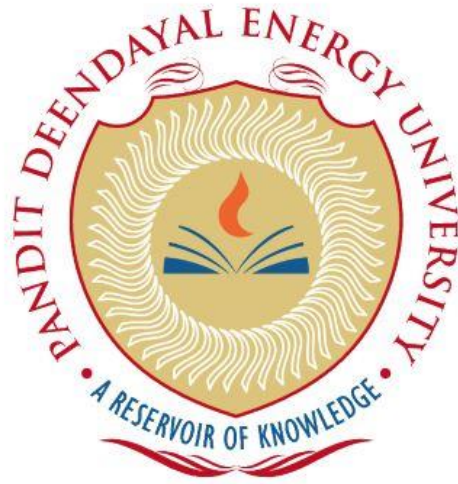


PANDIT DEENDAYAL ENERGY UNIVERSITY
SCHOOL OF TECHNOLOGY



Artificial Intelligence Lab

23CP307P

LAB MANUAL

B.Tech. (Computer Science and Engineering)

Semester 6

Submitted To:

Dr. Pooja Shah

Submitted By:

HARSH SHAH

21BCP359

G11 Batch

INDEX

23CP307P-Artificial Intelligence Lab

Exp. No.	Experiment Title	Date	Signature																		
1	WAP to implement DFS and BFS for traversing a graph from source node (S) to goal node (G), where source node and goal node is given by the user as an input.	10-01-24																			
2	Design water jug problem solver. You are given two jugs with m litres and a n litre capacity. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. You have to use the jugs to measure d litres of water where d is less than n. You are given two jugs with m litres and a n litre capacity. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. You have to use the jugs to measure d litres of water where d is less than n.	17-01-24																			
3	Solve 8 puzzle problem using A* algorithm where initial state and Goal state will be given by the users.	24-01-24																			
4	Implement using C/C++, the Fixed Increment Perceptron Learning algorithm as presented in the attachment. The training set for a 2-classificaiton problem is also attached. Iterate the perceptron through the training set and obtain the weights.	31-01-24																			
5	Given a C++ code bnp, identify the algorithm implemented through the code. Also document the code.	07-02-24																			
6a	Understand the project available on following link Project Link: https://github.com/aharley/nn_vis Project by: https://adamharley.com/ Reference in case needed: https://www.youtube.com/watch?v=pj9-rrlwDhM	14-02-24																			
6b	Part 2 Populate the table below to summarize your understanding of the project mentioned in part 1 <table><tr><th>Layer</th><th>Task</th><th>Rationale</th></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>	Layer	Task	Rationale																21-02-24	
Layer	Task	Rationale																			
How does the following hyper-parameters affect network performance?																					
Hyper-Parameter	One Line Definition	Effect on the CNN																			
Stride																					
Dilation Rate																					
Type of pooling layer																					
Kernel size																					
padding																					

	<p>References:</p> <p>An Intuitive Explanation of Convolutional Neural Networks – the data science blog (ujjwalkarn.me)</p> <p>Gentle Dive into Math Behind Convolutional Neural Networks by Piotr Skalski Towards Data Science</p> <p>Intuitively Understanding Convolutions for Deep Learning by Irhum Shafkat Towards Data Science</p> <p>An Introduction to different Types of Convolutions in Deep Learning by Paul-Louis Pröve Towards Data Science</p>		
7	<p>Prepare your version of CNN following the steps in the link shared here.</p> <p>https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f</p>	06-03-24	
8	<p>Design the Neural Network model for the project title submitted by you. Demonstrate "Over-fitting" and solve the same using "Dropout technique".</p> <p>Rubrics: Model Justification with respect to project domain - 5 marks Demonstration of over fitting and dropout technique - 5 marks</p>	13-03-24	
9	<p>For your project definition demonstrate applicable task out of prediction and classification.</p> <p>Explain the entire work flow of your project through a single diagram</p>	20-03-24	
10	<p>For your project demonstrate the following:</p> <ul style="list-style-type: none"> • need of optimizer - 5 marks • significance of your choice of optimizer - 5 marks • comparison of outcomes with and without optimization - 5 marks • Project Report including minimum (abstract, domain intro, data set description, implementation methodology with brief justification, results and discussion, future scope) - 10 marks 	27-03-24	
11a	Understanding the basics and IDE for Prolog Programming	03-04-24	
11b	<p>Implement any two of the following using Prolog:</p> <ul style="list-style-type: none"> - Medical diagnosis of common cold and flu using symptom inputs <p>Demonstrating list in prolog Monkey banana problem Find the factorial of a given number</p>	10-04-24	
12	WAP to design Tic Tac Toe games from O (Opponent) and X (Player) by using minimax algorithm.	24-04-24	

PRACTICAL 1

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	10-01-24	Batch:	G11
Aim:	WAP to implement DFS and BFS for traversing a graph from source node (S) to goal node (G), where source node and goal node is given by the user as an input.				

Program

```
from collections import deque
```

```
import timeit
```

```
def calculate_distance_bfs(graph, path, state, end):
```

```
    visited = set()
```

```
    distance = 0
```

```
    count = 0
```

```
    while state != end:
```

```
        for key in list((graph[state]).keys()):
```

```
            if key not in visited:
```

```
                distance = distance + graph[state][key]
```

```
                visited.add(key)
```

```
            if key == end:
```

```
                break
```

```
        if key == end:
```

```
            break
```

```
        count = count + 1
```

```
        state = path[count]
```

```
    return distance
```

```
def tsp_bfs(graph, start, end):
```

```
    visited = set()
```

```
    path = []
```

```
    distance = 0
```

```
    queue = deque([start])
```

```
    visited.add(start)
```

```
while queue:
    vertex = queue.popleft()
    path.append(vertex)

    if vertex == end:
        distance = calculate_distance_bfs(graph, path, start, end)
        return path, distance

    for adj in graph[vertex]:
        if adj not in visited:
            visited.add(adj)
            queue.append(adj)

return path, distance
```

```
def calculate_distance_dfs(graph, path):
    distance = 0
    for i in range(len(path) - 1):
        distance += graph[path[i]][path[i + 1]]
    return distance
```

```
def tsp_dfs(graph, start, stop):
    visited = set()
    stack = [start]
    path = []
    distance = 0

    while stack:
        vertex = stack.pop()
        path.append(vertex)
        visited.add(vertex)
        if vertex == stop:
            distance = calculate_distance_dfs(graph, path)
            return path, distance
```

```
temp_stack = []
for adj in graph[vertex]:
    if adj not in visited:
        temp_stack.append(adj)
    stack.extend(temp_stack[::-1])
return path, distance

if __name__ == "__main__":
    graph_1 = {
        "A": {"B": 22, "C": 48, "D": 28},
        "B": {"A": 22, "C": 20, "D": 18},
        "C": {"A": 48, "B": 20, "D": 32},
        "D": {"A": 28, "B": 18, "C": 32},
    }

    graph_2 = {
        "A": {"B": 2, "G": 6},
        "B": {"A": 2, "C": 7, "E": 2},
        "C": {"B": 7, "D": 3, "F": 3},
        "D": {"C": 3, "H": 2},
        "E": {"B": 2, "F": 2, "G": 1},
        "F": {"C": 3, "E": 2, "H": 2},
        "G": {"A": 6, "E": 1, "H": 4},
        "H": {"D": 2, "F": 2, "G": 4},
    }

    start = input("Enter the starting node: ")
    end = input("Enter the ending node: ")

    # DFS

    start_time_dfs = timeit.default_timer()
    path, dist = tsp_dfs(graph_2, start, end)
    execution_time_dfs = timeit.default_timer() - start_time_dfs
```

*BFS*

```
start_time_bfs = timeit.default_timer()
path, dist = tsp_bfs(graph_2, start, end)
execution_time_bfs = timeit.default_timer() - start_time_bfs

print("\nDFS Path:", ''.join(path))
print("DFS Cost:", dist)
print("DFS Execution Time:", execution_time_dfs)

print("\nBFS Path:", ''.join(path))
print("BFS Cost:", dist)
print("BFS Execution Time:", execution_time_bfs)
```

Output

```
Enter the starting node: A
Enter the ending node: F

DFS Path: ABGCEHDF
DFS Cost: 29
DFS Execution Time: 4.05999890062958e-05

BFS Path: ABGCEHDF
BFS Cost: 29
BFS Execution Time: 3.200001083314419e-05
```

Results

According to the results the **DFS** traversing **takes more time** than **BFS** traversing. Hence in the given example BFS outperforms DFS.

PRACTICAL 2

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	17-01-24	Batch:	G11
Aim:	You are given two jugs with m litres and a n litre capacity. Both the jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. You have to use the jugs to measure d litres of water where d is less than n.				

