|  |  |
| --- | --- |
| Ex No : 01DATE:23/3/24 | Program to help packman find his food using A\* search algorithm |

## AIM:

The aim of the experiment is to find an optimal path to move packman towards its goal using A star search algorithm.

ALGORITHM:

1. Initialize all the values like g, h, f, position by creating a init function
2. Create a base function for A-star searching, which takes in parameters like start point, destination goal. and entire grid layout.
3. The A-star function should have an open set and closed set, and the current point should be assigned and added into the openset.
4. By looping through the openset, we assign the set items that have the best F (i.e G+H) score as the current value.
5. If the current value is the destination goal value then backtrack its parents and print the path, else remove the current value from openset and add to the closed set.
6. Then check for nearby nodes to choose the node with the best G score, by calculating and comparing G score and H (heuristic) score.
7. Skip if the node is already in closed set or if the node value is 1, if node is already in openset compare the selected node G score with existing node G scores and sort the openset.
8. If the selected node with the best G score is not in any list, then add it to the open set.
9. In the main function hard code the grid with 0’s as walkable path and 1’s as objects blocking path.
10. get the start and destination goal position from user and call the astar() function

## PROGRAM:

class Node():

"""A node class for A\* Pathfinding"""

def \_\_init\_\_(self, parent=None, position=None):

self.parent = parent

self.position = position

self.g = 0

self.h = 0

self.f = 0

def \_\_eq\_\_(self, other):

return self.position == other.position

def astar(maze, start, end):

"""Returns a list of tuples as a path from the given start to the given end in the given maze"""

# Create start and end node

start\_node = Node(None, start)

start\_node.g = start\_node.h = start\_node.f = 0

end\_node = Node(None, end)

end\_node.g = end\_node.h = end\_node.f = 0

# Initialize both open and closed list

open\_list = []

closed\_list = []

# Add the start node

open\_list.append(start\_node)

# Loop until you find the end

while len(open\_list) > 0:

# Get the current node

current\_node = open\_list[0]

current\_index = 0

for index, item in enumerate(open\_list):

if item.f < current\_node.f:

current\_node = item

current\_index = index

# Pop current off open list, add to closed list

open\_list.pop(current\_index)

closed\_list.append(current\_node)

# Found the goal

if current\_node == end\_node:

path = []

current = current\_node

while current is not None:

path.append(current.position)

current = current.parent

return path[::-1] # Return reversed path

# Generate children

children = []

for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent squares

# Get node position

node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

# Make sure within range

if node\_position[0] > (len(maze) - 1) or node\_position[0] < 0 or node\_position[1] > (len(maze[len(maze)-1]) -1) or node\_position[1] < 0:

continue

# Make sure walkable terrain

if maze[node\_position[0]][node\_position[1]] != 0:

continue

# Create new node

new\_node = Node(current\_node, node\_position)

# Append

children.append(new\_node)

# Loop through children

for child in children:

# Child is on the closed list

for closed\_child in closed\_list:

if child == closed\_child:

continue

# Create the f, g, and h values

child.g = current\_node.g + 1

child.h = ((child.position[0] - end\_node.position[0]) \*\* 2) + ((child.position[1] - end\_node.position[1]) \*\* 2)

child.f = child.g + child.h

# Child is already in the open list

for open\_node in open\_list:

if child == open\_node and child.g > open\_node.g:

continue

# Add the child to the open list

open\_list.append(child)

def main():

maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

start = (0, 0)

end = (7, 6)

path = astar(maze, start, end)

print(path)

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

main()

## OUTPUT:



## RESULE:

Thus, the A-star search algorithm was implemented successfully in python to move packman to destination position in a given grid space.

|  |  |
| --- | --- |
| Ex No : 02DATE:8/3/24 | Use Minimax algorithm for solving Tic-Tac-Toe game |

## AIM:

The aim of this experiment is to create an AI playing tic-tac-toe game using minimax algorithm.

## ALGORITHM:

1. Import the necessary libraries like math, random, platform, time, os.
2. Initialize the values for Human point as -1 and Computer as +1 and initialize the board matrix.
3. Create a function to evaluate the state, returns +1 of computer wins, -1 if human wins.
4. Create a function to denote all possible win states by predefining a matrix of state combinations that will result in a win.
5. Create a function to return game over information to the player.
6. Create a function to find and crate an empty cells list (i.e cells that are not filled yet by computer or the human player)
7. Create a function to denote a valid move or not, because the human player can choose an already filled state by mistake.
8. Create a function to add the valid move to the play board matrix.
9. Create the minimax function that takes input parameters like state, depth, player.
10. If depth is equal to 0 or state is a game over state, then check and return the score +1 for Max win, -1 for Min win, or 0 for a draw.
11. recursively call the minimax function for the each valid empty state in the board based on Max or Min to find the best position to win, by comparing the state score with the best score.
12. Create a render function to display the playboard after every play turn.
13. Create a function for AI turn which calls the minimax function to get the best move for the AI player.
14. Create a function for the human player to validate the human’s choice and update the board.
15. Create a main function that gets user choice of side and also user move, coordinates the game until all the cells are filled.

## PROGRAM:

#!/usr/bin/env python3

from math import inf as infinity

from random import choice

import platform

import time

from os import system

"""

An implementation of Minimax AI Algorithm in Tic Tac Toe,

using Python.

This software is available under GPL license.

Author: Clederson Cruz

Year: 2017

License: GNU GENERAL PUBLIC LICENSE (GPL)

"""

HUMAN = -1

COMP = +1

board = [

[0, 0, 0],

[0, 0, 0],

[0, 0, 0],

]

def evaluate(state):

"""

Function to heuristic evaluation of state.

:param state: the state of the current board

:return: +1 if the computer wins; -1 if the human wins; 0 draw

"""

if wins(state, COMP):

score = +1

elif wins(state, HUMAN):

score = -1

else:

score = 0

return score

def wins(state, player):

"""

This function tests if a specific player wins. Possibilities:

\* Three rows [X X X] or [O O O]

\* Three cols [X X X] or [O O O]

\* Two diagonals [X X X] or [O O O]

:param state: the state of the current board

:param player: a human or a computer

:return: True if the player wins

"""

win\_state = [

[state[0][0], state[0][1], state[0][2]],

[state[1][0], state[1][1], state[1][2]],

[state[2][0], state[2][1], state[2][2]],

[state[0][0], state[1][0], state[2][0]],

[state[0][1], state[1][1], state[2][1]],

[state[0][2], state[1][2], state[2][2]],

[state[0][0], state[1][1], state[2][2]],

[state[2][0], state[1][1], state[0][2]],

]

if [player, player, player] in win\_state:

return True

else:

return False

def game\_over(state):

"""

This function test if the human or computer wins

:param state: the state of the current board

:return: True if the human or computer wins

"""

return wins(state, HUMAN) or wins(state, COMP)

def empty\_cells(state):

"""

Each empty cell will be added into cells' list

:param state: the state of the current board

:return: a list of empty cells

"""

cells = []

for x, row in enumerate(state):

for y, cell in enumerate(row):

if cell == 0:

cells.append([x, y])

return cells

def valid\_move(x, y):

"""

A move is valid if the chosen cell is empty

:param x: X coordinate

:param y: Y coordinate

:return: True if the board[x][y] is empty

"""

if [x, y] in empty\_cells(board):

return True

else:

return False

def set\_move(x, y, player):

"""

