**Instruction**

* **Necessary file will be in the folder with the experiment number**

**1)Implementation of Noise removal algorithms using OpenCV**

import cv2

from matplotlib import pyplot as plt

img=cv2.imread(‘C:/Users/thara/Downloads/bear.jpeg’)

import numpy as np

dst = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 15)

plt.subplot(121), plt.imshow(img)

plt.subplot(122), plt.imshow(dst)

plt.show()

**2)Implementation of Object detection based on Edge detection algorithms on any application using OpenCV**

import cv2

import numpy as np

# Read the image

original\_img = cv2.imread('C:/Users/thara/Downloads/bear.jpeg', cv2.IMREAD\_COLOR)

# Check if the image was successfully loaded

if original\_img is None:

print("Error: Unable to load image.")

else:

# Convert to grayscale

gray = cv2.cvtColor(original\_img, cv2.COLOR\_BGR2GRAY)

# Apply Gaussian Blur

blur\_img = cv2.GaussianBlur(gray, (3, 3), 0)

# Detect edges using Canny

edges = cv2.Canny(image=blur\_img, threshold1=100, threshold2=200)

# Display the results

cv2.imshow('Original Image', original\_img)

cv2.imshow('Edges', edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

**3) Implementation of Perspective projection of the lane borders using OpenCV**

from google.colab.patches import cv2\_imshow

import cv2

import numpy as np

def detect\_lane\_border(image):

    # Convert the image to grayscale

    gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    # Apply Gaussian blur to remove noise

    blurred = cv2.GaussianBlur(gray, (7, 7), 0)

    # Apply Canny edge detection

    edges = cv2.Canny(blurred, 50, 100)

    # Define region of interest (ROI)

    height, width = edges.shape

    mask = np.zeros\_like(edges)

    region\_of\_interest\_vertices = [

        (0, height),

        (width / 2, height / 2),

        (width, height)

    ]

    cv2.fillPoly(mask, [np.array(region\_of\_interest\_vertices, np.int32)], 255)

    masked\_edges = cv2.bitwise\_and(edges, mask)

    # Detect lines using Hough transform

    lines = cv2.HoughLinesP(masked\_edges, rho=1, theta=np.pi/180, threshold=50, minLineLength=50, maxLineGap=100)

    # Draw the detected lines on the original image

    if lines is not None:

        for line in lines:

            x1, y1, x2, y2 = line[0]

            cv2.line(image, (x1, y1), (x2, y2), (255, 0, 0), 5)

    return image

# Read the input image

input\_image = cv2.imread('/content/lane.jpeg')

# Detect lane borders

output\_image = detect\_lane\_border(input\_image)

# Display the result

cv2\_imshow(output\_image)

**4) Implementation of Principal Component Analysis**

import pandas as pd

import numpy as np

data=pd.read\_csv("/content/drive/MyDrive/cv lab experiment/wine.csv")

data.head()

data.isnull().sum()

X = data.iloc[:, :-1].values

y = data.iloc[:, -1].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.svm import SVC

cls=SVC()

cls.fit(X\_train,y\_train)

from sklearn.metrics import confusion\_matrix, accuracy\_score

y\_pred = cls.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)

from sklearn.decomposition import PCA

pca = PCA(n\_components = 2)

X\_train = pca.fit\_transform(X\_train)

X\_test = pca.transform(X\_test)

from sklearn.svm import SVC

cls=SVC()

cls.fit(X\_train,y\_train)

from sklearn.metrics import confusion\_matrix, accuracy\_score

y\_pred = cls.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)

**5) Implementation of Feature Extraction of an object using SIFT in OpenCV**

import cv2

import matplotlib.pyplot as plt

image = cv2.imread("/content/bear.jpeg")

img = cv2.imread('/content/bear.jpeg')

gray= cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

sift = cv2.SIFT\_create()

kp = sift.detect(gray, None)

img=cv2.drawKeypoints(gray ,kp ,img ,flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)

plt.figure(figsize=(25, 7))

plt.subplot(121).imshow(image)

plt.subplot(122).imshow(img)

plt.show()

**6) Implementation of Feature Extraction of an object using SURF in OpenCV**

from google.colab.patches import cv2\_imshow

import cv2

import numpy as np

# Load images

img\_scene = cv2.imread("/content/surf\_.png", cv2.IMREAD\_GRAYSCALE)  # Image containing the scene

img\_object = cv2.imread("/content/surf\_img.jpeg", cv2.IMREAD\_GRAYSCALE)  # Image of the object to be detected