Program

```
from collections import deque
```

```
def water_jug_BFS(x, y, z):  
    visited = set()  
    queue = deque([((0, 0), [])])  
  
    while queue:  
        (jug_a, jug_b), actions = queue.popleft()  
  
        if jug_a == z or jug_b == z or jug_a + jug_b == z:  
            return actions + ["Success"], True  
  
        if (jug_a, jug_b) in visited:  
            continue  
  
        visited.add((jug_a, jug_b))  
  
        # Fill jug A  
        if jug_a < x:  
            queue.append(((x, jug_b), actions + ["Fill A"]))  
  
        # Fill jug B  
        if jug_b < y:  
            queue.append(((jug_a, y), actions + ["Fill B"]))
```



```
# Empty jug A
if jug_a > 0:
    queue.append(((0, jug_b), actions + ["Empty A"]))

# Empty jug B
if jug_b > 0:
    queue.append(((jug_a, 0), actions + ["Empty B"]))

# Pour from A to B
if jug_a + jug_b >= y:
    queue.append(((jug_a - (y - jug_b), y), actions + ["Pour A to B"]))
else:
    queue.append(((0, jug_a + jug_b), actions + ["Pour A to B"]))

# Pour from B to A
if jug_a + jug_b >= x:
    queue.append(((x, jug_b - (x - jug_a)), actions + ["Pour B to A"]))
else:
    queue.append(((jug_a + jug_b, 0), actions + ["Pour B to A"]))

return [], False

if __name__ == "__main__":
    n = int(input("Enter jug A's capacity (n): "))
    m = int(input("Enter jug B's capacity (m): "))
    d = int(input("Enter capacity to measure (d): "))

    actions, result = water_jug_BFS(n, m, d)

    if result:
```

```
    print("The sequence of actions is:")
    for action in actions:
        print(action)
else:
    print("No solution found.")
```

Output

```
Enter jug A's capacity (n): 4
Enter jug B's capacity (m): 3
Enter capacity to measure (d): 2
The sequence of actions is:
Fill B
Pour B to A
Fill B
Pour B to A
Success
```

```
Enter jug A's capacity (n): 6
Enter jug B's capacity (m): 2
Enter capacity to measure (d): 5
No solution found.
```

PRACTICAL 3

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	24-01-24	Batch:	G11
Aim:	Solve 8 puzzle problem using A* algorithm where initial state and Goal state will be given by the users.				

Program

```
import numpy as np
```

```
# Function to get matrix input from the user
```

```
def get_matrix_input(prompt):
```

```
    print(prompt)
```

```
    matrix = []
```

```
    for i in range(3):
```

```
        # Get each row of the matrix from the user
```

```
        row = list(
```

```
            map(
```

```
                int,
```

```
                input(
```

```
                    "Enter row {} (separate numbers with space): ".format(i + 1)
```

```
                ).split(),
```

```
            )
```

```
        )
```

```
        matrix.append(row)
```

```
    return np.array(matrix)
```

```
# Function to calculate heuristic of a matrix
```

```
def heuristic(matrix, end_matrix):
```

```
    # Compare each element of the matrix with the end matrix
```

```
    res = matrix == end_matrix
```

```
    # Return the number of elements that are not in their correct position
```

```
    return 9 - np.count_nonzero(res)
```

```

# Function to generate possible children of a matrix
def possibleChildren(matrix, e_matrix):
    visited.append(matrix)

    [i, j] = np.where(matrix == 0) # Find the position of the empty space (0)
    direction = [
        [-1, 0],
        [0, -1],
        [1, 0],
        [0, 1],
    ] # Possible directions to move the empty space

    children = []

    for dir in direction:
        ni = i + dir[0]
        nj = j + dir[1]
        newMatrix = matrix.copy()

        # Check if the move is within the bounds of the matrix
        if ni >= 0 and ni <= 2 and nj >= 0 and nj <= 2:
            # Swap the empty space with the adjacent element
            newMatrix[i, j], newMatrix[ni, nj] = matrix[ni, nj], matrix[i, j]

            # Check if the new matrix has been visited before
            if not (any(np.array_equal(newMatrix, i) for i in visited)):
                visited.append(newMatrix)
                newMatrix_heu = heuristic(newMatrix, end_matrix)
                children.append([newMatrix_heu, newMatrix])

    # Sort the children based on their heuristic
    children = sorted(children, key=lambda x: x[0])
    for i in range(len(children)):
        children[i] = children[i][1]

```

```
    return children
```

```
# Function to solve the 8-puzzle problem using A* algorithm
```

```
def a_star_8_puzzle(start_matrix, end_matrix):  
    start_heuristic = heuristic(start_matrix, end_matrix)  
    if start_heuristic == 0:  
        for node in closed:  
            print(node)  
        return True  
    else:  
        children = possibleChildren(start_matrix, end_matrix)  
        if len(children) > 0:  
            for i in range(len(children)):  
                open.insert(i, children[i])  
  
        if len(open) > 0:  
            newHeu = heuristic(open[0], end_matrix)  
            newMatrix = open[0]  
            closed.append(open[0])  
            open.pop(0)  
  
            if newHeu == 0:  
                for node in closed:  
                    print(node)  
                return True  
            else:  
                a_star_8_puzzle(newMatrix, end_matrix)  
        else:  
            return False
```

```
# Get the start and end matrices from the user
```

```
start_matrix = get_matrix_input("Enter the start matrix:")
end_matrix = get_matrix_input("Enter the end matrix:")

visited = []
open = []
closed = []

closed.append(start_matrix)

if __name__ == "__main__":
    a_star_8_puzzle(start_matrix, end_matrix)
```

Output

```
Enter the start matrix:
Enter row 1 (separate numbers with space): 2 8 3
Enter row 2 (separate numbers with space): 1 6 4
Enter row 3 (separate numbers with space): 7 0 5
Enter the end matrix:
Enter row 1 (separate numbers with space): 1 2 3
Enter row 2 (separate numbers with space): 8 0 4
Enter row 3 (separate numbers with space): 7 6 5
```

```
[[2 8 3]
 [1 6 4]
 [7 0 5]]
[[2 8 3]
 [1 0 4]
 [7 6 5]]
[[2 0 3]
 [1 8 4]
 [7 6 5]]
[[0 2 3]
 [1 8 4]
 [7 6 5]]
[[1 2 3]
 [0 8 4]
 [7 6 5]]
[[1 2 3]
 [8 0 4]
 [7 6 5]]
```

PRACTICAL 4

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	31-01-24	Batch:	G11
Aim:	Implement the Fixed Increment Perceptron Learning algorithm as presented in the attachment. The training set for a 2- classification problem is also attached. Iterate the perceptron through the training set and obtain the weights.				

Program

```
#include <bits/stdc++.h>
using namespace std;
```

```
pair<int, int> input_dimensions = {8, 2};
```

```
vector<pair<double, double>> X = {{0.32, 0.41}, {0.27, 0.54}, {0.57, 0.42}, {0.59, 0.71}, {0.78, 0.82},
{0.79, 0.95}, {1.0, 0.85}, {1.0, 1.1}};
```

```
vector<double> coefficients, results, hidden_values, bias;
vector<double> targets = {0, 1, 0, 1, 0, 1, 0, 1};
```

```
void initialize_coefficients()
{
    uniform_real_distribution<double> unif(-0.00002, 0.00002);
    default_random_engine re;

    for (int i = 0; i < input_dimensions.second; i++)
    {
        coefficients.push_back(unif(re));
    }

    bias.push_back(unif(re));
}
```

```
void calculate_hidden_values()
{
    double res1, sum;
    sum = 0;
    hidden_values.clear();

    for (int i = 0; i < input_dimensions.first; i++)
    {
        res1 = (X[i].first) * (coefficients[0]) + (X[i].second) * (coefficients[1]);
        hidden_values.push_back((res1 + bias[0]));
    }
}
```

```
void make_predictions()
{

```

```
calculate_hidden_values();
results.clear();

for (int i = 0; i < input_dimensions.first; i++)
{
    if (hidden_values[i] > 0)
    {
        results.push_back(1);
    }
    else
    {
        results.push_back(0);
    }
}

double evaluate_accuracy()
{
    double acc = 0;
    for (int i = 0; i < input_dimensions.first; i++)
    {
        if (results[i] == targets[i])
        {
            acc++;
        }
    }
    return acc / input_dimensions.first;
}

void adjust_coefficients(double learning_rate)
{
    double error1, error2;
    error1 = 0;
    error2 = 0;

    for (int i = 0; i < input_dimensions.first; i++)
    {
        error1 += (targets[i] - results[i]) * X[i].first;
        error2 += (targets[i] - results[i]) * X[i].second;

        bias[0] += learning_rate * (targets[i] - results[i]);
    }

    coefficients[0] += learning_rate * error1;
    coefficients[1] += learning_rate * error2;
}

int main()
{
    double learning_rate, accuracy;
    learning_rate = 0.000000001;
```



```
initialize_coefficients();
make_predictions();
accuracy = evaluate_accuracy();

long long ep = 1;
while (accuracy < 1)
{
    adjust_coefficients(learning_rate);
    make_predictions();
    accuracy = evaluate_accuracy();
    // cout << "\tepoche: " << ep << "\taccuracy: " << accuracy << "\n";
    ep++;
}

cout << "Weights: " << coefficients[0] << ", " << coefficients[1] << endl;

return 0;
}
```

