NAME ; PARAKALA KARTHIK YADAV

EMAIL ID ; karthikyadavkarthik181@gmail.com

TASK 1 ; Tic Tac Toe AI

Overview This project implements a playable Tic Tac Toe game featuring an AI opponent powered by the Minimax algorithm. The AI is designed to play optimally, providing a challenging and engaging gameplay experience for players. The implementation is done in Python, making it easy to understand and extend.

** **Introduction**** Tic Tac Toe is a classic two-player game where players take turns marking spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. This project enhances the traditional game by introducing an AI opponent that uses the Minimax algorithm to play optimally.

****Features ****

Playable Tic Tac Toe game against an AI opponent

AI uses the Minimax algorithm to ensure optimal play

Easy-to-understand Python codebase

Command-line interface for simple interaction

Extensible code structure for further enhancements

****Technologies Used** ; Python 3.x

Prerequisites Before you begin, ensure you have met the following requirements:

Python 3.x installed on your machine

Basic understanding of Python programming

How the Minimax Algorithm Works The Minimax algorithm is a recursive algorithm used for decision-making and game theory. It provides an optimal move for the player assuming that the opponent also plays optimally.

Maximizing Player (AI): Tries to maximize the score.

Minimizing Player (Human): Tries to minimize the score.

The algorithm recursively explores all possible moves, evaluates them, and assigns scores based on the game outcome: Win: +1 Loss: -1 Draw: 0

The AI then selects the move with the highest score to ensure optimal play.

CODE;

Run Debug Stop Share Saved { } Beautify

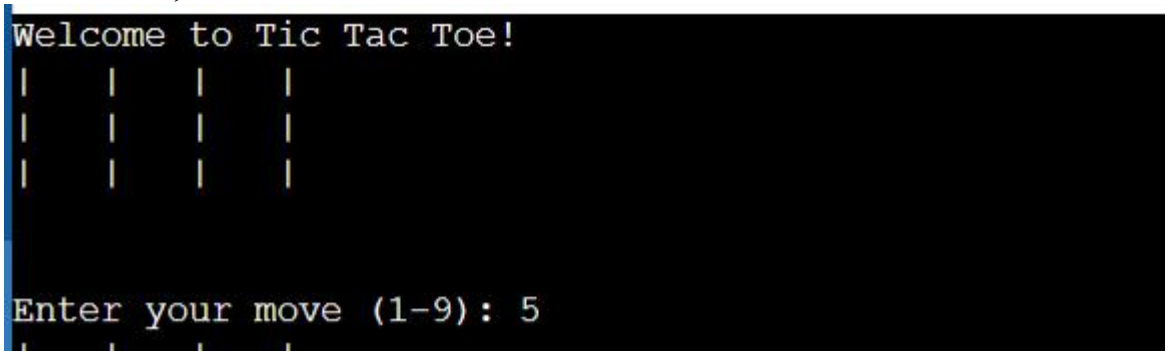
main.py

```python
import random

# Define the board
board = [' ' for _ in range(9)]

def print_board():
    for row in [board[i*3:(i+1)*3] for i in range(3)]:
        print('| ' + ' | '.join(row) + ' |')
    print("\n")

def is_winner(board, player):
    win_conditions = [(0, 1, 2), (3, 4, 5), (6, 7, 8),
                      (0, 3, 6), (1, 4, 7), (2, 5, 8),
                      (0, 4, 8), (2, 4, 6)]
    for condition in win_conditions:
        if board[condition[0]] == board[condition[1]] == board[condition[2]] == player:
            return True
    return False

def is_board_full(board):
    return ' ' not in board

def make_move(board, position, player):
    if board[position] == ' ':
        board[position] = player
        return True
    return False
```

```python

def minimax(board, depth, is_maximizing):
    if is_winner(board, 'O'):
        return 1
    elif is_winner(board, 'X'):
        return -1
    elif is_board_full(board):
        return 0

    if is_maximizing:
        best_score = -float('inf')
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'O'
                score = minimax(board, depth + 1, False)
                board[i] = ' '
                best_score = max(score, best_score)
        return best_score
    else:
        best_score = float('inf')
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                score = minimax(board, depth + 1, True)
                board[i] = ' '
                best_score = min(score, best_score)
        return best_score
```

```
55
56  def best_move():
57      best_score = -float('inf')
58      move = 0
59      for i in range(9):
60          if board[i] == ' ':
61              board[i] = 'O'
62              score = minimax(board, 0, False)
63              board[i] = ' '
64              if score > best_score:
65                  best_score = score
66                  move = i
67      make_move(board, move, 'O')
68
69  def main():
70      print("Welcome to Tic Tac Toe!")
71      print_board()
72
73      while True:
74          player_move = int(input("Enter your move (1-9): ")) - 1
75          if make_move(board, player_move, 'X'):
76              print_board()
77              if is_winner(board, 'X'):
78                  print("You win!")
79                  break
80              elif is_board_full(board):
81                  print("It's a tie!")
82                  break
83
84              best_move()
85              print_board()
86              if is_winner(board, 'O'):
87                  print("You lose!")
88                  break
89              elif is_board_full(board):
90                  print("It's a tie!")
91                  break
92          else:
93              print("Invalid move, try again.")
94
95  if __name__ == '__main__':
96      main()
97
```

