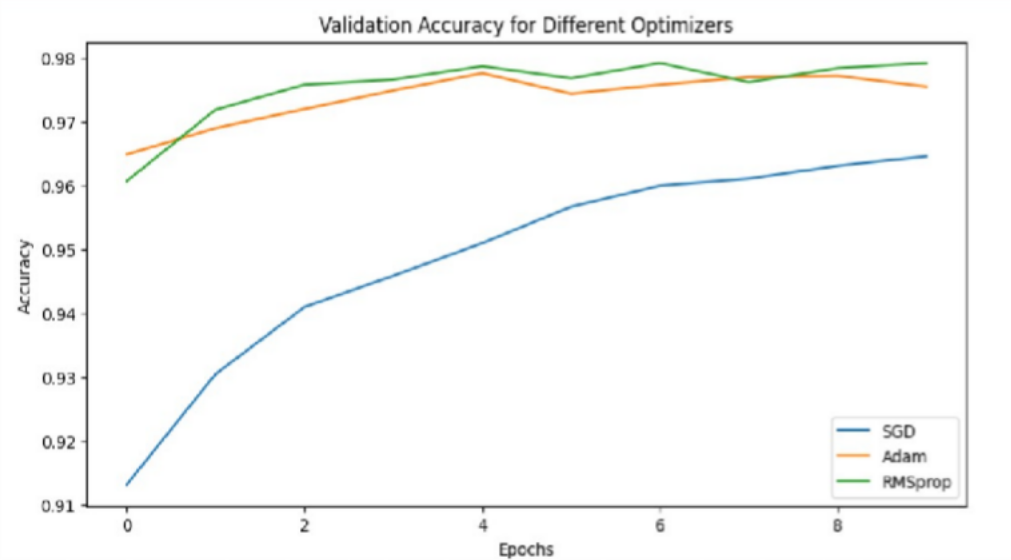
for name, val\_acc in results.items(): plt.plot(val\_acc, label=name)

plt.title('Validation Accuracy for Different Optimizers') plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

**Output :**

# Practical 2

**Aim : Write a program to implement regularization to prevent the model from overfitting**

# !pip install tensorflow matplotlib import tensorflow as tf

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Dropout from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

# Load and preprocess MNIST dataset

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

x\_train, x\_test = x\_train.reshape(-1, 784) / 255.0, x\_test.reshape(-1, 784) / 255.0 y\_train, y\_test = tf.keras.utils.to\_categorical(y\_train), tf.keras.utils.to\_categorical(y\_test)

# Build the model with regularization and dropout model = Sequential([

Dense(128, activation='relu', input\_shape=(784,), kernel\_regularizer=tf.keras.regularizers.l2(0.001)),

Dropout(0.5),

Dense(64, activation='relu', kernel\_regularizer=tf.keras.regularizers.l2(0.001)), Dropout(0.5),

Dense(10, activation='softmax')

])

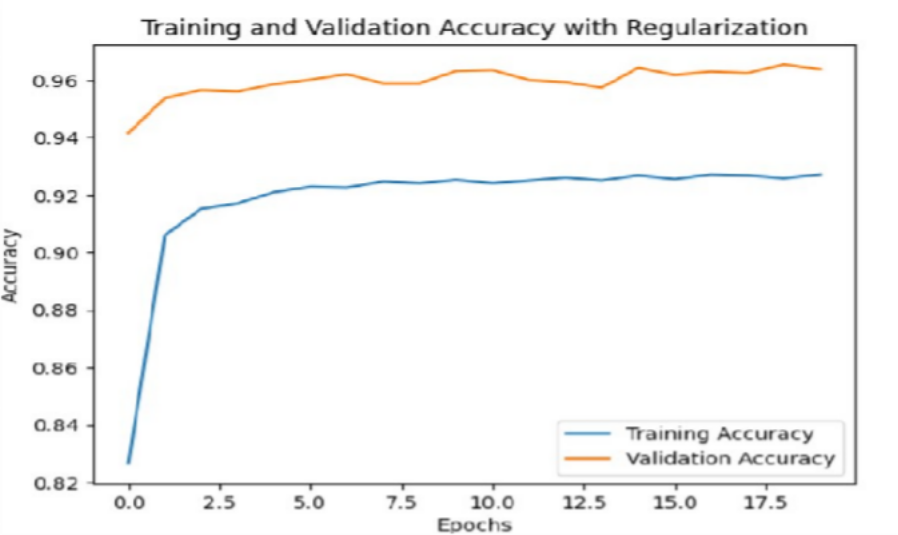
# Compile and train the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy']) history = model.fit(x\_train, y\_train, epochs=20, validation\_data=(x\_test, y\_test), verbose=2)