Output

```
PS C:\Users\harsh\OneDrive - pdpu.ac.in\HARSH_PDEU\SEM 6\Artificial Intelligence\LAB\Practical_4> cd "c:\Users\harsh\OneDrive - pdpu.ac.in\HARSH_PDEU\SEM 6\Artificial Intelligence\LAB\Practical_4\" ; if ($?) { g++ singleLayerPerceptron.cpp -o singleLayerPerceptron } ; if ($?) { .\singleLayerPerceptron }
Weights: -7.44657e-006, 7.33967e-006
```

PRACTICAL 5

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	07-02-24	Batch:	G11
Aim:	Given a C++ code, identify the algorithm implemented through the code. Also document the code.				

Program

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <cmath>

using namespace std;

// GLOBAL FILE NAME
char file_name[9], file_name_inf[14], file_name_wgt[14], file_name_rst[14];
char file_name_out[14], file_name_dat[14];

// Class representing a matrix
class matrix {
    int row, col;

public:
    float mat[15][15];
    matrix() {
        row = 0;
        col = 0;
    }
    void set(int, int);
    int getrows() {
```

```
        return row;
    }

    int getcols() {
        return col;
    }

    void getdata();
    FILE *fgetdata(FILE *);
    void displaydata();
    void displaydat();
    FILE *fputdata(FILE *);
    FILE *fputdat(FILE *);
    matrix operator+(matrix);
    matrix operator-();
    matrix operator*(matrix);
    matrix operator*(float);
};

// Set the dimensions of the matrix
void matrix::set(int i, int j) {
    row = i;
    col = j;
}

// Read matrix data from file
FILE *matrix::fgetdata(FILE *fmat) {
    char line;
    int i, j;
    fscanf(fmat, "%d%d", &(row), &(col));
    for (i = 1; i <= row; i++)
        for (j = 1; j <= col; j++)
            fscanf(fmat, "%f", &(mat[i][j]));
```

```
        return (fmat);
    }

// Read matrix data from user input
void matrix::getdata() {
    int i, j;
    cout << "Enter the size of the matrix:";
    cin >> row >> col;
    for (i = 1; i <= row; i++)
        for (j = 1; j <= col; j++) {
            cout << "element [" << i << " ] [ " << j << " ] ";
            cin >> mat[i][j];
        }
}
```

```
// Display matrix data
void matrix::displaydata() {
    int i, j;
    for (i = 1; i <= row; i++, printf("\n\r"))
        for (j = 1; j <= col; j++, printf("\t"))
            printf("\t\t%10.2f", mat[i][j]);
}
```

```
// Display matrix dimensions
void matrix::displaydat() {
    int i;
    cout << row;
}
```

```
// Write matrix data to file
FILE *matrix::fputdata(FILE *fmat) {
```

```
int i, j;

fprintf(fmat, "%d\n%d\n", row, col);

for (i = 1; i <= row; i++)

    for (j = 1; j <= col; j++)

        fprintf(fmat, "%f\n", mat[i][j]);

return (fmat);

}
```

// Write matrix dimensions to file

```
FILE *matrix::fputdat(FILE *fmat) {

    int i;

    fprintf(fmat, "%d", row);

    return (fmat);

}
```

// Overloaded operator for matrix addition

```
matrix matrix::operator+(matrix m) {

    matrix temp;

    int i, j;

    if ((row == m.row) && (col == m.col))

        for (i = 1; i <= row; i++)

            for (j = 1; j <= col; j++)

                temp.mat[i][j] = mat[i][j] + m.mat[i][j];

    else {

        cout << "The addition of the matrices is not possible";

        exit(1);

    }

    temp.row = row;

    temp.col = col;

    return (temp);

}
```

// Overloaded operator for matrix transposition

```
matrix matrix::operator-() {  
    matrix temp;  
    int i, j;  
    temp.row = col;  
    temp.col = row;  
    for (i = 1; i <= col; i++)  
        for (j = 1; j <= row; j++)  
            temp.mat[i][j] = mat[j][i];  
    return (temp);  
}
```

// Overloaded operator for matrix multiplication

```
matrix matrix::operator*(matrix m) {  
    matrix temp;  
    int i, j, k;  
    if (col == m.row) {  
        for (i = 1; i <= row; i++)  
            for (j = 1; j <= m.col; j++) {  
                temp.mat[i][j] = 0;  
                for (k = 1; k <= col; k++)  
                    temp.mat[i][j] = temp.mat[i][j] + (mat[i][k] * m.mat[k][j]);  
            }  
    } else {  
        cout << "The multiplication of the matrices is not possible";  
        exit(1);  
    }  
    temp.row = row;  
    temp.col = m.col;  
    return (temp);  
}
```

```
}
```

```
// Overloaded operator for scalar multiplication with matrix
```

```
matrix matrix::operator*(float svalue) {  
    matrix temp;  
    int i, j;  
    for (i = 1; i <= row; i++)  
        for (j = 1; j <= col; j++)  
            temp.mat[i][j] = mat[i][j] * svalue;  
    temp.row = row;  
    temp.col = col;  
    return (temp);  
}
```

```
// Class representing the training process of the neural network
```

```
class training {  
    FILE *fin, *fout, *fwt;  
    matrix Input[5], Output[5], Weights[5], dWeights[5], d, e, T;  
    float alpha, eta, err, theta, lamda, error;  
    int TotalLayers, HiddenLayers, l[5], ntest, iterates;  
    long filepos;  
  
public:  
    training();  
    void readinputs();  
    void printing();  
    void initweights();  
    void initdweights();  
    void train();  
    void io_values();  
    void backpropagate();
```

```
void errors();

void chgweights();

void newweights();

~training();

};

// Constructor for training class
training::training() {
    // Open files for input and output
    if ((fin = fopen(file_name_dat, "r")) == NULL) exit(1);
    if ((fout = fopen(file_name_out, "w")) == NULL) exit(1);
    if ((fwt = fopen(file_name_wgt, "w")) == NULL) exit(1);
    readinputs();
    printing();
    initweights();
    initdweights();
    train();
}

// Function to read input parameters for training
void training::readinputs() {
    int i;
    error = 0;
    char line;
    fscanf(fin, "%d", &HiddenLayers); // Get number of hidden layers
    TotalLayers = HiddenLayers + 1; // Calculate total number of layers
    for (i = 0; i <= TotalLayers; i++)
        fscanf(fin, "%d", &l[i]);
    fscanf(fin, "%f%f%f%f%f", &alpha, &err, &eta, &theta, &lamda);
    fscanf(fin, "%d%d", &ntest, &iterates);
    filepos = ftell(fin);
```



```
}
```

```
// Function to print input parameters for training
```

```
void training::printing() {  
    // Print parameters to output file  
    for (int i = 0; i <= TotalLayers; i++)  
        fprintf(fout, "\nNumber of Neurons in layer[%d]=%d", i + 1, l[i]);  
    fprintf(fout, "\nAlpha value(Momentum factor): %f", alpha);  
    fprintf(fout, "\nError constant : %f", err);  
    fprintf(fout, "\nLearning rate : %f", eta);  
    fprintf(fout, "\nThreshold value : %f", theta);  
    fprintf(fout, "\nScaling Parameter: %f", lamda);  
    fprintf(fout, "\nNo of Training data : %d", ntest);  
    fprintf(fout, "\nMaximum Iteration : %d", iterates);  
    system("cls");
```

```
// Print parameters to console
```

```
printf("\n\n\n");  
for (int i = 0; i <= TotalLayers; i++)  
    printf("\n\t\tNumber of Neurons in layer[%d]=%d", i + 1, l[i]);  
printf("\n\t\tAlpha value(Momentum factor): %f", alpha);  
printf("\n\t\tError constant : %f", err);  
printf("\n\t\tLearning rate : %f", eta);  
printf("\n\t\tThreshold value : %f", theta);  
printf("\n\t\tScaling Parameter : %f", lamda);  
printf("\n\t\tNo of Training data : %d", ntest);  
printf("\n\t\tMaximum Iteration : %d", iterates);  
cin.get();  
}
```