OUTPUT;



TASK 2

PING PONG AI

Overview

This project implements a playable Ping Pong game featuring an AI opponent using JavaScript. The AI is designed to play efficiently, providing a challenging and engaging gameplay experience for players.

Features

- Playable Ping Pong game against an AI opponent
- AI designed to be efficient and challenging
- Simple and clean user interface
- Easy-to-understand JavaScript codebase
- Extensible code structure for further enhancements

Technologies Used

- JavaScript

Prerequisites

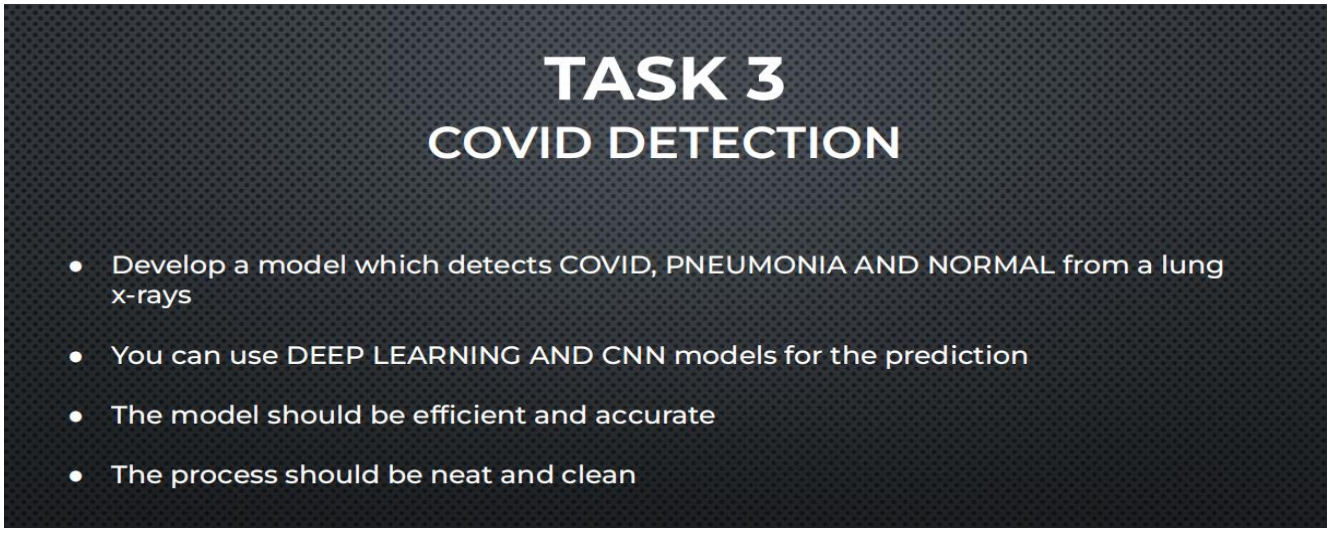
- A web browser (e.g., Chrome, Firefox, Safari)
- Basic understanding of HTML, CSS, and JavaScript