# Plot training and validation accuracy plt.plot(history.history['accuracy'], label='Training Accuracy') plt.plot(history.history['val\_accuracy'], label='Validation Accuracy') plt.title('Training and Validation Accuracy with Regularization') plt.xlabel('Epochs')

plt.ylabel('Accuracy') plt.legend() plt.show()

**Output :**



# Practical 3

**Aim : Implement deep learning for recognizing classes for datasets like CIFAR-10images for previously unseen images and assign them to one of the 10 classes**

# !pip install tensorflow matplotlib import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense from tensorflow.keras.datasets import cifar10

import matplotlib.pyplot as plt import numpy as np

# Load and preprocess the CIFAR-10 dataset

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

classes = ["airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck"]

# Build the CNN model model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

MaxPooling2D((2, 2)),

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

MaxPooling2D((2, 2)), Flatten(),

Dense(10, activation='softmax') ])

# Compile and train the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy']) history = model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test), verbose=2)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test) print(f'Test accuracy: {test\_acc:.2f}')

# Predict and visualize some test images predictions = model.predict(X\_test[:10]) plt.figure(figsize=(20, 4))

for i in range(10): plt.subplot(2, 10, i + 1) plt.imshow(X\_test[i]) plt.xticks([])

plt.yticks([]) plt.title(classes[np.argmax(predictions[i])])

plt.show()

**Output**



# Practical 4

**Aim : Implement deep learning for the prediction of the autoencoder from the testdata(e.g. MNIST data set)**

import tensorflow as tf

from tensorflow.keras.layers import Input, Dense from tensorflow.keras.models import Model from tensorflow.keras.datasets import mnist import matplotlib.pyplot as plt

# Load and Prepare Data

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

x\_train, x\_test = x\_train.reshape(-1, 784) / 255.0, x\_test.reshape(-1, 784) / 255.0

# Define the Autoencoder Model input\_img = Input(shape=(784,))

encoded = Dense(64, activation='relu')(input\_img) encoded = Dense(32, activation='relu')(encoded) decoded = Dense(64, activation='relu')(encoded) decoded = Dense(784, activation='sigmoid')(decoded)

autoencoder = Model(input\_img, decoded) # Compile and Train the Autoencoder

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

autoencoder.fit(x\_train, x\_train, epochs=10, batch\_size=256, validation\_data=(x\_test, x\_test))

# Predict and Display Results decoded\_imgs = autoencoder.predict(x\_test)

# Plotting original and reconstructed images plt.figure(figsize=(20, 4))

for i in range(10):

# Original images

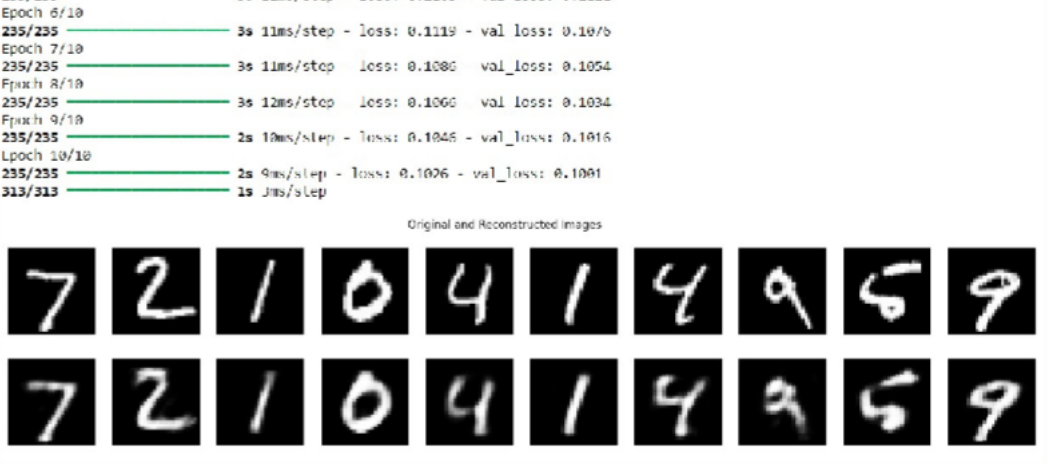
ax = plt.subplot(2, 10, i + 1) plt.imshow(x\_test[i].reshape(28, 28), cmap='gray') ax.axis('off')

# Reconstructed images

ax = plt.subplot(2, 10, i + 11) plt.imshow(decoded\_imgs[i].reshape(28, 28), cmap='gray') ax.axis('off')

plt.suptitle('Original and Reconstructed Images') plt.show()