Set the move on board, if the coordinates are valid

:param x: X coordinate

:param y: Y coordinate

:param player: the current player

"""

if valid\_move(x, y):

board[x][y] = player

return True

else:

return False

def minimax(state, depth, player):

"""

AI function that choice the best move

:param state: current state of the board

:param depth: node index in the tree (0 <= depth <= 9),

but never nine in this case (see iaturn() function)

:param player: an human or a computer

:return: a list with [the best row, best col, best score]

"""

if player == COMP:

best = [-1, -1, -infinity]

else:

best = [-1, -1, +infinity]

if depth == 0 or game\_over(state):

score = evaluate(state)

return [-1, -1, score]

for cell in empty\_cells(state):

x, y = cell[0], cell[1]

state[x][y] = player

score = minimax(state, depth - 1, -player)

state[x][y] = 0

score[0], score[1] = x, y

if player == COMP:

if score[2] > best[2]:

best = score # max value

else:

if score[2] < best[2]:

best = score # min value

return best

def clean():

"""

Clears the console

"""

os\_name = platform.system().lower()

if 'windows' in os\_name:

system('cls')

else:

system('clear')

def render(state, c\_choice, h\_choice):

"""

Print the board on console

:param state: current state of the board

"""

chars = {

-1: h\_choice,

+1: c\_choice,

0: ' '

}

str\_line = '---------------'

print('\n' + str\_line)

for row in state:

for cell in row:

symbol = chars[cell]

print(f'| {symbol} |', end='')

print('\n' + str\_line)

def ai\_turn(c\_choice, h\_choice):

"""

It calls the minimax function if the depth < 9,

else it choices a random coordinate.

:param c\_choice: computer's choice X or O

:param h\_choice: human's choice X or O

:return:

"""

depth = len(empty\_cells(board))

if depth == 0 or game\_over(board):

return

clean()

print(f'Computer turn [{c\_choice}]')

render(board, c\_choice, h\_choice)

if depth == 9:

x = choice([0, 1, 2])

y = choice([0, 1, 2])

else:

move = minimax(board, depth, COMP)

x, y = move[0], move[1]

set\_move(x, y, COMP)

time.sleep(1)

def human\_turn(c\_choice, h\_choice):

"""

The Human plays choosing a valid move.

:param c\_choice: computer's choice X or O

:param h\_choice: human's choice X or O

:return:

"""

depth = len(empty\_cells(board))

if depth == 0 or game\_over(board):

return

# Dictionary of valid moves

move = -1

moves = {

1: [0, 0], 2: [0, 1], 3: [0, 2],

4: [1, 0], 5: [1, 1], 6: [1, 2],

7: [2, 0], 8: [2, 1], 9: [2, 2],

}

clean()

print(f'Human turn [{h\_choice}]')

render(board, c\_choice, h\_choice)

while move < 1 or move > 9:

try:

move = int(input('Use numpad (1..9): '))

coord = moves[move]

can\_move = set\_move(coord[0], coord[1], HUMAN)

if not can\_move:

print('Bad move')

move = -1

except (EOFError, KeyboardInterrupt):

print('Bye')

exit()

except (KeyError, ValueError):

print('Bad choice')

def main():

"""

Main function that calls all functions

"""

clean()

h\_choice = '' # X or O

c\_choice = '' # X or O

first = '' # if human is the first

# Human chooses X or O to play

while h\_choice != 'O' and h\_choice != 'X':

try:

print('')

h\_choice = input('Choose X or O\nChosen: ').upper()

except (EOFError, KeyboardInterrupt):

print('Bye')

exit()

except (KeyError, ValueError):

print('Bad choice')

# Setting computer's choice

if h\_choice == 'X':

c\_choice = 'O'

else:

c\_choice = 'X'

# Human may starts first

clean()

while first != 'Y' and first != 'N':

try:

first = input('First to start?[y/n]: ').upper()

except (EOFError, KeyboardInterrupt):

print('Bye')

exit()

except (KeyError, ValueError):

print('Bad choice')

# Main loop of this game

while len(empty\_cells(board)) > 0 and not game\_over(board):

if first == 'N':

ai\_turn(c\_choice, h\_choice)

first = ''

human\_turn(c\_choice, h\_choice)

ai\_turn(c\_choice, h\_choice)

# Game over message

if wins(board, HUMAN):

clean()

print(f'Human turn [{h\_choice}]')

render(board, c\_choice, h\_choice)

print('YOU WIN!')

elif wins(board, COMP):

clean()

print(f'Computer turn [{c\_choice}]')

render(board, c\_choice, h\_choice)

print('YOU LOSE!')

else:

clean()

render(board, c\_choice, h\_choice)

print('DRAW!')

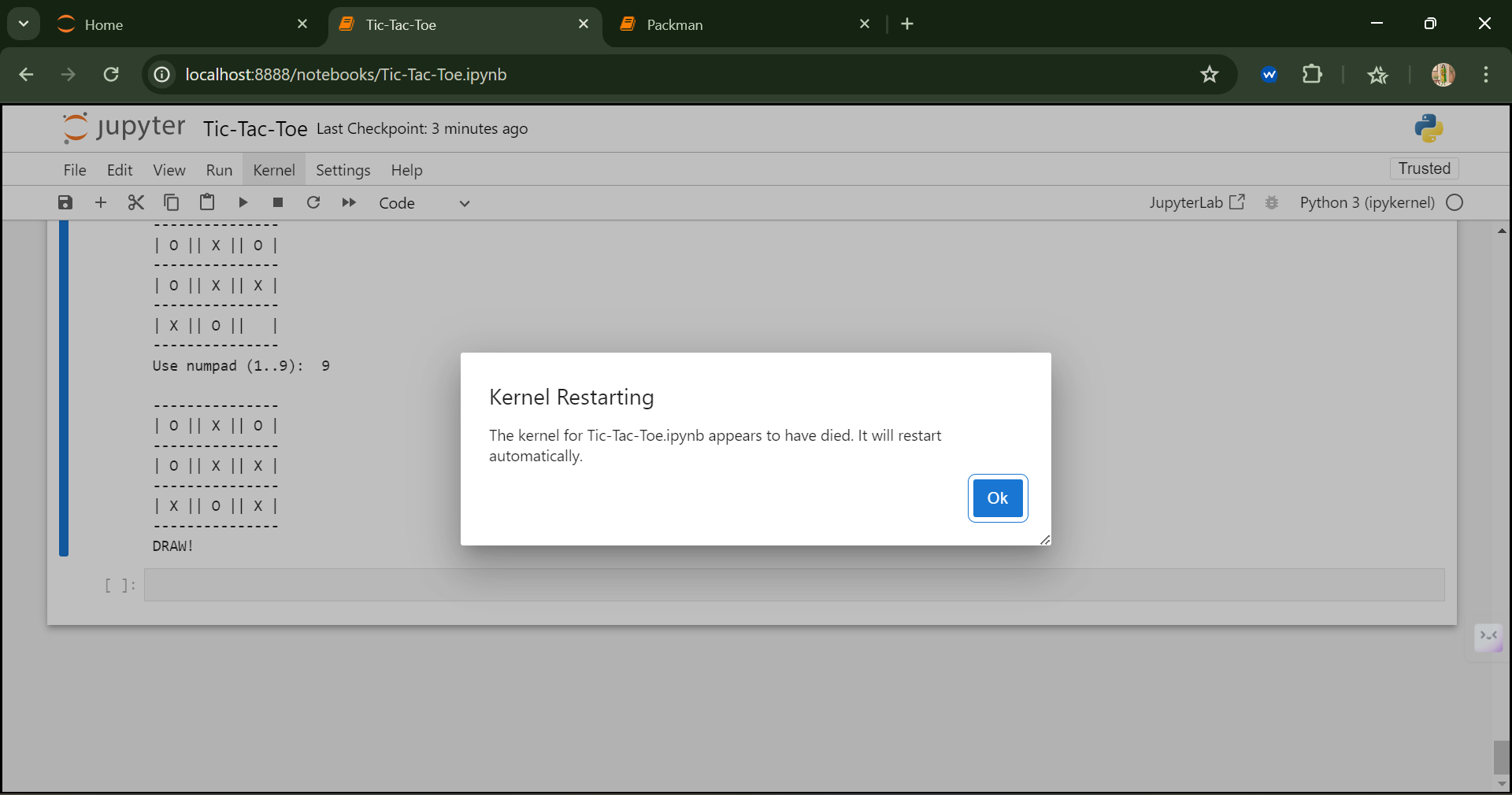
exit()