```
// Function to initialize weights randomly
```

```
void training::initweights() {  
    srand(2000);  
    srand(time(0));  
    for (int k = 0; k < TotalLayers; k++) {  
        Weights[k].set(l[k], l[k + 1]);  
        for (int i = 1; i <= l[k]; i++)  
            for (int j = 1; j <= l[k + 1]; j++)  
                Weights[k].mat[i][j] = ((float)rand() / 32767) - 0.5;  
        fprintf(fout, "\nWeights[%d]:", k);  
        Weights[k].fputdata(fout);  
    }  
}
```

// Function to initialize difference in weights

```
void training::initdweights() {  
    for (int k = 0; k < TotalLayers; k++) {  
        dWeights[k].set(l[k], l[k + 1]);  
        for (int i = 1; i <= l[k]; i++)  
            for (int j = 1; j <= l[k + 1]; j++)  
                dWeights[k].mat[i][j] = 0.0;  
    }  
}
```

// Function to perform neural network training

```
void training::train() {  
    int k;  
    for (int jtr = 1; jtr <= iterates; jtr++) {  
        error = 0.0;  
        fseek(fin, filepos, SEEK_SET);  
        cout << "\nIteration Number: " << jtr << endl;  
        for (int itr = 1; itr <= ntest; itr++) {
```

```

    Input[0].fgetdata(fin);
    T.fgetdata(fin);
    cout << "\rTraining Data Number: " << itr;
    io_values();
    backpropagate();
    errors();
    chgweights();
    newweights();
}

fprintf(fout, " %10.3E\n", error / ntest);
}

cin.get();
for (k = 0; k < TotalLayers; k++)
    fwt = Weights[k].fputdata(fwt);
}

// Function to calculate input/output values of neurons
void training::io_values() {
    Output[0] = Input[0];
    for (int m = 0; m <= TotalLayers - 1; m++) {
        Input[m + 1] = -Weights[m] * Output[m];
        Output[m + 1].set(l[m + 1], 1);
        for (int i = 1; i <= l[m + 1]; i++)
            Output[m + 1].mat[i][1] = 1.0 / (1.0 + exp(-lamda * (Input[m + 1].mat[i][1] + theta)));
    }
}

// Function to perform backpropagation
void training::backpropagate() {
    d.set(l[TotalLayers], 1);
    for (int i = 1; i <= l[TotalLayers]; i++)

```

```

    d.mat[i][1] = Output[TotalLayers].mat[i][1] * (1 - Output[TotalLayers].mat[i][1]) * (T.mat[i][1] -
Output[TotalLayers].mat[i][1]);

```

```

    dWeights[TotalLayers - 1] = (dWeights[TotalLayers - 1] * alpha) + ((Output[TotalLayers - 1] * -d) *
eta);

```

```

}

```

```

// Function to calculate errors

```

```

void training::errors() {
    float sum = 0.0, x, y1, y2;
    for (int j = 1; j <= l[TotalLayers]; j++) {
        y1 = T.mat[j][1];
        y2 = Output[TotalLayers].mat[j][1];
        x = fabs(y1 - y2);
        x = x * x;
        sum = sum + x;
    }
    sum = sqrt(sum / l[TotalLayers]);
    error = error + sum;
    cout << "\t\t Error =" << error;
}

```

```

// Function to calculate change in weights

```

```

void training::chgweights() {
    int k;
    for (int i = 0; i <= TotalLayers - 2; i++) {
        k = TotalLayers - i - 1;
        e = Weights[k] * d;
        d.set(l[k], 1);
        for (int j = 1; j <= l[k]; j++) {
            d.mat[j][1] = Output[k].mat[j][1] * (1 - Output[k].mat[j][1]) * e.mat[j][1];
        }
        dWeights[k - 1] = (dWeights[k - 1] * alpha) + ((Output[k - 1] * -d) * eta);
    }
}

```

```
    }  
}  
  
// Function to update weights  
void training::newweights() {  
    for (int k = 0; k < TotalLayers; k++)  
        Weights[k] = Weights[k] + dWeights[k];  
}  
  
// Destructor for training class  
training::~~training() {  
    fclose(fin);  
    fclose(fout);  
    fclose(fwt);  
}  
  
// Class representing the inference process of the neural network  
class inference {  
    FILE *fin, *fout, *fwt;  
    matrix Input[5], Output[5], Weights[5], T, CalculatedErr, NoOfTest;  
    float alpha, eta, err, theta, x1, x2, lamda, Calerror;  
    int TotalLayers, ntest, l[10];  
  
public:  
    inference();  
    void readinputs();  
    void initweights();  
    void i_values();  
    void calculate();  
    void error();  
    ~inference();
```

```
};
```

```
// Constructor for inference class
```

```
inference::inference() {  
    // Open files for input and output  
    if ((fin = fopen(file_name_inf, "r")) == NULL) exit(1);  
    if ((fout = fopen(file_name_rst, "w")) == NULL) exit(1);  
    if ((fwt = fopen(file_name_wgt, "r")) == NULL) exit(1);  
    readinputs();  
    initweights();  
    i_values();  
    calculate();  
    error();  
}
```

```
// Function to read input parameters for inference
```

```
void inference::readinputs() {  
    int i;  
    fscanf(fin, "%d", &TotalLayers); // Get number of hidden layers  
    for (i = 0; i <= TotalLayers; i++)  
        fscanf(fin, "%d", &l[i]); // Get number of neurons in each layer  
    fscanf(fin, "%f%f%f%f%f", &alpha, &err, &eta, &theta, &lamda); // Get other parameters  
    fscanf(fin, "%d", &ntest); // Get number of test cases  
}
```

```
// Function to initialize weights for inference
```

```
void inference::initweights() {  
    for (int k = 0; k < TotalLayers; k++) {  
        Weights[k].fgetdata(fwt);  
    }  
}
```

```
// Function to calculate input values for inference
```

```
void inference::i_values() {
    for (int itr = 1; itr <= ntest; itr++) {
        Input[0].fgetdata(fin);
        cout << "\rTesting Data Number: " << itr;
        Output[0] = Input[0];
        for (int m = 0; m <= TotalLayers - 1; m++) {
            Input[m + 1] = -Weights[m] * Output[m];
            Output[m + 1].set(l[m + 1], 1);
            for (int i = 1; i <= l[m + 1]; i++)
                Output[m + 1].mat[i][1] = 1.0 / (1.0 + exp(-lamda * (Input[m + 1].mat[i][1] + theta)));
        }
        Output[TotalLayers].displaydat();
    }
}
```

```
// Function to perform calculation for inference
```

```
void inference::calculate() {
    float sum = 0.0;
    for (int i = 1; i <= ntest; i++) {
        T.fgetdata(fin);
        CalculatedErr = Output[TotalLayers] - T;
        CalculatedErr = -CalculatedErr;
        CalculatedErr = CalculatedErr * CalculatedErr;
        x1 = CalculatedErr.mat[1][1];
        sum = sum + x1;
    }
    sum = sqrt(sum / ntest);
    Calerror = sum;
}
```

```
// Function to calculate error for inference
void inference::error() {
    printf("\nCalculated Error: %f", Calerror);
    fprintf(fout, "%f", Calerror);
}

// Destructor for inference class
inference::~inference() {
    fclose(fin);
    fclose(fout);
}

