SUBJECT CODE:**ITA0527**

SUBJECT NAME:

COMPUTER VISION

24	SHARPENING OF IMAGE USING HIGH BOOST MASKS	
25	SHARPENING OF IMAGE USING GRADIENT MASKING	
26	INSERT WATERMARK INTO THE IMAGE USING OPENCV	
27	CROPPING, COPYING AND PASTE AN IMAGE INSIDE ANOTHER IMAGE USING OPENCV	
28	FIND BOUNDARY OF AN IMAGE USING CONVOLUTIONAL KERNEL	
29	MORPHOLOGICAL OPERATION USING EROSION TECHNIQUE	
30	MORPHOLOGICAL OPERATION USING DILATE TECHNIQUE	
31	MORPHOLOGICAL OPERATION USING OPENING TECHNIQUE	
32	MORPHOLOGICAL OPERATION USING CLOSING TECHNIQUE	
33	MORPHOLOGICAL OPERATION USING MORPHOLOGICAL GRADIENT TECHNIQUE	
34	MORPHOLOGICAL OPERATION USING TOP HAT TECHNIQUE	
35	MORPHOLOGICAL OPERATION USING BLACK HAT TECHNIQUE	
36	OBJECT RECOGNITION	
37	VIDEO IN REVERSE MODE	
38	FACE DETECTION	
39	VEHICLE DETECTION IN A VIDEO FRAME	
40	DRAW RECTANGLE AND EXTRACT OBJECT	

1. Perform basic Image Handling and processing operations on the image. • Read an image in python and Convert an Image to Grayscale

AIM: To Perform Basic Operations to Read Image and Convert to Grayscale using Python

PROGRAM:

```
import cv2
from google.colab.patches import cv2_imshow
image = cv2.imread(r'/content/leaf.jpg')
if image is None:
    print("Error: Image not found.")
else:
    grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    cv2_imshow(image)
    cv2_imshow(grayscale_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

RESULT:

Gray scale conversion using python is successfully implemented.

2. Perform basic Image Handling and processing operations on the image.• Read an image in python and Convert an Image to Blur using GaussianBlur.

AIM: To Perform Basic Operations to Read Image and Convert to Blur using GaussianBlur.

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
kernel = np.ones((5,5),np.uint8)
print(kernel)
path = "/content/leaf.jpg"
img =cv2.imread(path)
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray,(7,7),0)
cv2_imshow(imgBlur)
cv2.waitKey(0)
```

INPUT:

OUTPUT:

RESULT:

The python program to convert image to blur using gaussian blur.

3. Perform basic Image Handling and processing operations on the image• Read an image in python and Convert an Image to show outline using Canny function

AIM: To Perform Basic Operations to Convert image to show outline Canny function in Python

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
kernel = np.ones((5,5),np.uint8)
print(kernel)
path = "/content/leaf.jpg"
img =cv2.imread(path)
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray,(7,7),0)
imgCanny = cv2.Canny(imgBlur,100,200)
cv2_imshow(imgCanny)
cv2.waitKey(0)
```

INPUT:

OUTPUT:

RESULT:

The python program to convert image to outline using canny function.

4. Perform basic Image Handling and processing operations on the image• Read an image in python and Dilate an Image using Dilate function

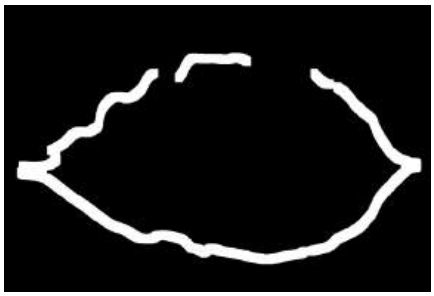
AIM: To Perform Basic Operations to Read Image and Dilate an Image using Python

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
kernel = np.ones((5,5),np.uint8)
print(kernel)
path = "/content/leaf.jpg"
img =cv2.imread(path)
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray,(7,7),0)
imgCanny = cv2.Canny(imgBlur,100,200)
imgDilation = cv2.dilate(imgCanny,kernel , iterations = 10)
imgEroded = cv2.erode(imgDilation,kernel,iterations=2)
cv2_imshow(imgEroded)
cv2.waitKey(0)
```

INPUT:

OUTPUT:



RESULT:

The python program to convert image to outline using dilate function.

5. Perform basic Image Handling and processing operations on the image• Read an image in python and Erode an Image using erode function

AIM: The Aim of the experiment is to Read an image in python and Erode an Image using erode function

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
path = "/content/leaf.jpg"
img = cv2.imread(path)
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
kernel = np.ones((5, 5), np.uint8)
imgEroded = cv2.erode(imgGray, kernel, iterations=1)
cv2_imshow(img)
cv2_imshow(imgEroded)
```

INPUT:

OUTPUT:

6. Perform basic video processing operations on the captured video• Read captured video in python and display the video, in slow motion and in fast motion.

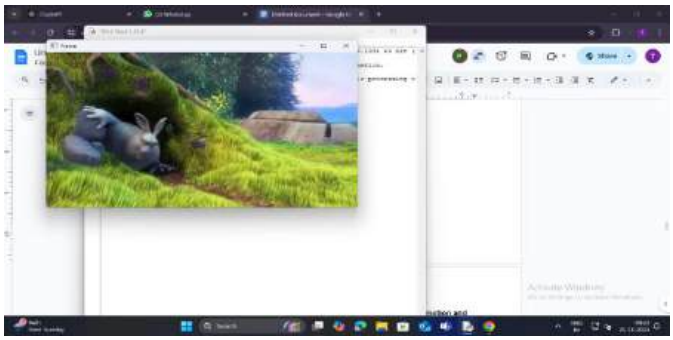
AIM: The Aim of the Experiment is to Read captured video in python and display the video, in slow motion and in fast motion

PROGRAM:

```
import cv2
import numpy as np
video_path = r"C:\Users\ADMIN\Downloads\SampleVideo_1280x720_1mb.mp4"
cap = cv2.VideoCapture(video_path)

if not cap.isOpened():
    print("Error opening video file")
else:
    while cap.isOpened():
        ret, frame = cap.read()
        if ret:
            cv2.imshow('Frame', frame)
            if cv2.waitKey(250) & 0xFF == ord('q'):
                break
        else:
            break
cap.release()
cv2.destroyAllWindows()
```

OUTPUT :



7. Capture video from web Camera and Display the video, in slow motion and in fast motion operations on the captured video

AIM: The Aim is to Capture video from web Camera and Display the video, in slow motion and in fast motion operations on the captured video

PROGRAM:

```
import cv2
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Error: Unable to access the webcam")
    exit()

print("Press 's' for slow motion, 'f' for fast motion, 'n' for normal speed, and 'q' to
quit.")
delay = 30

while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Unable to read from webcam")
        break
    cv2.imshow("Webcam Video", frame)
    key = cv2.waitKey(delay) & 0xFF

    if key == ord('q'):
        break
    elif key == ord('s'):
        delay = 100
    elif key == ord('f'):
        delay = 10
    elif key == ord('n'):
        delay = 30
cap.release()
cv2.destroyAllWindows()
```

8. Scaling an image to its Bigger and Smaller sizes.

AIM: The Aim is resize the image from bigger to smaller size

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
kernel = np.ones((5,5),np.uint8)
img = cv2.imread("/content/leaf.jpg",cv2.IMREAD_COLOR)
img = cv2.resize(img,(600,600))
cv2_imshow(img)
```

INPUT:

OUTPUT:

9.ROTATION 90 ALONG DEGREE

AIM: The Aim of the Experiment is to perform Rotation of an image along 90 degree

PROGRAM:

```
import cv2
from google.colab.patches import cv2_imshow
path = r"/content/leaf.jpg"
src = cv2.imread(path)
window_name = 'Image'
image = cv2.rotate(src, cv2.ROTATE_90_COUNTERCLOCKWISE)
cv2_imshow(image)
cv2.waitKey(0)
```

INPUT:

OUTPUT:

10.ROTATION 180 ALONG DEGREE

AIM: The Aim of the Experiment is to perform Rotation of an image along 180 degree

PROGRAM:

```
import cv2
from google.colab.patches import cv2_imshow
path=r"/content/leaf.jpg"
src = cv2.imread(path)
window_name = 'Image'
image = cv2.rotate(src, cv2.ROTATE_180)
cv2_imshow(image)
cv2.waitKey(0)
```

INPUT:

OUTPUT:

11.ROTATION 270 ALONG DEGREE

AIM: The Aim of the Experiment is to perform Rotation of an image along 270 degree

PROGRAM:

```
import cv2
from google.colab.patches import cv2_imshow
path = r"/content/leaf.jpg"
src = cv2.imread(path)
window_name = 'Image'
image = cv2.rotate(src, cv2.ROTATE_90_CLOCKWISE)
cv2_imshow(image)
cv2.waitKey(0)
```

INPUT:

OUTPUT:

12. Perform Affine Transformation on the image.

AIM: To perform affine transformation in an image

PROGRAM:

```
import cv2
```

```
import numpy as np
from google.colab.patches import cv2_imshow
img = cv2.imread("/content/leaf.jpg")
rows, cols, _ = img.shape
pts1 = np.float32([[50, 50], [200, 50], [50, 200]])
pts2 = np.float32([[10, 100], [200, 50], [100, 250]])
M = cv2.getAffineTransform(pts1, pts2)
dst = cv2.warpAffine(img, M, (cols, rows))
cv2_imshow(dst)
```

INPUT:

OUTPUT:

13. Perform Perspective Transformation on the image.

AIM: To perform perspective transformation on an image

PROGRAM:

```
import cv2
import numpy as np
```

```
from google.colab.patches import cv2_imshow
img = cv2.imread("/content/leaf.jpg")
rows,cols,ch = img.shape
pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2 = np.float32([[100,50],[300,0],[0,300],[300,300]])
M = cv2.getPerspectiveTransform(pts1,pts2)
dst = cv2.warpPerspective(img,M,(cols, rows))
cv2_imshow(dst)
```

INPUT:

OUTPUT:

14. Perform Perspective Transformation on the Video.

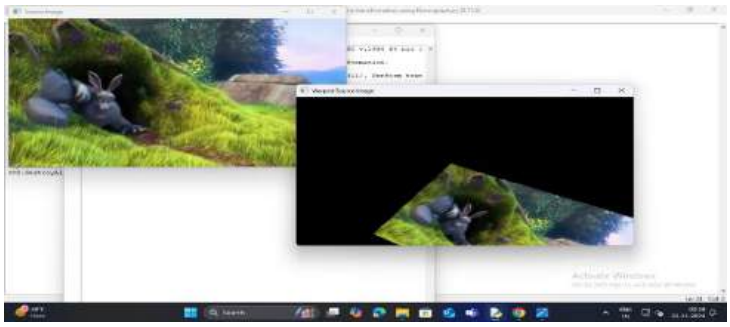
AIM: To perform perspective transformation on a video

PROGRAM:

import cv2
import numpy as np
video_path = r"C:\Users\ADMIN\Downloads\SampleVideo_1280x720_1mb.mp4"

```
cap = cv2.VideoCapture(video_path)
if not cap.isOpened():
    print("Error: Unable to open the video file")
    exit()
ret, im_src = cap.read()
if not ret:
    print("Error: Unable to read the video frame")
    cap.release()
    exit()
pts_src = np.array([[141, 131], [480, 159], [493, 630], [64, 601]])
pts_dst = np.array([[318, 256], [534, 372], [316, 670], [73, 473]])
im_dst = np.zeros_like(im_src)
h, status = cv2.findHomography(pts_src, pts_dst)
im_out = cv2.warpPerspective(im_src, h, (im_src.shape[1], im_src.shape[0]))
cv2.imshow("Source Image", im_src)
cv2.imshow("Destination Image", im_dst)
cv2.imshow("Warped Source Image", im_out)
cv2.waitKey(0)
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:



15. Perform transformation using Homography matrix

AIM: To perform transformation in an image using homography matrix

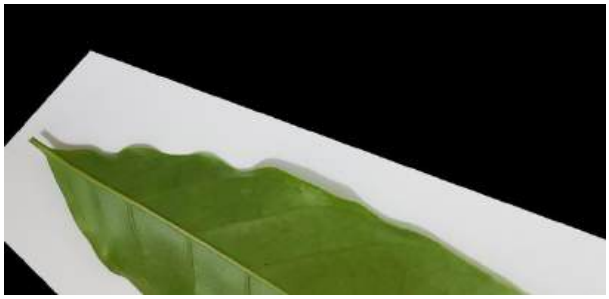
PROGRAM:

```
import cv2
import numpy as np
im_src = cv2.imread("/content/leaf.jpg")
```

```
pts_src = np.array([[141, 131], [480, 159], [493, 630],[64, 601]])
im_dst = cv2.imread("/content/leaf.jpg")
pts_dst = np.array([[318, 256],[534, 372],[316, 670],[73, 473]])
h, status = cv2.findHomography(pts_src, pts_dst)
if im_dst is not None:
    im_out = cv2.warpPerspective(im_src, h,
(im_dst.shape[1],im_dst.shape[0]))
    from google.colab.patches import cv2_imshow
    cv2_imshow(im_src)
    cv2_imshow(im_dst)
    cv2_imshow(im_out)
    cv2.waitKey(0)
else:
    print("Error: Could not load destination image.")
```

INPUT:

OUTPUT:



16.. Perform transformation using Direct Linear Transformation

AIM: To perform transformation in an image using direct linear transformation

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
img1 = cv2.imread("/content/leaf.jpg")
```

```
img2 = cv2.imread("/content/leaf.jpg")
pts1 = np.array([[50, 50], [200, 50], [50, 200], [200, 200]])
pts2 = np.array([[100, 100], [300, 100], [100, 300], [300, 300]])
H, _ = cv2.findHomography(pts1, pts2)
dst = cv2.warpPerspective(img1, H, (img2.shape[1], img2.shape[0]))
cv2_imshow(img1)
cv2_imshow(img2)
cv2_imshow(dst)
```

INPUT:

OU PUT:

17. Perform Edge detection using canny method

AIM: To detect edges of an image using canny method

PROGRAM:

```
import cv2
img = cv2.imread(r"C:/Users/ADMIN/Pictures/images (1).jfif")
cv2.imshow('Original', img)
cv2.waitKey(0)
```

img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
 # Blur the image for better edge detection
img_blur = cv2.GaussianBlur(img_gray, (3,3), 0)
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5)
cv2.imshow('Sobel X', sobelx)
cv2.waitKey(0)
input

OUTPUT:

18.Perform Edge detection using Sobel Matrix along X axis

AIM: To perform edge detection using sobel matrix along x-axis

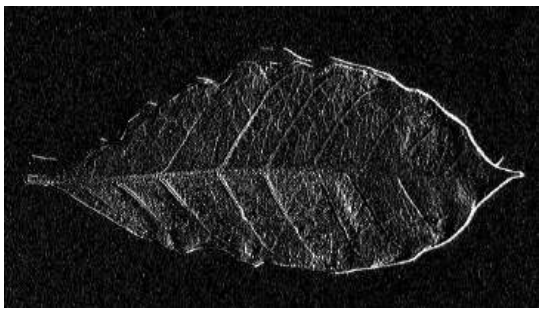
PROGRAM:

```
import cv2
from google.colab.patches import cv2_imshow
img = cv2.imread("/content/leaf.jpg")
cv2_imshow(img)
```

