Ex No : 01 **Program to help packman find his food using A* search algorithm**

DATE:23/3/24

AIM:

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

ALGORITHM:

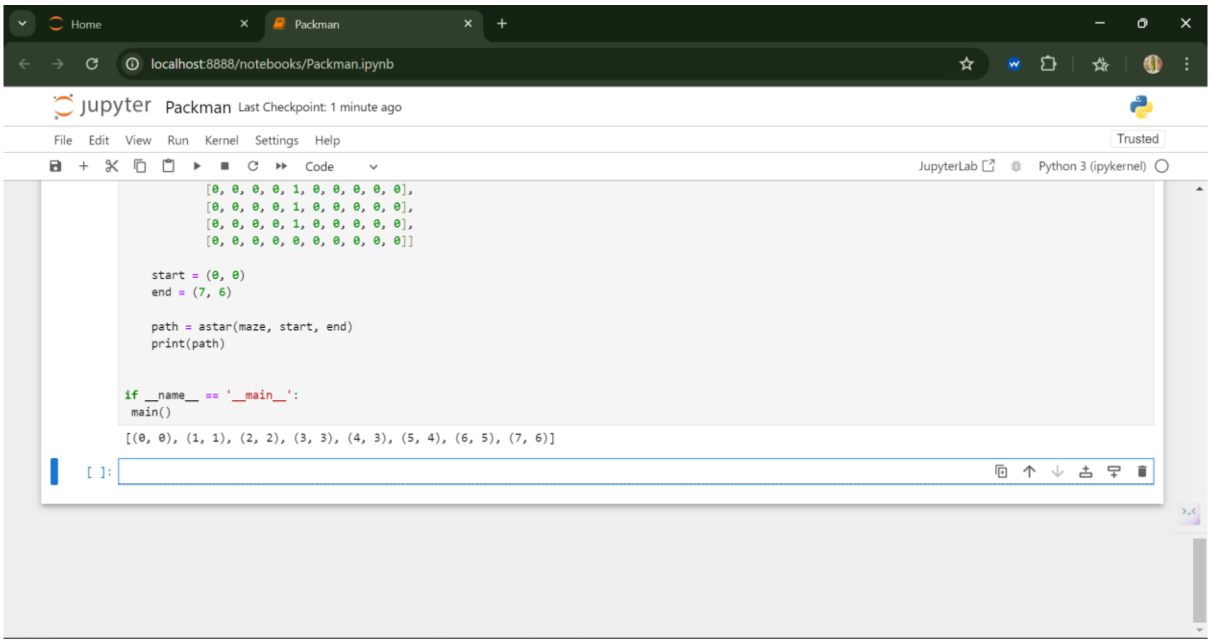
1. Initialize all the values like g, h, f, position by creating a init function
2. Create a base function for A-star searching, which takes in parameters like start point, destination goal. and entire grid layout.
3. The A-star function should have an open set and closed set, and the current point should 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:



Home
Packman
localhost:8888/notebooks/Packman.ipynb
jupyter Packman Last Checkpoint: 1 minute ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
            [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()
```

```
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 3), (5, 4), (6, 5), (7, 6)]
```

```
[ ]:
```

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 : 02 Use Minimax algorithm for solving Tic-Tac-Toe game

DATE:8/3/24

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.
```

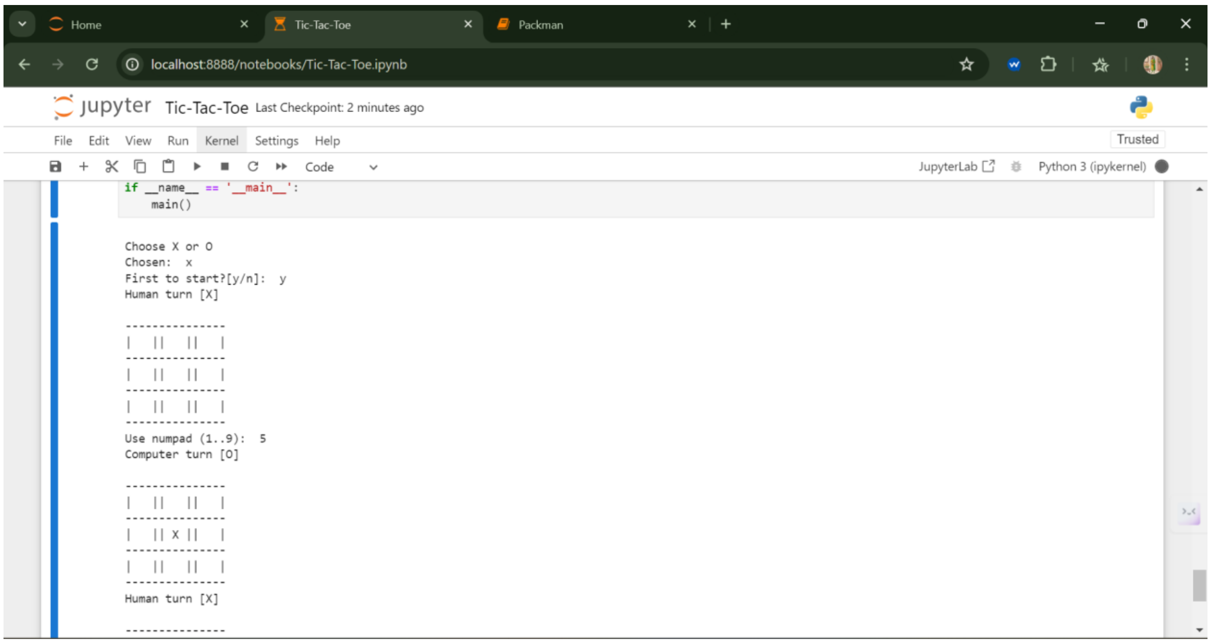
```
Author: Clederson Cruz
Year: 2017
License: GNU GENERAL PUBLIC LICENSE (GPL)
"""


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
```

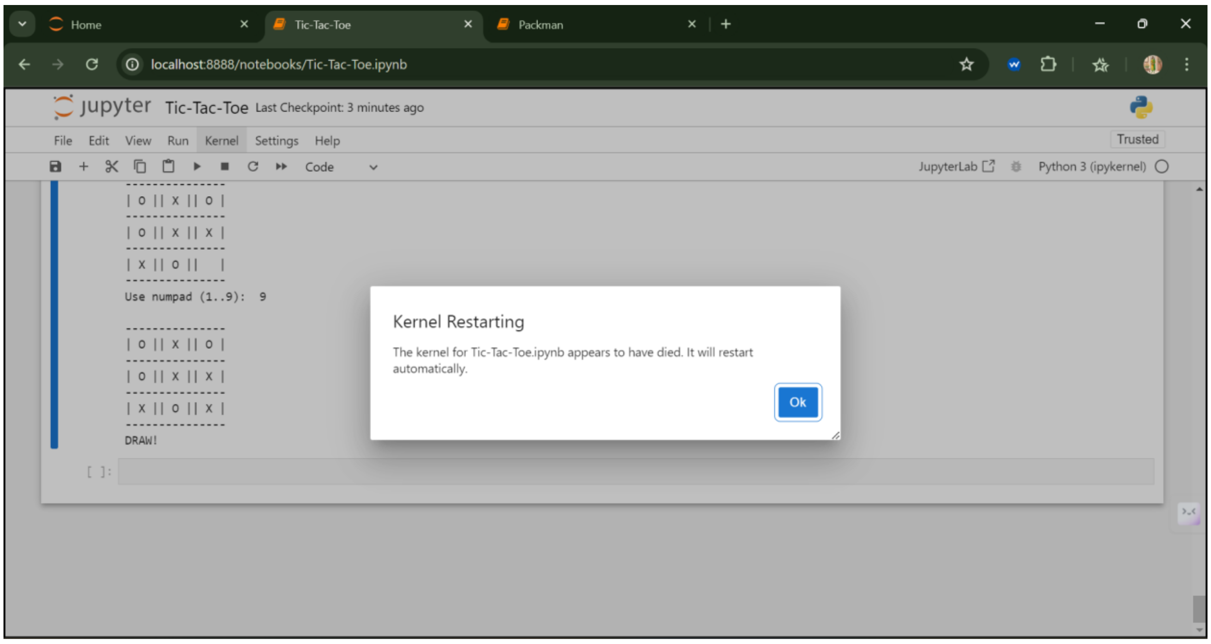
```
    if wins(board, HUMAN):
        clean()
        print(f'Human turn [{h_choice}]')
        render(board, c_choice, h_choice)
        print('YOU WIN!')
    elif wins(board, COMP):
        clean()
        print(f'Computer turn [{c_choice}]')
        render(board, c_choice, h_choice)
        print('YOU LOSE!')
    else:
        clean()
        render(board, c_choice, h_choice)
        print('DRAW!')

    exit()

if __name__ == '__main__':
    main()
```

OUTPUT:


Home
Tic-Tac-Toe
Packman
localhost:8888/notebooks/Tic-Tac-Toe.ipynb
jupyter Tic-Tac-Toe Last Checkpoint: 2 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
if __name__ == '__main__':
    main()
```

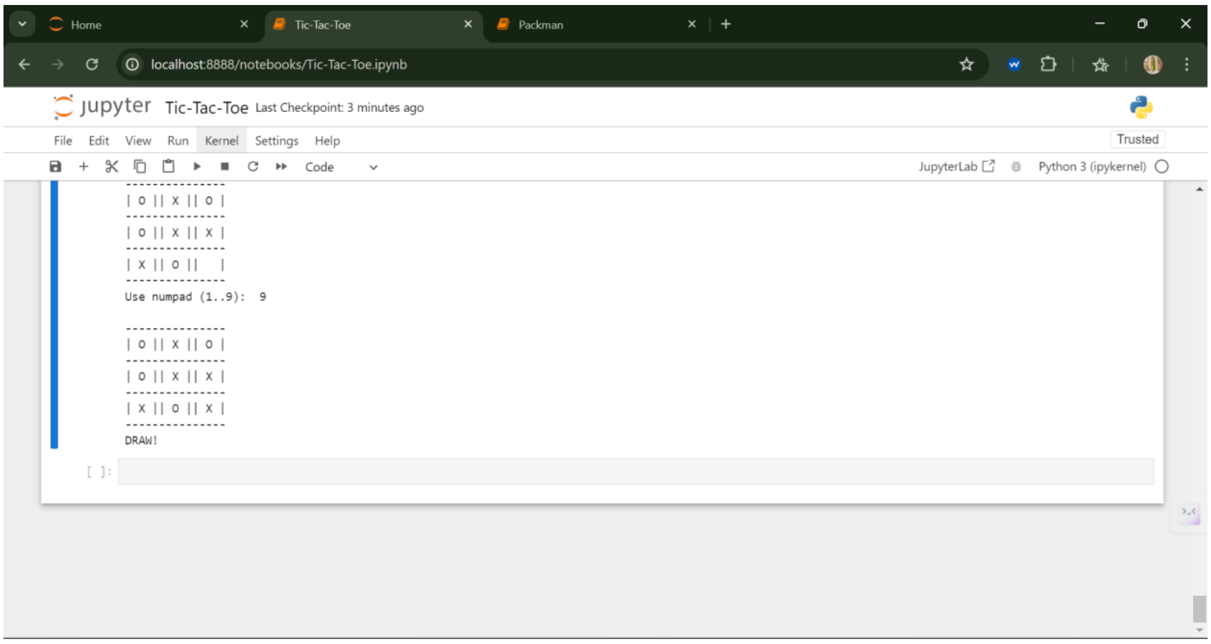
```
Choose X or O
Chosen:  x
First to start?[y/n]:  y
Human turn [X]

--------------
|   ||   ||   |
--------------
|   ||   ||   |
--------------
|   ||   ||   |
--------------
Use numpad (1..9):  5
Computer turn [O]

--------------
|   ||   ||   |
--------------
|   || X ||   |
--------------
|   ||   ||   |
--------------
Human turn [X]

--------------
```


Home
Tic-Tac-Toe
Packman
localhost:8888/notebooks/Tic-Tac-Toe.ipynb
Jupyter Tic-Tac-Toe Last Checkpoint: 3 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
 --------------
 | O || X || O |
 --------------
 | O || X || X |
 --------------
 | X || O ||   |
 --------------
Use numpad (1..9):  9

 --------------
 | O || X || O |
 --------------
 | O || X || X |
 --------------
 | X || O || X |
 --------------
DRAW!
```

Kernel Restarting

The kernel for Tic-Tac-Toe.ipynb appears to have died. It will restart automatically.


Home
Tic-Tac-Toe
Packman
localhost:8888/notebooks/Tic-Tac-Toe.ipynb
Jupyter Tic-Tac-Toe Last Checkpoint: 3 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
 --------------
 | O || X || O |
 --------------
 | O || X || X |
 --------------
 | X || O ||   |
 --------------
Use numpad (1..9):  9

 --------------
 | O || X || O |
 --------------
 | O || X || X |
 --------------
 | X || O || X |
 --------------
DRAW!
```

RESULE:

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

Ex No : 03 Develop an appropriate reasoning algorithm for question answering system

DATE:15/3/24

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:

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

Prerequisites to run the program:

13. Save the program and the text file in the same folder and do the following
14. $ sudo apt-get install python3-pip
15. Save the program and in the command prompt navigate to the folder.
16. cole@unisys-58:~/Building-a-Simple-Chatbot-in-Python-using-NLTK-master$
17. Install the following packages as given: $ sudo pip3 install numpy $ sudo pip3 install sklearn
18. Run the program cole@unisys-58:~/Building-a-Simple-Chatbot-in-Python-using-NLTK-master$ python3 chatbot.py
19. 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
---------------------------------------------------------------
```

20. Downloader> q
21. 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.

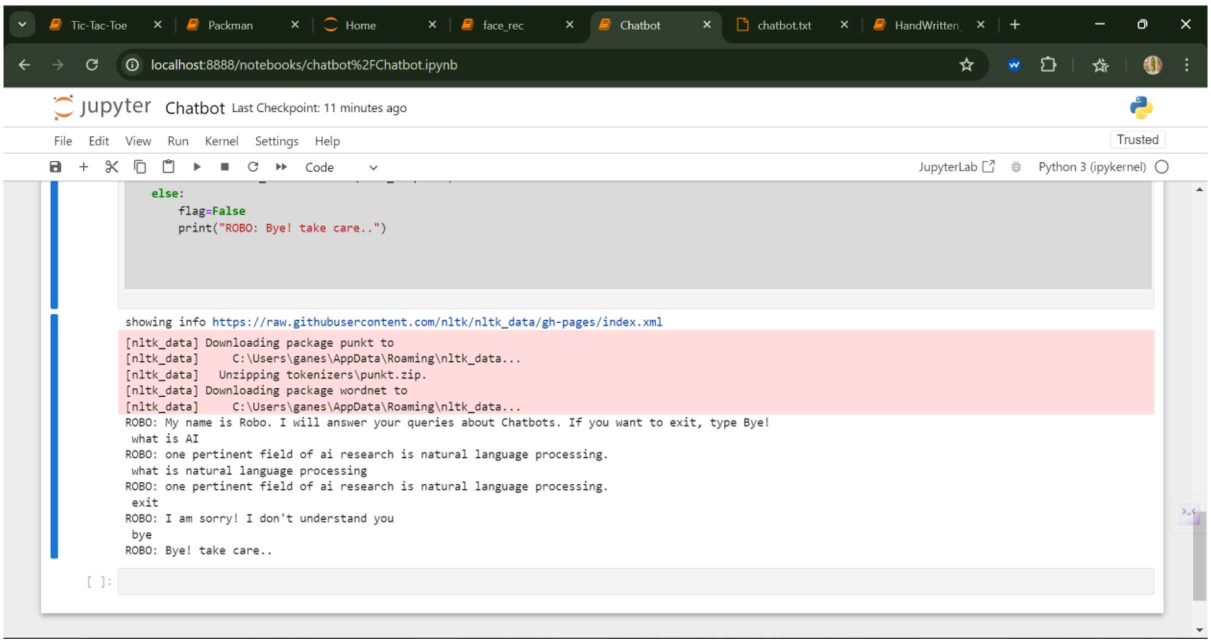
PROGRAM:

```
# 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:


Tic-Tac-Toe
Packman
Home
face_rec
Chatbot
chatbot.txt
HandWritten
localhost:8888/notebooks/chatbot%2FChatbot.ipynb
jupyter Chatbot Last Checkpoint: 11 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
    else:
        flag=False
        print("ROBO: Bye! take care..")
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ganes\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping tokenizers\punkt.zip.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\ganes\AppData\Roaming\nltk_data...
ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to exit, type Bye!
 what is AI
ROBO: one pertinent field of ai research is natural language processing.
 what is natural language processing
ROBO: one pertinent field of ai research is natural language processing.
 exit
ROBO: I am sorry! I don't understand you
 bye
ROBO: Bye! take care..
```

```
[ ]:
```

RESULE:

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

Ex No : 04 **Develop a hand-written character recognition learning model program**

DATE:5/4/24

AIM:

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

ALGORITHM:

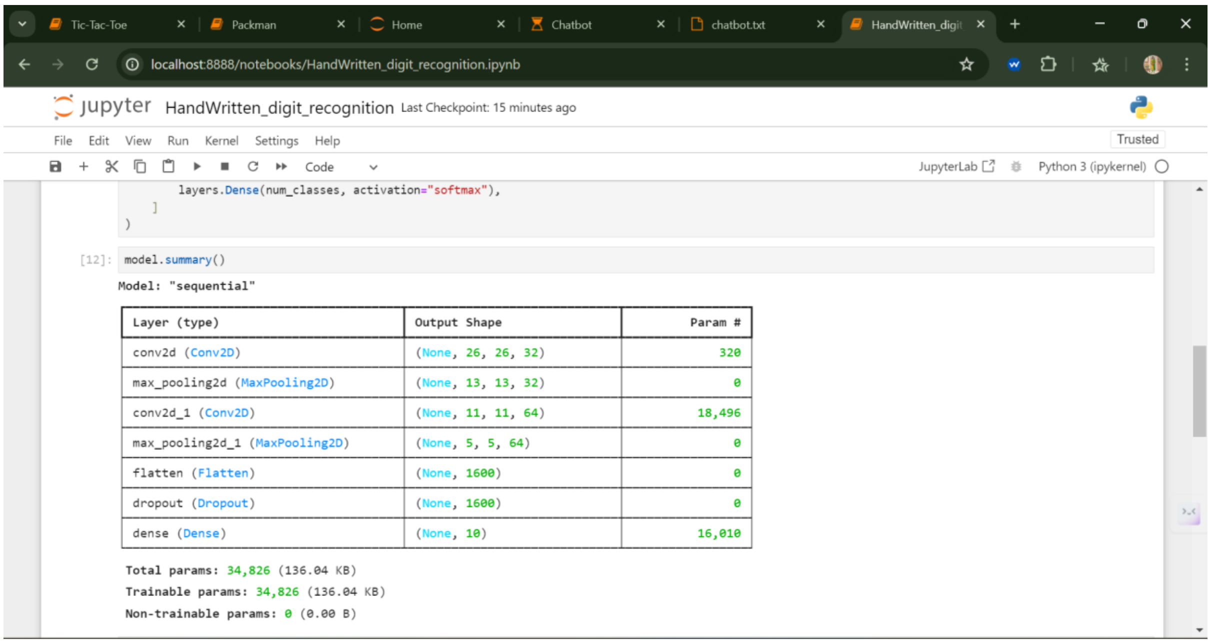
1. Import the required keras package to use keras functions in our python code
2. Import the mnist input database from keras.datasets
3. Initialize the batch size, number of classes and number of epochs.
4. Load the mnist dataset and split the dataset into training, test samples.
5. Convert the train 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])
```

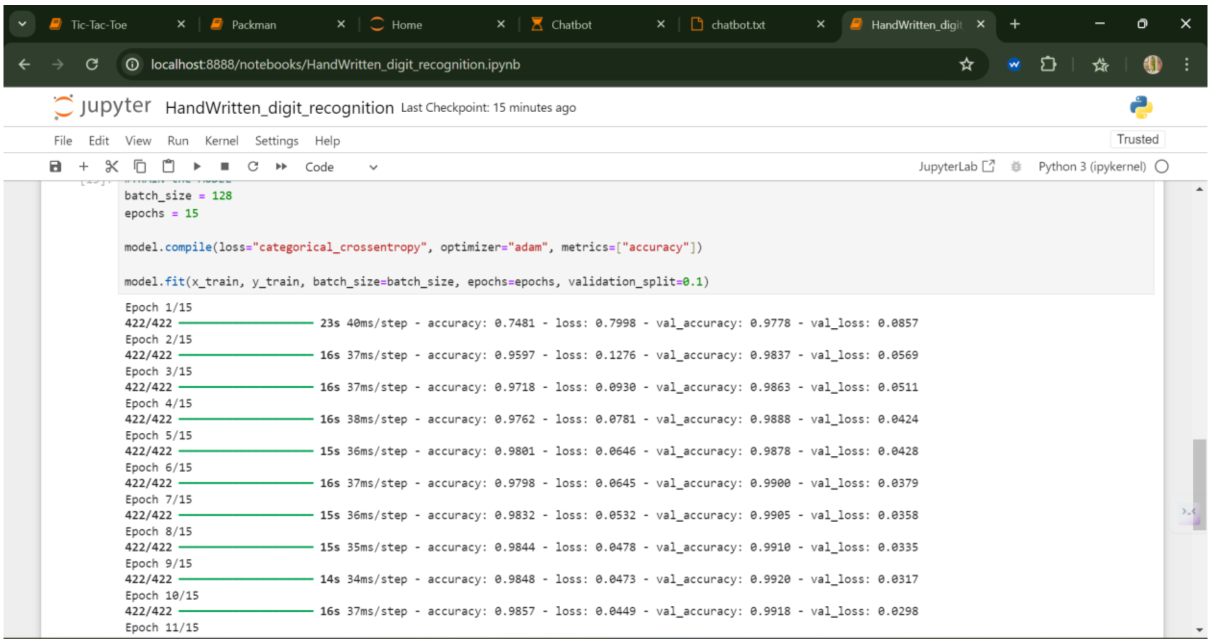
OUTPUT:



```
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

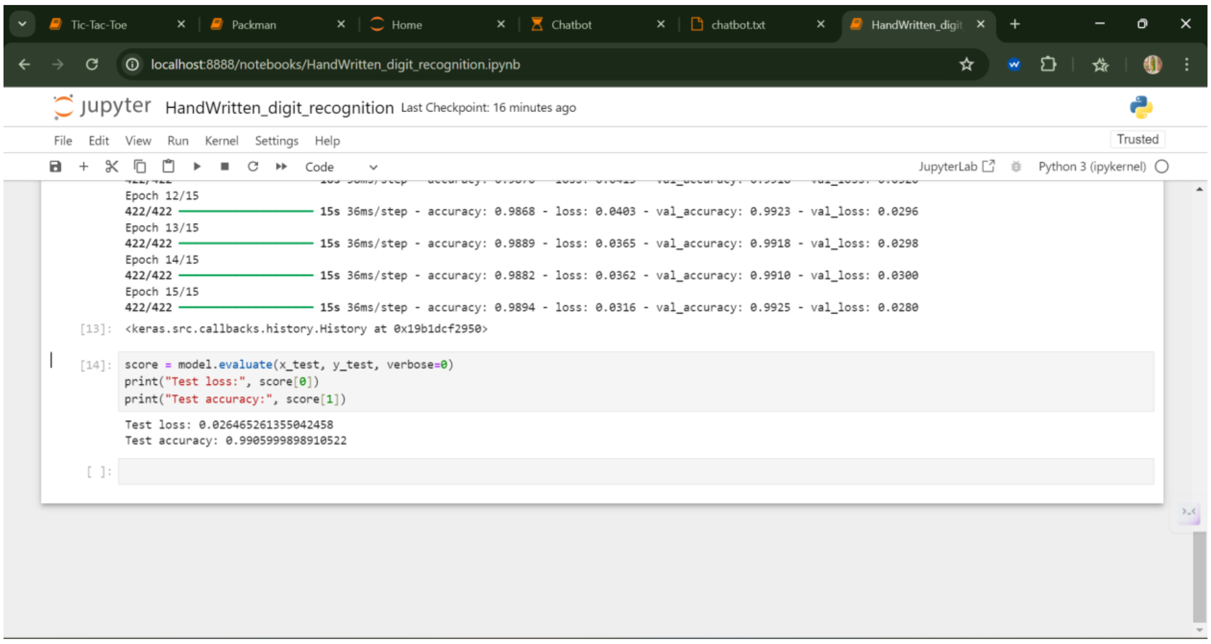
```
[12]: model.summary()
```

```
Model: "sequential"

┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d (Conv2D)                 │ (None, 26, 26, 32)     │           320 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 13, 13, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 11, 11, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 5, 5, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 1600)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 1600)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 10)             │        16,010 │
└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 34,826 (136.04 KB)
 Trainable params: 34,826 (136.04 KB)
 Non-trainable params: 0 (0.00 B)
```



```
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)
```

```
Epoch 1/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 23s 40ms/step - accuracy: 0.7481 - loss: 0.7998 - val_accuracy: 0.9778 - val_loss: 0.0857
Epoch 2/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 16s 37ms/step - accuracy: 0.9597 - loss: 0.1276 - val_accuracy: 0.9837 - val_loss: 0.0569
Epoch 3/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 16s 37ms/step - accuracy: 0.9718 - loss: 0.0930 - val_accuracy: 0.9863 - val_loss: 0.0511
Epoch 4/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 16s 38ms/step - accuracy: 0.9762 - loss: 0.0781 - val_accuracy: 0.9888 - val_loss: 0.0424
Epoch 5/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 15s 36ms/step - accuracy: 0.9801 - loss: 0.0646 - val_accuracy: 0.9878 - val_loss: 0.0428
Epoch 6/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 16s 37ms/step - accuracy: 0.9798 - loss: 0.0645 - val_accuracy: 0.9900 - val_loss: 0.0379
Epoch 7/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 15s 36ms/step - accuracy: 0.9832 - loss: 0.0532 - val_accuracy: 0.9905 - val_loss: 0.0358
Epoch 8/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 15s 35ms/step - accuracy: 0.9844 - loss: 0.0478 - val_accuracy: 0.9910 - val_loss: 0.0335
Epoch 9/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 14s 34ms/step - accuracy: 0.9848 - loss: 0.0473 - val_accuracy: 0.9920 - val_loss: 0.0317
Epoch 10/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 16s 37ms/step - accuracy: 0.9857 - loss: 0.0449 - val_accuracy: 0.9918 - val_loss: 0.0298
Epoch 11/15
```


Tic-Tac-Toe
Packman
Home
Chatbot
chatbot.txt
HandWritten_digit
localhost:8888/notebooks/HandWritten_digit_recognition.ipynb
jupyter HandWritten_digit_recognition Last Checkpoint: 16 minutes ago
Trusted
File Edit View Run Kernel Settings Help
Code
JupyterLab
Python 3 (ipykernel)

```
Epoch 12/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 15s 36ms/step - accuracy: 0.9868 - loss: 0.0403 - val_accuracy: 0.9923 - val_loss: 0.0296
Epoch 13/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 15s 36ms/step - accuracy: 0.9889 - loss: 0.0365 - val_accuracy: 0.9918 - val_loss: 0.0298
Epoch 14/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 15s 36ms/step - accuracy: 0.9882 - loss: 0.0362 - val_accuracy: 0.9910 - val_loss: 0.0300
Epoch 15/15
422/422 ━━━━━━━━━━━━━━━━━━━━ 15s 36ms/step - accuracy: 0.9894 - loss: 0.0316 - val_accuracy: 0.9925 - val_loss: 0.0280
[13]: <keras.src.callbacks.history.History at 0x19b1dcf2950>
```

```
[14]: score = model.evaluate(x_test, y_test, verbose=0)
      print("Test loss:", score[0])
      print("Test accuracy:", score[1])

      Test loss: 0.026465261355042458
      Test accuracy: 0.9905999898910522

[ ]:
```

RESULE:

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

Ex No : 05 **Construct a face recognition learning model using open source frameworks**

DATE:5/4/24

AIM:

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

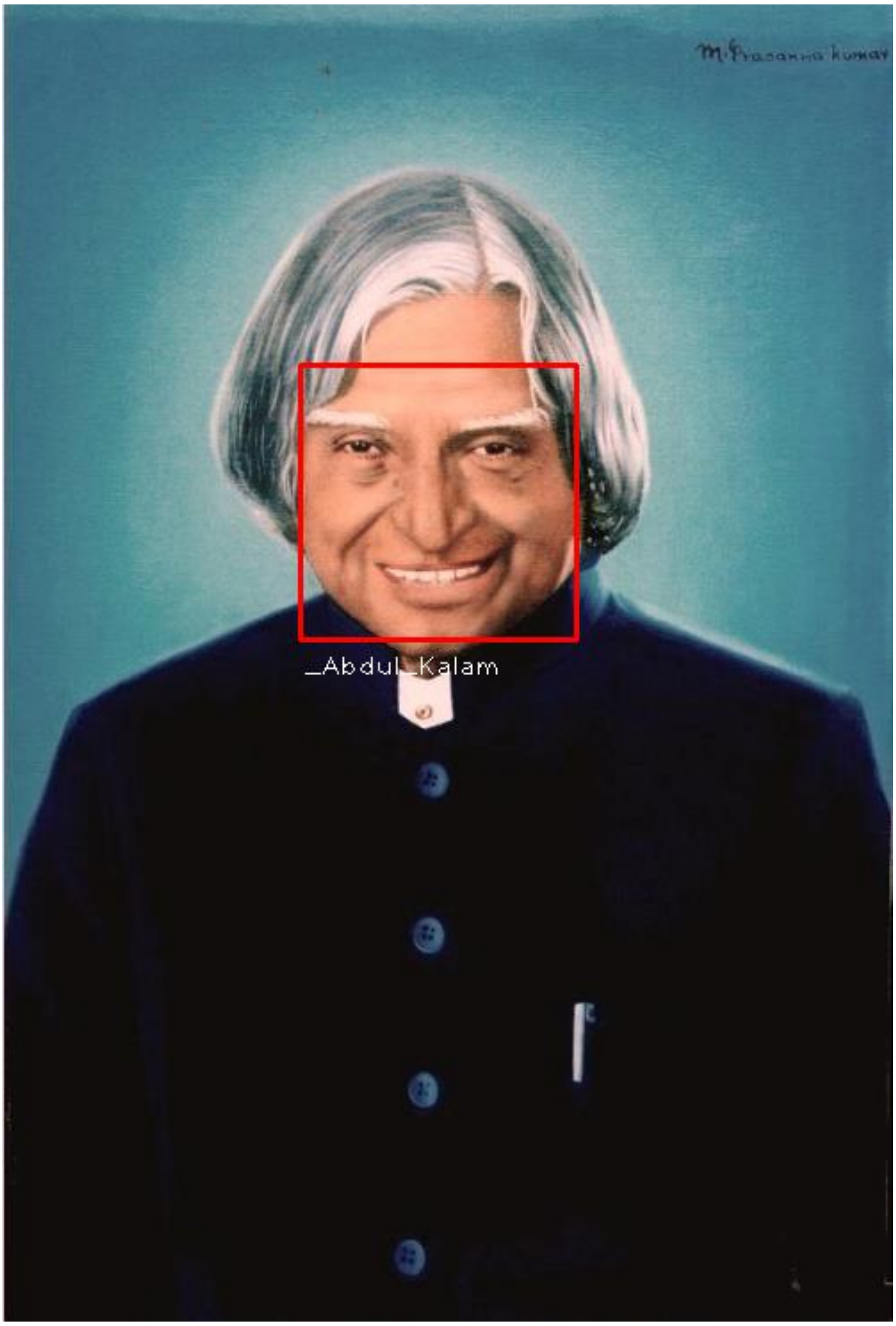
ALGORITHM:

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 : 06 **Construct an Object detection learning model for Robot Arm Tracking using open source frameworks**

DATE:12/4/24

AIM:

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

ALGORITHM:

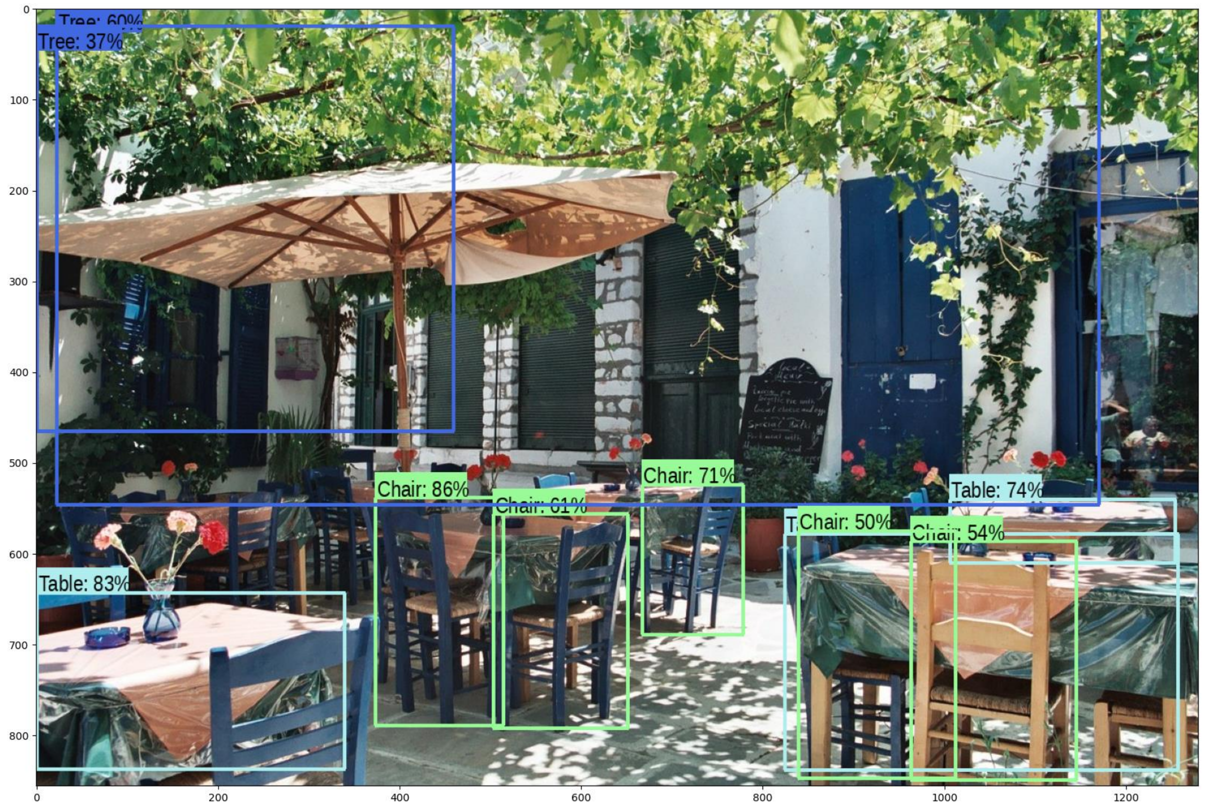
1. Import the necessary libraries required for object detection like os, matplotlib, numpy, tesorflow etc.
2. To display image outputs in the jupyter notebook include "%matplotlib inline"
3. Import the object detection modules to incorporate the models we have downloaded.
4. Download any pretrained learning 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.

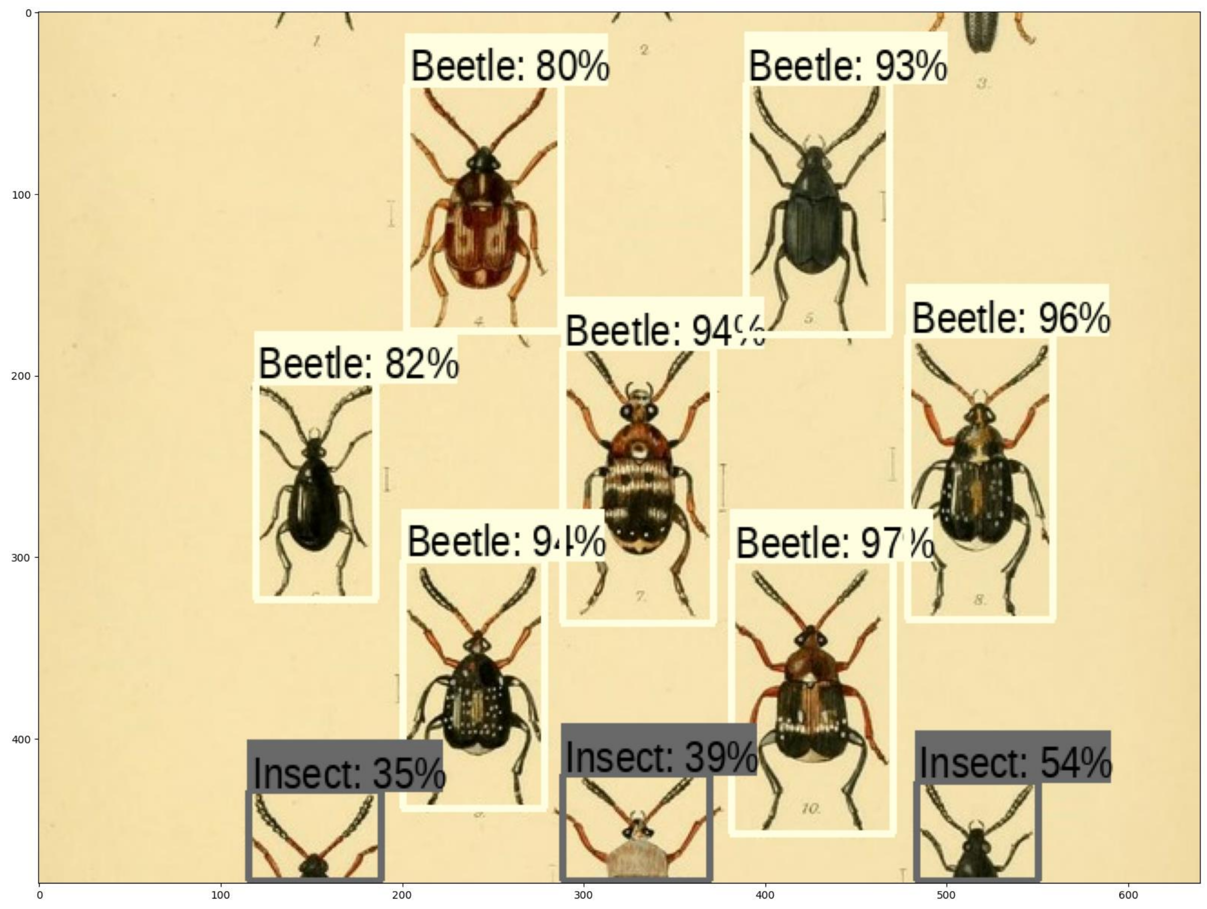
PROGRAM:

```
#@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)_(8592
917784).jpg

"https://upload.wikimedia.org/wikipedia/commons/1/1b/The_Coleoptera_of_the_British_islands_%28
Plate_125%29_%288592917784%29.jpg",
  # By Américo Toledano, Source:
https://commons.wikimedia.org/wiki/File:Biblioteca_Maim%C3%B3nides,_Campus_Universitario_d
e_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.