// Main function
int main() {
    strcpy(file_name, "Nndat.dat");
    strcpy(file_name_inf, "Nntst.dat");
    strcpy(file_name_wgt, "Nnwgt.dat");
    strcpy(file_name_out, "Nnout.dat");
    strcpy(file_name_rst, "Nnres.dat");
    training mlp1;
    inference mlp2;
    return 0;
}
```

Explanation

The algorithm implemented through the code is **Backpropagation for training a Multi-Layer Perceptron (MLP)** neural network.

Here's a breakdown of the code and its functionalities:

Classes:

- **matrix:** This class represents a matrix and provides methods for creating, manipulating, and displaying matrices.

- **training:** This class handles the training process of the MLP network. It includes methods for reading training data, initializing weights, performing backpropagation, calculating errors, and updating weights.
- **inference:** This class performs inference on the trained network. It reads test data, calculates the network's output, and compares it to the desired output.

Training Process (training class):

1. Initialization:

- Reads training data and network configuration from a file.
- Initializes weights and learning parameters.

2. Iteration Loop:

- Loops for a specified number of iterations.
- For each iteration:
 - Loops for each training data point:
 - Calculates the output of each layer using the forward pass.
 - Performs backpropagation to calculate the error gradients.
 - Updates the weights using the gradient descent algorithm with momentum.

3. Weight Update:

- Writes the final weights to a file.

Inference Process (inference class):

1. Loading Configuration:

- Reads network configuration and weights from files.

2. Test Data Loop:

- Loops for each test data point:
- Calculates the network's output using the forward pass.
- Compares the output to the desired output and calculates the error.
- Writes the calculated output, actual output, and error to a file.

Overall, the code implements a backpropagation algorithm to train a multilayer perceptron neural network. The trained network can then be used for inference on new data.

PRACTICAL 6

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	14-02-24	Batch:	G11
Aim:	<p>Part 1:</p> <p>Understand the project available on following link Project Link: https://github.com/aharley/nn_vis Project by: https://adamharley.com/ Reference in case needed: https://www.youtube.com/watch?v=pj9-rr1wDhM</p> <p>Part 2:</p> <p>Populate the table below to summarize your understanding of the project mentioned in part 1</p>				

Layer	Task	Rationale
Input layer	It receives user input and converts raw pixels from a sketchpad into data that the system can process.	To take raw input from the user and preprocess it.
Convolutional layer	It identifies patterns in the input data, like edges and corners, by applying mathematical operations and activation functions.	Extracts features, like edges and corners.
Pooling layer	It reduces the size of the data while keeping important information intact, making computations more efficient. It does this by condensing features and focusing on the most significant values.	To reduce space of the matrix (reduce spatial dimensions of feature maps) while conserving the original image. Consider the pixel having the highest value (illumination) using a stride of 2*2 pixels (2*2 max pooling) and taking that value to just one pixel in the new matrix.
Classifying layer	It utilizes the extracted features to accurately categorize the input data. It consists of interconnected neurons that analyze the features for classification.	The classifying layer takes the high-level abstracted features from previous layers and uses them to classify input data into different categories. There are 120 neurons in the first layer and 100 neurons in the second.
Output layer	It generates the final prediction based on the classification results, with each neuron representing the probability of a specific outcome, such as recognizing different digits.	Produces the final output or prediction of the network, representing the class probabilities.

How does the following hyper-parameters affect the network performance

Hyper-Parameter	One Line Definition	Effect on the CNN
Stride	Determines how much the filter moves across the input image.	Changing the stride impacts the size of the output feature maps. A larger stride means fewer calculations and smaller output maps, speeding up processing.
Dilation Rate	Controls how the elements of the convolutional filter are spread out.	Increasing dilation rate expands the filter's view, allowing it to capture broader features but at the cost of reduced detail in the output.
Type of pooling layer	Dictates how feature maps are condensed in pooling layers.	Various types like max pooling or average pooling determine how features are combined, impacting the network's capacity to maintain crucial details while decreasing size.
Kernel size	Determines the dimensions of the convolutional filters.	Bigger sizes gather more nearby details, enabling the network to learn complex patterns and demanding more computations. Smaller sizes concentrate on finer details but might miss broader patterns.
padding	Adding extra pixels around the input image.	It influences the size of the output feature maps. Zeropadding keeps the size unchanged, valid padding reduces it, and same padding maintains the input size.

References:

[An Intuitive Explanation of Convolutional Neural Networks – the data science blog \(ujjwalkarn.me\)](https://ujjwalkarn.me/2016/08/07/intuitive-explanation-convnets/)

[Gentle Dive into Math Behind Convolutional Neural Networks | by Piotr Skalski | Towards Data Science](#)

[Intuitively Understanding Convolutions for Deep Learning | by Irhum Shafkat | Towards Data Science](#)

[An Introduction to different Types of Convolutions in Deep Learning | by Paul-Louis Pröve | Towards Data Science](#)

PRACTICAL 7

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	06-03-24	Batch:	G11
Aim:	Prepare your version of CNN following the steps in the link shared here” https://towardsdatascience.com/build-your-own-convolution-neural-network-in-5-mins-4217c2cf964f				

Implementation:

```
import keras

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
import numpy as np

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 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)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=(28, 28, 1)))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation="softmax"))

model.compile(
    loss=keras.losses.categorical_crossentropy,
    optimizer=keras.optimizers.Adadelta(),
    metrics=["accuracy"],
)

model.fit(
    x_train,
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(x_test, y_test),
)

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

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

Output

```
(base) PS C:\Users\harsh> & C:/Users/harsh/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/harsh/OneDrive - pdpu.ac.in/HARSH/_POEU/SEM 6/Artificial Intelligence/ASSIGNMENTS/21BCP359 AI Class Assignment 5.py"
2024-03-20 09:11:28.730820: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-03-20 09:11:31.598119: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ————— 2s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
C:\Users\harsh\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not pass an `input_shape`/'input_dim' argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(
2024-03-20 09:11:40.064406: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/12
469/469 ————— 47s 97ms/step - accuracy: 0.1162 - loss: 39.1241 - val_accuracy: 0.4118 - val_loss: 4.7768
Epoch 2/12
469/469 ————— 52s 111ms/step - accuracy: 0.2310 - loss: 16.1297 - val_accuracy: 0.5825 - val_loss: 1.9206
Epoch 3/12
469/469 ————— 79s 105ms/step - accuracy: 0.3199 - loss: 7.9788 - val_accuracy: 0.5928 - val_loss: 1.3063
Epoch 4/12
469/469 ————— 53s 112ms/step - accuracy: 0.3507 - loss: 4.5408 - val_accuracy: 0.5516 - val_loss: 1.3733

Epoch 5/12
469/469 ————— 52s 110ms/step - accuracy: 0.3562 - loss: 3.0447 - val_accuracy: 0.5067 - val_loss: 1.5433
Epoch 6/12
469/469 ————— 47s 100ms/step - accuracy: 0.3543 - loss: 2.4200 - val_accuracy: 0.4842 - val_loss: 1.6276
Epoch 7/12
469/469 ————— 47s 100ms/step - accuracy: 0.3692 - loss: 2.1728 - val_accuracy: 0.5037 - val_loss: 1.6175
Epoch 8/12
469/469 ————— 46s 98ms/step - accuracy: 0.3781 - loss: 2.0311 - val_accuracy: 0.5177 - val_loss: 1.5752
Epoch 9/12
469/469 ————— 47s 101ms/step - accuracy: 0.3878 - loss: 1.9486 - val_accuracy: 0.5353 - val_loss: 1.5090
Epoch 10/12
469/469 ————— 59s 125ms/step - accuracy: 0.4073 - loss: 1.8606 - val_accuracy: 0.5694 - val_loss: 1.4370
Epoch 11/12
469/469 ————— 63s 133ms/step - accuracy: 0.4303 - loss: 1.8076 - val_accuracy: 0.5874 - val_loss: 1.3649
Epoch 12/12
469/469 ————— 52s 112ms/step - accuracy: 0.4500 - loss: 1.7371 - val_accuracy: 0.6071 - val_loss: 1.2898
Test loss: 1.2905181646347046
Test accuracy: 0.6071000099182129
```

PRACTICAL 8

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	13-03-24	Batch:	G11
Aim:	Design the Neural Network model for the project title submitted by you. Demonstrate Over-fitting and solve the same using Dropout technique .				

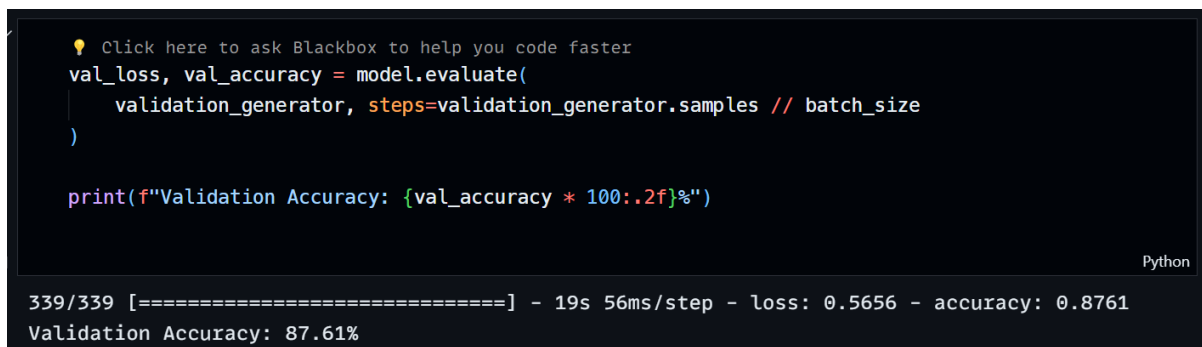
Overfitting and Regularization in Plant Disease Detection CNN

One of the key challenges in training deep learning models, particularly Convolutional Neural Networks (CNNs), is overfitting. Overfitting occurs when a model becomes too specialized on the training data and fails to generalize well to unseen data. This manifests as high training accuracy but low validation accuracy. In the context of plant disease detection, an overfitting model might achieve impressive results on the training images but struggle to accurately diagnose diseases in new images with slightly different characteristics.

