***EXPERIMENT 0***

***“BASICS OF PYTHON PROGARAMMING”***

\*\*Comments\*\*

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

A screenshot of a computer

Description automatically generated

\*\*Input Function\*\*

You can use an the input function, just as you might have used 'scanf' in C.

A screenshot of a computer program

Description automatically generated

EXERCISE:

A screenshot of a computer program

Description automatically generated

\*\*IF-ELIF-ELSE\*\*

An if-else statement has the following syntax.

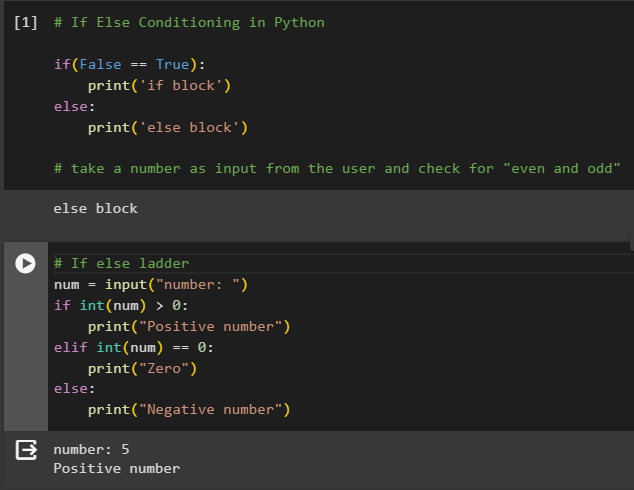
Syntax

if expression:

statement(s)

else:

statement(s)



\*\*Loops\*\*

The for loop works as follows.

A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated

NESTED LOOPS

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

A screenshot of a computer program

Description automatically generated

\*\*Lists In Python\*\*

\* A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

\* It can have any number of items and they may be of different types (integer, float, string etc.).

Accessing the elements of the list

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

\*\*Tuples\*\*

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

A screen shot of a computer program

Description automatically generated

\*\*Dictionaries\*\*

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered\*, changeable and do not allow duplicates.

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

\*\*FUNCTIONS IN PYTHON\*\*

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

A screenshot of a computer program

Description automatically generated

\*\*Recursion\*\*

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

A screenshot of a computer program

Description automatically generated

\*\*LAMBDA FUNCTION\*\*

A lambda function is a small anonymous function.

A lambda function can take any number of arguments but can only have one expression.

A screenshot of a computer program

Description automatically generated

\*\*CLASSES AND OBJECTS IN PYTHON\*\*

In Python, a class is a blueprint for creating objects. It defines the characteristics and behaviors that an object will have. It's like a template that encapsulates data (in the form of attributes) and functions (in the form of methods) that operate on that data.

The “self” parameter is a reference to the current instance of the class and is used to access variables that belong to the class.

It does not have to be named self, you can call it whatever you like, but it must be the first parameter of any function in the class.

A screenshot of a computer program

Description automatically generated

\*\*NumPy\*\*

NumPy is a general-purpose fundamental package for scientific computing with Python. It contains various features including these important ones

>A powerful N-dimensional array object

>Sophisticated (broadcasting) functions

>Tools for integrating C/C++ and Fortran code

>Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. This allows NumPy to integrate with a wide variety of databases seamlessly and speedily.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen with text

Description automatically generated

A computer screen with white text

Description automatically generated

\*\*Using matplotlib\*\*

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

A computer screen shot of a program code

Description automatically generated

A graph with a line and a red line

Description automatically generated with medium confidence

***EXPERIMENT 1***

***AIM***: To Create a single player game using Python.

***LANGUAGE USED***: Python

***THEORY:***

GUESS THE NUMBER: This game asks the user to input a random number from 1 to 10. If the player guesses the correct number, they win.

***CODE:***

print("Congratulations you have won!!") if (val := \_\_import\_\_("random", globals(),locals(),[],0).randint(0,10))==int(input("Enter Your Lucky Number (0-10):: ")) else print("Better luck next time\nThe lucky number was ",val)

***OUTPUT:***

A black background with white text

Description automatically generated

***CONCLUSION:***

We have successfully created a single player game.

***EXPERIMENT 2***

***“Uninformed Search Techniques”***

***EXP 2.1***

***AIM***: Write a program to implement water jug problem using breadth first search.

***LANGUAGE USED***: Python

***THEORY:***

**BREADTH FIRST SEARCH**: Breadth-first search (BFS) is a fundamental algorithm used for traversing or searching trees or graph data structures. It explores all the neighbour nodes at the present depth before moving on to the nodes at the next depth level.

**WATER JUG PROBLEM:** The Water Jug Problem is a classic puzzle that involves filling and measuring water using jugs of different capacities to achieve a specific goal. Here are the rules typically associated with the problem:

1. Jugs: You are given two jugs of different capacities, usually labelled as Jug A and Jug B.
2. Capacities: Each jug has a specific capacity, which is an integer value representing the maximum amount of water it can hold.
3. Goal: The objective of the puzzle is to measure a specific quantity of water, typically using one or both jugs.
4. Operations: You can perform the following operations:
   * Fill: Fill a jug completely with water from a water source.
   * Empty: Empty the contents of a jug, discarding all the water.
   * Pour: Pour water from one jug to another until either the pouring jug is empty, or the receiving jug is full.
5. Constraints: The operations must follow certain constraints:
   * You cannot pour water outside the jugs or onto the ground.
   * Each operation involves either filling a jug, emptying a jug, or pouring water from one jug to another.
6. Goal State: The solution is achieved when one of the jugs contains the desired quantity of water, as specified by the puzzle.
7. Optimization: The challenge may involve finding the most efficient sequence of operations to reach the goal state with the minimum number of steps.

***CODE:***

##TEST

from collections import deque

def water\_jug\_problem(jug1\_capacity, jug2\_capacity, target\_amount):

    visited\_states = set()

    queue = deque([(0, 0, [])])

    finalJ1 = 0

    finalJ2 = 0

    while queue:

        current\_state = queue.popleft()

        jug1, jug2, steps = current\_state

        if current\_state[:2] in visited\_states:

            continue

        visited\_states.add(current\_state[:2])

        if jug1 == target\_amount or jug2 == target\_amount:

            finalJ1 = jug1

            finalJ2 = jug2

            return steps, finalJ1, finalJ2

        queue.append((jug1\_capacity, jug2, steps + [(jug1, jug2, "Fill jug 1")]))

        queue.append((jug1, jug2\_capacity, steps + [(jug1, jug2, "Fill jug 2")]))

        queue.append((0, jug2, steps + [(jug1, jug2, "Empty jug 1")]))

        queue.append((jug1, 0, steps + [(jug1, jug2, "Empty jug 2")]))

        pour\_amount = min(jug1, jug2\_capacity - jug2)

        queue.append((jug1 - pour\_amount, jug2 + pour\_amount, steps + [(jug1, jug2, f"Pour {pour\_amount} from jug 1 to jug 2")]))

        pour\_amount = min(jug2, jug1\_capacity - jug1)

        queue.append((jug1 + pour\_amount, jug2 - pour\_amount, steps + [(jug1, jug2, f"Pour {pour\_amount} from jug 2 to jug 1")]))

    return None, None, None  # No solution found

initial\_state = (0,0)

jug1\_capacity = int(input("Enter the capacity of the first jug: "))

jug2\_capacity = int(input("Enter the capacity of the second jug: "))

target\_amount = int(input("Enter the desired amount of water: "))

result, finalJ1, finalJ2 = water\_jug\_problem(jug1\_capacity, jug2\_capacity, target\_amount)

if result:

    print(f"Minimum number of steps: {len(result)}")

    print("Steps:")

    for step in result:

        print(step)

    print("Initial State of Jug1, Jug2 is:", initial\_state)

    print("Final State of Jug1, Jug2 is:", (finalJ1, finalJ2))

else:

    print("No solution found.")