CODE;

```
const canvas = document.getElementById("pingPongTable");
const context = canvas.getContext("2d");

const net = {
    x: canvas.width / 2 - 1,
    y: 0,
    width: 2,
    height: canvas.height,
    color: "#FFF"
};

const player = {
    x: 0,
    y: canvas.height / 2 - 50,
    width: 10,
    height: 100,
    color: "#FFF",
    score: 0
};

const ai = {
    x: canvas.width - 10,
    y: canvas.height / 2 - 50,
    width: 10,
    height: 100,
    color: "#FFF",
    score: 0
};

const ball = {
    x: canvas.width / 2,
    y: canvas.height / 2,
    radius: 10,
    speed: 5,
    velocityX: 5,
    velocityY: 5,
    color: "#05EDFF"
};

function drawRect(x, y, w, h, color) {
    context.fillStyle = color;
    context.fillRect(x, y, w, h);
}

function drawCircle(x, y, r, color) {
    context.fillStyle = color;
    context.beginPath();
    context.arc(x, y, r, 0, Math.PI * 2, false);
    context.closePath();
    context.fill();
}

function drawText(text, x, y, color) {
    context.fillStyle = color;
    context.font = "35px sans-serif";
    context.fillText(text, x, y);
}

function drawNet() {
    drawRect(net.x, net.y, net.width, net.height, net.color);
}

function resetBall() {
    ball.x = canvas.width / 2;
    ball.y = canvas.height / 2;
    ball.speed = 5;
    ball.velocityX = -ball.velocityX;
}

function update() {
    ball.x += ball.velocityX;
    ball.y += ball.velocityY;

    if (ball.y + ball.radius > canvas.height || ball.y - ball.radius < 0) {
        ball.velocityY = -ball.velocityY;
    }

    let playerOrAI = (ball.x < canvas.width / 2) ? player : ai;
```

```
    if (collision(ball, playerOrAI)) {
        let collidePoint = ball.y - (playerOrAI.y + playerOrAI.height / 2);
        collidePoint = collidePoint / (playerOrAI.height / 2);
        let angleRad = collidePoint * Math.PI / 4;

        let direction = (ball.x < canvas.width / 2) ? 1 : -1;
        ball.velocityX = direction * ball.speed * Math.cos(angleRad);
        ball.velocityY = ball.speed * Math.sin(angleRad);

        ball.speed += 0.5;
    }

    if (ball.x - ball.radius < 0) {
        ai.score++;
        resetBall();
    } else if (ball.x + ball.radius > canvas.width) {
        player.score++;
        resetBall();
    }

    ai.y += ((ball.y - (ai.y + ai.height / 2)) * 0.1);

    if (player.y < 0) player.y = 0;
    if (player.y + player.height > canvas.height) player.y = canvas.height - player.height;
}

function collision(b, p) {
    p.top = p.y;
    p.bottom = p.y + p.height;
    p.left = p.x;
    p.right = p.x + p.width;

    b.top = b.y - b.radius;
    b.bottom = b.y + b.radius;
    b.left = b.x - b.radius;
    b.right = b.x + b.radius;

    return b.right > p.left && b.bottom > p.top && b.left < p.right && b.top < p.bottom;
}

canvas.addEventListener("mousemove", movePaddle);

function movePaddle(evt) {
    let rect = canvas.getBoundingClientRect();

    player.y = evt.clientY - rect.top - player.height / 2;
}

function render() {
    drawRect(0, 0, canvas.width, canvas.height, "#000");
    drawNet();
    drawText(player.score, canvas.width / 4, canvas.height / 5, "#FFF");
    drawText(ai.score, 3 * canvas.width / 4, canvas.height / 5, "#FFF");
    drawRect(player.x, player.y, player.width, player.height, player.color);
    drawRect(ai.x, ai.y, ai.width, ai.height, ai.color);
    drawCircle(ball.x, ball.y, ball.radius, ball.color);
}

function game() {
    update();
    render();
}

const framePerSecond = 50;
setInterval(game, 1000 / framePerSecond);
```

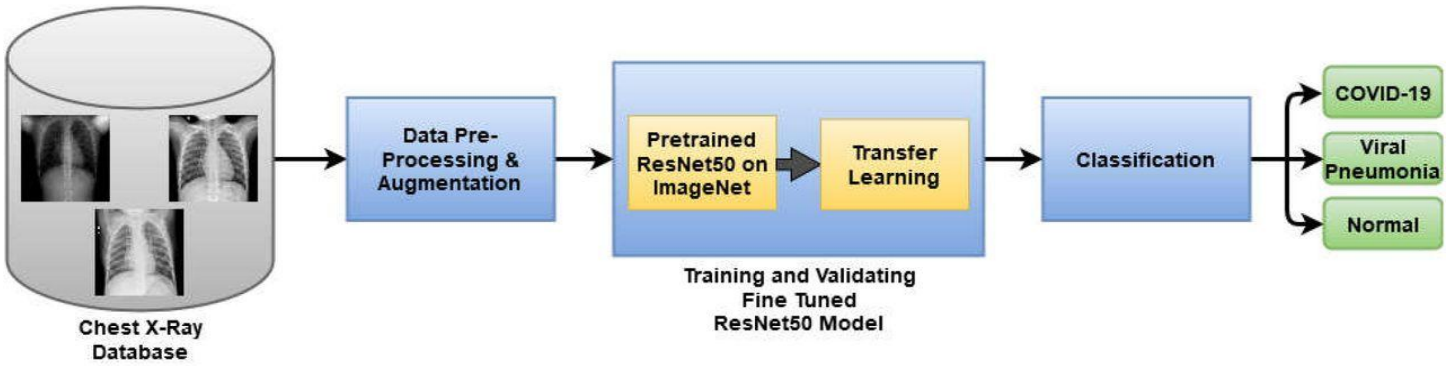
TASK 3 COVID DETECTION



COVID Detection Model

This repository contains a deep learning model for detecting COVID-19, Pneumonia, and Normal cases from lung X-ray images using Convolutional Neural Networks (CNN).