To mitigate overfitting and improve the generalizability of our Plant Disease detection CNN model, we can employ a regularization technique called dropout. Dropout randomly drops out a certain percentage of neurons during training. This forces the model to learn redundant representations and prevents it from relying on any single neuron or its specific quirks in the training data. By encountering different "reduced versions" of the training data during each epoch, the model is encouraged to learn more generalizable features that are robust to variations in the data.

Before Dropout Implementation

```
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Conv2D(64, (3, 3), activation="relu"))
model.add(layers.MaxPooling2D(2, 2))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation="relu"))
model.add(layers.Dense(train_generator.num_classes, activation="softmax"))
```



```

Click here to ask Blackbox to help you code faster
val_loss, val_accuracy = model.evaluate(
    validation_generator, steps=validation_generator.samples // batch_size
)

print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")

```

Python

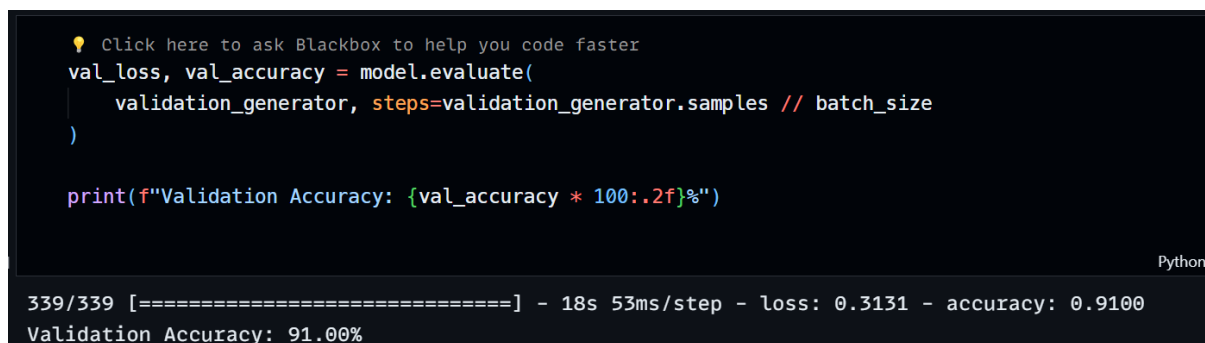
```

339/339 [=====] - 19s 56ms/step - loss: 0.5656 - accuracy: 0.8761
Validation Accuracy: 87.61%

```

After Dropout Implementation

```
model.add(layers.Conv2D(32, (3, 3), activation="relu",  
input_shape=(img_size, img_size, 3)))  
model.add(layers.MaxPooling2D(2, 2))  
model.add(layers.Dropout(0.25))  
model.add(layers.Conv2D(64, (3, 3), activation="relu"))  
model.add(layers.MaxPooling2D(2, 2))  
model.add(layers.Dropout(0.25))
```



```
Click here to ask Blackbox to help you code faster  
val_loss, val_accuracy = model.evaluate(  
    validation_generator, steps=validation_generator.samples // batch_size  
)  
  
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

Python

```
339/339 [=====] - 18s 53ms/step - loss: 0.3131 - accuracy: 0.9100  
Validation Accuracy: 91.00%
```

In this example, Dropout(0.25) is used after each convolutional layer. This means that during training, 25% of the neurons in those layers will be randomly dropped out. The dropout rate is a hyperparameter that can be tuned for optimal performance. Experimenting with different dropout rates (between 0.2 and 0.5) can help achieve the best balance between preventing overfitting and maintaining model accuracy.

The dropout rate (e.g., 0.25 in the code) determines the percentage of neurons dropped in each layer during training. It's a hyperparameter you can tune. Here's a general guideline:

- **Low Dropout (0-0.2):** Might not be effective enough in preventing overfitting.
- **Moderate Dropout (0.2-0.5):** Often a good starting point for many CNN architectures.
- **High Dropout (0.5-0.8):** Can be too aggressive and might hurt performance if not tuned carefully.

By incorporating dropout as a regularization technique, we can enhance the robustness and generalizability of our Plant Disease detection CNN model. This allows the model to perform more effectively on unseen plant images, ultimately leading to more accurate disease diagnoses.

PRACTICAL 9

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	20-03-2427-03-24	Batch:	G11
Aim:	For your project definition demonstrate applicable task out of prediction and classification. Explain the entire work flow of your project through a single diagram.				

Plant Disease Classification using CNN

1. Data Collection:

- Gather high-quality images of plant leaves.
- Include images of healthy leaves and leaves with various diseases you aim to classify.
- Ensure the image collection is representative of the target diseases.

2. Data Pre-processing:

- Prepare the images for training the model.

3. Model Development

• Model Architecture:

- This model utilizes a Convolutional Neural Network (CNN).
- CNNs excel at extracting spatial features from images, making them ideal for image classification tasks.

• Building Blocks:

- **Convolutional Layers:** These layers apply filters to the image, identifying patterns and extracting features.
- **Pooling Layers:** Reduce image dimensionality by summarizing features from previous layers (e.g., Max Pooling).
- **Activation Functions:** Introduce non-linearity to the network, allowing it to learn complex relationships – ReLU and Softmax.
- **Flatten Layer:** Transform the extracted features into a one-dimensional vector for feeding into fully-connected layers.
- **Fully-Connected Layers:** Dense layers that process the flattened features and make the final classification decision.

4. Training Process:

- Divide the pre-processed data into training, validation, and test sets.
- The training set is used to train the model, the validation set monitors training progress, and the test set evaluates final model performance on unseen data.
- The model learns by iteratively adjusting its internal weights based on the training data and the chosen loss function - Categorical Cross entropy for multi-class classification.

- The optimizer - Adam guides these adjustments to minimize the loss function and improve classification accuracy.

Evaluation and Deployment

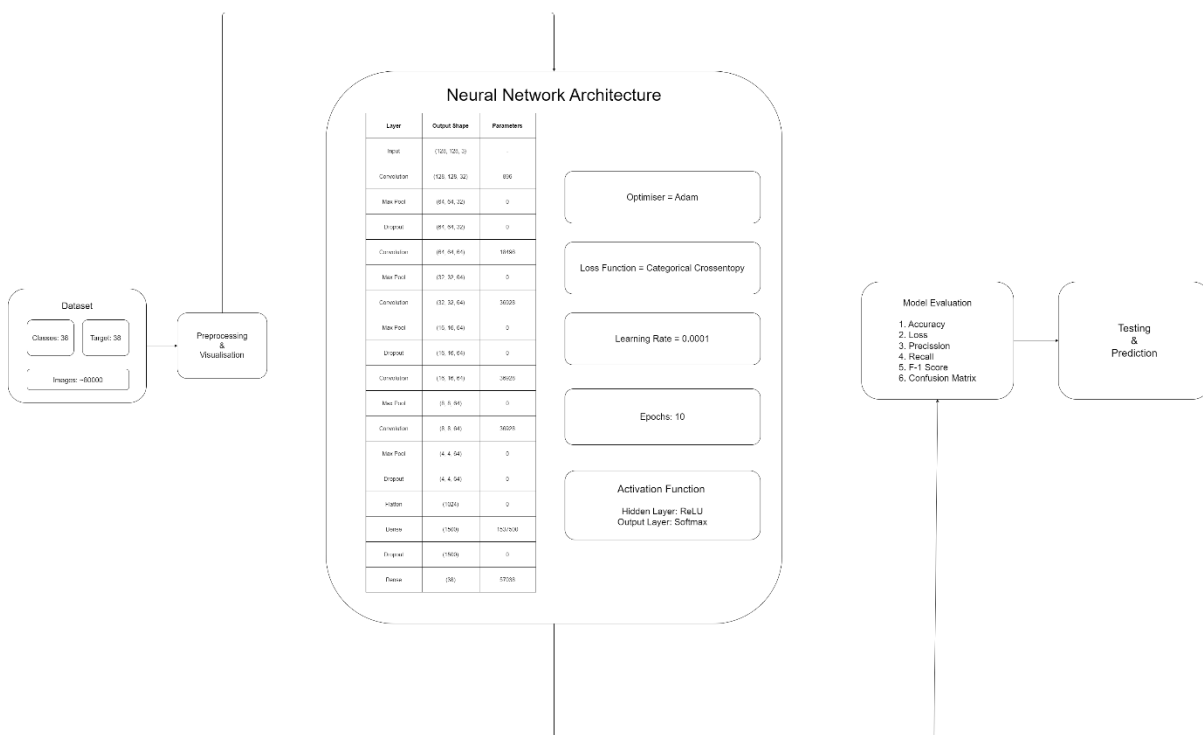
1. Model Evaluation:

- After training, assess the model's performance on the held-out test set.
- Common metrics include **accuracy**, **precision**, **recall**, and **F1-score**.

2. Deployment:

- Once satisfied with the model's performance, deploy it for real-world use.
- This could involve integrating it into a mobile application or web service for on-demand plant disease prediction.

Flow Chart



PRACTICAL 10

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	27-03-24	Batch:	G11
Aim:	For your project demonstrate the following: <ul style="list-style-type: none"> • need of optimizer - 5 marks • significance of your choice of optimizer - 5 marks • comparison of outcomes with and without optimization - 5 marks 				

Need of Optimizer

In a neural network, an optimizer plays a crucial role in training the model. The primary purpose of an optimizer is to minimise the error between the predicted output of the neural network and the actual target output by adjusting the weights and biases of the network during the training process.

- **Error Minimization:** Neural networks are trained by adjusting their parameters (weights and biases) to minimise the difference between predicted outputs and actual targets. Optimizers determine how these parameters should be updated to reduce this error effectively.
- **Gradient Descent:** Most optimizers use some form of gradient descent algorithm to update the weights and biases of the neural network. Gradient descent computes the gradients of the loss function with respect to the network parameters, indicating the direction and magnitude of the steepest decrease in the error. By following these gradients, the optimizer updates the parameters to reach a minimum point in the error surface.
- **Speed and Efficiency:** Optimizers help in training the neural network efficiently by adjusting the learning rate and other hyperparameters. They prevent the model from getting stuck in local minima and help it converge to the global minimum of the loss function.
- **Stability:** Some optimizers incorporate techniques to stabilise the training process, such as momentum, adaptive learning rates, and regularisation. These techniques help prevent the model from oscillating or diverging during training.
- **Flexibility:** Different optimizers have different characteristics and are suitable for different types of problems or architectures. For instance, Adam optimizer is widely used for its adaptive learning rate properties, while SGD (Stochastic Gradient Descent) is simpler and may be more suitable for certain scenarios.

Overall, optimizers are essential components of neural network training, ensuring that the model learns effectively and efficiently from the training data to make accurate predictions on new data.

Significance of ADAM Optimizer

Adam (Adaptive Moment Estimation) optimizer is a popular optimization algorithm used in training neural networks. Its significance lies in several key features that make it effective for a wide range of deep learning tasks:

- **Adaptive Learning Rates:** Adam dynamically adjusts the learning rates for each parameter based on the magnitude of gradients and the past history of gradients for that parameter. This adaptivity helps in faster convergence and efficient training, as it allows larger updates for infrequent parameters and smaller updates for frequent ones.
- **Momentum Optimization:** Adam incorporates momentum, which helps accelerate convergence by accumulating gradients from past time steps. This helps to navigate through areas of high curvature and reach the minimum of the loss function more efficiently.
- **Bias Correction:** Adam performs bias correction, particularly in the early stages of training when the estimates of the moments are biased towards zero. This correction helps to improve the stability of training and ensures that the optimization process starts with more accurate estimates.
- **Efficient Memory Usage:** Adam maintains exponentially decaying averages of past gradients and squared gradients, requiring only first-order moments (mean) and second-order moments (variance). This results in efficient memory usage compared to other optimization algorithms like RMSprop.
- **Robustness to Hyperparameters:** Adam is relatively less sensitive to hyperparameters compared to other optimization algorithms like SGD. It performs well with default hyperparameters across a wide range of tasks and architectures, making it easier to use for practitioners.
- **Widely Used:** Adam has become a standard optimizer in many deep learning frameworks and is often the default choice for training neural networks. Its widespread adoption and proven performance on various tasks make it a go-to optimizer for many practitioners.

Overall, the significance of Adam optimizer lies in its ability to provide efficient and effective optimization for neural networks, leading to faster convergence, better generalisation, and easier hyperparameter tuning.

Comparison with and without Optimizer

Without Optimiser