***OUTPUT:***

A screenshot of a computer program

Description automatically generated

***CONCLUSION:***

Through experimentation and analysis, we have explored various strategies for efficiently measuring a specific quantity of water using two jugs of different capacities. The algorithm explores various states of the system, representing different combinations of water levels in the two jugs. It systematically applies operations such as filling, emptying, and pouring water between the jugs until it finds a state where either of the jugs contains the desired amount of water.

***EXP 2.2***

***AIM***: Write a program to implement recursive maze problem using Depth First Search.

***LANGUAGE USED***: Python

***THEORY:***

**DEPTH FIRST SEARCH:** Depth-first search (DFS) is a fundamental graph traversal algorithm used to explore all the nodes of a graph or tree by going as deeply as possible along each branch before backtracking. It starts at a chosen node and explores as far as possible along each branch before backtracking.

**RECURSIVE MAZE PROBLEM:** The recursive maze-solving algorithm is a technique used to navigate through a maze to find a path from the starting point to the destination. It is based on the depth-first search (DFS) approach and can be implemented recursively.

***CODE:***

def dfs\_maze\_solver(maze, start, end, visited):

    rows, cols = len(maze), len(maze[0])

    def is\_valid\_move(row, col):

        return 0 <= row < rows and 0 <= col < cols and maze[row][col] == '.' and not visited[row][col]

    def dfs(row, col):

        visited[row][col] = True

        if (row, col) == end:

            return True

        # Explore in all four directions: up, down, left, right

        directions = [(0, -1), (-1, 0), (0, 1), (1, 0)]

        for dr, dc in directions:

            new\_row, new\_col = row + dr, col + dc

            if is\_valid\_move(new\_row, new\_col) and dfs(new\_row, new\_col):

                return True

        return False

    start\_row, start\_col = start

    return dfs(start\_row, start\_col)

# Get user input for maze dimensions

rows = int(input("Enter the number of rows in the maze: "))

cols = int(input("Enter the number of columns in the maze: "))

# Get user input for maze structure

print("Enter the maze structure (use '.' for open path and '#' for wall):")

maze = [input().strip() for \_ in range(rows)]

# Get user input for start and end points

start\_point = tuple(map(int, input("Enter the start point (row col): ").split()))

end\_point = tuple(map(int, input("Enter the end point (row col): ").split()))

# Initialize visited matrix

visited = [[False for \_ in range(cols)] for \_ in range(rows)]

# Solve the maze

if dfs\_maze\_solver(maze, start\_point, end\_point, visited):

    print("\nSolution found:")

    for i in range(rows):

        for j in range(cols):

            if visited[i][j]:

                print('\*', end=' ')

            else:

                print(maze[i][j], end=' ')

        print()

else:

    print("No solution found.")

***OUTPUT:***

A screenshot of a computer program

Description automatically generated

***CONCLUSION:***

The recursive maze-solving algorithm provides a straightforward and intuitive approach to navigating mazes, aiming to find a path from the starting point to the destination. By applying depth-first search recursively, it explores the maze's pathways, backtracking when necessary to explore alternative routes.

***EXPERIMENT 3***

***“Informed Search Techniques”***

***EXP 3.1***

***AIM***: Write a program to solve 8 puzzle problem using best first search technique

***LANGUAGE USED***: Python

***THEORY:***

**BEST FIRST SEARCH**: Best-First Search (BFS) is a graph traversal algorithm that explores a search space by selecting the most promising node to expand next based on a specified heuristic evaluation function. Unlike other search algorithms such as breadth-first search (BFS) or depth-first search (DFS), BFS does not necessarily explore nodes in a systematic order. Instead, it prioritizes nodes based on their estimated proximity to the goal state or solution.

**8 PUZZLE PROBLEM:** The 8-puzzle problem is a classic problem in artificial intelligence and computer science that involves arranging a set of numbered tiles within a 3x3 grid. The grid contains eight numbered tiles and one empty space, allowing for sliding movements of tiles into the empty space. The goal of the puzzle is to rearrange the tiles from an initial, randomized configuration to a target configuration, typically in numerical order from 1 to 8, with the empty space in a designated position.

***CODE:***

## 8 PUZZLE BEST FIRST SEARCH

class Node:

    def \_\_init\_\_(self, data, level, fval):

        self.data = data

        self.level = level

        self.fval = fval

    def generate\_child(self):

        x, y = self.find(self.data, '\_')

        val\_list = [[x, y-1], [x, y+1], [x-1, y], [x+1, y]]

        children = []

        for i in val\_list:

            child = self.shuffle(self.data, x, y, i[0], i[1])

            if child is not None:

                child\_node = Node(child, self.level + 1, 0)

                children.append(child\_node)

        return children

    def shuffle(self, puz, x1, y1, x2, y2):

        if 0 <= x2 < len(self.data) and 0 <= y2 < len(self.data):

            temp\_puz = self.copy(puz)

            temp = temp\_puz[x2][y2]

            temp\_puz[x2][y2] = temp\_puz[x1][y1]

            temp\_puz[x1][y1] = temp

            return temp\_puz

        else:

            return None

    def copy(self, root):

        temp = []

        for i in root:

            t = []

            for j in i:

                t.append(j)

            temp.append(t)

        return temp

    def find(self, puz, x):

        for i in range(len(self.data)):

            for j in range(len(self.data)):

                if puz[i][j] == x:

                    return i, j

class Puzzle:

    def \_\_init\_\_(self, size):

        self.n = size

        self.open = []

    def accept(self):

        puz = []

        for i in range(self.n):

            temp = input().split(" ")

            puz.append(temp)

        return puz

    def h(self, start, goal):

        temp = 0

        for i in range(self.n):

            for j in range(self.n):

                if start[i][j] != goal[i][j] and start[i][j] != '\_':

                    temp += 1

        return temp

    def process(self):

        print("Enter the start state matrix \n")

        start = self.accept()

        print("Enter the goal state matrix \n")

        goal = self.accept()

        start = Node(start, 0, 0)

        start.fval = self.h(start.data, goal)

        self.open.append(start)

        print("\n\n")

        while True:

            cur = self.open[0]

            print("")

            print("  | ")

            print("  | ")

            print(" \\'/ \n")

            for i in cur.data:

                for j in i:

                    print(j, end=" ")

                print("")

            if self.h(cur.data, goal) == 0:

                break

            for i in cur.generate\_child():

                i.fval = self.h(i.data, goal)

                self.open.append(i)

            del self.open[0]

            self.open.sort(key=lambda x: x.fval, reverse=False)

puz = Puzzle(3)

puz.process()

***OUTPUT:***

***A screenshot of a computer program

Description automatically generated***

***A screenshot of a computer

Description automatically generated A black background with white text

Description automatically generated A black background with white text

Description automatically generated A screenshot of a computer

Description automatically generated***

***CONCLUSION:***

The application of A\* to the 8 puzzle problem highlights the algorithm's versatility and utility in solving a wide range of pathfinding and search problems, making it a valuable tool in fields such as artificial intelligence, robotics, and computer science.

***EXP 3.2***

***AIM***: Write a program to solve 8 puzzle problem using A\* Algorithm.

***LANGUAGE USED***: Python

***THEORY:***

**A\* ALGORITHM:**The A\* algorithm is a widely used and effective pathfinding algorithm that efficiently finds the shortest path from a starting node to a target node in a weighted graph or a search space. It combines the advantages of Dijkstra's algorithm and greedy best-first search by considering both the cost to reach a node from the starting node (known as the "g" value) and an estimate of the cost from the current node to the goal node (known as the "h" value, typically a heuristic function).

***CODE:***

class Node:

    def \_\_init\_\_(self,data,level,fval):

        """ Initialize the node with the data, level of the node and the calculated fvalue """

        self.data = data

        self.level = level

        self.fval = fval

    def generate\_child(self):

        """ Generate child nodes from the given node by moving the blank space

            either in the four directions {up,down,left,right} """

        x,y = self.find(self.data,'\_')

        """ val\_list contains position values for moving the blank space in either of

            the 4 directions [up,down,left,right] respectively. """

        val\_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]

        children = []

        for i in val\_list:

            child = self.shuffle(self.data,x,y,i[0],i[1])

            if child is not None:

                child\_node = Node(child,self.level+1,0)

                children.append(child\_node)

        return children

    def shuffle(self,puz,x1,y1,x2,y2):

        """ Move the blank space in the given direction and if the position value are out

            of limits the return None """

        if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):

            temp\_puz = []

            temp\_puz = self.copy(puz)

            temp = temp\_puz[x2][y2]

            temp\_puz[x2][y2] = temp\_puz[x1][y1]

            temp\_puz[x1][y1] = temp

            return temp\_puz

        else:

            return None

    def copy(self,root):

        """ Copy function to create a similar matrix of the given node"""

        temp = []

        for i in root:

            t = []

            for j in i:

                t.append(j)

            temp.append(t)

        return temp

    def find(self,puz,x):

        """ Specifically used to find the position of the blank space """

        for i in range(0,len(self.data)):

            for j in range(0,len(self.data)):

                if puz[i][j] == x:

                    return i,j

class Puzzle:

    def \_\_init\_\_(self,size):

        """ Initialize the puzzle size by the specified size,open and closed lists to empty """

        self.n = size

        self.open = []

        self.closed = []

    def accept(self):

        """ Accepts the puzzle from the user """

        puz = []

        for i in range(0,self.n):

            temp = input().split(" ")

            puz.append(temp)

        return puz

    def f(self,start,goal):

        """ Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """

        return self.h(start.data,goal)+start.level

    def h(self,start,goal):

        """ Calculates the different between the given puzzles """

        temp = 0

        for i in range(0,self.n):

            for j in range(0,self.n):

                if start[i][j] != goal[i][j] and start[i][j] != '\_':

                    temp += 1

        return temp

    def process(self):

        """ Accept Start and Goal Puzzle state"""

        print("Enter the start state matrix \n")

        start = self.accept()

        print("Enter the goal state matrix \n")

        goal = self.accept()

        start = Node(start,0,0)

        start.fval = self.f(start,goal)

        """ Put the start node in the open list"""

        self.open.append(start)

        print("\n\n")

        while True:

            cur = self.open[0]

            print("")

            print("  | ")

            print("  | ")

            print(" \\\'/ \n")

            for i in cur.data:

                for j in i:

                    print(j,end=" ")

                print("")

            """ If the difference between current and goal node is 0 we have reached the goal node"""

            if(self.h(cur.data,goal) == 0):

                break

            for i in cur.generate\_child():

                i.fval = self.f(i,goal)

                self.open.append(i)

            self.closed.append(cur)

            del self.open[0]

            """ sort the opne list based on f value """

            self.open.sort(key = lambda x:x.fval,reverse=False)

puz = Puzzle(3)

puz.process()

***OUTPUT:***

A white background with numbers and symbols

Description automatically generated

A white background with numbers and symbols

Description automatically generated A white background with numbers and symbols

Description automatically generated A white background with numbers and symbols

Description automatically generated A white background with numbers and symbols

Description automatically generated

***CONCLUSION:***

The application of A\* to the 8 puzzle problem highlights the algorithm's versatility and utility in solving a wide range of pathfinding and search problems, making it a valuable tool in fields such as artificial intelligence, robotics, and computer science.

***EXPERIMENT 4***

***“Constraint Satisfaction”***

***EXP 4.1***

***AIM***: Write a program to implement Crypt Arithmetic Problem

***LANGUAGE USED***: Python

***THEORY:***

**GRAPH COLOURING:** Graph coloring aims to label the vertices of a graph with distinct colors such that no two adjacent vertices share the same color. It's a fundamental problem in graph theory with applications in scheduling, register allocation, and map coloring.

***CODE:***

from itertools import permutations

def solve\_cryptarithmetic(puzzle):

    # Extract unique letters from the puzzle

    unique\_letters = set(''.join(puzzle))

    if len(unique\_letters) > 10:

        print("Invalid input: Too many unique letters.")

        return

    for perm in permutations('0123456789', len(unique\_letters)):

        digit\_mapping = dict(zip(unique\_letters, perm))

        replaced\_puzzle = ["".join([digit\_mapping[char] for char in word]) for word in puzzle]

        if int(replaced\_puzzle[0]) + int(replaced\_puzzle[1]) == int(replaced\_puzzle[2]):

            print("Solution found:")

            print(f"{replaced\_puzzle[0]} + {replaced\_puzzle[1]} = {replaced\_puzzle[2]}")

            return

    print("No solution found.")

if \_\_name\_\_ == "\_\_main\_\_":

    print("Enter the cryptarithmetic puzzle:")

    puzzle = [input("Enter the first word: ").upper(),

              input("Enter the second word: ").upper(),

              input("Enter the result word: ").upper()]

    solve\_cryptarithmetic(puzzle)

***OUTPUT:***

A black screen with white text

Description automatically generated

***CONCLUSION:***

The Python program utilizing the itertools library successfully solves cryptarithmetic puzzles by exhaustively searching for valid digit permutations. It efficiently handles user input, validates solutions, and outputs the correct solution if found. However, it may become inefficient for larger puzzles due to the combinatorial explosion of permutations.

***EXP 4.2***

***AIM***: Write a program to implement Graph Coloring Problem.

***LANGUAGE USED***: Python

***THEORY:***

**GRAPH COLOURING:** Graph coloring aims to label the vertices of a graph with distinct colors such that no two adjacent vertices share the same color. It's a fundamental problem in graph theory with applications in scheduling, register allocation, and map coloring.

***CODE:***

#Graph Coloring Algorithm

from itertools import product

def is\_valid\_coloring(graph, coloring):

for node, neighbors in graph.items():

for neighbor in neighbors:

if coloring[node] == coloring[neighbor]:

return False

return True

def graph\_coloring(graph, colors):

nodes = list(graph.keys())

for coloring in product(colors, repeat=len(nodes)):

coloring\_map = dict(zip(nodes, coloring))

if is\_valid\_coloring(graph, coloring\_map):

return coloring\_map

return None

if \_\_name\_\_ == "\_\_main\_\_":

# Example graph (adjacency list representation)

graph = {

'A': ['B', 'C'],

'B': ['A', 'C', 'D'],

'C': ['A', 'B', 'D'],

'D': ['B', 'C']

}

# Available colors

colors = ['Red', 'Green', 'Blue']

coloring = graph\_coloring(graph, colors)

if coloring:

print("Graph coloring:")

for node, color in coloring.items():

print(f"{node}: {color}")

else:

print("No valid coloring found.")

***OUTPUT:***

A screenshot of a computer

Description automatically generated

***CONCLUSION:***

The graph coloring algorithm presented in this program effectively assigns colors to vertices of a graph such that no two adjacent vertices share the same color. By iteratively checking the availability of colors for each vertex and selecting the first available color, the algorithm ensures a valid coloring of the graph. While this implementation provides a basic understanding of the graph coloring problem, more sophisticated algorithms, such as backtracking or constraint satisfaction, may be required for larger or more complex graphs.