**Output :**



# Practical 5

**Aim : Implement Convolutional Neural Network for Digit recognition on the MNISTdataset**

import tensorflow as tf

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense from tensorflow.keras.models import Sequential

from tensorflow.keras.datasets import mnist import matplotlib.pyplot as plt

# Load and Prepare Data

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data() x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Build the CNN Model model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

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

MaxPooling2D((2, 2)), Flatten(),

Dense(64, activation='relu'), Dense(10, activation='softmax')

])

# Compile and Train the Model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy']) history = model.fit(x\_train.reshape(-1, 28, 28, 1), y\_train, epochs=5, batch\_size=64,

validation\_data=(x\_test.reshape(-1, 28, 28, 1), y\_test))

# Evaluate the Model

test\_loss, test\_acc = model.evaluate(x\_test.reshape(-1, 28, 28, 1), y\_test) print(f'Test accuracy: {test\_acc:.4f}')

# Predictions and Visualization

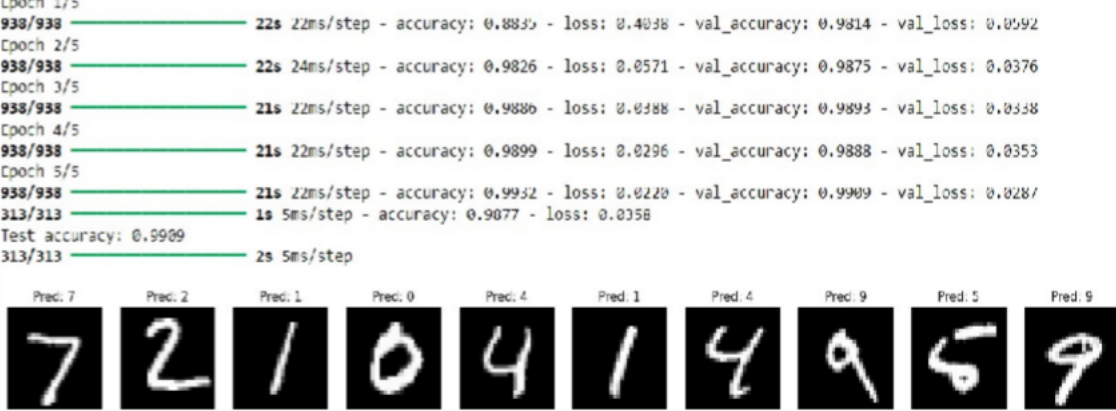
predictions = model.predict(x\_test.reshape(-1, 28, 28, 1))

plt.figure(figsize=(20, 4)) for i in range(10):

ax = plt.subplot(2, 10, i + 1) plt.imshow(x\_test[i], cmap='gray') ax.axis('off')

ax.set\_title(f'Pred: {predictions[i].argmax()}') plt.show()

**Output :**



# Practical 6

**Aim : Write a program to implement Transfer Learning on the suitable dataset**

import tensorflow as tf

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.layers import GlobalAveragePooling2D, Dense from tensorflow.keras.models import Model

import matplotlib.pyplot as plt

# Load and preprocess CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data() x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Load pre-trained MobileNetV2 model

base\_model = MobileNetV2(weights='imagenet', include\_top=False, input\_shape=(32, 32, 3)) x = GlobalAveragePooling2D()(base\_model.output)

predictions = Dense(10, activation='softmax')(x)

model = Model(inputs=base\_model.input, outputs=predictions)