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_blur = cv2.GaussianBlur(img_gray, (3,3), 0)
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0,
ksize=5)
cv2_imshow(sobelx)
```

INPUT:

OUTPUT:



19.Perform Edge detection using Sobel Matrix along Y axis

AIM: To perform edge detection using sobel matrix along y-axis

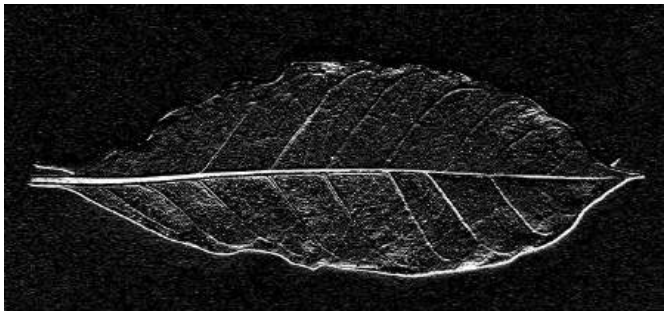
PROGRAM:

```
import cv2
img = cv2.imread("/content/leaf.jpg")
from google.colab.patches import cv2_imshow
cv2_imshow(img)
```

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_blur = cv2.GaussianBlur(img_gray, (3,3), 0)
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1,
ksize=5)
cv2_imshow(sobely)
```

INPUT:

OUTPUT:



20.Perform Edge detection using Sobel Matrix along XY axis

AIM: To perform edge detection in an image using sobel matrix along xy axis

PROGRAM:

```
import cv2
import numpy as np
img = cv2.imread(r"C:\Users\ADMIN\Pictures\images (1).jfif")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
kernel = np.array([[0,1,0], [1,-8,1], [0,1,0]])
sharpened = cv2.filter2D(gray, -1, kernel)
cv2.imshow('Original', gray)
cv2.imshow('Sharpened', sharpened)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Input

OUTPUT:

21. Perform Sharpening of Image using Laplacian mask with negative centre coefficient.

AIM: To perform sharpening of image using laplacian mask with negative centre coefficient.

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
```

```
img = cv2.imread("/content/leaf.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
kernel = np.array([[0,1,0], [1,-8,1], [0,1,0]])
sharpened = cv2.filter2D(gray, -1, kernel)
cv2_imshow(gray)
cv2_imshow(sharpened)
```

INPUT:

OUTPUT:

22.Perform Sharpening of Image using Laplacian mask implemented with an extension of diagonal neighbours

AIM: To Perform Sharpening of Image using Laplacian mask implemented with an extension of diagonal neighbours

PROGRAM:

import cv2
import numpy as np
img = cv2.imread(r"C:\Users\ADMIN\Pictures\images (1).jfif")

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
kernel = np.array([[0,1,0], [1,-4,1], [0,1,0]])
sharpened = cv2.filter2D(gray, -1, kernel)
cv2.imshow('Original', gray)
cv2.imshow('Sharpened', sharpened)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

23. Perform Sharpening of Image using Laplacian mask with positive centre coefficient.

AIM: To Perform Sharpening of Image using Laplacian mask with positive centre coefficient.

PROGRAM:

```
import cv2
import numpy as np
img = cv2.imread(r"C:\Users\ADMIN\Pictures\images (1).jfif")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
kernel = np.array([[0,1,0], [1,-8,1], [0,1,0]])
sharpened = cv2.filter2D(gray, -1, kernel)
cv2.imshow('Original', gray)
cv2.imshow('Sharpened', sharpened)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

24.. Perform Sharpening of Image using unsharp masking.

AIM: To perform sharpening of image using unsharp masking.

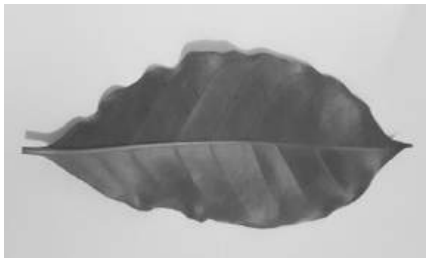
PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
img = cv2.imread("/content/leaf.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
laplacian_kernel = np.array([[0, 1, 0],
 [1, -4, 1],
```

```
 [0, 1, 0]])
laplacian = cv2.filter2D(gray, -1, laplacian_kernel)
sharpened = cv2.add(gray, laplacian)
cv2_imshow(gray)
cv2_imshow(sharpened)
```

INPUT:

OUTPUT:



25. Perform Sharpening of Image using High-Boost Masks.

AIM:

To Perform Sharpening of Image using High-Boost Masks.