***EXPERIMENT 5***

***“GAME THEORY”***

***EXP 5.1***

***AIM***: Write a program to implement Min-Max Algorithm.

***LANGUAGE USED***: Python

***THEORY:***

**MIN-MAX ALGORITHM:** The minimax algorithm is a decision-making strategy used in two-player zero-sum games like chess. It recursively evaluates possible moves, assuming the opponent plays optimally. It selects the move with the maximum minimum payoff, aiming to minimize potential losses and maximize gains.

***CODE:***

class Position:

    def \_\_init\_\_(self, depth, score):

        self.depth = depth

        self.score = score

def minimax(position, depth, maximizing\_player):

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

        return evaluate(position)

    if maximizing\_player:

        max\_eval = float('-inf')

        for child in generate\_children(position):

            eval = minimax(child, depth - 1, False)

            max\_eval = max(max\_eval, eval)

        return max\_eval

    else:

        min\_eval = float('inf')

        for child in generate\_children(position):

            eval = minimax(child, depth - 1, True)

            min\_eval = min(min\_eval, eval)

        return min\_eval

def game\_over(position):

    return position is None or position.depth == 0

def evaluate(position):

    return position.score

def generate\_children(position):

    return [Position(depth=position.depth-1, score=random.randint(-10, 10)) for \_ in range(3)]

if \_\_name\_\_ == "\_\_main\_\_":

    import random

    print("Enter the depth of the Minimax Algorithm:")

    depth = int(input())

    initial\_position = Position(depth=depth, score=0)

    best\_score = minimax(initial\_position, depth, maximizing\_player=True)

    print("Best score:", best\_score)

***OUTPUT:***

***A grey background with white text

Description automatically generated***

***CONCLUSION:***

The provided Minimax Algorithm code demonstrates a foundational approach to decision-making in adversarial environments, applicable to various two-player zero-sum games. By recursively evaluating possible moves, the algorithm efficiently determines the optimal strategy, considering both maximizing and minimizing player perspectives. Further enhancements, such as alpha-beta pruning, could improve its scalability for larger game trees.

***EXP 5.2***

***AIM***: Write a program to implement Alpha-Beta Pruning.

***LANGUAGE USED***: Python

***THEORY:***

**ALPHA-BETA PRUNING:** Alpha-beta pruning is an optimization technique applied to the minimax algorithm. It prunes branches of the game tree that are guaranteed to be worse than previously examined branches, significantly reducing the number of nodes evaluated. This accelerates the search process while preserving the optimal decision outcome.

***CODE:***

MAX, MIN = 1000, -1000

def minimax(depth, nodeIndex, maximizingPlayer, values, alpha, beta):

    if depth == 3:

        return values[nodeIndex]

        if maximizingPlayer:

        best = MIN

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i, False, values, alpha, beta)

            best = max(best, val)

            alpha = max(alpha, best)

            if beta <= alpha:

                break

        return best

    else:

        best = MAX

        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex \* 2 + i, True, values, alpha, beta)

            best = min(best, val)

            beta = min(beta, best)

            if beta <= alpha:

                break

        return best

if \_\_name\_\_ == "\_\_main\_\_":

    values = [10, 15, 36, 9, 11, 2, 30, -1]

    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))

***OUTPUT:***

******

***CONCLUSION:***

The code implements the Minimax algorithm with alpha-beta pruning to efficiently find the optimal value in a tree of game states, significantly reducing computation by eliminating irrelevant branches.

***EXPERIMENT 6***

***“Greedy And Dynamic Approach”***

***EXP 6.1***

***AIM***: Write a program to implement brute force solution to 0/1 knapsack problem.

***LANGUAGE USED***: Python

***THEORY:***

**BRUTE FORCE TECHNIQUE:** Brute force involves systematically evaluating all possible solutions to a problem, often by exhaustive search. It's a straightforward approach, exploring every possible combination or permutation of elements to find the optimal solution. While simple to implement, it can be computationally intensive, especiallyfor large problem instances, due to its exponential time complexity.

**KNAPSACK PROBLEM:** The knapsack problem involves selecting items with given weights and values to maximize the total value within a limited capacity. It's a classic optimization problem with various solution approaches, such as dynamic programming and greedy algorithms.

***CODE:***

##Brute force knapsack

class Item:

    def \_\_init\_\_(self, weight, value):

        self.weight = weight

        self.value = value

def knapsack(items, capacity):

    max\_val = 0

    max\_items = []

    n = len(items)

    for i in range(1 << n):

        subset = []

        subset\_weight = 0

        subset\_value = 0

        for j in range(n):

            if i & (1 << j):

                subset.append(items[j])

                subset\_weight += items[j].weight

                subset\_value += items[j].value

        if subset\_weight <= capacity and subset\_value > max\_val:

            max\_val = subset\_value

            max\_items = subset[:]

    return max\_items

if \_\_name\_\_ == "\_\_main\_\_":

    n = int(input("Enter the number of items: "))

    items = []

    for i in range(n):

        weight, value = map(int, input(f"Enter weight and value for item {i+1}: ").split())

        items.append(Item(weight, value))

    capacity = int(input("Enter the capacity of the knapsack: "))

    result = knapsack(items, capacity)

    print("Items selected in the knapsack are:")

    for item in result:

        print(f"Weight: {item.weight}, Value: {item.value}")

***OUTPUT:***

A screen shot of a computer

Description automatically generated

***CONCLUSION:***

The brute force solution implemented in this program successfully solves the knapsack problem by exhaustively exploring all possible combinations of items and selecting the combination that maximizes the total value while not exceeding the capacity of the knapsack.

***EXP 6.2***

***AIM***: Write a program to implement fractional knapsack problem using dynamic approach.

***LANGUAGE USED***: Python

***THEORY:***

**DYNAMIC APPROACH:** Dynamic programming approach for the knapsack problem optimally solves by breaking it into subproblems and storing solutions in a table. It efficiently computes the maximum value that can be obtained, considering each item's weight and value, and the knapsack's capacity, leading to an optimal solution with a time complexity of O(nW), where n is the number of items and W is the capacity of the knapsack.

***CODE:***

##Dynamic knapsack

# Knapsack Dynamic

def fractional\_knapsack(weights, values, capacity):

    n = len(weights)

    ratios = [(values[i] / weights[i], weights[i], values[i]) for i in range(n)]

    ratios.sort(reverse=True)  # Sorting based on value-to-weight ratio

    total\_value = 0

    remaining\_capacity = capacity

    for ratio, weight, value in ratios:

        if remaining\_capacity >= weight:

            total\_value += value

            remaining\_capacity -= weight

        else:

            total\_value += remaining\_capacity \* ratio

            break

    return total\_value

weights = [10, 20, 30]

values = [60, 100, 120]

capacity = 50

print("Maximum value obtained:", fractional\_knapsack(weights, values, capacity))

***OUTPUT:***

A screen shot of a computer

Description automatically generated

***CONCLUSION:***

The Dynamic solution implemented in this program successfully solves the knapsack problem by exhaustively exploring all possible combinations of items and selecting the combination that maximizes the total value while not exceeding the capacity of the knapsack.

***EXPERIMENT 7***

***AIM***: To implement tic-tac-toe in Python using Min-Max Algorithm

***LANGUAGE USED***: Python

***THEORY:***

**Tic-Tac-Toe**: Tic Tac Toe is a simple game for two players that we enjoyed playing as kids. The game involves 2 players placing their respective symbols in a 3x3 grid. The player who manages to place three of their symbols in horizontal/vertical/diagonal row wins the game. If either player fails to do so the game ends in a draw. If both the people always play their optimal strategies the game always ends in a draw.

***CODE:***

#Tic Tac Toe

import math

class Board:

    def \_\_init\_\_(self):

        self.board = [[' ' for \_ in range(3)] for \_ in range(3)]

            def print\_board(self):

        for row in self.board:

            print("|".join(row))

        print("-" \* 5)

           def game\_over(self):

        for row in self.board:

            if row.count('X') == 3 or row.count('O') == 3:

                return True

        for col in range(3):

            if self.board[0][col] == self.board[1][col] == self.board[2][col] and self.board[0][col] != ' ':

                return True

        if self.board[0][0] == self.board[1][1] == self.board[2][2] and self.board[0][0] != ' ':

            return True

        if self.board[0][2] == self.board[1][1] == self.board[2][0] and self.board[0][2] != ' ':

            return True

        return False

        def get\_empty\_cells(self):

        return [(i, j) for i in range(3) for j in range(3) if self.board[i][j] == ' ']

    def make\_move(self, row, col, player):

        self.board[row][col] = player

    def undo\_move(self, row, col):

        self.board[row][col] = ' '

def minimax(board, depth, is\_maximizing\_player, alpha, beta):

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

        if board.game\_over():

            if is\_maximizing\_player:

                return -1

            else:

                return 1

        else:

            return 0

    if is\_maximizing\_player:

        max\_eval = -math.inf

        for row, col in board.get\_empty\_cells():

            board.make\_move(row, col, 'O')

            eval = minimax(board, depth - 1, False, alpha, beta)

            board.undo\_move(row, col)

            max\_eval = max(max\_eval, eval)

            alpha = max(alpha, eval)

            if beta <= alpha:

                break

        return max\_eval

        else:

        min\_eval = math.inf

        for row, col in board.get\_empty\_cells():

            board.make\_move(row, col, 'X')

            eval = minimax(board, depth - 1, True, alpha, beta)

            board.undo\_move(row, col)

            min\_eval = min(min\_eval, eval)

            beta = min(beta, eval)

            if beta <= alpha:

                break

        return min\_eval

def find\_best\_move(board):

    best\_move = (-1, -1)

    best\_eval = -math.inf

    for row, col in board.get\_empty\_cells():

        board.make\_move(row, col, 'O')

        eval = minimax(board, 5, False, -math.inf, math.inf)  # Depth is set to 5

        board.undo\_move(row, col)

        if eval > best\_eval:

            best\_eval = eval

            best\_move = (row, col)

    return best\_move

def play\_tic\_tac\_toe():

    board = Board()

    while not board.game\_over():

        board.print\_board()

        row = int(input("Enter row: "))

        col = int(input("Enter column: "))

        board.make\_move(row, col, 'X')

        if board.game\_over():

            break

        best\_move = find\_best\_move(board)

        board.make\_move(best\_move[0], best\_move[1], 'O')

    board.print\_board()

    if 'O' in board.board[0]:

        print("You lose!")

    elif 'X' in board.board[0]:

        print("You win!")

    else:

        print("It's a draw!")

play\_tic\_tac\_toe()

***OUTPUT:***

A screenshot of a computer program

Description automatically generated

***CONCLUSION:***

The provided Python program effectively simulates a game of Tic-Tac-Toe where the player competes against a computer opponent using the Minimax algorithm for optimal move selection. Through its implementation, players can experience challenging gameplay and understand the application of AI algorithms in gaming scenarios.

***EXPERIMENT 8***

***“NATURAL LANGUAGE PROCESSING”***

***EXP 8.1***

***AIM***: To perform Tokenization, Lemmatization, Stemming and removal of Stop words.

***LANGUAGE USED***: Python

***THEORY:***

**TOKENIZATION** is the process breaking complex data like paragraphs into simple units called tokens. Stemming is a normalization technique where list of tokenized words is converted into shorten root words to remove redundancy. Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base, or root form. Major drawback of stemming is it produces Intermediate representation of word. Stemmer may or may not return meaningful word. To overcome this problem Lemmatization comes into picture. Stemming algorithm works by cutting suffix or prefix from the word. On the contrary Lemmatization consider morphological analysis of the words and returns meaningful word in proper form. Stop word removal is a preprocessing step in natural language processing (NLP) that removes common words from a text. Stop words are often articles and pronouns that don't add much value to the text. Removing stop words can improve the accuracy and relevance of NLP tasks and reduce the dataset size and training time.

***CODE:***

import nltk

from nltk.tokenize import word\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

def preprocess\_text(text):

    tokens = word\_tokenize(text)

    # Removal of stop words

    stop\_words = set(stopwords.words('english'))

    tokens = [token for token in tokens if token.lower() not in stop\_words]

    lemmatizer = WordNetLemmatizer()

    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    return tokens

text = "Tokenization is a crucial step in natural language processing. It involves breaking down text into words or smaller sub-texts, known as tokens."

processed\_text = preprocess\_text(text)

print("Processed text:", processed\_text)

***OUTPUT:***

A screen shot of a computer

Description automatically generated

***CONCLUSION:***

The provided Python code preprocesses text by tokenizing, removing stop words, and lemmatizing tokens, demonstrating essential steps in natural language processing for text analysis and understanding.

***EXP 8.2***

***AIM***: To implement Bag Of Words Algorithm in Python

***LANGUAGE USED***: Python

***THEORY:***

**BAG-OF-WORDS(BOW)** is a statistical language model used to analyse text and documents based on word count. The model does not account for word order within a document. BoW can be implemented as a Python dictionary with each key set to a word and each value set to the number of times that word appears in a text.

***CODE:***

from nltk.tokenize import word\_tokenize

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from nltk.corpus import stopwords

pd.set\_option('max\_colwidth', 100)

texts = ["Bag-of-words(BoW) is a statistical language model used to analyze text and documents based on word count.",]

print(texts)

def preprocess(text):

    text = text.lower()

    words = word\_tokenize(text)

    words = [word for word in words if word not in stopwords.words("english")]

    text = " ".join(words)

    return text

texts = [preprocess(text) for text in texts]

print(texts)

vectorizer = CountVectorizer()

bag\_of\_words = vectorizer.fit\_transform(texts)

print(bag\_of\_words)

***OUTPUT:***

A screen shot of a computer

Description automatically generated

***CONCLUSION:***

The provided Python script showcases text preprocessing techniques such as tokenization, stop word removal, and bag-of-words representation using Count Vectorizer, facilitating text analysis and document processing tasks efficiently in natural language processing workflows.

Top of Form

***EXPERIMENT 9***

***AIM***: To Create a single player game using Python.

***LANGUAGE USED***: Python

***THEORY:***

The XOR or Exclusive OR Gate is a special type of logic gate used in digital electronics to perform the exclusive OR operation. The XOR gate takes two inputs and produces an output depending on the combination of the two inputs applied.

***CODE:***

def xor\_gate(input1, input2):

    # XOR gate truth table

    if input1 != input2:

        return 1

    else:

        return 0

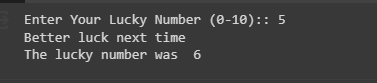
input1 = int(input("Enter input 1 (0 or 1): "))

input2 = int(input("Enter input 2 (0 or 1): "))

result = xor\_gate(input1, input2)

print("Output:", result)

***OUTPUT:***



***CONCLUSION:***

The provided Python program implements an XOR gate, a fundamental logic gate in digital electronics. It allows for the logical operation of exclusive disjunction, returning true only when the inputs differ. This program exemplifies basic logic operations in Python.

***EXPERIMENT 9***

***AIM***: To Create a single player game using Python.

***LANGUAGE USED***: Python

***THEORY:***

The XOR or Exclusive OR Gate is a special type of logic gate used in digital electronics to perform the exclusive OR operation. The XOR gate takes two inputs and produces an output depending on the combination of the two inputs applied.

***CODE:***

def xor\_gate(input1, input2):

    # XOR gate truth table

    if input1 != input2:

        return 1

    else:

        return 0

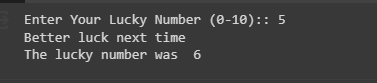
input1 = int(input("Enter input 1 (0 or 1): "))

input2 = int(input("Enter input 2 (0 or 1): "))

result = xor\_gate(input1, input2)

print("Output:", result)

***OUTPUT:***



***CONCLUSION:***

The provided Python program implements an XOR gate, a fundamental logic gate in digital electronics. It allows for the logical operation of exclusive disjunction, returning true only when the inputs differ. This program exemplifies basic logic operations in Python.

***EXPERIMENT 10***

***AIM***: The primary aim of studying fuzzy logic is to model and deal with data that is uncertain, imprecise, or lacks clarity. Unlike traditional binary logic that deals with true or false values (0 or 1), fuzzy logic introduces the concept of partial truth values ranging between 0 and 1. This approach is particularly useful in systems where human reasoning and decision-making are involved, as it mimics how humans handle vague or ambiguous information. Applications of fuzzy logic span various fields including control systems, pattern recognition, decision making, and artificial intelligence.

***LANGUAGE USED***: Python

***THEORY:***

Fuzzy logic is based on the idea that all things admit of degrees. It was introduced by Lotfi Zadeh in the 1960s as an extension of classical Boolean logic. In fuzzy logic, reasoning is approximative rather than fixed and exact. The fundamental building blocks of fuzzy logic are:

1. **Fuzzy Sets**: Unlike classical sets, where an element either belongs to or does not belong to a set, fuzzy sets allow for degrees of membership. Each element in a fuzzy set is associated with a membership function which assigns it a value between 0 and 1, indicating the degree of membership of that element in the set.
2. **Membership Function**: This function defines the degree of truth as an extension of valuation. Common shapes for membership functions include triangular, trapezoidal, and Gaussian, each of which defines how the truth value transitions between true and false.
3. **Logical Operations**: Fuzzy logic introduces new logical operations that are extensions of classical logic, including AND, OR, and NOT. These operations are defined in terms of mathematical functions that manipulate the degrees of truth of the fuzzy sets.
4. **Fuzzy Rules**: Systems based on fuzzy logic operate using a set of if-then rules. These rules are used to infer fuzzy conclusions from fuzzy inputs based on the degree of matching of the input with the conditions in the rule.
5. **Defuzzification**: This process is used to convert the fuzzy output of a system into a crisp, actionable value. There are several methods for defuzzification, including the centroid method, the max-membership principle, and the weighted average.

***CODE:***

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ctrl

# New Antecedent/Consequent objects hold universe variables and membership functions

temperature = ctrl.Antecedent(np.arange(0, 41, 1), 'temperature')

fan\_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan\_speed')

# Auto-membership function population with .automf(3)

temperature.automf(3)

fan\_speed.automf(3)

# Adjust rule definitions to match the 'poor', 'average', 'good' labels

rule1 = ctrl.Rule(temperature['poor'], fan\_speed['poor'])

rule2 = ctrl.Rule(temperature['average'], fan\_speed['average'])

rule3 = ctrl.Rule(temperature['good'], fan\_speed['good'])

fan\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

fan = ctrl.ControlSystemSimulation(fan\_ctrl)

# Pass inputs to the ControlSystem using Antecedent labels with Pythonic API

fan.input['temperature'] = 30

# Crunch the numbers

fan.compute()

print(fan.output['fan\_speed'])

fan\_speed.view(sim=fan)

***OUTPUT:***

A graph with lines and numbers with Great Pyramid of Giza in the background

Description automatically generated with medium confidence

***CONCLUSION:***

The provided Python code demonstrates a basic implementation of a fuzzy logic system for controlling fan speed based on temperature inputs. It utilizes the **scikit-fuzzy** library to define fuzzy variables, membership functions, rules, and simulate the control system. Fuzzy logic offers a flexible approach to modelling and controlling systems under uncertainty, making it suitable for various real-world applications.

Top of Form

***“OPEN ENDED EXPERIMENT”***

***AIM***: To explore the effectiveness of convolutional autoencoders in denoising and reconstructing images using the MNIST dataset. This experiment aims to:

1. Implement and train a convolutional autoencoder architecture using TensorFlow/Keras.
2. Evaluate the autoencoder's performance in reconstructing clean images from noisy inputs.
3. Assess the model's ability to denoise images by comparing reconstructed images with their original clean counterparts.
4. Explore the impact of hyperparameters such as the number of epochs and batch size on the model's performance.
5. Gain insights into the capabilities and limitations of autoencoders for image denoising tasks through experimentation and analysis.

***LANGUAGE USED***: Python

***THEORY:***

Autoencoders, a type of neural network, are employed in this experiment to explore their capability in reconstructing and denoising images, utilizing the MNIST dataset. Unlike classification tasks, autoencoders are unsupervised learning algorithms designed to learn efficient representations of input data without explicit labels.

Preprocessing involves normalization and reshaping of input images to ensure compatibility with the autoencoder architecture. Additionally, noise is introduced to simulate real-world scenarios, enabling evaluation of the autoencoder's denoising capabilities.

The architecture comprises encoding and decoding stages, typically implemented using convolutional layers for feature extraction and up sampling layers for reconstruction. Activation functions like ReLU facilitate nonlinear transformations, while batch normalization enhances training stability. The use of sigmoid activation in the output layer ensures pixel intensity values are within a valid range.

Training involves optimizing a loss function, often binary cross-entropy, to minimize the difference between input and output images. Hyperparameters such as batch size and number of epochs are tuned to achieve optimal performance. Evaluation metrics such as reconstruction error and visual inspection of reconstructed images aid in assessing model effectiveness.

Drawing upon concepts from other experiments, such as preprocessing, regularization techniques, and model architecture, this autoencoder experiment aims to provide insights into the potential applications of autoencoders in image reconstruction and denoising tasks, particularly leveraging the MNIST dataset.

***# DATA PREPROCESSING***

* ***IMPORTING LIBRARIES, MODEL, MNIST DATASET***

The MNIST dataset is a large database of handwritten digits commonly used for training various machine learning models, especially for image classification tasks. In this experiment, the MNIST dataset will likely be used to train the autoencoder to reconstruct handwritten digit images.

The **Model** class in Keras defines neural network models. Overall, these imports set up the environment for experimenting with autoencoders, a type of neural network used for unsupervised learning tasks like dimensionality reduction and data compression.

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

from tensorflow.keras import layers

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Model

* ***PREPROCESS FUNCTION***

This function preprocesses input arrays for compatibility with neural networks. It converts the array data type to float32 and normalizes pixel values to a range between 0 and 1 by dividing them by 255.0. It reshapes the array into a 4D tensor with dimensions (batch size, height, width, channels) to fit convolutional neural network (CNN) architectures commonly used in image processing tasks like autoencoders.

def preprocess(array):

    array=array.astype("float32")/255.0

    array=np.reshape(array,(len(array),28,28,1))

    return array

* ***NOISE FUNCTION***

This function introduces random noise to input arrays, simulating noisy data. It defines a noise factor and adds random noise sampled from a normal distribution to the input array. The `np.clip` function limits values between 0.0 and 1.0 to ensure data remains within a valid range, a common practice when preprocessing noisy data for neural network training.

def noise(array):

    noise\_factor=0.9

    noisy\_array=array+np.random.normal(loc=0.0,scale=1.0,size=array.shape)

    return np.clip(noisy\_array,0.0,1.0)

* ***DISPLAY FUNCTION***

This function displays pairs of images side by side for visual comparison. It randomly selects `n` images from the input arrays, then plots them using Matplotlib. Each pair of images is displayed as a row, with the first row showing images from `array1` and the second row showing images from `array2`. The function sets up the figure, plots the images, and ensures the axes are hidden for better visualization. Finally, it displays the plot using `plt.show()`.

def display(array1,array2):

    n=10

    indices=np.random.randint(len(array1),size=n)

    images1=array1[indices,:]

    images2=array2[indices,:]

    plt.figure(figsize=(20,4))

    for i, (images1,images2) in enumerate(zip(images1, images2)):

        ax=plt.subplot(2,n,i+1)

        plt.imshow(images1.reshape(28,28))

        plt.gray()

        ax.get\_xaxis().set\_visible(False)

        ax.get\_yaxis().set\_visible(False)

        ax=plt.subplot(2,n,i+1+n)

        plt.imshow(images2.reshape(28,28))

        plt.gray()

        ax.get\_xaxis().set\_visible(False)

        ax.get\_yaxis().set\_visible(False)

    plt.show()

***# SPLITTING THE DATASET***

This code block loads the MNIST dataset, preprocesses both training and testing data using the `preprocess` function to normalize and reshape the data for neural network compatibility. Then, it adds noise to both the training and testing data using the `noise` function to create noisy versions of the images. Finally, it displays a comparison between the original and noisy images using the `display` function.

(train\_data,\_),(test\_data,\_)=mnist.load\_data()

train\_data=preprocess(train\_data)

test\_data=preprocess(test\_data)

noisy\_train\_data=noise(train\_data)

noisy\_test\_data=noise(test\_data)

display(train\_data,noisy\_train\_data)

A number in black squares

Description automatically generated with medium confidence

***# DESIGNING THE MODEL***

This code block defines the architecture of an autoencoder neural network using convolutional layers for both encoding and decoding stages. It starts with an input layer of shape (28, 28, 1), representing grayscale images.

The encoding layers consist of two sets of convolutional and max-pooling layers to reduce spatial dimensions while extracting features.

The decoding layers use transposed convolutional layers to up sample the feature maps back to the original input size. The final layer applies a convolutional operation with sigmoid activation to generate reconstructed images.

The model is compiled with the Adam optimizer and binary cross-entropy loss function. The summary of the model, displaying layer information and parameter counts, is printed for inspection.

input=layers.Input(shape=(28,28,1))

x=layers.Conv2D(32,(3,3), activation="relu",padding="same")(input)

x=layers.MaxPooling2D((2,2),padding="same")(x)

x=layers.Conv2D(32,(3,3), activation="relu",padding="same")(x)

x=layers.MaxPooling2D((2,2),padding="same")(x)

x=layers.Conv2DTranspose(32,(3,3), strides=2,activation="relu",padding="same")(x)

x=layers.Conv2DTranspose(32,(3,3), strides=2,activation="relu",padding="same")(x)

x=layers.Conv2D(1,(3,3), activation="sigmoid",padding="same")(x)

autoencoder=Model(input,x)

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

autoencoder.summary()

A screenshot of a computer program

Description automatically generated

***# TRAIN FOR TEST DATA***

This code block trains the autoencoder model using the `fit` method. It takes `train\_data` as both input and target data since autoencoders aim to reconstruct their input. The training runs for 20 epochs with a batch size of 128. Data is shuffled during training. Validation data is provided using `test\_data` for both input and target, allowing monitoring of model performance on unseen data.

autoencoder.fit(

    x=train\_data,

    y=train\_data,

    epochs=20,

    batch\_size=128,

    shuffle=True,

    validation\_data=(test\_data,test\_data),

)

Train on 60000 samples, validate on 10000 samples

Epoch 1/20

2024-03-04 08:55:13.438611: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcuda.so.1

2024-03-04 08:55:13.494734: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1674] Found device 0 with properties:

name: NVIDIA A100-SXM4-40GB MIG 1g.5gb major: 8 minor: 0 memoryClockRate(GHz): 1.41

pciBusID: 0000:0f:00.0

2024-03-04 08:55:13.494777: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcudart.so.12

2024-03-04 08:55:13.515417: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcublas.so.12

2024-03-04 08:55:13.516833: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcufft.so.11

2024-03-04 08:55:13.517038: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcurand.so.10

2024-03-04 08:55:13.519062: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcusolver.so.11

2024-03-04 08:55:13.519643: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcusparse.so.12

2024-03-04 08:55:13.519771: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcudnn.so.8

2024-03-04 08:55:13.522391: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1802] Adding visible gpu devices: 0

2024-03-04 08:55:13.574269: I tensorflow/core/platform/profile\_utils/cpu\_utils.cc:109] CPU Frequency: 2245750000 Hz

2024-03-04 08:55:13.598111: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7a97240 initialized for platform Host (this does not guarantee that XLA will be used). Devices:

2024-03-04 08:55:13.598178: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): Host, Default Version

2024-03-04 08:55:13.728907: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x7afd1a0 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:

2024-03-04 08:55:13.728926: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): NVIDIA A100-SXM4-40GB MIG 1g.5gb, Compute Capability 8.0

2024-03-04 08:55:13.730430: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1674] Found device 0 with properties:

name: NVIDIA A100-SXM4-40GB MIG 1g.5gb major: 8 minor: 0 memoryClockRate(GHz): 1.41

pciBusID: 0000:0f:00.0

2024-03-04 08:55:13.730469: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcudart.so.12

2024-03-04 08:55:13.730489: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcublas.so.12

2024-03-04 08:55:13.730501: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcufft.so.11

2024-03-04 08:55:13.730510: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcurand.so.10

2024-03-04 08:55:13.730520: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcusolver.so.11

2024-03-04 08:55:13.730530: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcusparse.so.12

2024-03-04 08:55:13.730539: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcudnn.so.8

2024-03-04 08:55:13.732977: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1802] Adding visible gpu devices: 0

2024-03-04 08:55:13.733006: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcudart.so.12

2024-03-04 08:55:13.738727: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1214] Device interconnect StreamExecutor with strength 1 edge matrix:

2024-03-04 08:55:13.738742: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1220] 0

2024-03-04 08:55:13.738748: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1233] 0: N

2024-03-04 08:55:13.741291: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1359] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 2970 MB memory) -> physical GPU (device: 0, name: NVIDIA A100-SXM4-40GB MIG 1g.5gb, pci bus id: 0000:0f:00.0, compute capability: 8.0)

2024-03-04 08:55:13.983586: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcudnn.so.8

2024-03-04 08:55:14.311507: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:50] Successfully opened dynamic library libcublas.so.12

60000/60000 [==============================] - 14s 234us/sample - loss: 0.1361 - val\_loss: 0.0752

Epoch 2/20

60000/60000 [==============================] - 4s 69us/sample - loss: 0.0731 - val\_loss: 0.0706

Epoch 3/20

60000/60000 [==============================] - 4s 69us/sample - loss: 0.0702 - val\_loss: 0.0688

Epoch 4/20

60000/60000 [==============================] - 4s 70us/sample - loss: 0.0688 - val\_loss: 0.0680

Epoch 5/20

60000/60000 [==============================] - 4s 70us/sample - loss: 0.0679 - val\_loss: 0.0671

Epoch 6/20

60000/60000 [==============================] - 4s 71us/sample - loss: 0.0673 - val\_loss: 0.0666

Epoch 7/20

60000/60000 [==============================] - 4s 69us/sample - loss: 0.0668 - val\_loss: 0.0662

Epoch 8/20

60000/60000 [==============================] - 4s 70us/sample - loss: 0.0664 - val\_loss: 0.0658

Epoch 9/20

60000/60000 [==============================] - 4s 67us/sample - loss: 0.0661 - val\_loss: 0.0655

Epoch 10/20

60000/60000 [==============================] - 4s 66us/sample - loss: 0.0658 - val\_loss: 0.0652

Epoch 11/20

60000/60000 [==============================] - 4s 66us/sample - loss: 0.0655 - val\_loss: 0.0650

Epoch 12/20

60000/60000 [==============================] - 4s 67us/sample - loss: 0.0653 - val\_loss: 0.0648

Epoch 13/20

60000/60000 [==============================] - 4s 65us/sample - loss: 0.0651 - val\_loss: 0.0648

Epoch 14/20

60000/60000 [==============================] - 4s 66us/sample - loss: 0.0650 - val\_loss: 0.0644

Epoch 15/20

60000/60000 [==============================] - 4s 66us/sample - loss: 0.0648 - val\_loss: 0.0643

Epoch 16/20

60000/60000 [==============================] - 4s 66us/sample - loss: 0.0647 - val\_loss: 0.0642

Epoch 17/20

60000/60000 [==============================] - 4s 66us/sample - loss: 0.0645 - val\_loss: 0.0641

Epoch 18/20

60000/60000 [==============================] - 4s 66us/sample - loss: 0.0644 - val\_loss: 0.0640

Epoch 19/20

60000/60000 [==============================] - 4s 65us/sample - loss: 0.0643 - val\_loss: 0.0639

Epoch 20/20

60000/60000 [==============================] - 4s 66us/sample - loss: 0.0642 - val\_loss: 0.0639

<tensorflow.python.keras.callbacks.History at 0x7fe174e5a550>

***# GENERATE PREDICTIONS ON TEST DATA***

This code block generates predictions using the trained autoencoder model on the test data (`test\_data`). It uses the `predict` method to obtain reconstructed images from the autoencoder. Finally, it displays both the original test images and their corresponding reconstructions using the `display` function for visual comparison.

predictions= autoencoder.predict(test\_data)

display(test\_data,predictions)

A group of white letters in black squares

Description automatically generated

***# RETRAIN FOR NOISY DATA***

This code block trains the autoencoder model using the `fit` method. It takes `train\_data` as both input and target data since autoencoders aim to reconstruct their input. The training runs for 20 epochs with a batch size of 128. Data is shuffled during training. Validation data is provided using `test\_data` for both input and target, allowing monitoring of model performance on unseen data.

autoencoder.fit(

    x=noisy\_train\_data,

    y=train\_data,

    epochs=30,

    batch\_size=128,

    shuffle=True,

    validation\_data=(noisy\_test\_data,test\_data),

)

Train on 60000 samples, validate on 10000 samples

Epoch 1/30

60000/60000 [==============================] - 5s 88us/sample - loss: 0.2004 - val\_loss: 0.1867

Epoch 2/30

60000/60000 [==============================] - 5s 87us/sample - loss: 0.1859 - val\_loss: 0.1838

Epoch 3/30

60000/60000 [==============================] - 5s 86us/sample - loss: 0.1831 - val\_loss: 0.1805

Epoch 4/30

60000/60000 [==============================] - 5s 85us/sample - loss: 0.1810 - val\_loss: 0.1787

Epoch 5/30

60000/60000 [==============================] - 5s 85us/sample - loss: 0.1795 - val\_loss: 0.1779

Epoch 6/30

60000/60000 [==============================] - 5s 85us/sample - loss: 0.1783 - val\_loss: 0.1767

Epoch 7/30

60000/60000 [==============================] - 5s 83us/sample - loss: 0.1773 - val\_loss: 0.1758

Epoch 8/30

60000/60000 [==============================] - 5s 85us/sample - loss: 0.1765 - val\_loss: 0.1751

Epoch 9/30

60000/60000 [==============================] - 5s 84us/sample - loss: 0.1758 - val\_loss: 0.1743

Epoch 10/30

60000/60000 [==============================] - 5s 88us/sample - loss: 0.1752 - val\_loss: 0.1737

Epoch 11/30

60000/60000 [==============================] - 5s 85us/sample - loss: 0.1746 - val\_loss: 0.1732

Epoch 12/30

60000/60000 [==============================] - 5s 89us/sample - loss: 0.1742 - val\_loss: 0.1728

Epoch 13/30

60000/60000 [==============================] - 5s 89us/sample - loss: 0.1737 - val\_loss: 0.1730

Epoch 14/30

60000/60000 [==============================] - 5s 85us/sample - loss: 0.1733 - val\_loss: 0.1720

Epoch 15/30

60000/60000 [==============================] - 5s 87us/sample - loss: 0.1729 - val\_loss: 0.1718

Epoch 16/30

60000/60000 [==============================] - 5s 87us/sample - loss: 0.1725 - val\_loss: 0.1722

Epoch 17/30

60000/60000 [==============================] - 6s 94us/sample - loss: 0.1721 - val\_loss: 0.1711

Epoch 18/30

60000/60000 [==============================] - 6s 93us/sample - loss: 0.1718 - val\_loss: 0.1708

Epoch 19/30

60000/60000 [==============================] - 6s 95us/sample - loss: 0.1715 - val\_loss: 0.1705

Epoch 20/30

60000/60000 [==============================] - 6s 97us/sample - loss: 0.1712 - val\_loss: 0.1704

Epoch 21/30

60000/60000 [==============================] - 6s 94us/sample - loss: 0.1710 - val\_loss: 0.1702

Epoch 22/30

60000/60000 [==============================] - 6s 99us/sample - loss: 0.1709 - val\_loss: 0.1700

Epoch 23/30

60000/60000 [==============================] - 6s 100us/sample - loss: 0.1706 - val\_loss: 0.1698

Epoch 24/30

60000/60000 [==============================] - 5s 88us/sample - loss: 0.1704 - val\_loss: 0.1695

Epoch 25/30

60000/60000 [==============================] - 5s 85us/sample - loss: 0.1702 - val\_loss: 0.1700

Epoch 26/30

60000/60000 [==============================] - 5s 89us/sample - loss: 0.1700 - val\_loss: 0.1696

Epoch 27/30

60000/60000 [==============================] - 5s 88us/sample - loss: 0.1699 - val\_loss: 0.1690

Epoch 28/30

60000/60000 [==============================] - 5s 89us/sample - loss: 0.1698 - val\_loss: 0.1694

Epoch 29/30

60000/60000 [==============================] - 5s 89us/sample - loss: 0.1696 - val\_loss: 0.1689

Epoch 30/30

60000/60000 [==============================] - 5s 89us/sample - loss: 0.1694 - val\_loss: 0.1688

<tensorflow.python.keras.callbacks.History at 0x7fe323e832b0>

***# GENERATE PREDICTIONS ON TEST DATA***

This code block generates predictions using the trained autoencoder model on the noisy test data (`noisy\_test\_data`). It uses the `predict` method to obtain reconstructed images from the autoencoder. Finally, it displays both the noisy test images and their corresponding denoised reconstructions using the `display` function for visual comparison.

predictions= autoencoder.predict(noisy\_test\_data)

display(noisy\_test\_data,predictions)

A collage of images of different shapes

Description automatically generated

***CONCLUSION:***

The autoencoder experiment demonstrated the efficacy of convolutional neural networks (CNNs) in reconstructing and denoising images using the MNIST dataset. Through preprocessing and noise introduction, the models successfully learned compact representations and produced clean reconstructions. Leveraging techniques from image classification experiments enhanced performance and robustness. This underscores autoencoders' versatility in image processing tasks, promising applications in denoising, dimensionality reduction, and feature extraction. Continued exploration and experimentation will further advance our understanding and utilization of these powerful neural network models in real-world scenarios.