# Freeze base model layers base\_model.trainable = False

# Compile model

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

# Train model

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

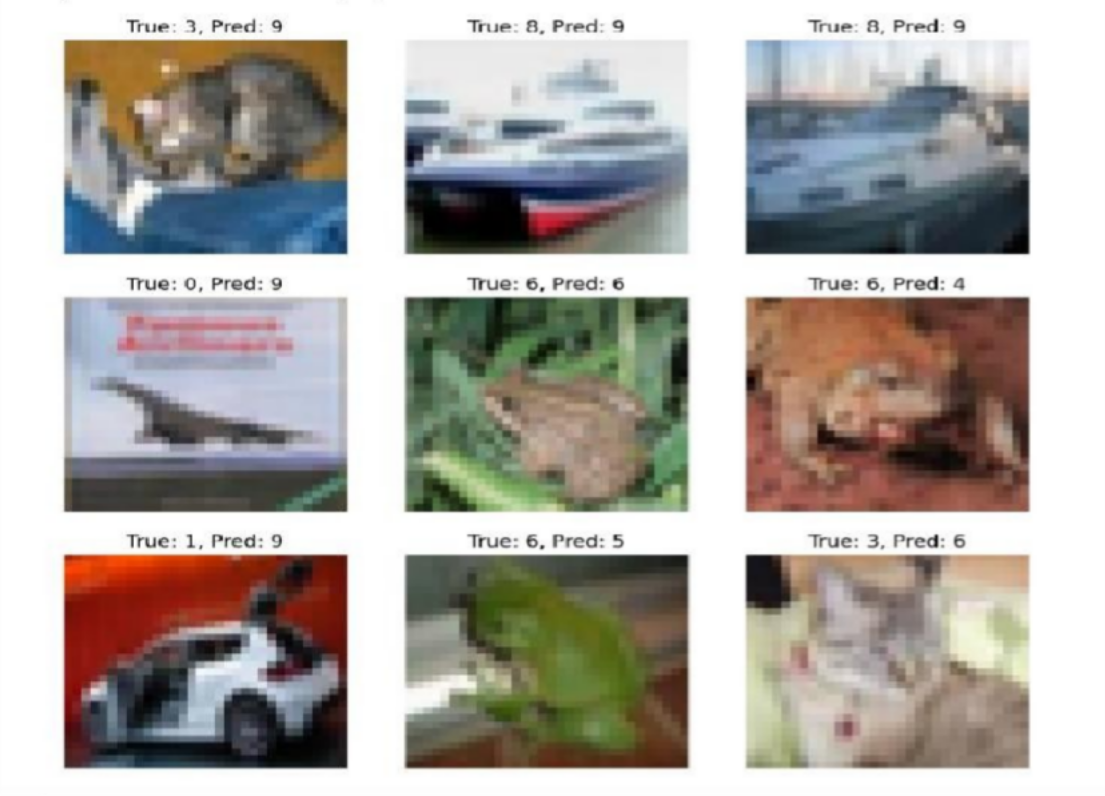
# Predict and visualize some test images predictions = model.predict(x\_test[:9]) plt.figure(figsize=(10, 10))

for i in range(9): plt.subplot(3, 3, i+1) plt.imshow(x\_test[i])

plt.title(f"True: {y\_test[i][0]}, Pred: {predictions[i].argmax()}") plt.axis('off')

plt.show()

**Output :**



# Practical 7

**Aim : Write a program to implement a simple form of a recurrent neural network.**

1. **E.g. (4-to-1 RNN) to show that the quantity of rain on a certain day also depends on the values of the previous day**

import numpy as np import tensorflow as tf

# Generate synthetic data

data = np.random.rand(100, 1) X, y = [], []

for i in range(len(data) - 4): X.append(data[i:i+4]) y.append(data[i+4])

X, y = np.array(X), np.array(y)

# Build and train the RNN model model = tf.keras.Sequential([

tf.keras.layers.SimpleRNN(50, input\_shape=(4, 1)), tf.keras.layers.Dense(1)

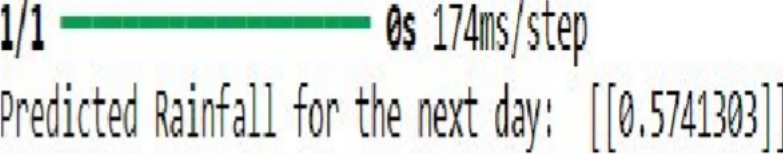
])

model.compile(optimizer='adam', loss='mse') model.fit(X, y, epochs=200, verbose=0)

# Predict the next value

print('Predicted Rainfall for the next day: ',model.predict(data[-4:].reshape(1, 4, 1)))

**Output :**



1. **LSTM for sentiment analysis on datasets like UMICH SI650 for similar.**

import tensorflow as tf

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

from nltk.corpus import movie\_reviews nltk.download('movie\_reviews')

# Load and preprocess data

sentences = [" ".join(movie\_reviews.words(fileid)) for fileid in movie\_reviews.fileids()] labels = [1 if fileid.startswith('pos') else 0 for fileid in movie\_reviews.fileids()]

tokenizer = tf.keras.preprocessing.text.Tokenizer(num\_words=5000) X = tokenizer.texts\_to\_sequences(sentences)

X = tf.keras.preprocessing.sequence.pad\_sequences(X, maxlen=100) y = np.array(labels)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) # Build and train the LSTM model

model = tf.keras.Sequential([

tf.keras.layers.Embedding(5000, 128, input\_length=100), tf.keras.layers.LSTM(128),