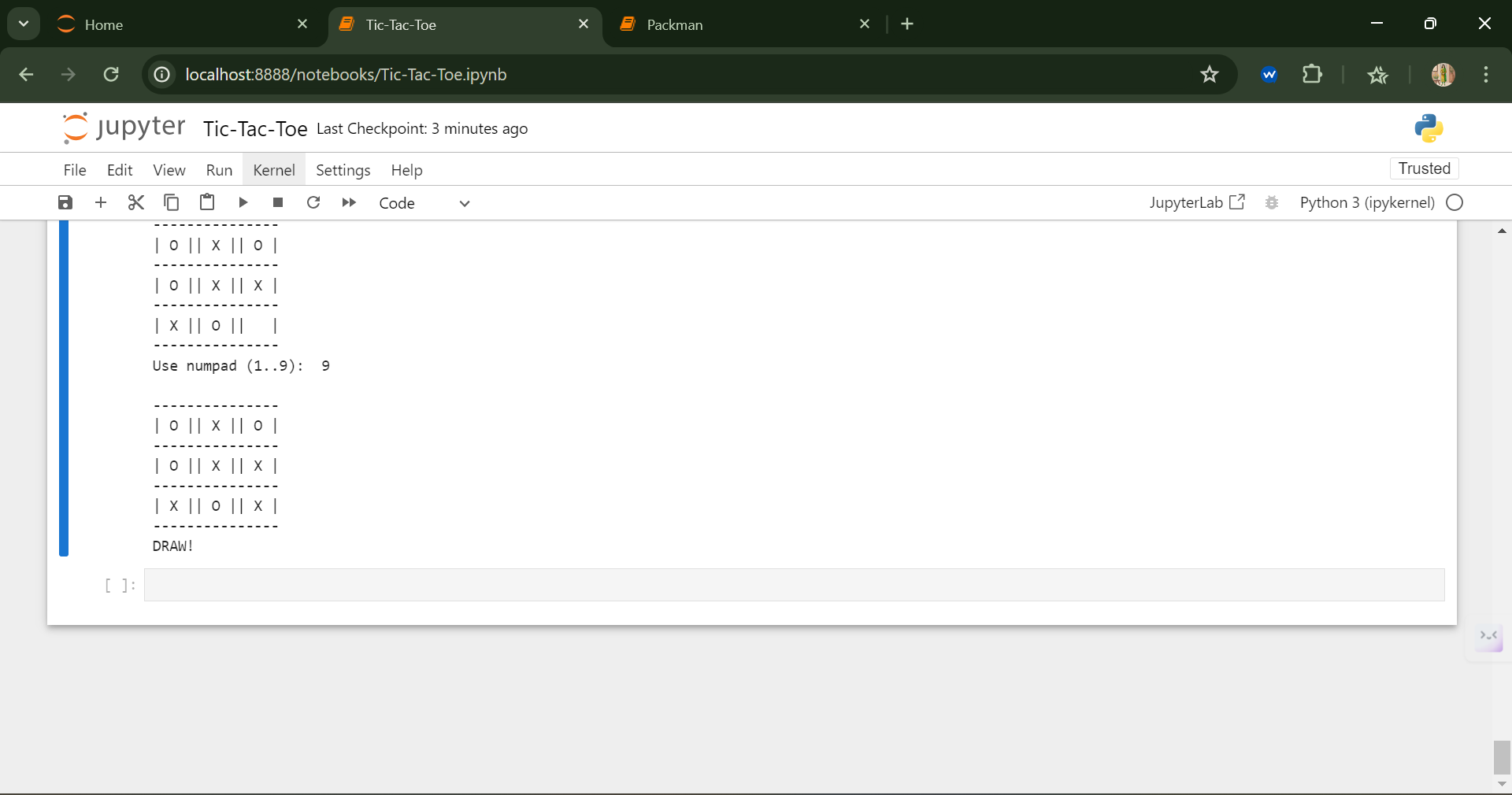
if \_\_name\_\_ == '\_\_main\_\_':

main()

## OUTPUT:







## RESULE:

Thus, an appropriate reasoning algorithm for question answering chatbot using python has been implemented and executed successfully.

|  |  |
| --- | --- |
| Ex No : 03DATE:15/3/24 | Develop an appropriate reasoning algorithm for question answering system |

## AIM:

To develop an appropriate reasoning algorithm for question answering chatbot using python.

## ALGORITHM:

1. Import the natural language toolkit (nltk) and the appropriate packages for processing natural language.
2. Download the necessary data using nltk.download(‘punkt’). This package can be used only for the first time of compilation.
3. Create a file variable to read the text file.

### Text Pre-processing:

1. Convert all the contents in the file to lowercase letters and perform sentence and word tokenization on them.
2. Perform lemmatization over the tokenized words.
3. Mention the common greeting post and responses.
4. Create a function for greeting.
5. Importing of TF-IDF and Cosine similarity is used in order to find the occurrence of the word in the document and to determine the similarity between the post and the response from the given text file.
6. Create a response function which process to retrieve the appropriate response from the text file given as input.
7. When the user sends the post the response function tokenizs the user’s post and determine the TF-IDF value and cosine similarity.
8. Sorting is performed over the retrieved response and it is displayed to the user.
9. If the TF-IDF value is 0 then the respective machine response is generated.

### Prerequisites to run the program:

1. Save the program and the text file in the same folder and do the following
2. $ sudo apt-get install python3-pip
3. Save the program and in the command prompt navigate to the folder.
4. cole@unisys-58:~/Building-a-Simple-Chatbot-in-Python-using-NLTK-master$
5. Install the following packages as given: $ sudo pip3 install numpy $ sudo pip3 install sklearn
6. Run the program cole@unisys-58:~/Building-a-Simple-Chatbot-in-Python-using-NLTK-master$ python3 chatbot.py
7. Do the following

NLTK Downloader

d) Download l) List u) Update c) Config h) Help q) Quit

Downloader> d

Download which package (l=list; x=cancel)?

Identifier> all-nltk

Done downloading collection all-nltk

d) Download l) List u) Update c) Config h) Help q) Quit

1. Downloader> q
2. If you try to run the file for the second time comment (#) the following lines in the code and run the file #nltk.download() #nltk.download('punkt') # first-time use only #nltk.download('wordnet') # first-time use only.