```

Evaluating Model

🔗 Click here to ask Blackbox to help you code faster
# Training Accuracy
train_loss, train_acc = cnn.evaluate(training_set)
print('Training accuracy:', train_acc)

Python

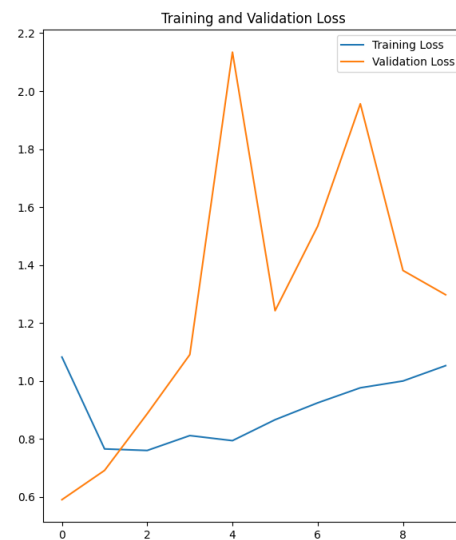
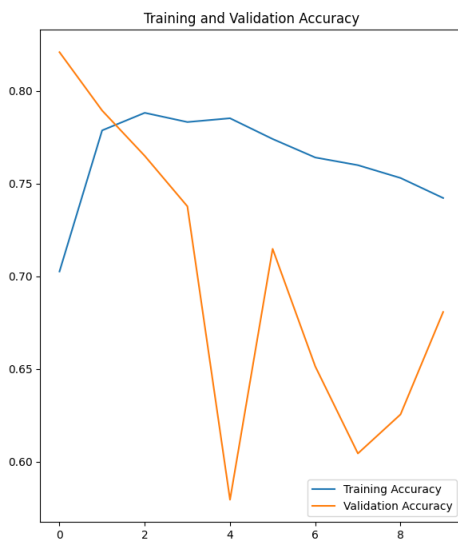
2197/2197 [=====] - 53s 24ms/step - loss: 1.2183 - accuracy: 0.6906
Training accuracy: 0.6905612349510193

🔗 Click here to ask Blackbox to help you code faster
# Validation Accuracy
val_loss, val_acc = cnn.evaluate(validation_set)
print('Validation accuracy:', val_acc)

Python

550/550 [=====] - 14s 24ms/step - loss: 1.2975 - accuracy: 0.6807
Validation accuracy: 0.6806851625442505

```



With Optimiser

Evaluating Model

Click here to ask Blackbox to help you code faster

Training Accuracy

```
train_loss, train_acc = cnn.evaluate(training_set)
print('Training accuracy:', train_acc)
```

Python

2197/2197 [=====] - 54s 24ms/step - loss: 0.1681 - accuracy: 0.9456
Training accuracy: 0.945600688457489

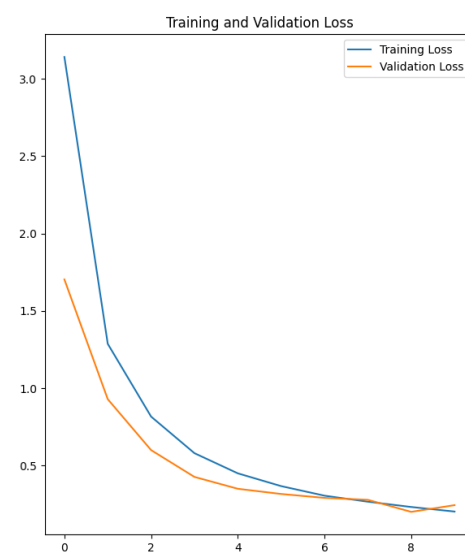
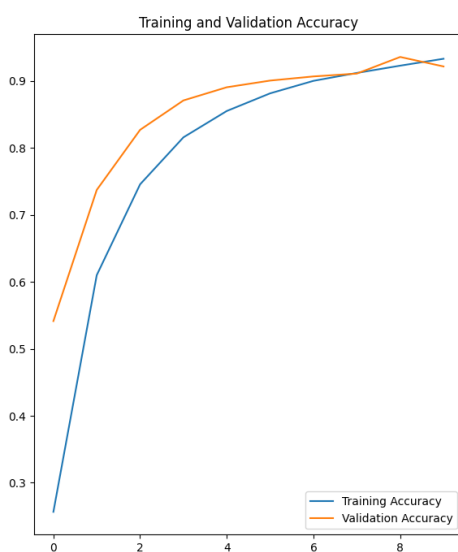
Click here to ask Blackbox to help you code faster

Validation Accuracy