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

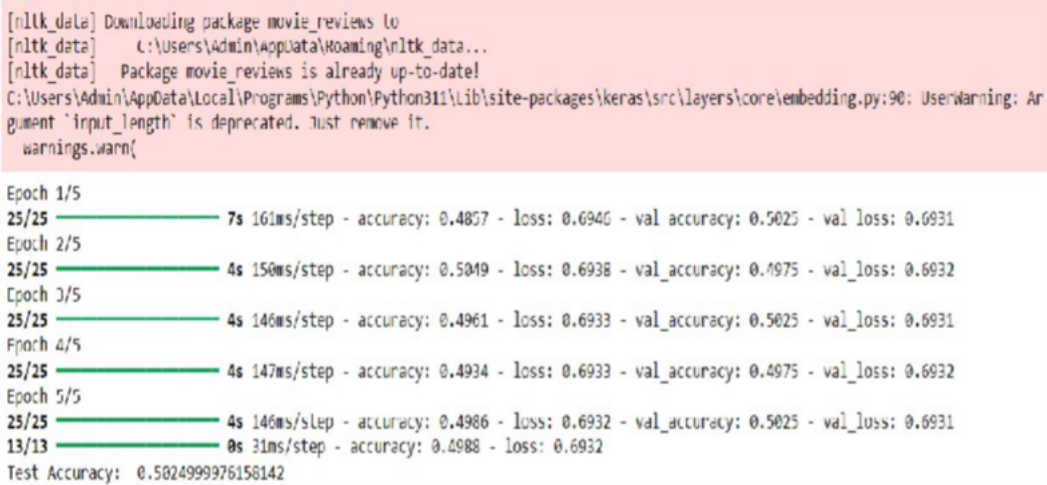
])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy']) model.fit(X\_train, y\_train, epochs=5, batch\_size=64, validation\_data=(X\_test, y\_test))

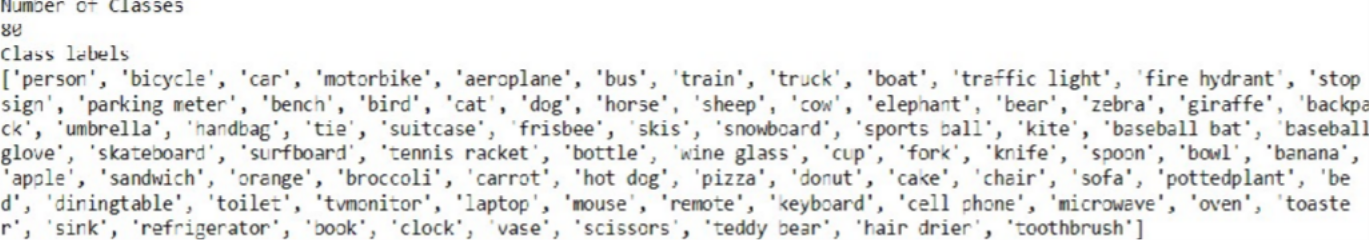
# Evaluate the model

print('Test Accuracy: ',model.evaluate(X\_test, y\_test)[1])

**Output:**



# Practical 8



**Aim : Write a program for object detection using pre-trained models to use objectDetection**

#importing libraries import cv2

import matplotlib.pyplot as plt from matplotlib import ft2font

#importing and using necessary files

config\_file=r'C:\Users\HARDIK PATIL\ssd\_mobilenet\_v3\_large\_coco\_2020\_01\_14(1).pbtxt' frozen\_model=r'C:\Users\HARDIK PATIL\frozen\_inference\_graph.pb'

#Tenserflow object detection model

model = cv2.dnn\_DetectionModel(frozen\_model,config\_file)

#Reading Coco dataset classLabels=[]

filename=r'C:\Users\HARDIK PATIL\coco.names' with open(filename,'rt') as fpt:

classLabels = fpt.read().rstrip('\n').split('\n')

print("Number of Classes") print(len(classLabels)) print("Class labels") print(classLabels)

#Model training model.setInputSize(320,320) model.setInputScale(1.0/127.5) model.setInputMean((127.5,127.5,127.5)) model.setInputSwapRB(True)

#reading image

img = cv2.imread(r'C:\Users\HARDIK PATIL\test.jpg') plt.imshow(img)



#object detection

ClassIndex, confidence, bbox = model.detect(img, confThreshold=0.5)

for class\_id, conf, box in zip(ClassIndex.flatten(), confidence.flatten(), bbox): if class\_id in desired\_classes:

class\_counts[class\_id] += 1 # Increment counter for the detected class label = f'{classLabels[class\_id-1]}: {conf:.2f}'

cv2.rectangle(img, box, color=(0, 255, 0), thickness=2)

cv2.putText(img, label, (box[0], box[1] - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0,255, 0),2)

# Display the image with detected objects print(class\_counts)

plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)) plt.show()

**Output :**