## PROGRAM:

# coding: utf-8

# # Meet Robo: your friend

import nltk

import warnings

warnings.filterwarnings("ignore")

nltk.download() # for downloading packages

import numpy as np

import random

import string # to process standard python strings

f=open('chatbot.txt','r',errors = 'ignore')

raw=f.read()

raw=raw.lower()# converts to lowercase

nltk.download('punkt') # first-time use only

nltk.download('wordnet') # first-time use only

sent\_tokens = nltk.sent\_tokenize(raw)# converts to list of sentences

word\_tokens = nltk.word\_tokenize(raw)# converts to list of words

sent\_tokens[:2]

word\_tokens[:5]

lemmer = nltk.stem.WordNetLemmatizer()

def LemTokens(tokens):

return [lemmer.lemmatize(token) for token in tokens]

remove\_punct\_dict = dict((ord(punct), None) for punct in string.punctuation)

def LemNormalize(text):

return LemTokens(nltk.word\_tokenize(text.lower().translate(remove\_punct\_dict)))

GREETING\_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)

GREETING\_RESPONSES = ["hi", "hey", "\*nods\*", "hi there", "hello", "I am glad! You are talking to me"]

# Checking for greetings

def greeting(sentence):

"""If user's input is a greeting, return a greeting response"""

for word in sentence.split():

if word.lower() in GREETING\_INPUTS:

return random.choice(GREETING\_RESPONSES)

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

# Generating response

def response(user\_response):

robo\_response=''

sent\_tokens.append(user\_response)

TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop\_words='english')

tfidf = TfidfVec.fit\_transform(sent\_tokens)

vals = cosine\_similarity(tfidf[-1], tfidf)

idx=vals.argsort()[0][-2]

flat = vals.flatten()

flat.sort()

req\_tfidf = flat[-2]

if(req\_tfidf==0):

robo\_response=robo\_response+"I am sorry! I don't understand you"

return robo\_response

else:

robo\_response = robo\_response+sent\_tokens[idx]

return robo\_response

flag=True

print("ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye!")

while(flag==True):

user\_response = input()

user\_response=user\_response.lower()

if(user\_response!='bye'):

if(user\_response=='thanks' or user\_response=='thank you' ):

flag=False

print("ROBO: You are welcome..")

else:

if(greeting(user\_response)!=None):

print("ROBO: "+greeting(user\_response))

else:

print("ROBO: ",end="")

print(response(user\_response))

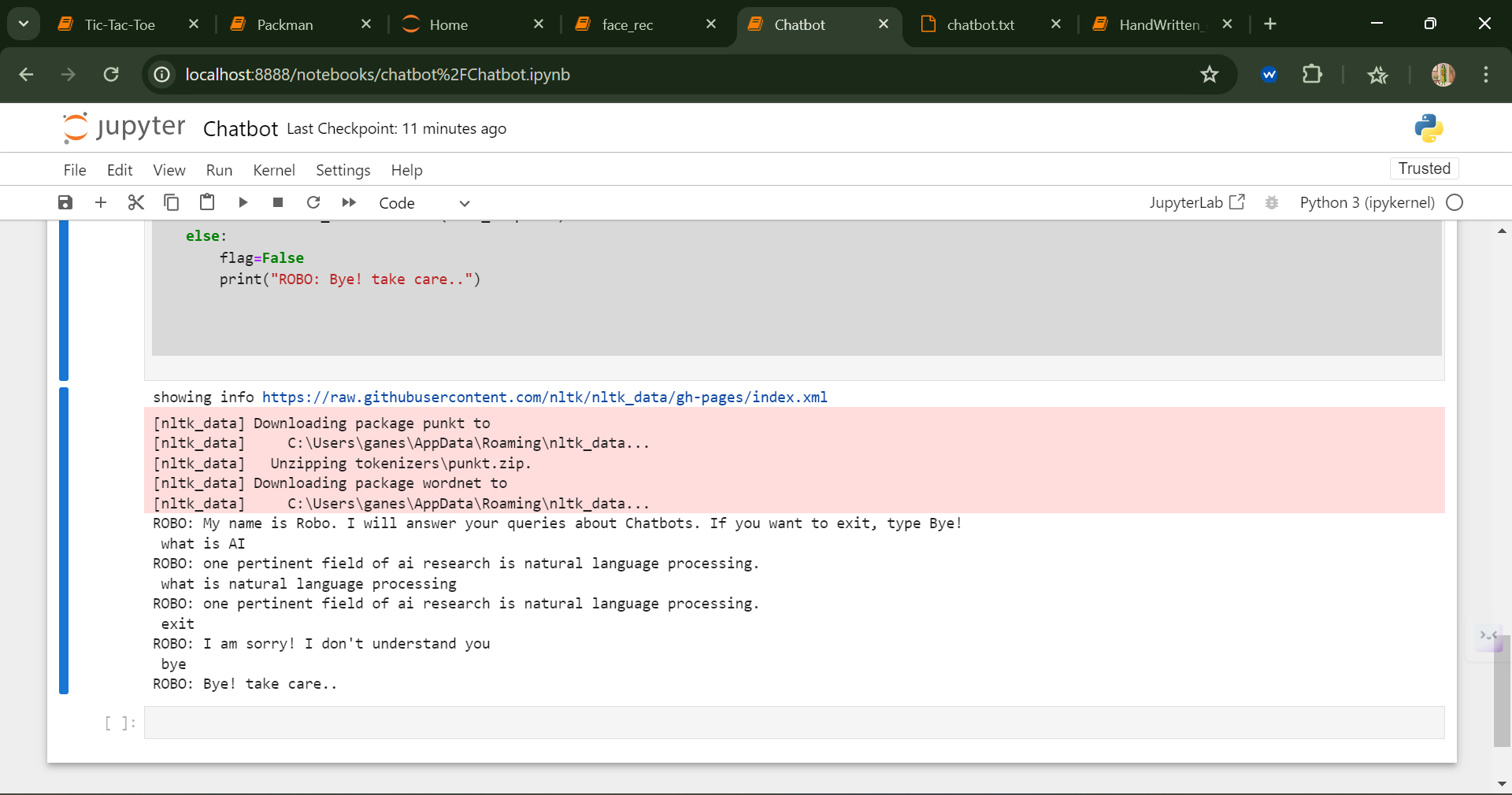
sent\_tokens.remove(user\_response)

else:

flag=False

print("ROBO: Bye! take care..")

## OUTPUT:



## RESULE:

Thus, an appropriate reasoning algorithm for question answering chatbot using python has been implemented and executed successfully.

|  |  |
| --- | --- |
| Ex No : 04DATE:5/4/24 | Develop a hand-written character recognition learning model program |

## AIM:

The aim of the experiment is to code the CNN Handwritten recognition program using Keras.

## ALGORITHM:

1. Import the required keras package to use keras functions in our python code
2. Import the mnist input database from keras.datasets
3. Initialize the batch size, number of classes and number of epochs.
4. Load the mnist dataset and split the dataset into training, test samples.
5. Convert the train and test vectors into binary using keras.utils.to\_categorical function.
6. Load the sequential model and declare the activation functions relu and softmax.
7. Fit the model with the training data by providing the corresponding parameter values.
8. Evaluate the model with the test data.
9. Display the error and accuracy value after the test data.
10. Run the code in GPU and CPU machine to check the time taken.