PROGRAM:

```
import cv2
resized_img = cv2.imread(r"C:\Users\ADMIN\Pictures\girl.png")
if resized_img is None:
    print("Error: Could not load the main image. Check the file path.")
    exit()
resized_wm = cv2.imread(r"C:\Users\ADMIN\Pictures\images (1).jfif")
```

```
if resized_wm is None:
    print("Error: Could not load the watermark image. Check the file path.")
    exit()
resized_wm = cv2.resize(resized_wm, (100, 100))
h_img, w_img, _ = resized_img.shape
h_wm, w_wm, _ = resized_wm.shape

center_y = int(h_img / 2)
center_x = int(w_img / 2)

top_y = center_y - int(h_wm / 2)
left_x = center_x - int(w_wm / 2)
bottom_y = top_y + h_wm
right_x = left_x + w_wm

if top_y < 0 or left_x < 0 or bottom_y > h_img or right_x > w_img:
    print("Error: Watermark size exceeds the boundaries of the main image.")
    exit()

roi = resized_img[top_y:bottom_y, left_x:right_x]

result = cv2.addWeighted(roi, 1, resized_wm, 0.3, 0)

resized_img[top_y:bottom_y, left_x:right_x] = result

filename =(r"C:\Users\ADMIN\Pictures\images (1).jfif")
cv2.imwrite(filename, resized_img)

cv2.imshow("Resized Input Image", resized_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

26.Perform Sharpening of Image using Gradient masking

AIM: To perform sharpening of image using gradient sharpening.

PROGRAM:

```
import cv2
import numpy as np
image = cv2.imread(r"C:\Users\ADMIN\Pictures\images (1).jfif",
cv2.IMREAD_COLOR)
if image is None:
    print("Error: Could not load the image. Check the file path.")
    exit()
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
sobel_x = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=3)
sobel_y = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=3)
gradient_magnitude = cv2.magnitude(sobel_x, sobel_y)
gradient_magnitude = cv2.convertScaleAbs(gradient_magnitude)
alpha = 1
beta = 0.5
sharpened_image = cv2.addWeighted(gray_image, alpha, gradient_magnitude,
beta,0)
cv2.imshow("Original Image", gray_image)
cv2.imshow("Gradient Magnitude (Mask)", gradient_magnitude)
cv2.imshow("Sharpened Image", sharpened_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

27. Insert water marking to the image using OpenCV.

AIM: To perform watermarking to the image using opencv.

PROGRAM:

```
import cv2
from google.colab.patches import cv2_imshow
img = cv2.imread("/content/leaf.jpg")
wm = cv2.imread("/content/leaf.jpg")
h_wm, w_wm = wm.shape[:2]
h_img, w_img = img.shape[:2]
center_x = int(w_img/2)
center_y = int(h_img/2)
top_y = center_y - int(h_wm/2)
```

```
left_x = center_x - int(w_wm/2)
bottom_y = top_y + h_wm
right_x = left_x + w_wm
roi = img[top_y:bottom_y, left_x:right_x]
result = cv2.addWeighted(roi, 1, wm, 0.3, 0)
img[top_y:bottom_y, left_x:right_x] = result
cv2_imshow(img)
```

INPUT:

OUTPUT:

28.Do Cropping, Copying and pasting image inside another image using OpenCV

AIM: To perform cropping,copying and pasting an image inside another image using opencv.

PROGRAM:

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
image = cv2.imread("/content/leaf.jpg")
img2 = cv2.imread("/content/leaf.jpg")
print(image.shape)
cv2_imshow(image)
imageCopy = image.copy()
```

```
cv2.circle(imageCopy, (100, 100), 30, (255, 0, 0), -1)
cv2_imshow(image)
cv2_imshow(imageCopy)
cropped_image = image[80:280, 150:330]
cv2_imshow(cropped_image)
cv2.imwrite("Cropped Image.jpg", cropped_image)
dst = cv2.addWeighted(image, 0.5, img2, 0.7, 0)
img_arr = np.hstack((image, img2))
cv2_imshow(img_arr)
cv2_imshow(dst)
```

INPUT:

OUTPUT:

29.. Find the boundary of the image using Convolution kernel for the given image

AIM: To find the boundary of the image using a convolution kernel for the given image.

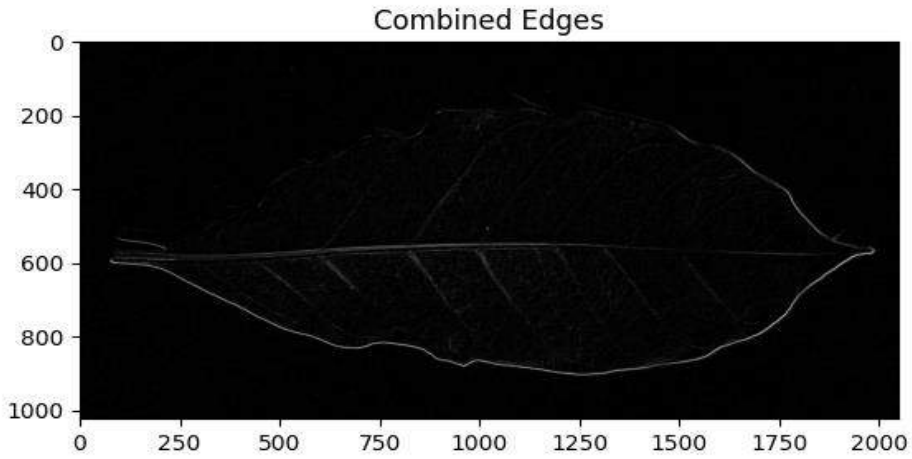
PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/leaf.jpg', cv2.IMREAD_GRAYSCALE)
sobel_x = np.array([[-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]])
sobel_y = np.array([[-1, -2, -1],
                    [ 0,  0,  0],
                    [ 1,  2,  1]])
```

```
edges_x = cv2.filter2D(image, -1, sobel_x)
edges_y = cv2.filter2D(image, -1, sobel_y)
edges = cv2.magnitude(edges_x.astype(np.float32),
edges_y.astype(np.float32))
edges = cv2.normalize(edges, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.subplot(1, 3, 2)
plt.title('Edges X')
plt.imshow(edges_x, cmap='gray')
plt.subplot(1, 3, 3)
plt.title('Edges Y')
plt.imshow(edges_y, cmap='gray')
plt.figure()
plt.title('Combined Edges')
plt.imshow(edges, cmap='gray')
plt.show()
```