```
val_loss, val_acc = cnn.evaluate(validation_set)
print('Validation accuracy:', val_acc)
```

Python

550/550 [=====] - 14s 25ms/step - loss: 0.2447 - accuracy: 0.9218
Validation accuracy: 0.9218074083328247



PRACTICAL 11

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	10-04-24	Batch:	G11
Aim:	11a - Understanding the basics and IDE for Prolog Programming 11b - Implement any two of the following using Prolog: <ul style="list-style-type: none"> • Medical diagnosis of common cold and flu using symptom inputs • Demonstrating list in prolog • Monkey banana problem • Find the factorial of a given number 				

11a: Understanding the basics and IDE for Prolog Programming

Prolog is a logic programming language associated with artificial intelligence and computational linguistics. In Prolog, programs are expressed in terms of relations and rules. It's based on a formal system called Horn clauses, which consist of facts and rules. Here's a brief overview:

- Facts:** Facts are statements about relationships between entities. They are represented as predicates.
Example:
`human(socrates).`
- Rules:** Rules define relationships based on conditions. They consist of a head and a body.
Example:
`mortal(X) :- human(X).`
- Queries:** In Prolog, you can ask queries to the knowledge base to retrieve information or verify facts.
Example:
`?- mortal(socrates).`

IDEs for Prolog Programming:

There are several IDEs (Integrated Development Environments) available for Prolog programming. Some popular ones include:

- **SWI-Prolog:** It's a comprehensive Prolog environment with a graphical debugger and IDE-like features. It's available for multiple platforms.
- **GNU Prolog:** This is a free Prolog compiler with a command-line interface. It provides a basic environment for Prolog development.
- **SICStus Prolog:** It's a commercial Prolog development system with a comprehensive IDE and advanced features for debugging and optimization.

11b: Implementing tasks in Prolog

Medical Diagnosis of Common Cold and Flu Using Symptom Inputs: `symptom(fever).`

```
symptom(cough). symptom(sore_throat).  
symptom(runny_nose).  
symptom(headache).  
symptom(muscle_aches).  
symptom(fatigue).
```

```
diagnosis(cold) :symptom(fever),  
    symptom(cough),  
    symptom(runny_nose),  
    not(symptom(headache)),  
    not(symptom(muscle_aches)),  
    not(symptom(sore_throat)),  
    not(symptom(fatigue)).
```

```
diagnosis(flu) :symptom(fever),  
    symptom(cough),  
    symptom(runny_nose),  
    symptom(headache),  
    symptom(muscle_aches),  
    symptom(fatigue),  
    not(symptom(sore_throat)).
```

Demonstrating Lists in Prolog:

% Predicate to check if X is a member of the list.

```
member(X, [X|_]).
```

```
member(X, [_|T]) :member(X, T).
```

% Predicate to append two lists.

```
append([], L, L).
```

```
append([H|T], L, [H|R]) :append(T, L, R).
```

PRACTICAL 12

Name:	Harsh Shah	Semester:	VI	Division:	6
Roll No.:	21BCP359	Date:	24-04-24	Batch:	G11
Aim:	WAP to design Tic Tac Toe games from O (Opponent) and X (Player) by using minimax algorithm.				

Program

```
import sys
```

```
# Function to print the Tic Tac Toe board
```

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("- " * 3)
```

```
# Function to check if the board is full
```

```
def is_board_full(board):
    for row in board:
        for cell in row:
            if cell == " ":
                return False
    return True
```

```
# Function to check if a player has won
```

```
def check_winner(board, player):
    # Check rows
    for row in board:
        if all(cell == player for cell in row):
            return True
    # Check columns
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    # Check diagonals
    if all(board[i][i] == player for i in range(3)) or all(
        board[i][2 - i] == player for i in range(3)
    ):
        return True
    return False
```

```
# Function to evaluate the current state of the board
```

```
def evaluate(board):
    if check_winner(board, "O"):
        return 1
    elif check_winner(board, "X"):
        return -1
```



```
elif is_board_full(board):
    return 0
else:
    return None

# Minimax algorithm implementation
def minimax(board, depth, is_maximizing):
    score = evaluate(board)
    if score is not None:
        return score

    if is_maximizing:
        best_score = -sys.maxsize
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "O"
                    score = minimax(board, depth + 1, False)
                    board[i][j] = " "
                    best_score = max(best_score, score)
            return best_score
    else:
        best_score = sys.maxsize
        for i in range(3):
            for j in range(3):
                if board[i][j] == " ":
                    board[i][j] = "X"
                    score = minimax(board, depth + 1, True)
                    board[i][j] = " "
                    best_score = min(best_score, score)
            return best_score

# Function to find the best move for the opponent using Minimax
def find_best_move(board):
    best_score = -sys.maxsize
    best_move = None
    for i in range(3):
        for j in range(3):
            if board[i][j] == " ":
                board[i][j] = "O"
                score = minimax(board, 0, False)
                board[i][j] = " "
                if score > best_score:
                    best_score = score
                    best_move = (i, j)
    return best_move

# Main function to play the game
def play_game():
    board = [[" " for _ in range(3)] for _ in range(3)]
    print("Welcome to Tic Tac Toe!")
    print_board(board)
```

```

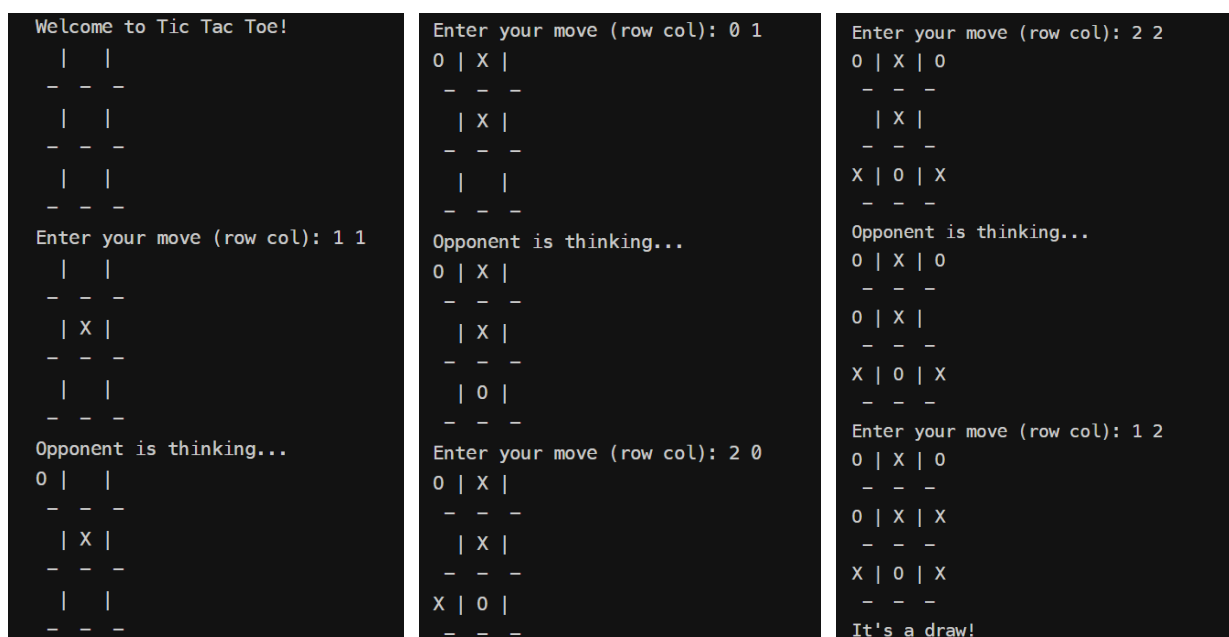
while True:
    # Player's move
    row, col = map(int, input("Enter your move (row col): ").split())
    if board[row][col] != " ":
        print("Invalid move. Try again.")
        continue
    board[row][col] = "X"
    print_board(board)
    if check_winner(board, "X"):
        print("Congratulations! You win!")
        break
    if is_board_full(board):
        print("It's a draw!")
        break

    # Opponent's move
    print("Opponent is thinking...")
    opponent_row, opponent_col = find_best_move(board)
    board[opponent_row][opponent_col] = "O"
    print_board(board)
    if check_winner(board, "O"):
        print("Sorry, you lose!")
        break
    if is_board_full(board):
        print("It's a draw!")
        break

if __name__ == "__main__":
    play_game()

```

Output



```

Welcome to Tic Tac Toe!
  |  |
-- --
  |  |
-- --
  |  |
-- --
Enter your move (row col): 1 1
  |  |
-- --
  | X |
-- --
  |  |
-- --
Opponent is thinking...
0 |  |
-- --
  | X |
-- --
  |  |
-- --
Enter your move (row col): 0 1
0 | X |
-- --
  | X |
-- --
  | O |
-- --
Enter your move (row col): 2 0
0 | X |
-- --
  | X |
-- --
X | O |
-- --
Enter your move (row col): 2 2
0 | X | O
-- --
  | X |
-- --
X | O | X
-- --
Opponent is thinking...
0 | X | O
-- --
0 | X |
-- --
X | O | X
-- --
Enter your move (row col): 1 2
0 | X | O
-- --
0 | X | X
-- --
X | O | X
-- --
It's a draw!

```