## PROGRAM:

import numpy as np

from tensorflow import keras

from tensorflow.keras import layers

#Prepare the data

# Model / data parameters

num\_classes = 10

input\_shape = (28, 28, 1)

#Load the data and split it between train and test sets

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

# Scale images to the [0, 1] range

x\_train = x\_train.astype("float32") / 255

x\_test = x\_test.astype("float32") / 255

# Make sure images have shape (28, 28, 1)

x\_train = np.expand\_dims(x\_train, -1)

x\_test = np.expand\_dims(x\_test, -1)

print("x\_train shape:", x\_train.shape)

print(x\_train.shape[0], "train samples")

print(x\_test.shape[0], "test samples")

# convert class vectors to binary class matrices

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

#Build the modela

model = keras.Sequential(

[

keras.Input(shape=input\_shape),

layers.Conv2D(32, kernel\_size=(3, 3), activation="relu"),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Conv2D(64, kernel\_size=(3, 3), activation="relu"),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Flatten(),

layers.Dropout(0.5),

layers.Dense(num\_classes, activation="softmax"),

]

)

model.summary()

#TRAIN the MODEL

batch\_size = 128

epochs = 15

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

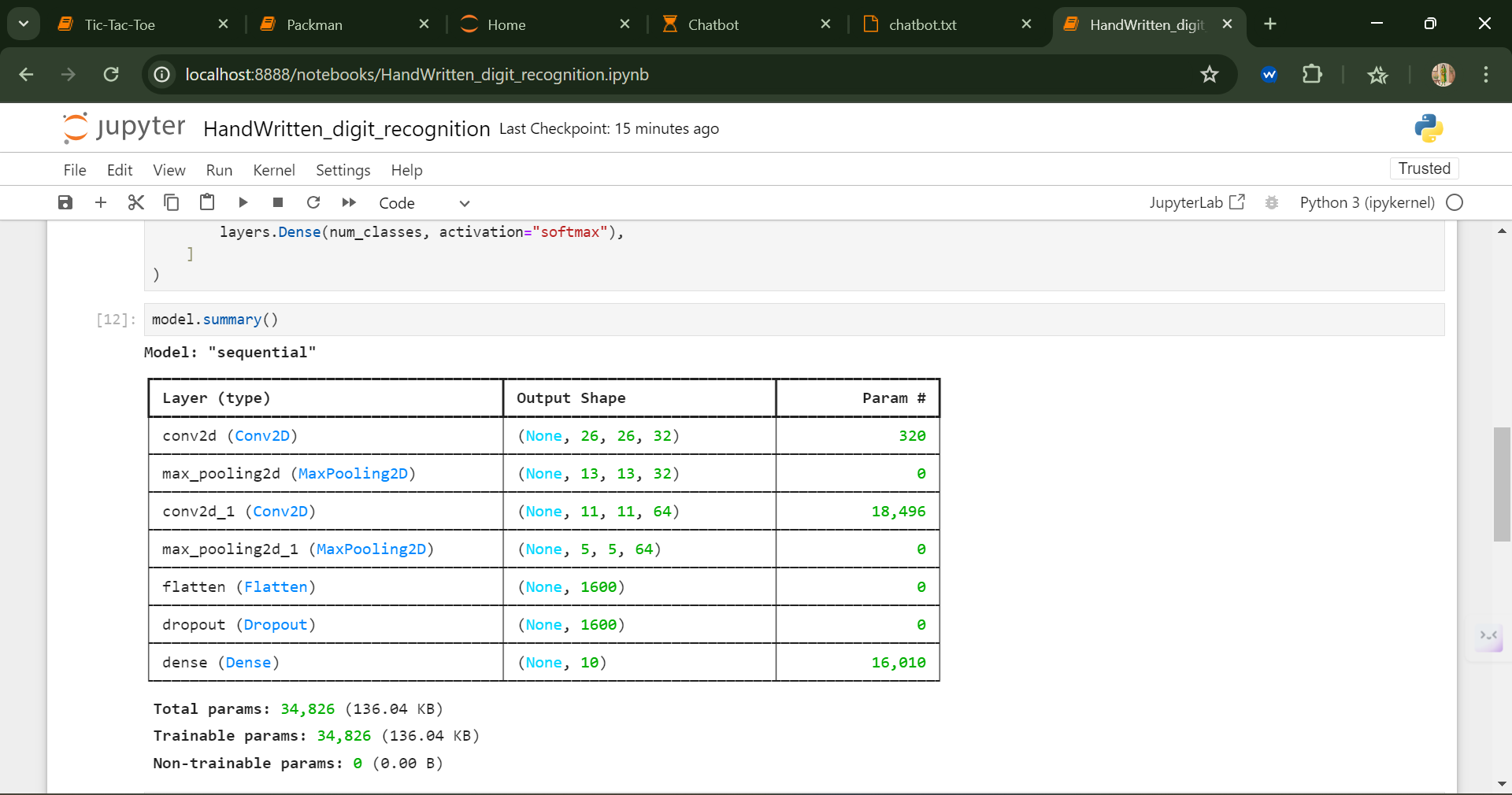
model.fit(x\_train, y\_train, batch\_size=batch\_size, epochs=epochs, validation\_split=0.1)

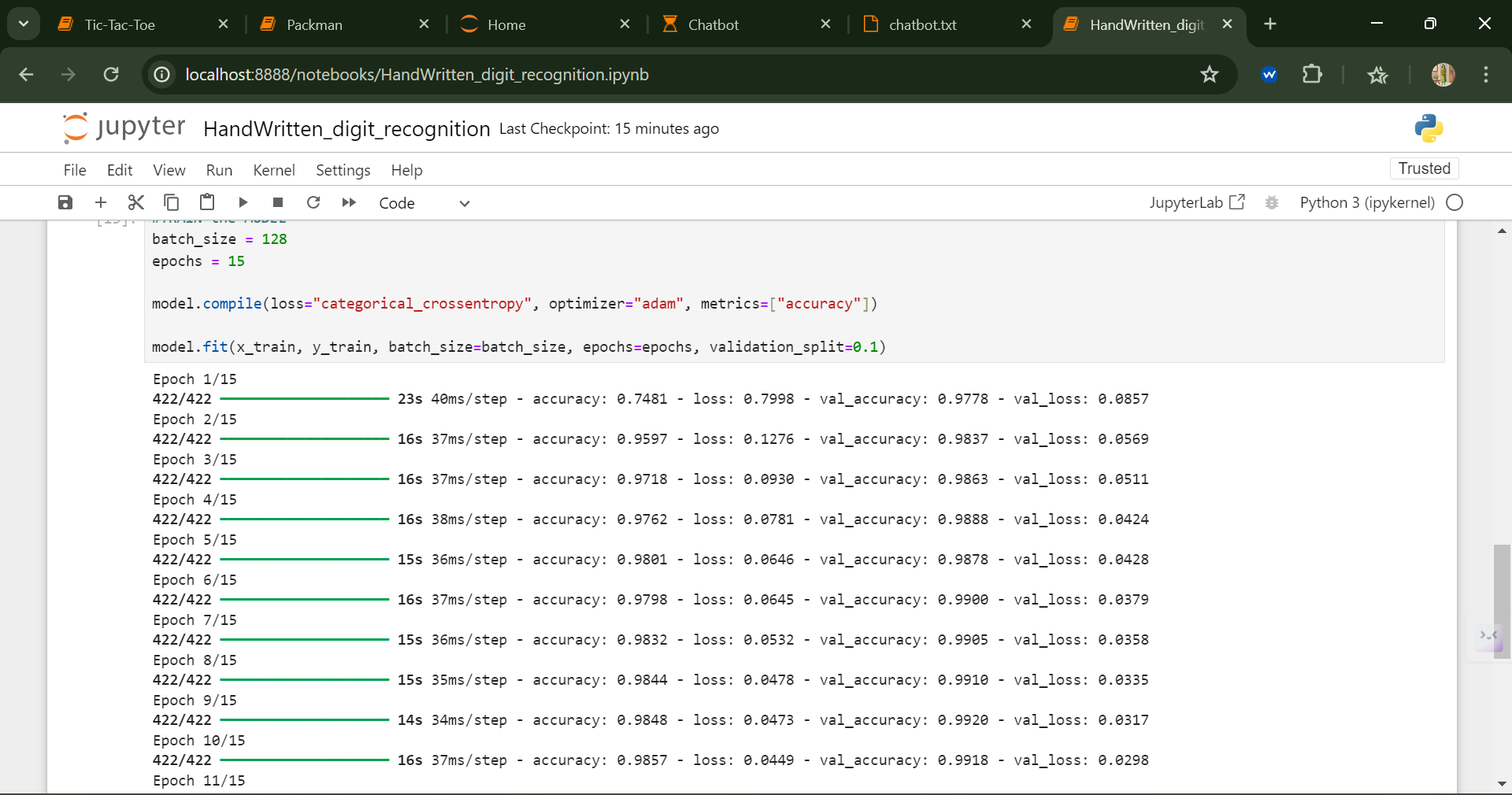
score = model.evaluate(x\_test, y\_test, verbose=0)