INPUT:

OUTPUT:



30.Morphological operations based on OpenCV using Erosion technique

AIM: To perform morphological operations on opencv using erosion technique.

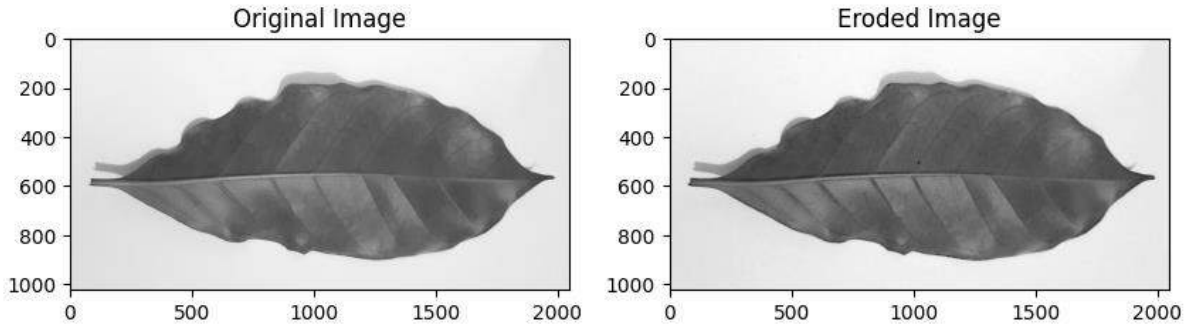
PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/leaf.jpg', cv2.IMREAD_GRAYSCALE)
```

```
kernel = np.ones((5, 5), np.uint8)
eroded_image = cv2.erode(image, kernel, iterations=1)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.subplot(1, 2, 2)
plt.title('Eroded Image')
plt.imshow(eroded_image, cmap='gray')
plt.show()
```

INPUT:

OUTPUT:



31.Morphological operations based on OpenCV using Dilation technique

AIM: To perform morphological operations based on opencv using dilation technique.

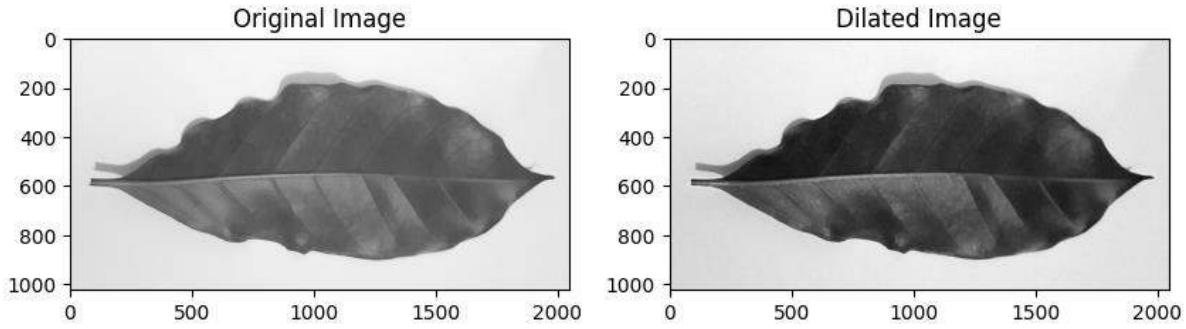
PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/leaf.jpg', cv2.IMREAD_GRAYSCALE)
```

```
kernel = np.ones((5, 5), np.uint8)
dilated_image = cv2.dilate(image, kernel, iterations=1)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.subplot(1, 2, 2)
plt.title('Dilated Image')
plt.imshow(dilated_image, cmap='gray')
plt.show()
```

INPUT:

OUTPUT:



32. Morphological operations based on OpenCV using Opening technique.

AIM: To perform morphological operations based on open cv using opening technique.

