NITTE (Deemed to be University) | NMAM INSTITUTE OF TECHNOLOGY

Nitte(DU) established under Section 3 of UGC Act 1956 | Accredited with ‘A+’ Grade by NAAC

Department of Artificial Intelligence & Machine Learning Engineering

LAB MANUAL

Artificial Intelligence and Computer Vision Lab 21AM505

Academic Year

2023-2024

ARTIFICIAL INTELLIGENCE AND COMPUTER VISION LAB

(Effective from the academic year 2023 -2024)

SEMESTER – V

Course Code	21AM505	**CIE Marks**	50
Number of Contact Hours/Week	0:0:2	**SEE Marks**	50
Total Number of Contact Hours	26	**Exam Hours**	03

Credits – 1

PART-A

1. Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem
2. Implement and Demonstrate Best First Search Algorithm on any AI problem
3. Implement AO* Search algorithm.
4. Solve 8-Queens Problem with suitable assumptions
5. Implementation of TSP using heuristic approach
6. Implementation of the problem-solving strategies: either using Forward Chaining or Backward Chaining
7. Implement K- means algorithm.
8. Implement K- nearest neighbour algorithm
9. Implement SVM

PART-B

1. Write a program in python to demonstrate working with images and videos using OpenCV.
2. Write a program in python to demonstrate Bitwise Operations on Binary Images using OpenCV.
3. Write a program in python to Draw different geometric shapes and to write text on images using OpenCV.
4. Write a program in python to perform different Morphological operations on images based on OpenCV
5. Implement different Thresholding techniques, Edge detection and Contour detection on images using openCV.
6. Demonstrate Haar feature-based cascade classifiers for Face and Eye Detection on images.
7. Develop a classification model using YOLO object detection algorithm using OpenCV.
8. Write a program in python to demonstrate Handwritten Digit Recognition on MNIST dataset.

1. Implement and Demonstrate Depth First Search Algorithm on Water Jug Problem

Program:
```
def water_jug_dfs(jug1_capacity, jug2_capacity, target_capacity):
    def dfs(jug1, jug2, path):
        if jug1 == target_capacity or jug2 == target_capacity:
            print("Solution found:", path)
            return

        # Fill jug1
        if jug1 < jug1_capacity:
            new_jug1 = jug1_capacity
            new_jug2 = jug2
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Fill Jug1\n")

        # Fill jug2
        if jug2 < jug2_capacity:
            new_jug1 = jug1
            new_jug2 = jug2_capacity
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Fill Jug2\n")

        # Pour water from jug1 to jug2
        if jug1 > 0 and jug2 < jug2_capacity:
            pour_amount = min(jug1, jug2_capacity - jug2)
            new_jug1 = jug1 - pour_amount
            new_jug2 = jug2 + pour_amount
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Pour Jug1 into 
Jug2\n")

        # Pour water from jug2 to jug1
        if jug2 > 0 and jug1 < jug1_capacity:
            pour_amount = min(jug2, jug1_capacity - jug1)
            new_jug1 = jug1 + pour_amount
            new_jug2 = jug2 - pour_amount
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Pour Jug2 into 
Jug1\n")

        # Empty jug1
        if jug1 > 0:
            new_jug1 = 0
            new_jug2 = jug2
            if (new_jug1, new_jug2) not in visited:
```

```
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Empty Jug1\n")

        # Empty jug2
        if jug2 > 0:
            new_jug1 = jug1
            new_jug2 = 0
            if (new_jug1, new_jug2) not in visited:
                visited.add((new_jug1, new_jug2))
                dfs(new_jug1, new_jug2, path + f"Empty Jug2\n")

    visited = set()
    dfs(0, 0, "")

# Example usage
jug1_capacity = 4
jug2_capacity = 3
target_capacity = 2

water_jug_dfs(jug1_capacity, jug2_capacity, target_capacity)
```

Output:

```
Solution Found: Fill Jug1
Fill Jug2
Empty Jug1
Pour Jug2 to Jug1
Fill Jug2
Pour Jug2 to Jug1

Solution Found: Fill Jug1
Pour Jug1 to Jug2
Empty Jug2
Pour Jug1 to Jug2
Fill Jug1
Pour Jug1 to Jug2
```

2. Implement and Demonstrate Best First Search Algorithm on any AI problem

Program:

```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]

# Function For Implementing Best First Search
# Gives output path having lowest cost


def best_first_search(actual_Src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True

    while pq.empty() == False:
        u = pq.get()[1]
        # Displaying the path having lowest cost
        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()

# Function for adding edges to graph

 def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))


# The nodes shown in above example(by alphabets) are
# implemented using integers addedge(x,y,cost);
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
```

```
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
best_first_search(source, target, v)
```

Output:

```
 0 1 3 2 8 9
```

3. Implement AO* Search algorithm.

Program:

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):
#instantiate graph object with graph topology, heuristic values,
start node
        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}

    def applyAOStar(self): # starts a recursive AO* algorithm
        self.aoStar(self.start, False)

    def getNeighbors(self, v): # gets the Neighbors of a given node
        return self.graph.get(v,'')

    def getStatus(self,v): # return the status of a given node
        return self.status.get(v,0)

    def setStatus(self,v, val): # set the status of a given node
        self.status[v]=val

    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0) # always return the heuristic value
of a given node

    def setHeuristicNodeValue(self, n, value):
        self.H[n]=value # set the revised heuristic value of a
given node

    def printSolution(self):
        print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE
START NODE:",self.start)
        print("-----------------------------------------------------
---------")
        print(self.solutionGraph)
        print("-----------------------------------------------------
---------")

    def computeMinimumCostChildNodes(self, v): # Computes the
Minimum Cost of child nodes of a given node v
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True
        for nodeInfoTupleList in self.getNeighbors(v): # iterate
over all the set of child node/s
```

```
            cost=0
            nodeList=[]
            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)
            if flag==True: # initialize Minimum Cost with the cost
of first set of child node/s
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList # set
the Minimum Cost child node/s
                flag=False
            else: # checking the Minimum Cost nodes with the
current Minimum Cost
                if minimumCost>cost:
                    minimumCost=cost
                    costToChildNodeListDict[minimumCost]=nodeList #
set the Minimum Cost child node/s
        return minimumCost, costToChildNodeListDict[minimumCost] #
return Minimum Cost and Minimum Cost child node/s

    def aoStar(self, v, backTracking): # AO* algorithm for a start
node and backTracking status flag
        print("HEURISTIC VALUES :", self.H)
        print("SOLUTION GRAPH :", self.solutionGraph)
        print("PROCESSING NODE :", v)
        print("-----------------------------------------------------
-------------------------------------")
        if self.getStatus(v) >= 0: # if status node v >= 0, compute
Minimum Cost nodes of v
            minimumCost, childNodeList =
self.computeMinimumCostChildNodes(v)
            print(minimumCost, childNodeList)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))
            solved=True # check the Minimum Cost nodes of v are
solved
            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False
            if solved==True: # if the Minimum Cost nodes of v are
solved, set the current node status as solved(-1)
                self.setStatus(v,-1)
                self.solutionGraph[v]=childNodeList # update the
solution graph with the solved nodes which may be a part of
solution
            if v!=self.start: # check the current node is the start
node for backtracking the current node value
                self.aoStar(self.parent[v], True) # backtracking
the current node value with backtracking status set to true
            if backTracking==False: # check the current call is not
for backtracking
```

```
                for childNode in childNodeList: # for each Minimum
Cost child node
                    self.setStatus(childNode,0) # set the status of
child node to 0(needs exploration)
                    self.aoStar(childNode, False) # Minimum Cost
child node is further explored with backtracking status as false

h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H':
7, 'I': 7, 'J': 1}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}

G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()
```

Output:

```
HEURISTIC VALUES : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1,
'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
--------------------------------------------------------------------
---------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : B
--------------------------------------------------------------------
---------------------
6 ['G']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
--------------------------------------------------------------------
---------------------
10 ['B', 'C']
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 5, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : G
--------------------------------------------------------------------
---------------------
8 ['I']
```

```
HEURISTIC VALUES : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : B
--------------------------------------------------------------------
---------------------
8 ['H']
HEURISTIC VALUES : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}
PROCESSING NODE : A
--------------------------------------------------------------------
---------------------
12 ['B', 'C']
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 8, 'H': 7, 'I': 7, 'J': 1}
SOLUTION GRAPH : {}

PROCESSING NODE : I
--------------------------------------------------------------------
---------------------
0 []
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 8, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': []}
PROCESSING NODE : G
--------------------------------------------------------------------
---------------------
1 ['I']
HEURISTIC VALUES : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I']}
PROCESSING NODE : B
--------------------------------------------------------------------
---------------------
2 ['G']
HEURISTIC VALUES : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F':
1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A
--------------------------------------------------------------------
---------------------
6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1,
'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : C
--------------------------------------------------------------------
---------------------
2 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1,
'G': 1, 'H': 7, 'I': 0, 'J': 1}
```

```
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
6 ['B', 'C']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE : J
-----------------------------------------------------------------------------------------
0 []
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE : C
-----------------------------------------------------------------------------------------
1 ['J']
HEURISTIC VALUES : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0}
SOLUTION GRAPH : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE : A
-----------------------------------------------------------------------------------------
5 ['B', 'C']
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
```

4. Solve 8-Queens Problem with suitable assumptions

Program:

```
# Taking number of queens as input from user
print ("Enter the number of queens")
N = int(input())
# here we create a chessboard
# NxN matrix with all elements set to 0
board = [[0]*N for _ in range(N)]
def attack(i, j):
    #checking vertically and horizontally
    for k in range(0,N):
        if board[i][k]==1 or board[k][j]==1:
            return True
    #checking diagonally
    for k in range(0,N):
        for l in range(0,N):
            if (k+l==i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return True
    return False
def N_queens(n):
    if n==0:
        return True
    for i in range(0,N):
        for j in range(0,N):
            if (not(attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1
                if N_queens(n-1)==True:
                    return True
                board[i][j] = 0
    return False
N_queens(N)
for i in board:
    print (i)
```

Output:

```
Enter the Number of queeens: 8
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
```

5. Implementation of TSP using heuristic approach

Program:

```
import math
# Define a function to calculate the Euclidean distance between
two points
def distance(point1, point2):
    return math.sqrt((point1[0] - point2[0])**2 + (point1[1] -
point2[1])**2)

# Define the Nearest Neighbor algorithm
def nearest_neighbor(points):
    n = len(points)
    unvisited = set(range(n))
    tour = [0]  # Start from the first point
    unvisited.remove(0)

    while unvisited:
        current_point = tour[-1]
        nearest_point = min(unvisited, key=lambda x:
distance(points[current_point], points[x]))
        tour.append(nearest_point)
        unvisited.remove(nearest_point)

    # Complete the tour by returning to the starting point
    tour.append(tour[0])

    return tour

# Example usage
if __name__ == "__main__":
    # Define the points as (x, y) coordinates
    points = [(0, 0), (1, 2), (2, 3), (3, 4),(4,2)]

    # Find the tour using the Nearest Neighbor algorithm
    tour = nearest_neighbor(points)

    print("Optimal Tour:", tour)
```

Output:

```
Optimal Tour: [0, 1, 2, 3, 4, 0]
```

6. **Implementation of the problem-solving strategies: either using Forward Chaining or Backward Chaining**

Forward Chaining Program:

```
class Rule:
    def __init__(self, antecedents, consequent):
        self.antecedents = antecedents
        self.consequent = consequent

class KnowledgeBase:
    def __init__(self):
        self.facts = set()
        self.rules = []

    def add_fact(self, fact):
        self.facts.add(fact)

    def add_rule(self, rule):
        self.rules.append(rule)

    def apply_forward_chaining(self):
        new_facts_derived = True
        while new_facts_derived:
            new_facts_derived = False
            for rule in self.rules:
                if all(antecedent in self.facts for antecedent in
rule.antecedents) and rule.consequent not in self.facts:
                    self.facts.add(rule.consequent)
                    new_facts_derived = True

if __name__ == "__main__":
    kb = KnowledgeBase()

    # Define rules and facts
    rule1 = Rule(["A", "C"], "E")
    rule2 = Rule(["A", "E"], "G")
    rule3 = Rule(["B"], "E")
    rule4 = Rule(["G"], "D")
    kb.add_rule(rule1)
    kb.add_rule(rule2)
    kb.add_rule(rule3)
    kb.add_rule(rule4)
    kb.add_fact("A")
    kb.add_fact("C")
    # Apply forward chaining
    kb.apply_forward_chaining()
    # Print the derived facts
    print("Derived Facts:", kb.facts)
```

Output:

```
Derieved Facts: {'A', 'C', 'D', 'E', 'G'}
```

Backward Chaining Program:

```
# Define the knowledge base as a dictionary of rules
knowledge_base = {
    "rule1": {
        "if": ["A", "B"],
        "then": "C"
    },
    "rule2": {
        "if": ["D"],
        "then": "A"
    },
    "rule3": {
        "if": ["E"],
        "then": "B"
    },
    "rule4": {
        "if": ["F"],
        "then": "D"
    },
    "rule5": {
        "if": ["G"],
        "then": "E"
    }
}
# Define a function to perform backward chaining
def backward_chaining(goal, known_facts):
    if goal in known_facts:
        return True
    for rule, value in knowledge_base.items():
        if goal in value["if"]:
            all_conditions_met = all(condition in known_facts for
condition in value["if"])
            if all_conditions_met and
backward_chaining(value["then"], known_facts):
                return True
    return False
# Define the goal and known facts
goal = "C"
known_facts = ["G", "F", "E"]

# Check if the goal can be reached using backward chaining
if backward_chaining(goal, known_facts):
    print(f"The goal '{goal}' can be reached.")
else:
    print(f"The goal '{goal}' cannot be reached.")
```

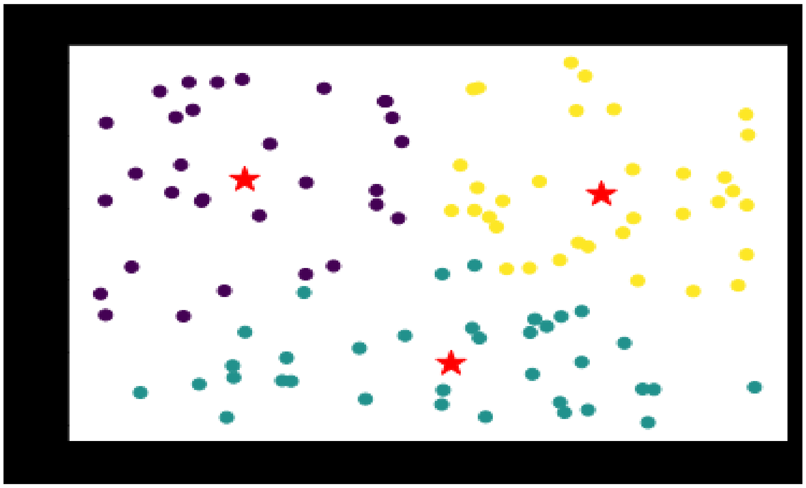
Output:

The Goal'C' cannot be Reached

7. Implement K- means algorithm.

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Generate some sample data for clustering
np.random.seed(0)
X = np.random.rand(100, 2)

# Number of clusters (k)
k = 3
# Initialize the KMeans model
kmeans = KMeans(n_clusters=k)

# Fit the model to the data
kmeans.fit(X)
# Get cluster centers and labels
cluster_centers = kmeans.cluster_centers_
labels = kmeans.labels_
# Plot the data points and cluster centers
plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1],
marker='x', s=200, color='red')
plt.title(f'K-Means Clustering (k={k})')
plt.show()
```

Output:



8. Implement K- nearest neighbour algorithm

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Generate some sample data for classification
np.random.seed(0)
X = np.random.rand(100, 2)  # Feature matrix
y = np.random.choice([0, 1], size=100)  # Target vector (binary
classification)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create a K-Nearest Neighbors classifier with k=3
k = 3
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Fit the classifier to the training data
knn_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn_classifier.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Output:

```
Accuracy: 85.00%
```

9. Implement SVM

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Create a synthetic dataset for classification (you can replace
this with your own dataset)
X, y = datasets.make_classification(n_samples=500, n_features=3,
n_informative=2, n_redundant=0, random_state=42)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create an SVM classifier
svm_classifier = SVC(kernel='linear', C=1.0)

# Train the SVM classifier on the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = svm_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Output:

```
Accuracy: 90.00%
```

PART B

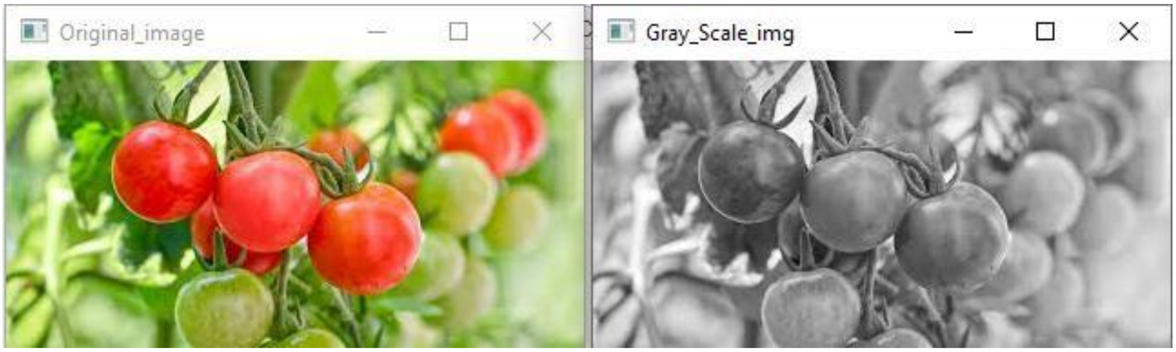
1. **Write a program in python to demonstrate working with images and videos using OpenCV.**

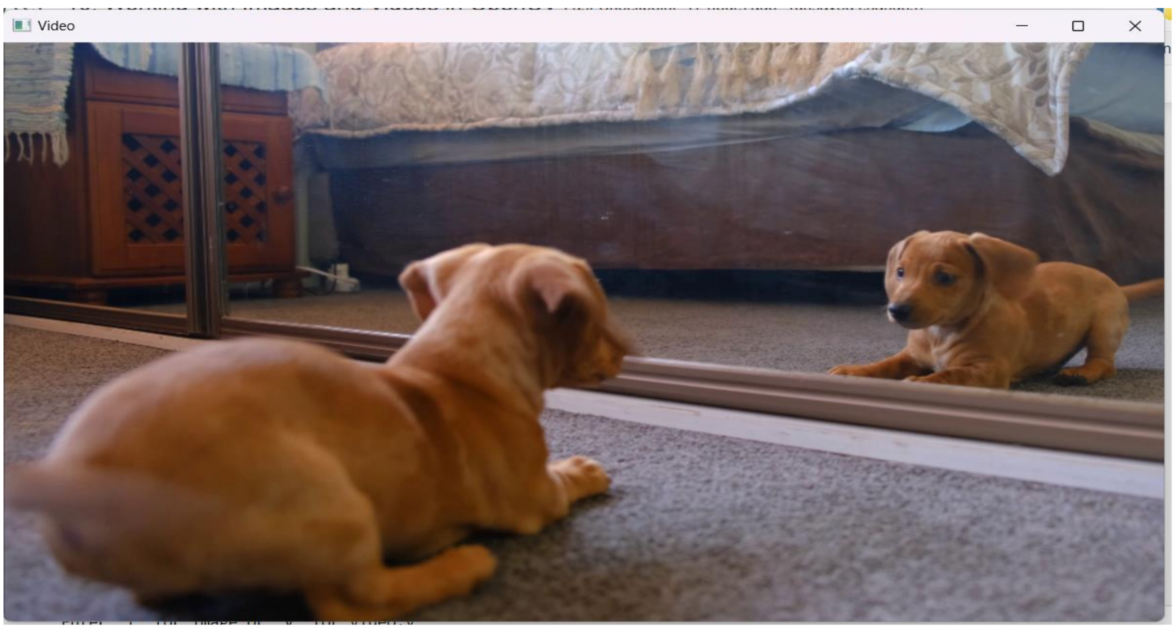
Program:

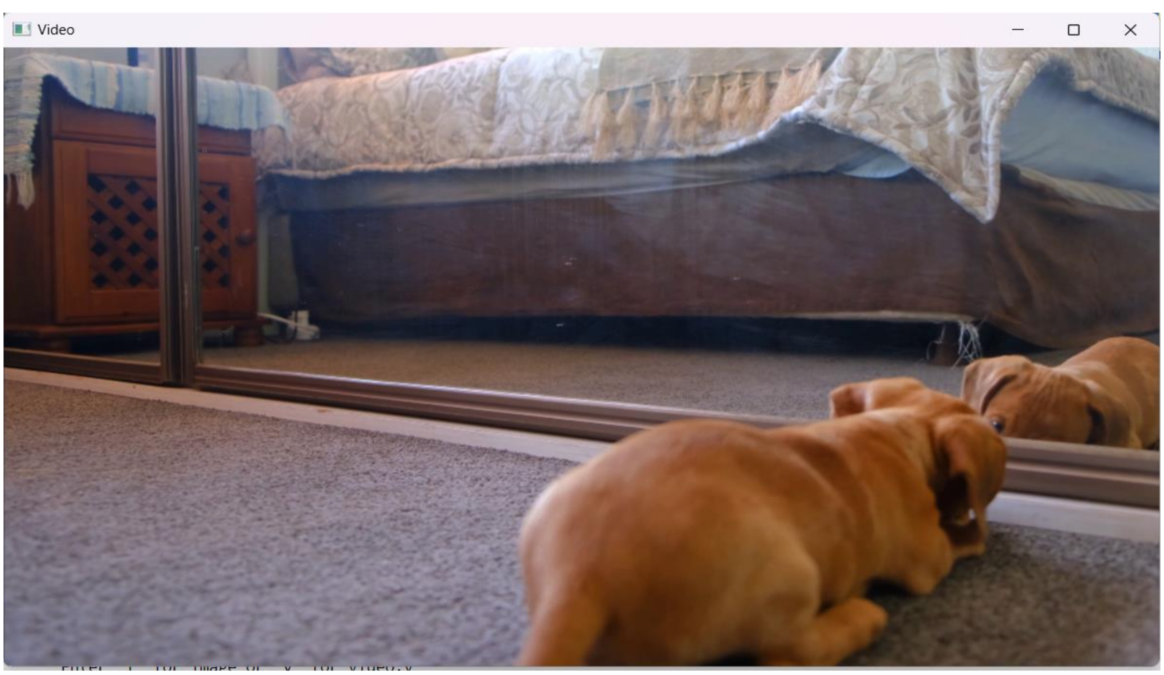
```
import cv2
#Function to process images
def process_image(image):
  #convert the image to grayscale
  gray=cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
  #Display the original image
  cv2.imshow("original image",image)


  #Display the grayscale image
  cv2.imshow("Grayscale image",gray)
  #wait for a key press and close the window
  cv2.waitKey(0)
  cv2.destroyAllWindows()

def process_video(video_path):
  #open a video capture object
  Cap=cv2.VideoCapture(video_path)
  while cap.isOpened():
       ret,frame=cap.read()
       frame=cv2.resize(frame,(960,540))
       #Display the video frame
       cv2.imshow("Vi the video capturedeo",frame)
       #Press 'q' to exit the video playback
       if cv2.waitKey(250) & 0xFF==ord('q'):
            Break
       #Release the video capture object and close all the
windows

cap.release()
cv2.destroyAllwindows()

#Choose whether to process an image or video

choice=input("Enter 'I' for image or 'V' for video:")

if choice.lower()=='I':
  #Load an image
  image=cv2.imread('image.jpg')
  process_image(image)
elif choice.lower()=='v':
  #Load a video
  video_path='video.mp4'
  process_video(video_path)
else:
```

```
print(“invalid choice. Please enter ‘I’ or ‘V’)
```

Output:







2. **Write a program in python to demonstrate Bitwise Operations on Binary Images using OpenCV.**

Program:

```
import cv2
import numpy as np

# images are loaded with imread command
img1 = cv2.imread('input1.jpg)
img2 = cv2.imread('input2.jpg')

dest_and = cv2.bitwise_and(img2, img1, mask = None)
dest_or = cv2.bitwise_or(img2, img1, mask = None)
dest_not1 = cv2.bitwise_not(img1, mask = None)
dest_not2 = cv2.bitwise_not(img2, mask = None)
dest_xor = cv2.bitwise_xor(img2, img1,mask = None)

# the window showing input image
cv2.imshow('Image 1', img1)
cv2.imshow('Image 2', img2)
# the window showing output image
cv2.imshow('AND', dest_and)
cv2.imshow('OR', dest_or)
cv2.imshow('NOT OF IMAGE 1', dest_not1)
cv2.imshow('NOT OF IMAGE 2', dest_not2)
cv2.imshow('XOR', dest_xor)
 cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:

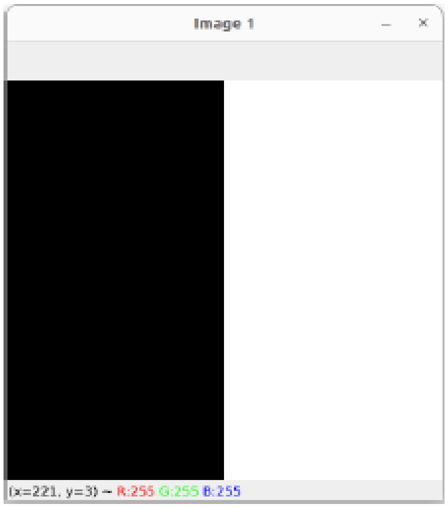


Image 1
(x=221, y=3) ~ R:255 G:255 B:255

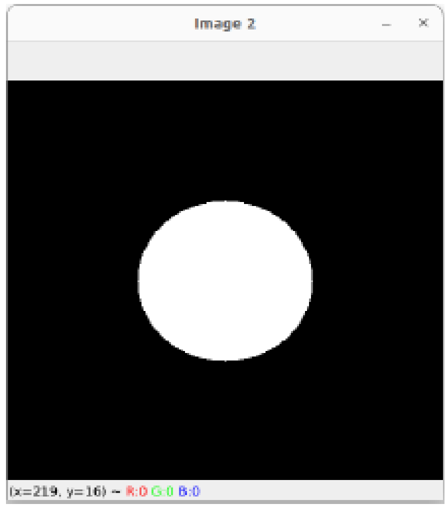
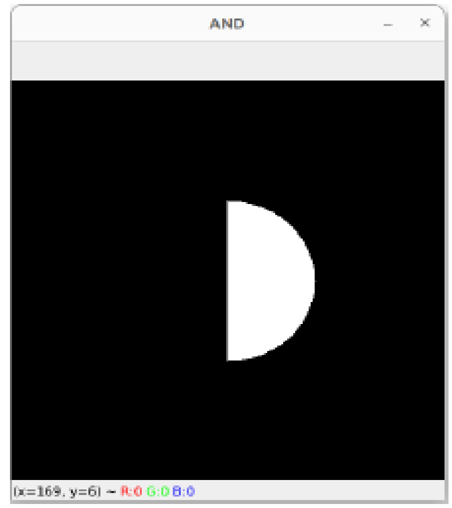
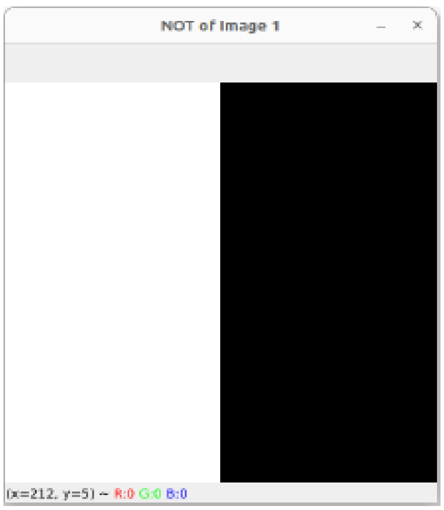
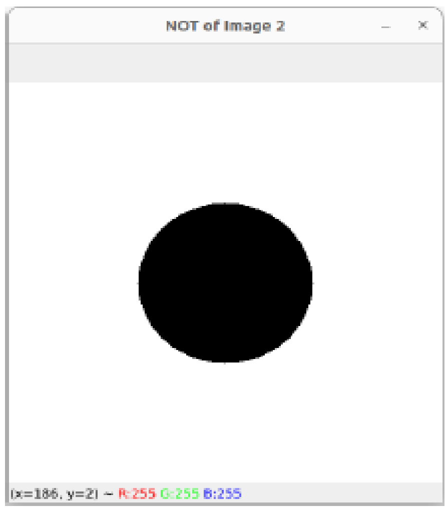
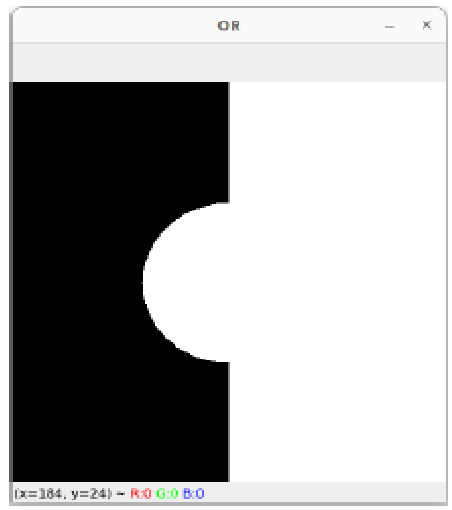


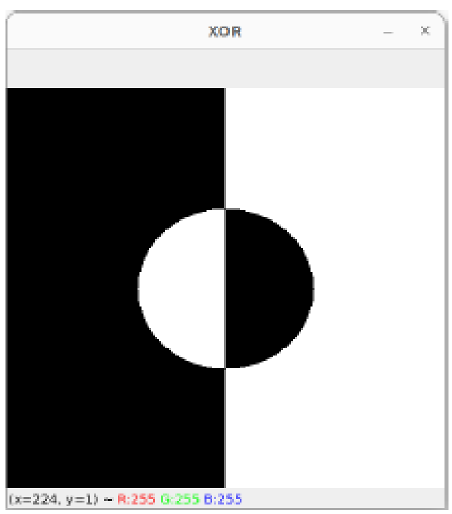
Image 2
(x=219, y=16) ~ R:0 G:0 B:0


AND
(x=169, y=6) ~ R:0 G:0 B:0


NOT of Image 1
(x=212, y=5) ~ R:0 G:0 B:0


NOT of Image 2
(x=186, y=2) ~ R:255 G:255 B:255


OR
(x=184, y=24) ~ R:0 G:0 B:0

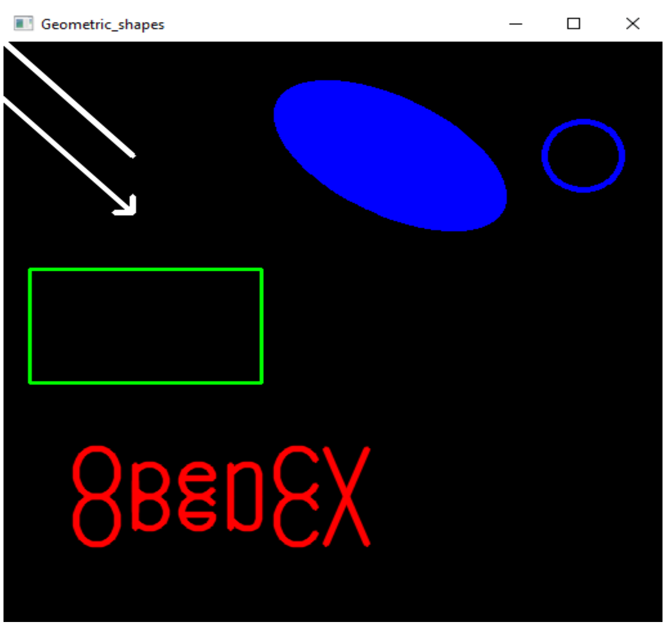

XOR
(x=224, y=1) ~ R:255 G:255 B:255

3. **Write a program in python to Draw different geometric shapes and to write text on images using OpenCV.**

```
import numpy
as np import
cv2

# Creating a black screen image using
nupy.zeros function Img = np.zeros((512, 512,
3), dtype='uint8')

# Using cv2.line() method to draw a diagonal white line with
thickness of 4px
Img =cv2.line(Img, (0,0), (100,100), (255,255,255),4)

# Using cv2.arrowedLine() method Draw a diagonal
arrow line
Img = cv2.arrowedLine(Img, (0, 50), (100, 150), (255, 255, 255),
4)

# Using cv2.ellipse() method
#center_coordinates = (300, 100)
#axesLength = (100, 50)
#angle = 30
#startAngle = 0
#endAngle = 360
#color = (255, 0, 0)
#thickness = -1
Img = cv2.ellipse(Img, (300, 100), 100, 50), 30,0,360, (255, 0,
0), -1)

# Using cv2.circle() method
Img =cv2.circle(Img, (450,100), 30, color, (255, 0, 0),4)

# Using cv2.rectangle() method
# Draw a rectangle with green line borders of
thickness of 2 px Img = cv2.rectangle(Img, (20, 200),
(200, 300), (0, 255, 0),2)

# Using cv2.putText()
method font =
cv2.FONT_HERSHEY_SIMPLEX
org = (50, 400)
fontScale = 2
color = (0, 0, 255)
thickness = 3
Img = cv2.putText(Img, 'OpenCV', org, font,fontScale, color,
```

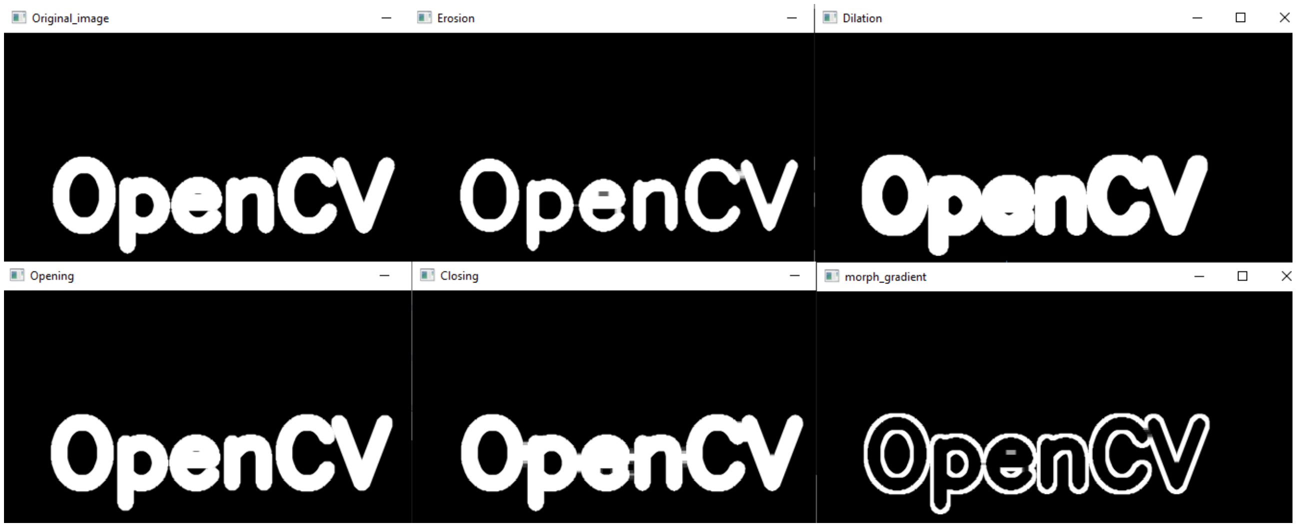
```
thickness,cv2.LINE_AA, False)
Img = cv2.putText(Img, 'OpenCV', org, font,fontScale, color,
thickness,cv2.LINE_AA, True)

# Displaying the image
cv2.imshow('Geometric_shapes', Img)

cv2.waitKey(0)
cv2.destroyAllWindows(
)
```

Output:



4. **Write a program in python to perform different Morphological operations on images based on OpenCV**

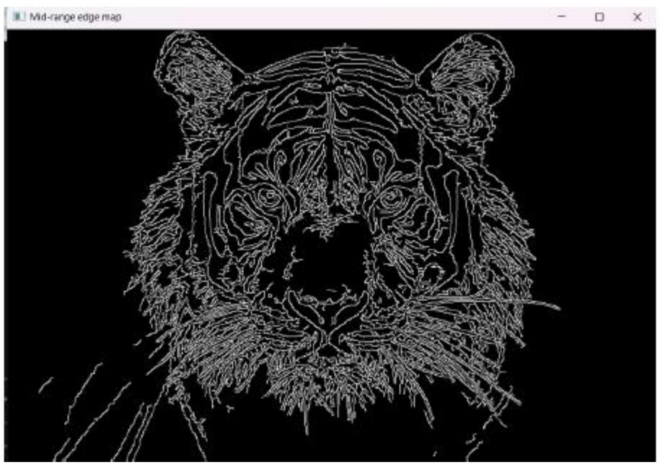
```
import cv2
import numpy as np

img = cv.imread(morph.jpg)


# Creating kernel
kernel =np.ones((5, 5), np.uint8)


# Using cv2.erode() method
img_erosion =cv2.erode(Img, kernel,
iterations=1)# Using cv2.dilate()
method
img_dilation =cv2.dilate(Img, kernel,
iterations=1)# opening the image
img_opening = cv2.morphologyEx(Img, cv2.MORPH_OPEN,
kernel, iterations=1)# closing the image
img_closing = cv2.morphologyEx(Img, cv2.MORPH_CLOSE,
kernel, iterations=1)# use morph gradient
img_morph_gradient = cv2.morphologyEx(Img, cv2.MORPH_GRADIENT,
kernel)

# Displaying the image
cv2.imshow('Original_image', Img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Dilation', img_dilation)
cv2.imshow('Opening', img_opening)
cv2.imshow('Closing', img_closing)
cv2.imshow('morph_gradient',
img_morph_gradient)

cv2.waitKey(0)
cv2.destroyAllWindo
ws()
```

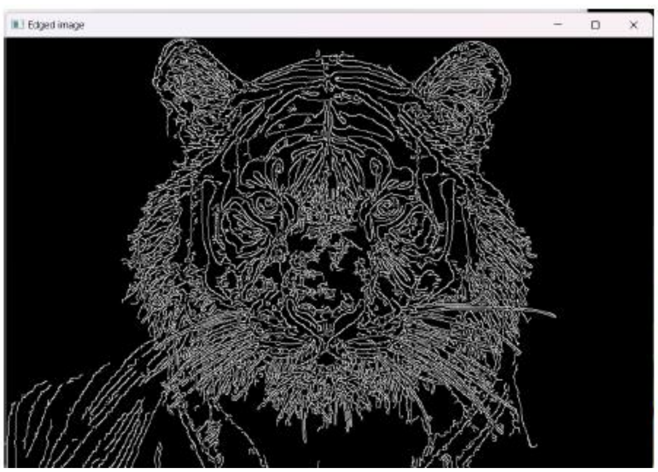
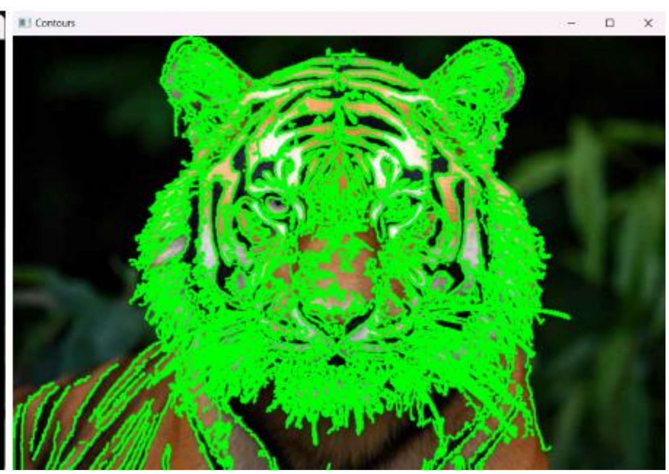

Original_image
Erosion
Dilation
Opening
Closing
morph_gradient

5. **Implement different Thresholding techniques, Edge detection and Contour detection on images using openCV.**

```
import cv2

image = cv2.imread('tiger.jpg')

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

blurred = cv2.GaussianBlur(gray, (3, 3), 0)


wide_Edge = cv2.Canny(blurred, 10, 100)

Mid_Edge = cv2.Canny(blurred, 30, 150)

Tight_Edge = cv2.Canny(blurred, 240, 250)



cv2.imshow("Original image", image)

cv2.imshow("wide Edged image", wide_Edge)

cv2.imshow("Mid Edged image", Mid_Edge)

cv2.imshow("Tight Edged image", Tight_Edge)

cv2.waitKey(0)


contours,_=cv2.findContours(edged,cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

image_copy = image.copy()

# draw the contours on a copy of the original image

cv2.drawContours(image_copy, contours, -1, (0, 255, 0), 2)

print(len(contours), "objects were found in this image.")


cv2.imshow("Edged image", edged)

cv2.imshow("contours", image_copy)

cv2.waitKey(0)
```


Original image

Wide range edge map

Mid-range edge map

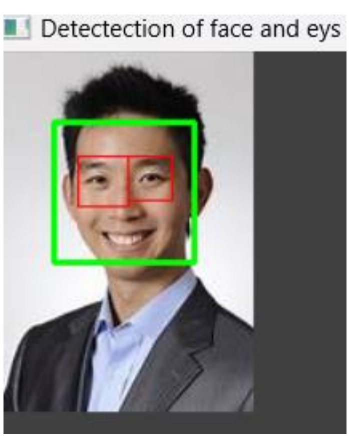
Tight range edge map

Edged image

Contours

6. **Demonstrate Haar feature-based cascade classifiers for Face and Eye Detection on images.**

<u>Program:</u>

```
import cv2

# Reading the image
img = cv2.imread('img1.jpg')
# Converting image to grayscale
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Loading the required haar-cascade xml classifier file
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades
+'Haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades
+'haarcascade_eye.xml')


# Applying the face detection method on the grayscale image
faces_rect =face_cascade.detectMultiScale(gray_img, 1.1, 9)
# Iterating through rectangles of detected faces
for (x, y, w, h) in faces_rect:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Applying the eye detection method on the grayscale image
eyes_rect =eye_cascade.detectMultiScale(gray_img, 1.1, 1)

for (x, y, w, h) in eyes_rect:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 1)

cv2.imshow('Detectection of face and eys', img)
cv2.waitKey(0)
```

<u>Output:</u>



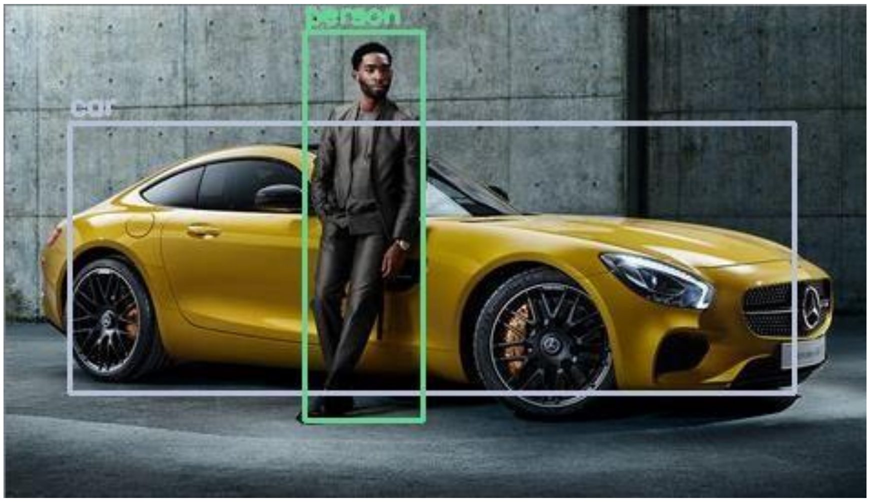
7. Develop a classification model using YOLO object detection algorithm using OpenCV.

Program:

```
import cv2
import numpy as
np# Load Yolo
print("LOADING YOLO")
net = cv2.dnn.readNet("yolov3.weights",
"yolov3.cfg")#save all the names in file o
the list classes classes = []
with open("coco.names", "r") as f:
     classes = [line.strip() for line in
f.readlines()]#get layers of the network
layer_names = net.getLayerNames()
#Determine the output layer names from the YOLO model
output_layers = [layer_names[i - 1] for i in
net.getUnconnectedOutLayers()]print("YOLO LOADED")
# Capture frame-by-frame
img = cv2.imread("test_img.jpg")
#img = cv2.resize(img, None, fx=0.4,
fy=0.4)height, width, channels =
img.shape
# USing blob function of opencv to preprocess image
blob = cv2.dnn.blobFromImage(img, 1 / 255.0, (416,
416),swapRB=True,crop=False)
#Detecting
objects
net.setInput(bl
ob)
outs = net.forward(output_layers)
# Showing informations on the
screenclass_ids = []
confidences =
[]boxes = []
for out in outs:
     for detection in out:
          scores =
          detection[5:]
          class_id =
          np.argmax(scores)
          confidence =
          scores[class_id]if
          confidence > 0.5:
               # Object detected
```

```
                center_x = int(detection[0] *
                width) center_y =
                int(detection[1] * height) w =
                int(detection[2] * width)

                h = int(detection[3] *
                height)# Rectangle
                coordinates
                x = int(center_x - w
                / 2) y = int(center_y
                - h / 2)
                boxes.append([x, y,
                w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)
#We use NMS function in opencv to perform Non-maximum
Suppression#we give it score threshold and nms
threshold as arguments. indexes =
cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
colors = np.random.uniform(0, 255,
size=(len(classes), 3))for i in range(len(boxes)):
     if i in indexes:
          x, y, w, h = boxes[i]
          label = str(classes[class_ids[i]])

          color = colors[class_ids[i]]
          cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
          cv2.putText(img, label, (x, y -
          5),cv2.FONT_HERSHEY_SIMPLEX,1/2,color, 2)
               cv2.imshow("Image",img) cv2.waitKey(0)
```

Output:



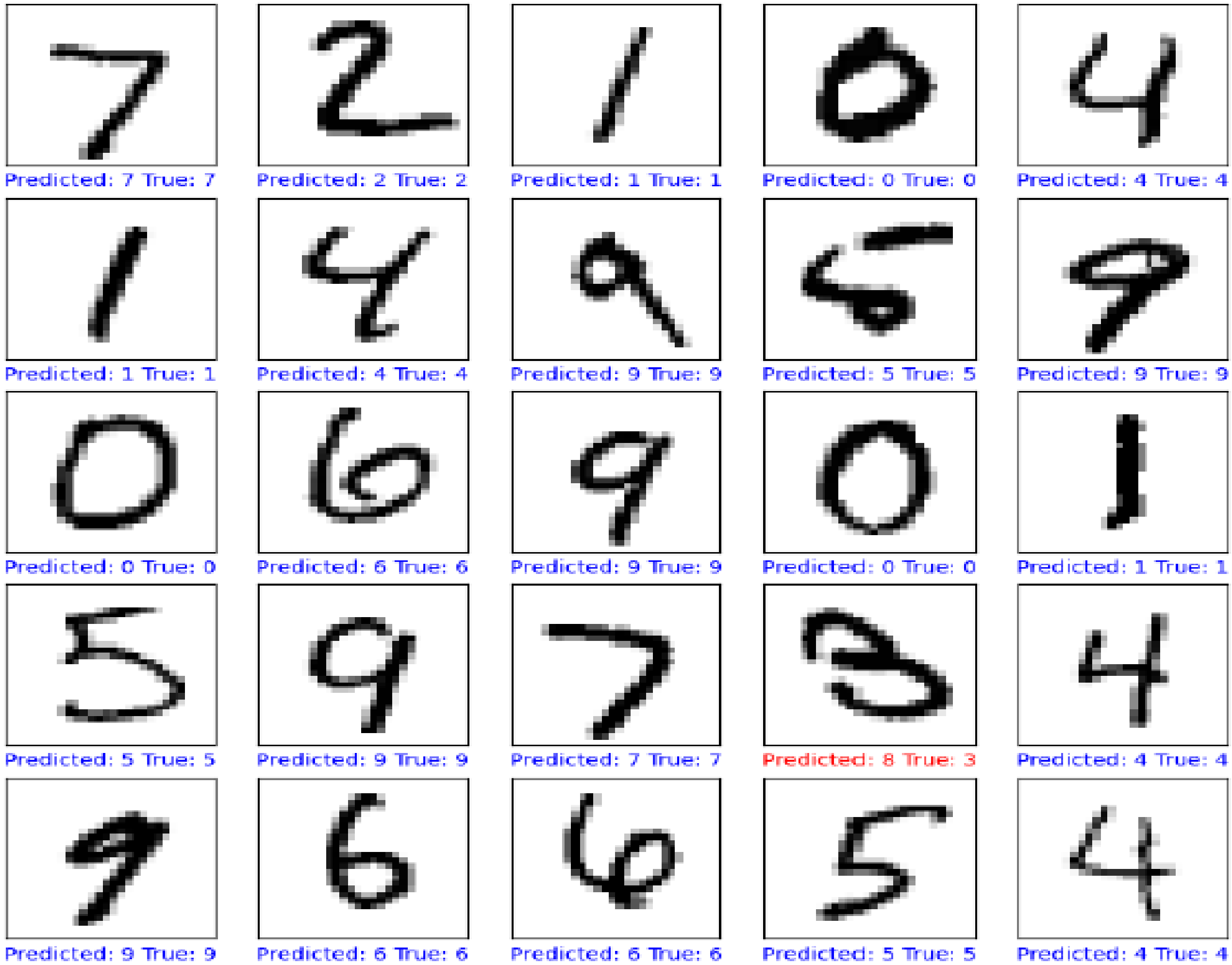
8. Write a program in python to demonstrate Handwritten Digit Recognition on MNIST dataset.

Program:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images /
255.0

encode the labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Build the neural network model
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))
```

```
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=5)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Make predictions on some test images
predictions = model.predict(test_images)

# Display the first few test images and their predicted labels
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(test_images[i], cmap=plt.cm.binary)
    predicted_label = predictions[i].argmax()
    true_label = test_labels[i].argmax()
    color = 'blue' if predicted_label == true_label else 'red'
    plt.xlabel(f'Predicted: {predicted_label} True: {true_label}',
color=color)

plt.show()
```

Output:


Predicted: 7 True: 7
Predicted: 2 True: 2
Predicted: 1 True: 1
Predicted: 0 True: 0
Predicted: 4 True: 4
Predicted: 1 True: 1
Predicted: 4 True: 4
Predicted: 9 True: 9
Predicted: 5 True: 5
Predicted: 9 True: 9
Predicted: 0 True: 0
Predicted: 6 True: 6
Predicted: 9 True: 9
Predicted: 0 True: 0
Predicted: 1 True: 1
Predicted: 5 True: 5
Predicted: 9 True: 9
Predicted: 7 True: 7
Predicted: 8 True: 3
Predicted: 4 True: 4
Predicted: 9 True: 9
Predicted: 6 True: 6
Predicted: 6 True: 6
Predicted: 5 True: 5
Predicted: 4 True: 4