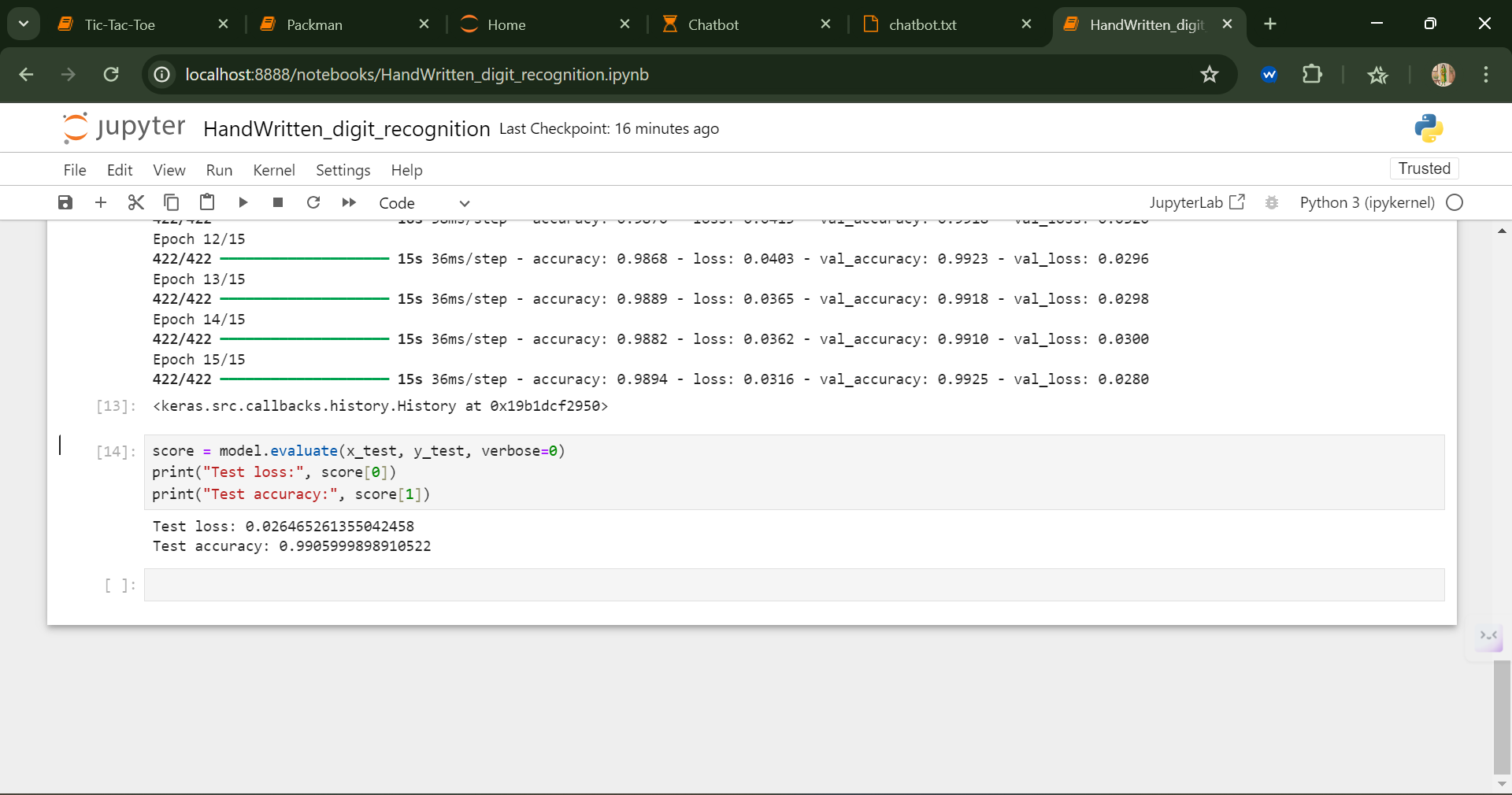
print("Test loss:", score[0])

print("Test accuracy:", score[1])

## OUTPUT:







## RESULE:

Thus, the CNN handwritten digit recognition program was coded using Keras successfully

|  |  |
| --- | --- |
| Ex No : 05DATE:5/4/24 | Construct a face recognition learning model using open source frameworks |

## AIM:

The aim of this experiment is to create a program that could recognizes faces.

## ALGORITHM:

1. Import the os and facerecognition library.
2. Create a images list by using listdir() function by specifying the corpus directory path
3. Load the image to be tested using load\_image\_file() function of face recognition library.
4. Convert the loaded image into feature vector using face\_encodings() function.
5. Similarly loop through the corpus directory and compare each image feature vector with the test image feature vector for a match.
6. Stop the loop if there is a match or when there are no more images in the corpus.

## PROGRAM:

!mkdir known

!wget https://upload.wikimedia.org/wikipedia/commons/f/f9/Obama\_portrait\_crop.jpg -O known/Obama.jpg

!wget https://upload.wikimedia.org/wikipedia/commons/a/a5/President\_of\_the\_United\_States\_Joe\_Biden\_%282021%29.jpg -O known/biden.jpg

!wget https://upload.wikimedia.org/wikipedia/commons/6/6e/A.\_P.\_J.\_Abdul\_Kalam.jpg -O known/\_Abdul\_Kalam.jpg

#!wget https://www.biography.com/.image/t\_share/MTE4MDAzNDEwNzg5ODI4MTEw/barack-obama-12782369-1-402.jpg -O known/obama.jpg

!mkdir unknown

#!wget https://i.insider.com/5ddfa893fd9db26b8a4a2df7 -O unknown/1.jpg

#!wget https://cdn-images-1.medium.com/max/1200/1\*aEoYLgy4z1lT1kW7dqWzBg.jpeg -O unknown/2.jpg

#!wget https://upload.wikimedia.org/wikipedia/commons/6/68/Joe\_Biden\_presidential\_portrait.jpg -O unknown/3.jpg

!wget https://upload.wikimedia.org/wikipedia/commons/a/a0/A\_P\_J\_Abdul\_Kalam.jpg -O unknown/5.jpg

#!wget https://specials-images.forbesimg.com/imageserve/1184274010/960x0.jpg -O unknown/4.jpg

import face\_recognition

import cv2

import os

from google.colab.patches import cv2\_imshow

def read\_img(path):

img = cv2.imread(path)

(h, w) = img.shape[:2]

width = 500

ratio = width / float(w)

height = int(h \* ratio)

return cv2.resize(img, (width, height))

for file in os.listdir(known\_dir):

img = read\_img(known\_dir + '/' + file)

img\_enc = face\_recognition.face\_encodings(img)[0]

known\_encodings.append(img\_enc)

known\_names.append(file.split('.')[0])

unknown\_dir = 'unknown'

for file in os.listdir(unknown\_dir):

print("Processing", file)

img = read\_img(unknown\_dir + '/' + file)

img\_enc = face\_recognition.face\_encodings(img)[0]

results = face\_recognition.compare\_faces(known\_encodings, img\_enc)

# print(face\_recognition.face\_distance(known\_encodings, img\_enc))

for i in range(len(results)):

if results[i]:

name = known\_names[i]

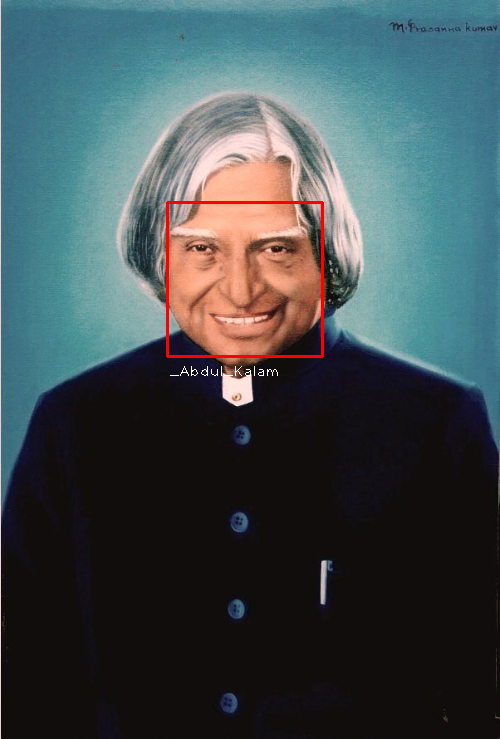
(top, right, bottom, left) = face\_recognition.face\_locations(img)[0]

cv2.rectangle(img, (left, top), (right, bottom), (0, 0, 255), 2)

cv2.putText(img, name, (left+2, bottom+20), cv2.FONT\_HERSHEY\_PLAIN, 1, (255, 255, 255), 1)

cv2\_imshow(img)

## OUTPUT:



## RESULE

Thus, the face recognition is successfully implemented using a learning library in python.

|  |  |
| --- | --- |
| Ex No : 06DATE:12/4/24 | Construct an Object detection learning model for Robot Arm Tracking using open source frameworks |

## AIM:

The aim of this experiment is to identify objects from a given image using learning frameworks.

## ALGORITHM:

1. Import the necessary libraries required for object detection like os, matplotlib, numpy, tesorflow etc.
2. To display image outputs in the jupyter notebook include “%matplotlib inline”
3. Import the object detection modules to incorporate the models we have downloaded.
4. Download any pretrained learning models frozen graph and load it.
5. Download the label text files to create bounding boxes.
6. Download the model using the retrieve() function and extract it using tar.
7. Load the downloaded label map.
8. Create a function to convert an image into a numpy array, so that it could be used in the learning model.
9. Provide the path to the image corpus for testing the model for object detection.
10. Create a function to process the image to find objects and place bounding boxes with labels on them.
11. Loop through the image corpus and process all images to detect objects in each processed image and display images with labeled bounding boxes.

## PROGRAM:

#@title Imports and function definitions

# For running inference on the TF-Hub module.

import tensorflow as tf

import tensorflow\_hub as hub

# For downloading the image.

import matplotlib.pyplot as plt

import tempfile

from six.moves.urllib.request import urlopen

from six import BytesIO

# For drawing onto the image.

import numpy as np

from PIL import Image

from PIL import ImageColor

from PIL import ImageDraw

from PIL import ImageFont

from PIL import ImageOps

# For measuring the inference time.

import time

# Print Tensorflow version

print(tf.\_\_version\_\_)

# Check available GPU devices.

print("The following GPU devices are available: %s" % tf.test.gpu\_device\_name())

def display\_image(image):

fig = plt.figure(figsize=(20, 15))

plt.grid(False)

plt.imshow(image)

def download\_and\_resize\_image(url, new\_width=256, new\_height=256,

display=False):

\_, filename = tempfile.mkstemp(suffix=".jpg")

response = urlopen(url)

image\_data = response.read()

image\_data = BytesIO(image\_data)

pil\_image = Image.open(image\_data)

pil\_image = ImageOps.fit(pil\_image, (new\_width, new\_height), Image.ANTIALIAS)

pil\_image\_rgb = pil\_image.convert("RGB")

pil\_image\_rgb.save(filename, format="JPEG", quality=90)

print("Image downloaded to %s." % filename)

if display:

display\_image(pil\_image)

return filename

def draw\_bounding\_box\_on\_image(image,

ymin,

xmin,

ymax,

xmax,

color,

font,

thickness=4,

display\_str\_list=()):

"""Adds a bounding box to an image."""

draw = ImageDraw.Draw(image)

im\_width, im\_height = image.size

(left, right, top, bottom) = (xmin \* im\_width, xmax \* im\_width,

ymin \* im\_height, ymax \* im\_height)

draw.line([(left, top), (left, bottom), (right, bottom), (right, top),

(left, top)],

width=thickness,

fill=color)

# If the total height of the display strings added to the top of the bounding

# box exceeds the top of the image, stack the strings below the bounding box

# instead of above.

display\_str\_heights = [font.getsize(ds)[1] for ds in display\_str\_list]

# Each display\_str has a top and bottom margin of 0.05x.

total\_display\_str\_height = (1 + 2 \* 0.05) \* sum(display\_str\_heights)

if top > total\_display\_str\_height:

text\_bottom = top

else:

text\_bottom = top + total\_display\_str\_height

# Reverse list and print from bottom to top.

for display\_str in display\_str\_list[::-1]:

text\_width, text\_height = font.getsize(display\_str)

margin = np.ceil(0.05 \* text\_height)

draw.rectangle([(left, text\_bottom - text\_height - 2 \* margin),

(left + text\_width, text\_bottom)],

fill=color)

draw.text((left + margin, text\_bottom - text\_height - margin),

display\_str,

fill="black",

font=font)

text\_bottom -= text\_height - 2 \* margin

def draw\_boxes(image, boxes, class\_names, scores, max\_boxes=10, min\_score=0.1):

"""Overlay labeled boxes on an image with formatted scores and label names."""

colors = list(ImageColor.colormap.values())

try:

font = ImageFont.truetype("/usr/share/fonts/truetype/liberation/LiberationSansNarrow-Regular.ttf",

25)

except IOError:

print("Font not found, using default font.")

font = ImageFont.load\_default()

for i in range(min(boxes.shape[0], max\_boxes)):

if scores[i] >= min\_score:

ymin, xmin, ymax, xmax = tuple(boxes[i])

display\_str = "{}: {}%".format(class\_names[i].decode("ascii"),

int(100 \* scores[i]))

color = colors[hash(class\_names[i]) % len(colors)]

image\_pil = Image.fromarray(np.uint8(image)).convert("RGB")

draw\_bounding\_box\_on\_image(

image\_pil,

ymin,

xmin,

ymax,

xmax,

color,

font,

display\_str\_list=[display\_str])

np.copyto(image, np.array(image\_pil))

return image

# By Heiko Gorski, Source: https://commons.wikimedia.org/wiki/File:Naxos\_Taverna.jpg

image\_url = "https://upload.wikimedia.org/wikipedia/commons/6/60/Naxos\_Taverna.jpg" #@param

downloaded\_image\_path = download\_and\_resize\_image(image\_url, 1280, 856, True)

module\_handle = "https://tfhub.dev/google/faster\_rcnn/openimages\_v4/inception\_resnet\_v2/1" #@param ["https://tfhub.dev/google/openimages\_v4/ssd/mobilenet\_v2/1", "https://tfhub.dev/google/faster\_rcnn/openimages\_v4/inception\_resnet\_v2/1"]

detector = hub.load(module\_handle).signatures['default']

def load\_img(path):

img = tf.io.read\_file(path)

img = tf.image.decode\_jpeg(img, channels=3)

return img

def run\_detector(detector, path):

img = load\_img(path)

converted\_img = tf.image.convert\_image\_dtype(img, tf.float32)[tf.newaxis, ...]

start\_time = time.time()

result = detector(converted\_img)

end\_time = time.time()

result = {key:value.numpy() for key,value in result.items()}

print("Found %d objects." % len(result["detection\_scores"]))

print("Inference time: ", end\_time-start\_time)

image\_with\_boxes = draw\_boxes(

img.numpy(), result["detection\_boxes"],

result["detection\_class\_entities"], result["detection\_scores"])

display\_image(image\_with\_boxes)

run\_detector(detector, downloaded\_image\_path)

image\_urls = [

# Source: https://commons.wikimedia.org/wiki/File:The\_Coleoptera\_of\_the\_British\_islands\_(Plate\_125)\_(8592917784).jpg

"https://upload.wikimedia.org/wikipedia/commons/1/1b/The\_Coleoptera\_of\_the\_British\_islands\_%28Plate\_125%29\_%288592917784%29.jpg",

# By Américo Toledano, Source: https://commons.wikimedia.org/wiki/File:Biblioteca\_Maim%C3%B3nides,\_Campus\_Universitario\_de\_Rabanales\_007.jpg

"https://upload.wikimedia.org/wikipedia/commons/thumb/0/0d/Biblioteca\_Maim%C3%B3nides%2C\_Campus\_Universitario\_de\_Rabanales\_007.jpg/1024px-Biblioteca\_Maim%C3%B3nides%2C\_Campus\_Universitario\_de\_Rabanales\_007.jpg",

# Source: https://commons.wikimedia.org/wiki/File:The\_smaller\_British\_birds\_(8053836633).jpg

"https://upload.wikimedia.org/wikipedia/commons/0/09/The\_smaller\_British\_birds\_%288053836633%29.jpg",

]

def detect\_img(image\_url):

start\_time = time.time()

image\_path = download\_and\_resize\_image(image\_url, 640, 480)

run\_detector(detector, image\_path)

end\_time = time.time()

print("Inference time:",end\_time-start\_time)

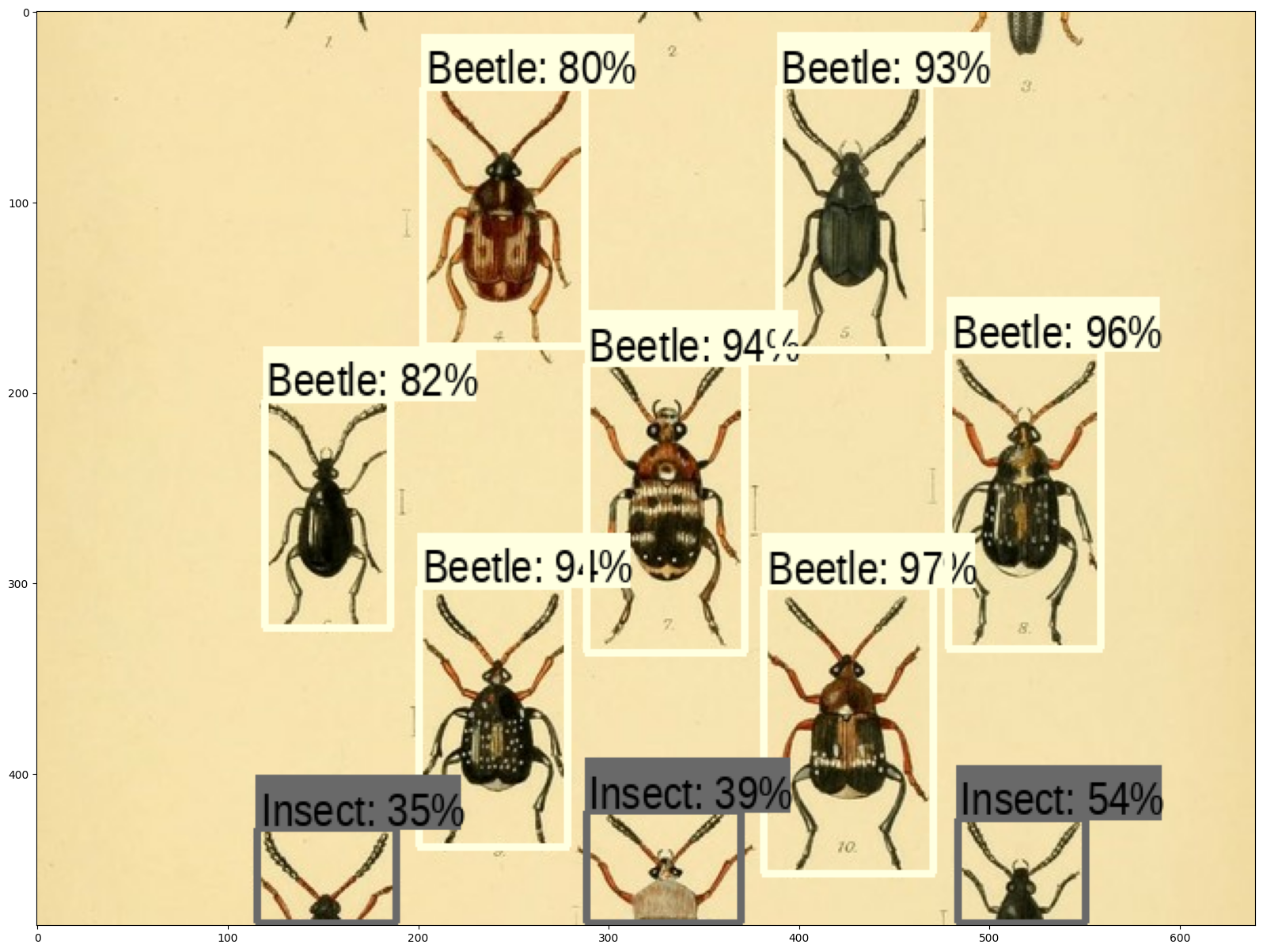
detect\_img(image\_urls[0])

detect\_img(image\_urls[1])

detect\_img(image\_urls[2])

## OUTPUT:





## RESULE:

Thus, the object detection learning model is successfully implemented using deep learning frameworks.