PROGRAM:

import cv2
import numpy as np

```
img = cv2.imread(r"C:\Users\ADMIN\Pictures\images (1).jfif",
cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
cv2.imshow("Original", img)
cv2.imshow("opening", opening)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

33.. Morphological operations based on OpenCV using Closing technique.

AIM: To perform operations based on opencv using closure technique.

PROGRAM:

```
import cv2
import numpy as np
img = cv2.imread(r"C:\Users\ADMIN\Pictures\images (1).jfif",
cv2.IMREAD_GRAYSCALE)
kernel = np.ones((5,5), np.uint8)
```

```
closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
cv2.imshow("Original", img)
cv2.imshow("Closing", closing)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT:

34.. Morphological operations based on OpenCV using Morphological Gradient technique

AIM: To perform morphological operations based on opencv using morphological gradient technique.

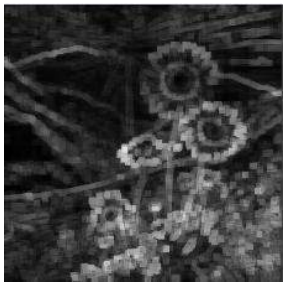
PROGRAM:

```
import cv2
import numpy as np
img = cv2.imread(r"C:\Users\ADMIN\Pictures\images (1).jfif",
cv2.IMREAD_GRAYSCALE)
```

```
kernel = np.ones((5,5), np.uint8)
grad = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)
cv2.imshow("Original", img)
cv2.imshow("Gradient", grad)
cv2.waitKey
```

INPUT:

OUTPUT:



35.Morphological operations based on OpenCV using Top hat technique.

AIM: To perform morphological operations based on open cv using top hat technique.

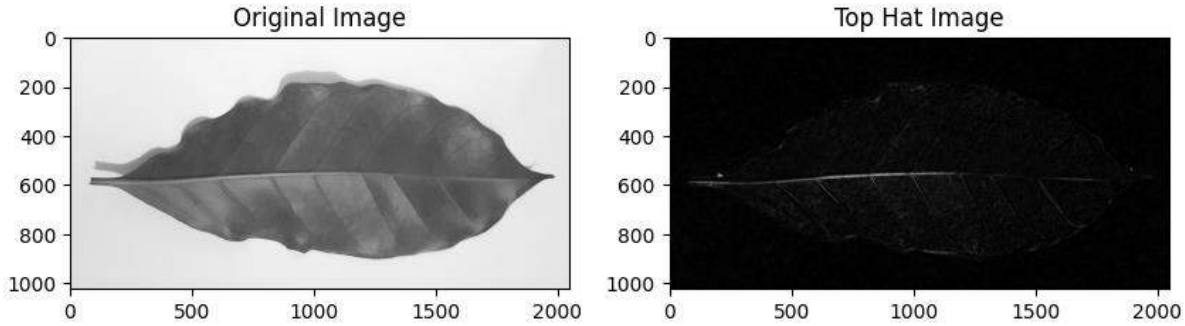
PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/leaf.jpg', cv2.IMREAD_GRAYSCALE)
kernel = np.ones((15, 15), np.uint8)
top_hat = cv2.morphologyEx(image, cv2.MORPH_TOPHAT, kernel)
plt.figure(figsize=(10, 5))
```

```
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.subplot(1, 2, 2)
plt.title('Top Hat Image')
plt.imshow(top_hat, cmap='gray')
plt.show()
```

INPUT:

OUTPUT:



36.. Morphological operations based on OpenCV using Black hat technique.

AIM: To perform morphological operations based on opencv using black hat technique.

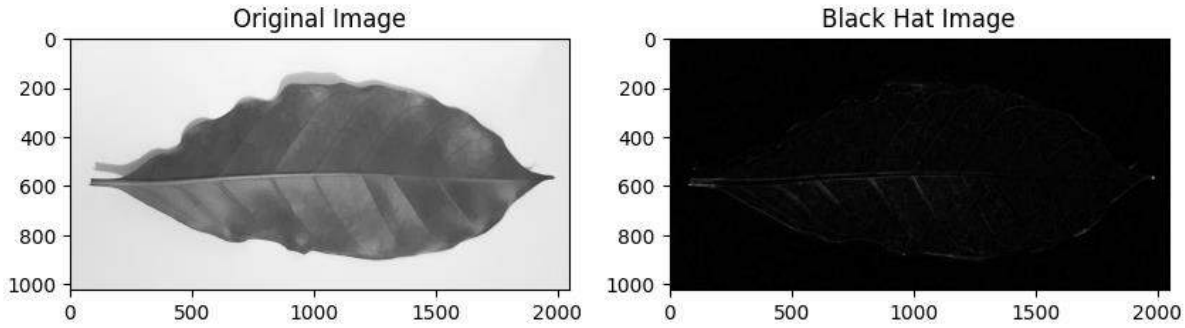
PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/leaf.jpg', cv2.IMREAD_GRAYSCALE)
kernel = np.ones((15, 15), np.uint8)
black_hat = cv2.morphologyEx(image, cv2.MORPH_BLACKHAT, kernel)
```

```
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image, cmap='gray')
plt.subplot(1, 2, 2)
plt.title('Black Hat Image')
plt.imshow(black_hat, cmap='gray')
plt.show()
```

INPUT:

OUTPUT:



37. Recognise LEAF from the given image by general Object recognition using OpenCV.

AIM: To recognize leaf from the given image by general object recognition using opencv.