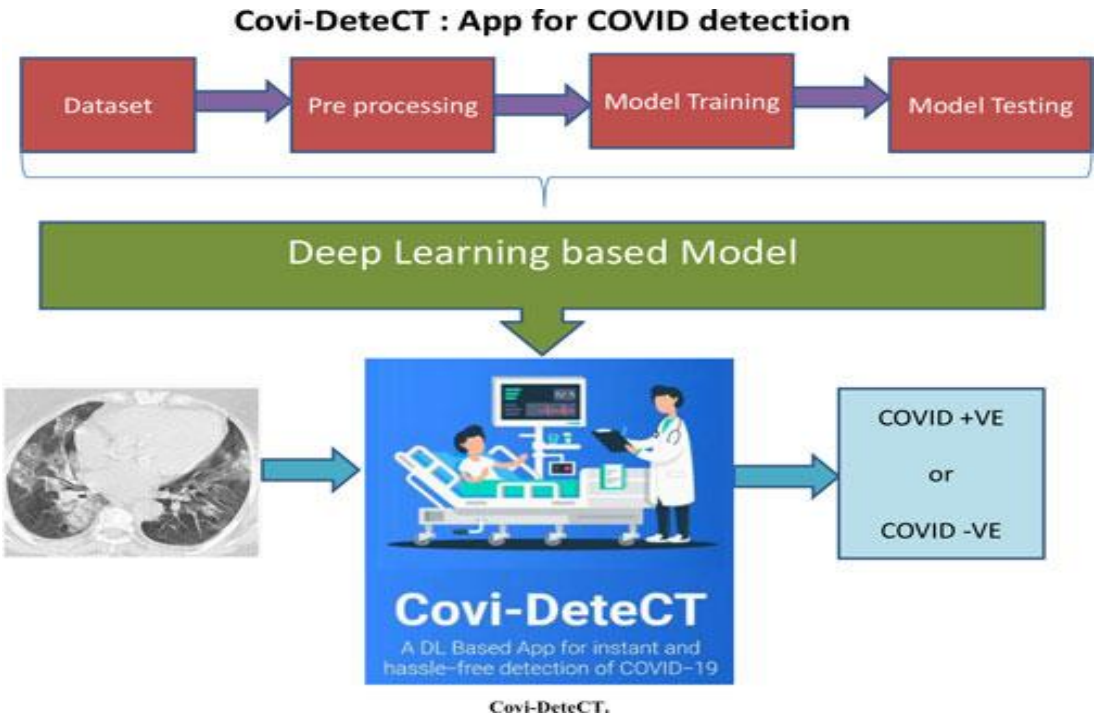
Overview



The goal of this project is to develop an accurate and efficient model that can classify lung X-ray images into three categories:

- COVID-19
- Pneumonia
- Normal

The model architecture is based on CNN, which is a powerful technique for image classification tasks.



Covi-DeteCT.

Dataset

The dataset used for training and evaluation consists of lung X-ray images collected from various sources. The dataset is organized into three classes:

- COVID-19
- Pneumonia
- Normal

Model Architecture

The CNN model used for this task consists of multiple convolutional layers followed by max-pooling layers for feature extraction, and then fully connected layers for classification. The final layer uses softmax activation to output probabilities for each class.

Evaluation

The model is evaluated using accuracy, precision, recall, and F1-score metrics to ensure its performance on all classes. The evaluation is done on a separate test dataset that was not used during training.

Requirements

- Python 3.x
- TensorFlow
- Keras
- NumPy
- Matplotlib
- Pandas

(a) Normal (b) Pneumonia (c) COVD-19

TASK 4 DIGIT RECOGNISER

Digit Recognizer

This repository contains a deep learning model for recognizing handwritten digits using Convolutional Neural Networks (CNN) trained on the MNIST dataset.

Overview

The goal of this project is to develop an efficient and accurate model that can classify handwritten digits from 0 to 9. The model is trained using a CNN architecture, which is well-suited for image classification tasks.

Dataset

The model is trained on the MNIST dataset, which is a widely used dataset for training and testing machine learning models. The dataset consists of 60,000 training images and 10,000 test images, each of size 28x28 pixels.

Model Architecture

The CNN model used for this task consists of multiple convolutional layers followed by max-pooling layers for feature extraction, and then fully connected layers for classification. The final layer uses softmax activation to output probabilities for each digit class (0-9).