# Check if the images are loaded successfully

if img\_scene is None:

    print("Error: Unable to load scene image.")

elif img\_object is None:

    print("Error: Unable to load object image.")

else:

    # Create ORB detector

    orb = cv2.ORB\_create()

    # Detect keypoints and descriptors for the object and scene images

    keypoints\_object, descriptors\_object = orb.detectAndCompute(img\_object, None)

    keypoints\_scene, descriptors\_scene = orb.detectAndCompute(img\_scene, None)

    # Create a Brute-Force Matcher object

    bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)

    # Match descriptors of the object and scene images

    matches = bf.match(descriptors\_object, descriptors\_scene)

    # Sort the matches based on distance

    matches = sorted(matches, key=lambda x: x.distance)

    # Draw the first 10 matches

    img\_matches = cv2.drawMatches(img\_object, keypoints\_object, img\_scene, keypoints\_scene, matches[:10], None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)

    # Display the matched keypoints

    cv2\_imshow(img\_matches)

**7) Implementation of Emotion Recognition in OpenCV**

import cv2

import numpy as np

from tensorflow.keras.models import load\_model

# Load the trained model

model = load\_model("C:/Users/thara/Downloads/model.h5")

# Prevent OpenCL usage and unnecessary logging messages

cv2.ocl.setUseOpenCL(False)

# Dictionary mapping class labels with corresponding emotions

emotion\_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4: "Neutral", 5: "Sad", 6: "Surprised"}

# Start the webcam feed

cap = cv2.VideoCapture(0) # You can also specify a video file path instead of 0 for webcam

while True:

# Capture frame

ret, frame = cap.read()

if not ret:

break

# Find Haar cascade to draw bounding box around face

face\_casc = cv2.CascadeClassifier('C:/Users/thara/Downloads/Emotion-detection-master/Emotion-detection-master/src/haarcascade\_frontalface\_default.xml')

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces = face\_casc.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (255, 0, 0), 2)

roi\_gray = gray[y:y + h, x:x + w]

# Resize and convert to color image

roi\_color = cv2.resize(roi\_gray, (112, 112))

roi\_color = cv2.cvtColor(roi\_color, cv2.COLOR\_GRAY2RGB)

# Normalize

roi\_color = roi\_color / 255.0

# Expand dimensions and make prediction

cropped\_img = np.expand\_dims(roi\_color, axis=0)

prediction = model.predict(cropped\_img)

maxindex = int(np.argmax(prediction))

cv2.putText(frame, emotion\_dict[maxindex], (x, y), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE\_AA)

# Display the frame

cv2.imshow('Video', cv2.resize(frame, (500, 500), interpolation=cv2.INTER\_CUBIC))

# Break the loop if 'q' is pressed

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the webcam and close OpenCV windows

cap.release()

cv2.destroyAllWindows()

**8) Implementation of Gesture Recognition in OpenCV**

import cv2

import numpy as np

# Load the pre-trained Haar Cascade hand detection model

hand\_cascade = cv2.CascadeClassifier('C:/Users/thara/Downloads/haarcascade\_hand.xml')

# Function to detect and recognize hand gestures

def detect\_gestures(frame):

# Convert the frame to grayscale

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Detect hands in the frame

hands = hand\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Draw lines and dots representing the detected hands

for (x, y, w, h) in hands:

# Calculate the center of the hand

center\_x = x + w // 2

center\_y = y + h // 2

# Draw a crosshair at the center of the hand

cv2.line(frame, (center\_x - 10, center\_y), (center\_x + 10, center\_y), (0, 255, 0), 2)

cv2.line(frame, (center\_x, center\_y - 10), (center\_x, center\_y + 10), (0, 255, 0), 2)

# Draw a circle around the hand

cv2.circle(frame, (center\_x, center\_y), max(w, h) // 2, (0, 255, 0), 2)

return frame

# Main function for capturing video from webcam

def main():

cap = cv2.VideoCapture(0)

if not cap.isOpened():

print("Error: Unable to access the webcam.")

return

while True:

ret, frame = cap.read()

if not ret:

print("Error: Failed to capture frame.")

break

# Detect and recognize gestures in the frame

frame = detect\_gestures(frame)

cv2.imshow('Gesture Recognition', frame)

if cv2.waitKey(1) & 0xFF == ord('q'): # Press 'q' to exit

break

cap.release()

cv2.destroyAllWindows()

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

main()

**9) Implementation of Face Detection in OpenCV**

import cv2

# Load the pre-trained Haar Cascade face detection model

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

# Function to detect faces in a frame

def detect\_faces(frame):

# Convert the frame to grayscale

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Detect faces in the frame

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

# Draw rectangles around the detected faces

for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

return frame

# Main function for capturing video from the camera

def main():

cap = cv2.VideoCapture(0) # Use default camera (index 0)

if not cap.isOpened():

print("Error: Unable to access the camera.")

return

while True:

ret, frame = cap.read()

if not ret:

print("Error: Failed to capture frame.")

break

# Detect faces in the frame

frame = detect\_faces(frame)

# Display the resulting frame

cv2.imshow('Face Detection', frame)

# Press 'q' to exit the video feed

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Release the camera and close all OpenCV windows

cap.release()

cv2.destroyAllWindows()

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

main()

**10) Implementation of Object detection using AdaBoost in OpenCV**

pip install opencv-python==4.5.5.64  
  
  
import cv2  
# Load pre-trained AdaBoost object detection model  
object\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')  
# Function to detect objects using AdaBoost  
def detect\_objects(frame):  
    gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
    objects = object\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))  
  
    for (x, y, w, h) in objects:  
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)  
  
    return frame  
# Main function for capturing video from webcam  
def main():  
    cap = cv2.VideoCapture(0)  
  
    while True:  
        ret, frame = cap.read()  
        if not ret:  
            break  
        frame = detect\_objects(frame)  
        cv2.imshow('Object Detection using AdaBoost', frame)  
        if cv2.waitKey(1) & 0xFF == ord('a'):  
            break  
    cap.release()  
    cv2.destroyAllWindows()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    main()

**or**

import cv2  
import numpy as np  
  
# Load YOLOv3 model and configuration files  
net = cv2.dnn.readNet('yolov3.weights', 'yolov3.cfg')  
with open('coco.names', 'r') as f:  
    classes = f.read().strip().split('\n')  
  
# Function to perform object detection on a frame  
def detect\_objects(frame):  
    # Create a blob from the input image  
    blob = cv2.dnn.blobFromImage(frame, 1/255.0, (416, 416), swapRB=True, crop=False)  
  
    # Set the input for the network and perform a forward pass  
    net.setInput(blob)  
    layer\_outputs = net.forward(net.getUnconnectedOutLayersNames())  
  
    # Process the detections  
    conf\_threshold = 0.5  
    nms\_threshold = 0.4  
    boxes = []  
    confidences = []  
    class\_ids = []  
  
    for output in layer\_outputs:  
        for detection in output:  
            scores = detection[5:]  
            class\_id = np.argmax(scores)  
            confidence = scores[class\_id]  
            if confidence > conf\_threshold:  
                center\_x = int(detection[0] \* frame.shape[1])  
                center\_y = int(detection[1] \* frame.shape[0])  
                width = int(detection[2] \* frame.shape[1])  
                height = int(detection[3] \* frame.shape[0])  
                left = int(center\_x - width / 2)  
                top = int(center\_y - height / 2)  
                boxes.append([left, top, width, height])  
                confidences.append(float(confidence))  
                class\_ids.append(class\_id)  
  
    # Apply non-maximum suppression to remove overlapping boxes  
    indices = cv2.dnn.NMSBoxes(boxes, confidences, conf\_threshold, nms\_threshold)  
  
    # Draw bounding boxes and labels on the frame  
    if len(indices) > 0:  
        for i in indices.flatten():  
            (x, y, w, h) = boxes[i]  
            label = f"{classes[class\_ids[i]]}: {confidences[i]:.2f}"  
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)  
            cv2.putText(frame, label, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)  
  
    return frame  
  
# Main function for capturing video from the camera  
def main():  
    cap = cv2.VideoCapture(0)  # Use default camera (index 0)  
  
    if not cap.isOpened():  
        print("Error: Unable to access the camera.")  
        return  
     
    while True:  
        ret, frame = cap.read()  
        if not ret:  
            print("Error: Failed to capture frame.")  
            break  
  
        # Perform object detection on the frame  
        frame = detect\_objects(frame)  
  
        # Display the resulting frame  
        cv2.imshow('Object Detection', frame)  
  
        # Press 'q' to exit the video feed  
        if cv2.waitKey(1) & 0xFF == ord('q'):  
            break  
  
    # Release the camera and close all OpenCV windows  
    cap.release()  
    cv2.destroyAllWindows()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    main()