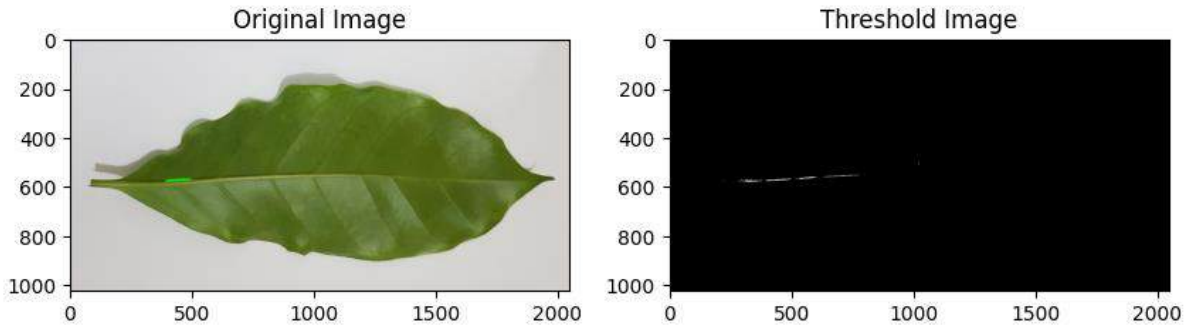
PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/leaf.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
```

```
_, thresh = cv2.threshold(blurred, 60, 255, cv2.THRESH_BINARY_INV)
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
leaf_contour = max(contours, key=cv2.contourArea)
cv2.drawContours(image, [leaf_contour], -1, (0, 255, 0), 3)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title('Threshold Image')
plt.imshow(thresh, cmap='gray')
plt.show()
```

INPUT:

OUTPUT:



38.Using Opencv play Video in Reverse mode

AIM: To play video in reverse using opencv.

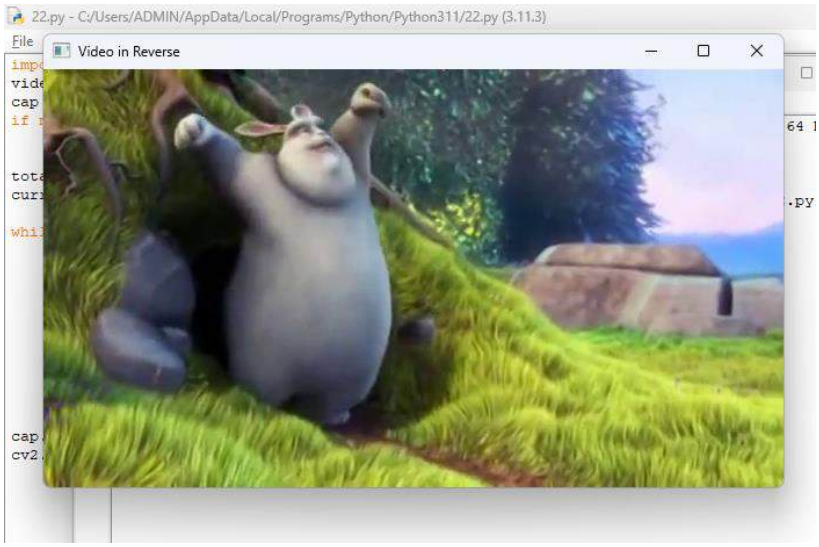
PROGRAM

```
import cv2
video_path = r"C:\Users\ADMIN\Downloads\SampleVideo_1280x720_1mb.mp4"
cap = cv2.VideoCapture(video_path)
if not cap.isOpened():
    print("Error: Could not open the video file. Check the file path.")
    exit()
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
current_frame = total_frames - 1
```

```
while current_frame >= 0:
    cap.set(cv2.CAP_PROP_POS_FRAMES, current_frame)
    ret, frame = cap.read()
    if not ret:
        print(f"Error: Unable to read frame {current_frame}.")
        break
    cv2.imshow('Video in Reverse', frame)

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break
    current_frame -= 1
cap.release()
cv2.destroyAllWindows()
```

OUTPUT:



39.. Face Detection using Opencv

AIM: To recognize face using opencv.

PROGRAM:

```
import cv2
image_path = r"C:\Users\ADMIN\Pictures\images (1).jfif"
img = cv2.imread(image_path)
if img is None:
    print("Error: Could not load the image. Check the file path.")
    exit()
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
haar_cascade_path = cv2.data.haarcascades +
"haarcascade_frontalface_default.xml"
face_cascade = cv2.CascadeClassifier(haar_cascade_path)

if face_cascade.empty():
    print("Error: Could not load the Haar Cascade file.")
    exit()
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
cv2.imshow('Faces Detected', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

40. Draw Rectangular shape and extract objects

AIM: To draw a rectangular shape in the image to extract objects

PROGRAM:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import c cv2_imshow
image = cv2.imread('/content/leaf.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5, 5), 0)
_, thresh = cv2.threshold(blurred, 60, 255, cv2.THRESH_BINARY_INV)
```

```
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
output_image = image.copy()
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    cv2.rectangle(output_image, (x, y), (x + w, y + h), (0, 255, 0), 2)
    extracted_object = image[y:y + h, x:x + w]
    cv2_imshow(extracted_object)
cv2_imshow(output_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INPUT:

OUTPUT: