**SIMATS ENGINEERING**

**DSA0201-COMPUTER VISION WITH OPENCV FOR ROBOTICS AND AUTOMATION**

**NAME: C.PRAVEENA**

**REG NO :- 192324272**

**1. PERFORM BASIC IMAGE HANDLING AND PROCESSING OPERATIONS ON THE IMAGE.**

**• Read an image in python and Convert an Image to Grayscale**

**AIM:** To read an image in python and convert an image to grayscale

**INPUT:**

****

**PROGRAM:**

import cv2

from google.colab.patches import cv2\_imshow

# Read the image

image = cv2.imread('/content/Screenshot 2025-04-30 084404.png')

# Convert to grayscale

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

# Display the grayscale image using matplotlib

cv2\_imshow(image)

cv2\_imshow(gray\_image)

**OUTPUT:**

****

**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 Read an image in python and Convert an Image to Blur using GaussianBlur.

**INPUT:**

****

**PROGRAM:**

import cv2

from google.colab.patches import cv2\_imshow

image = cv2.imread('/content/Screenshot 2025-04-30 091205.png')

blurred\_image = cv2.GaussianBlur(image, (15, 15), 0)

cv2\_imshow(image)

cv2\_imshow(blurred\_image)

**OUTPUT:**

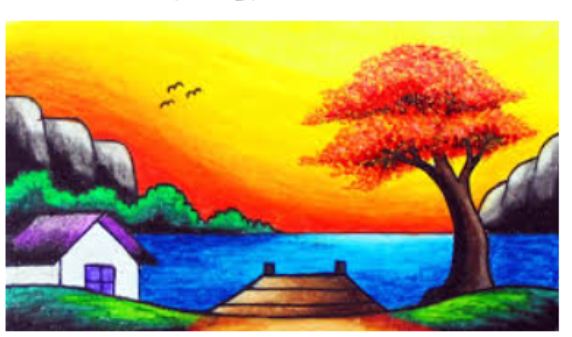


**3. PERFORM BASIC IMAGE HANDLING AND PROCESSING OPERATIONS ON THE IMAGE**

**• Read an image in python  and  Convert an Image to show outline using Canny function.**

**AIM:** To Read an image in python  and  Convert an Image to show outline using Canny function.

**INPUT:**

****

**PROGRAM:**

import cv2

from google.colab.patches import cv2\_imshow

image = cv2.imread('/content/Screenshot 2025-04-30 100739.png')

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

edges = cv2.Canny(gray\_image, 100, 200)

cv2\_imshow(image)

cv2\_imshow(edges)

**OUTPUT:**

****

**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 Read an image in python  and  Dilate an Image using Dilate function.

****

**PROGRAM:**

import cv2

from google.colab.patches import cv2\_imshow

image = cv2.imread('/content/Screenshot 2025-04-30 101600.png')

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

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5))

dilated\_image = cv2.dilate(gray\_image, kernel, iterations=1)

cv2\_imshow(image)

cv2\_imshow(dilated\_image)

**OUTPUT:**

****

**5. PERFORM BASIC IMAGE HANDLING AND PROCESSING OPERATIONS ON THE IMAGE**

**• Read an image in python  and  Erode an Image using erode function.**

**AIM:** To Read an image in python  and  Erode an Image using erode function.



**PROGRAM:**

import cv2

from google.colab.patches import cv2\_imshow

image = cv2.imread('/content/Screenshot 2025-04-30 102323.png')

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

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5))

eroded\_image = cv2.erode(gray\_image, kernel, iterations=1)

cv2\_imshow(image)

cv2\_imshow(eroded\_image)

**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:** To Read captured video in python  and  display the video, in slow motion and in fast motion.

**PROGRAM:**

import cv2

from google.colab.patches import cv2\_imshow

import time

cap = cv2.VideoCapture('/content/WhatsApp Video 2025-04-30 at 10.47.36 AM.mp4')  # Replace with your video path

if not cap.isOpened():

    print("Error: Cannot open video file")

    exit()

speed\_mode = 'slow'  # Change to 'normal', 'slow', or 'fast'

while cap.isOpened():

    ret, frame = cap.read()

    if not ret:

        break

    cv2\_imshow(frame)

    if speed\_mode == 'normal':

        time.sleep(1 / 5)  # ~30 FPS

    elif speed\_mode == 'slow':

        time.sleep(1 / 2)  # Slower (~10 FPS)

    elif speed\_mode == 'fast':

        time.sleep(1 / 10)  # Faster (~60 FPS)

cap.release()

**INPUT & OUTPUT:**



**7. CAPTURE VIDEO FROM WEB CAMERA AND  DISPLAY THE VIDEO, IN SLOW MOTION AND IN FAST MOTION.**

**AIM:** To Perform basic Operations To capture live video from a webcam in Python and display it in normal, slow motion, and fast motion.

**PROGRAM:**

import cv2

cap = cv2.VideoCapture(0)

speed\_mode = input("Enter speed (normal/slow/fast): ").lower()

if speed\_mode == 'slow':

delay = 100

elif speed\_mode == 'fast':

delay = 10

else:

delay = 30

while True:

ret, frame = cap.read()

if not ret:

break

cv2.imshow('Webcam Video', frame)

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

break

cap.release()

cv2.destroyAllWindows()

**INPUT & OUTPUT:**



**8. SCALING AN IMAGE TO ITS BIGGER AND SMALLER SIZES.**

**AIM:**

To Perform basic Operations to read an image and resize it to both a larger and a smaller size using Python.

**PROGRAM:**

import cv2

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

smaller = cv2.resize(image, None, fx=0.5, fy=0.5, interpolation=cv2.INTER\_AREA)

bigger = cv2.resize(image, None, fx=2.0, fy=2.0, interpolation=cv2.INTER\_LINEAR)

cv2.imshow('Original Image', image)

cv2.imshow('Smaller Image', smaller)

cv2.imshow('Bigger Image', bigger)

cv2.waitKey(0)

cv2.destroyAllWindows()

**INPUT:**

****

**OUTPUT:**

****

**9. Perform Rotation of an image to clockwise and counter clockwise direction.**

**AIM**: To Perform Rotation of an image to clockwise and counter clockwise direction.

**PROGRAM:**

from PIL import Image

import matplotlib.pyplot as plt

image = Image.open('/content/pikachu.webp')

# Rotate images

rotated\_clockwise = image.rotate(-90, expand=True)

rotated\_counter\_clockwise = image.rotate(90, expand=True)

plt.figure(figsize=(15, 5))

# Original image

plt.subplot(1, 3, 1)

plt.imshow(image)

plt.title("Original")

plt.axis('off')

# Rotated clockwise

plt.subplot(1, 3, 2)

plt.imshow(rotated\_clockwise)

plt.title("Clockwise")

plt.axis('off')

# Rotated counter-clockwise

plt.subplot(1, 3, 3)

plt.imshow(rotated\_counter\_clockwise)

plt.title("Counter Clockwise")

plt.axis('off')

plt.show()

**INPUT/OUTPUT**



**10. PERFORM MOVING OF AN IMAGE FROM ONE PLACE TO ANOTHER.**

**AIM**: To Perform moving of an image from one place to another.

**PROGRAM:**

import cv2

import numpy as np

from google.colab import files

from PIL import Image

import matplotlib.pyplot as plt

uploaded = files.upload()

image\_path = list(uploaded.keys())[0]

image = Image.open(image\_path)

image\_cv = np.array(image)

image\_cv = cv2.cvtColor(image\_cv, cv2.COLOR\_RGB2BGR)

Tx = 100

Ty = 50

translation\_matrix = np.float32([[1, 0, Tx], [0, 1, Ty]])

rows, cols, \_ = image\_cv.shape

translated\_image = cv2.warpAffine(image\_cv, translation\_matrix, (cols, rows))

translated\_image\_rgb = cv2.cvtColor(translated\_image, cv2.COLOR\_BGR2RGB)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(image)

plt.title("Original Image")

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(translated\_image\_rgb)

plt.title("Translated Image")

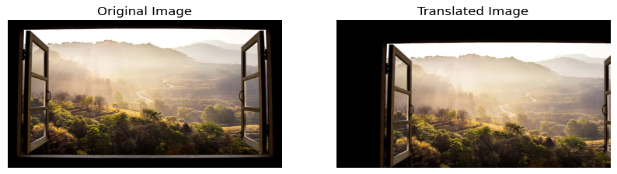
plt.axis('off')

plt.show()

cv2.imwrite("translated\_image.jpg", translated\_image)

files.download("translated\_image.jpg")

**INPUT/OUTPUT:**



**11. PERFORM AFFINE TRANSFORMATION ON THE IMAGE.**

**AIM:** To Perform Affine Transformation on the image.

**PROGRAM:**

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

from IPython.display import display

image = Image.open('/content/pikachu.webp')

# Define the affine transformation matrix

matrix = [1, 0.1, 50, 0, 1, 5]

transformed\_image = image.transform(image.size, Image.AFFINE, matrix)

# Create a figure with subplots

plt.figure(figsize=(12, 6))

# Original image

plt.subplot(1, 2, 1)  # 1 row, 2 columns, first subplot

plt.imshow(image)

plt.title("Original Image")

plt.axis('off')

# Transformed image

plt.subplot(1, 2, 2)  # 1 row, 2 columns, second subplot

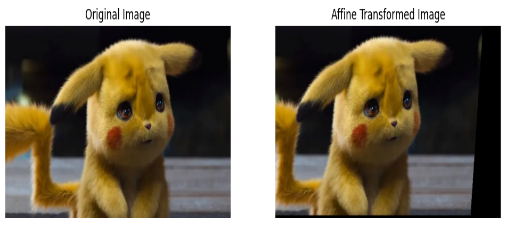
plt.imshow(transformed\_image)

plt.title("Affine Transformed Image")

plt.axis('off')

plt.show()

**INPUT/OUTPUT:**



**12. PERFORM PERSPECTIVE TRANSFORMATION ON THE IMAGE.**

**AIM:** To Perform Perspective Transformation on the image.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image

image = cv2.imread('/content/BTS.jfif')

src\_points = np.float32([[50, 50], [500, 50], [500, 500], [50, 500]])

dst\_points = np.float32([[100, 100], [450, 80], [500, 450], [100, 500]])

# Calculate the perspective transformation matrix

matrix = cv2.getPerspectiveTransform(src\_points, dst\_points)

# Apply the perspective transformation to the image

transformed\_image = cv2.warpPerspective(image, matrix, (image.shape[1], image.shape[0]))

# Convert the BGR image to RGB for displaying with Matplotlib

transformed\_image\_rgb = cv2.cvtColor(transformed\_image, cv2.COLOR\_BGR2RGB)

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Plot the original and transformed images side by side

plt.figure(figsize=(12, 6))

# Original image

plt.subplot(1, 2, 1)

plt.imshow(image\_rgb)

plt.title("Original Image")

plt.axis('off')

# Transformed image

plt.subplot(1, 2, 2)

plt.imshow(transformed\_image\_rgb)

plt.title("Perspective Transformed Image")

plt.axis('off')

plt.show()

**INPUT/OUTPUT:**



**13. Perform Perspective Transformation on the Video.**

**AIM:**

To perform Perspective Transformation on each frame of a video using Python and OpenCV**.**

**PROGRAM:**

import cv2

import numpy as np

cap = cv2.VideoCapture('video.mp4')

pts1 = np.float32([[50, 50], [300, 50], [50, 300], [300, 300]])

pts2 = np.float32([[0, 0], [300, 0], [0, 300], [300, 300]])

matrix = cv2.getPerspectiveTransform(pts1, pts2)

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

transformed = cv2.warpPerspective(frame, matrix, (300, 300))

cv2.imshow('Original Video', frame)

cv2.imshow('Perspective Transform', transformed)

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

break

cap.release()

cv2.destroyAllWindows()

**INPUT & OUTPUT:**



**14. PERFORM TRANSFORMATION USING HOMOGRAPHY MATRIX.**

**AIM**: To Perform transformation using Homography matrix.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('/content/PLACES.jpg')

src\_points = np.float32([[100, 100], [400, 100], [400, 400], [100, 400]])

dst\_points = np.float32([[150, 150], [450, 50], [500, 400], [100, 500]])

matrix, \_ = cv2.findHomography(src\_points, dst\_points)

height, width, \_ = image.shape

transformed\_image = cv2.warpPerspective(image, matrix, (width, height))

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

transformed\_image\_rgb = cv2.cvtColor(transformed\_image, cv2.COLOR\_BGR2RGB)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.imshow(image\_rgb)

plt.title("Original Image")

plt.axis('off')

plt.subplot(1, 2, 2)

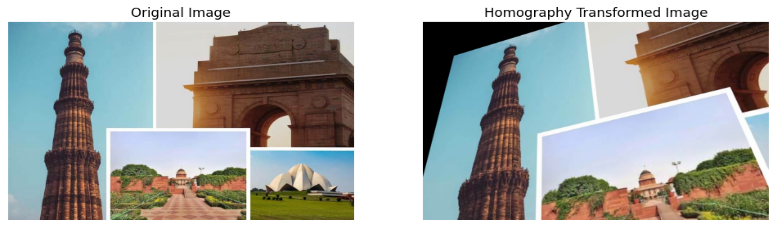
plt.imshow(transformed\_image\_rgb)

plt.title("Homography Transformed Image")

plt.axis('off')

plt.show()

**INPUT/OUTPUT:**



**15. PERFORM TRANSFORMATION USING DIRECT LINEAR TRANSFORMATION.**

**AIM:** To Perform transformation using Direct Linear Transformation.

**PROGRAM:**

import numpy as np

import cv2

import matplotlib.pyplot as plt

image = cv2.imread('/content/Gkf0aexWoAA89zd.jpg')

src\_points = np.array([[100, 100], [400, 100], [400, 400], [100, 400]], dtype=np.float32)

dst\_points = np.array([[150, 150], [450, 50], [500, 400], [100, 500]], dtype=np.float32)

matrix, \_ = cv2.findHomography(src\_points, dst\_points)

height, width, \_ = image.shape

transformed\_image = cv2.warpPerspective(image, matrix, (width, height))

image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

transformed\_image\_rgb = cv2.cvtColor(transformed\_image, cv2.COLOR\_BGR2RGB)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.imshow(image\_rgb)

plt.title("Original Image")

plt.axis('off')

plt.subplot(1, 2, 2)

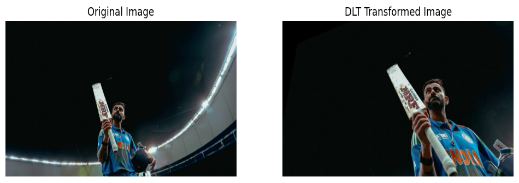
plt.imshow(transformed\_image\_rgb)

plt.title("DLT Transformed Image")

plt.axis('off')

plt.show()

**INPUT/OUTPUT:**



**16. PERFORM EDGE DETECTION USING CANNY METHOD**

**AIM:** To Perform Edge detection using canny method.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image

image = cv2.imread('/content/BTS.jfif', cv2.IMREAD\_GRAYSCALE)

# Apply Gaussian Blur to reduce noise and improve edge detection

blurred\_image = cv2.GaussianBlur(image, (5, 5), 0)

# Perform Canny edge detection

edges = cv2.Canny(blurred\_image, threshold1=100, threshold2=200)

# Display the original image and the edges side by side

plt.figure(figsize=(12, 6))

# Original image

plt.subplot(1, 2, 1)

plt.imshow(image, cmap='gray')

plt.title("Original Image")

plt.axis('off')

# Edge-detected image

plt.subplot(1, 2, 2)

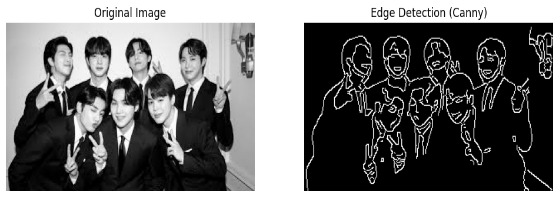
plt.imshow(edges, cmap='gray')

plt.title("Edge Detection (Canny)")

plt.axis('off')

plt.show()

**INPUT/OUTPUT:**



**17. PERFORM EDGE DETECTION USING SOBEL MATRIX ALONG X AXIS**

**AIM:** To Perform Edge detection using Sobel Matrix along X axis

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('/content/pikachu.webp', cv2.IMREAD\_GRAYSCALE)

# Apply Sobel operator along the X-axis (to detect vertical edges)

sobel\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=3)

# Convert the result to absolute values and then to uint8 for display

sobel\_x\_abs = cv2.convertScaleAbs(sobel\_x)

# Display the original image and the Sobel X edges side by side

plt.figure(figsize=(12, 6))

# Original image

plt.subplot(1, 2, 1)

plt.imshow(image, cmap='gray')

plt.title("Original Image")

plt.axis('off')

# Sobel X edge-detected image

plt.subplot(1, 2, 2)

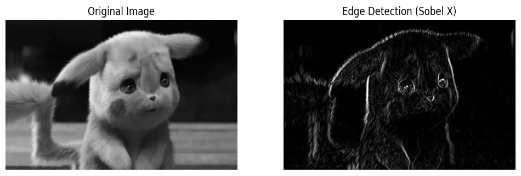
plt.imshow(sobel\_x\_abs, cmap='gray')

plt.title("Edge Detection (Sobel X)")

plt.axis('off')

plt.show()

**INPUT/OUTPUT:**



**18. PERFORM EDGE DETECTION USING SOBEL MATRIX ALONG Y AXIS.**

**AIM:** To Perform Edge detection using Sobel Matrix along Y axis

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('/content/pikachu.webp', cv2.IMREAD\_GRAYSCALE)

# Apply Sobel operator along the Y-axis (to detect horizontal edges)

sobel\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=3)

# Convert the result to absolute values and then to uint8 for display

sobel\_y\_abs = cv2.convertScaleAbs(sobel\_y)

# Display the original image and the Sobel Y edges side by side

plt.figure(figsize=(12, 6))

# Original image

plt.subplot(1, 2, 1)

plt.imshow(image, cmap='gray')

plt.title("Original Image")

plt.axis('off')

# Sobel Y edge-detected image

plt.subplot(1, 2, 2)

plt.imshow(sobel\_y\_abs, cmap='gray')

plt.title("Edge Detection (Sobel Y)")

plt.axis('off')

plt.show()

**INPUT/OUTPUT:**



**19. PERFORM EDGE DETECTION USING SOBEL MATRIX ALONG XY AXIS**

**AIM:** To Perform Edge detection using Sobel Matrix along XY axis

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('/content/pikachu.webp', cv2.IMREAD\_GRAYSCALE)

# Apply Sobel operator along the X-axis (to detect vertical edges)

sobel\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=3)

sobel\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=3)

sobel\_magnitude = cv2.sqrt(sobel\_x\*\*2 + sobel\_y\*\*2)

sobel\_magnitude\_abs = cv2.convertScaleAbs(sobel\_magnitude)

# Display the original image and the Sobel XY edges side by side

plt.figure(figsize=(12, 6))

# Original image

plt.subplot(1, 2, 1)

plt.imshow(image, cmap='gray')

plt.title("Original Image")

plt.axis('off')

# Sobel XY edge-detected image

plt.subplot(1, 2, 2)

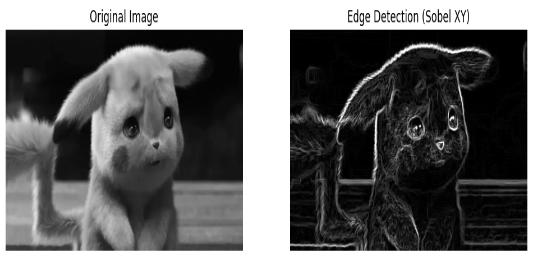
plt.imshow(sobel\_magnitude\_abs, cmap='gray')

plt.title("Edge Detection (Sobel XY)")

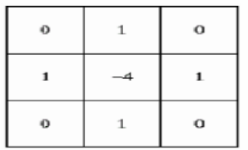
plt.axis('off')

plt.show()

**INPUT/OUTPUT:**



**20. PERFORM SHARPENING OF IMAGE USING LAPLACIAN MASK WITH NEGATIVE CENTER COEFFICIENT.**

****

**AIM:** To Perform Sharpening of Image using Laplacian mask with negative center coefficient.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load image (you can change the path to your image)

image = cv2.imread('/content/Screenshot (61).png')  # Use your uploaded image

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

# Define Laplacian kernel with negative center coefficient

laplacian\_kernel = np.array([[0,  1, 0],

                             [1, -4, 1],

                             [0,  1, 0]])

# Apply the kernel to the grayscale image

laplacian = cv2.filter2D(gray, -1, laplacian\_kernel)

# Sharpen the image: Original - Laplacian

sharpened = cv2.subtract(gray, laplacian)

# Display results

plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1), plt.imshow(gray, cmap='gray'), plt.title('Original Grayscale')

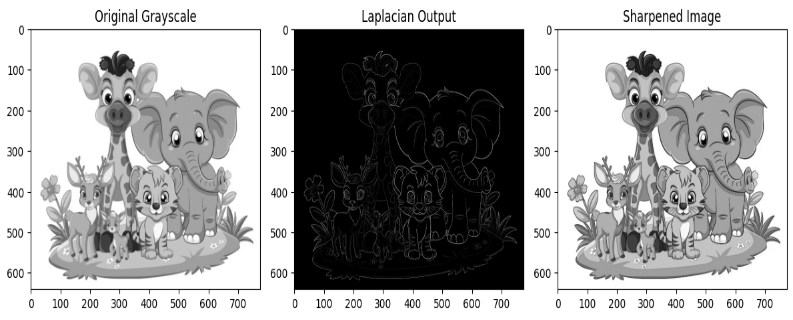
plt.subplot(1, 3, 2), plt.imshow(laplacian, cmap='gray'), plt.title('Laplacian Output')

plt.subplot(1, 3, 3), plt.imshow(sharpened, cmap='gray'), plt.title('Sharpened Image')

plt.tight\_layout()

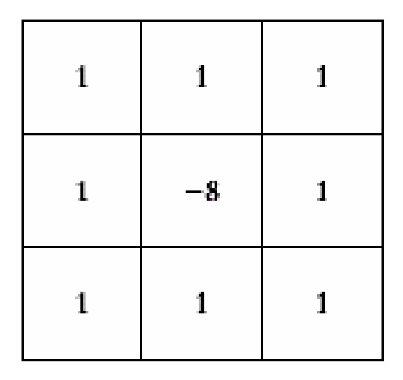
plt.show()

**INPUT /OUTPUT:**



**21. PERFORM SHARPENING OF IMAGE USING LAPLACIAN MASK IMPLEMENTED WITH AN EXTENSION OF**

**DIAGONAL NEIGHBORS,**



**AIM:** To Perform Sharpening of Image using Laplacian mask implemented with an extension of diagonal neighbors.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image in grayscale (for simplicity)

image = cv2.imread('/content/jerry.jpeg', cv2.IMREAD\_GRAYSCALE)

# Define the Laplacian kernel with diagonal neighbors

laplacian\_kernel = np.array([[ 1,  1,  1],

                             [ 1, -8,  1],

                             [ 1,  1,  1]])

# Apply convolution to the image with the Laplacian kernel

laplacian\_image = cv2.filter2D(image, -1, laplacian\_kernel)

# Sharpen the image: original + (original - laplacian\_image)

sharpened\_image = cv2.addWeighted(image, 1.5, laplacian\_image, -0.5, 0)

# Display the images

plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)

plt.title('Original Image')

plt.imshow(image, cmap='gray')

plt.axis('off')

plt.subplot(1, 3, 2)

plt.title('Laplacian Image')

plt.imshow(laplacian\_image, cmap='gray')

plt.axis('off')

plt.subplot(1, 3, 3)

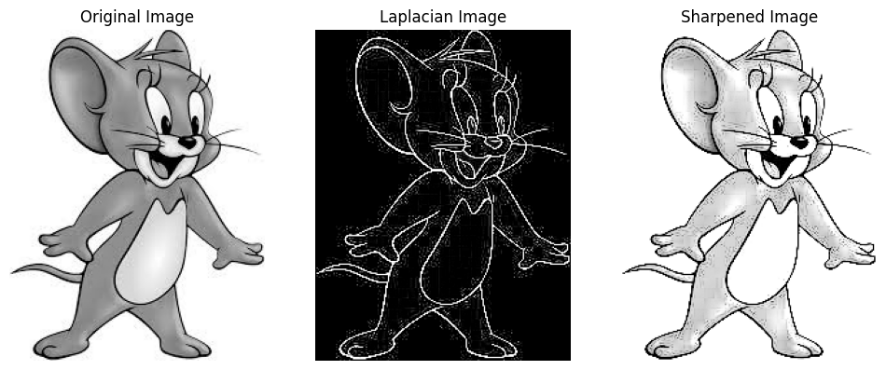
plt.title('Sharpened Image')

plt.imshow(sharpened\_image, cmap='gray')

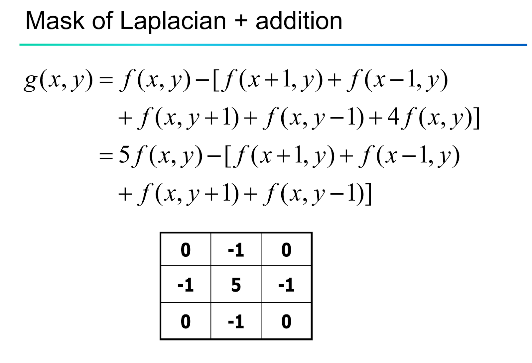
plt.axis('off')

plt.show()

**INPUT & OUTPUT:**



**22. PERFORM SHARPENING OF IMAGE USING LAPLACIAN MASK WITH POSITIVE CENTER COEFFICIENT.**



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

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image in grayscale

image = cv2.imread('/content/pexels-jriika-1704899.jpg', cv2.IMREAD\_GRAYSCALE)

# Define Laplacian kernel with positive center

laplacian\_kernel = np.array([[-1, -1, -1],

[-1, 8, -1],

[-1, -1, -1]])

# Apply the Laplacian kernel

laplacian = cv2.filter2D(image, -1, laplacian\_kernel)

# Sharpened image: add the Laplacian to the original

sharpened = cv2.add(image, laplacian)

# Clip to valid range

sharpened = np.clip(sharpened, 0, 255).astype(np.uint8)

# Display the results

plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)

plt.title("Original Image")

plt.imshow(image, cmap='gray')

plt.subplot(1, 3, 2)

plt.title("Laplacian (Positive Center)")

plt.imshow(laplacian, cmap='gray')

plt.subplot(1, 3, 3)

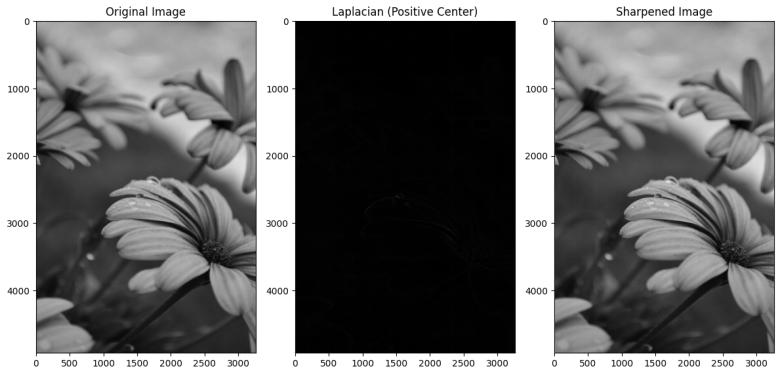
plt.title("Sharpened Image")

plt.imshow(sharpened, cmap='gray')

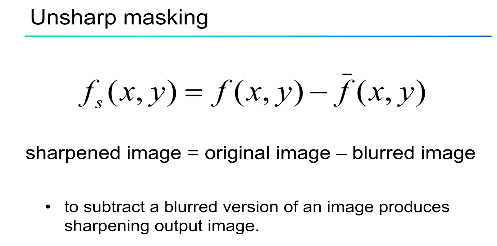
plt.tight\_layout()

plt.show()

**INPUT & OUTPUT:**

****

**23. PERFORM SHARPENING OF IMAGE USING UNSHARP MASKING.**

**AIM:** To Perform Sharpening of Image using unsharp masking.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image in grayscale

image = cv2.imread('/content/pexels-souvenirpixels-1519088.jpg', cv2.IMREAD\_GRAYSCALE)

# Apply Gaussian blur to get the "unsharp" image

blurred = cv2.GaussianBlur(image, (5, 5), sigmaX=1.0)

# Create the unsharp mask

mask = cv2.subtract(image, blurred)

# Set sharpening amount

amount = 1.5

# Add mask to the original image to get the sharpened image

sharpened = cv2.addWeighted(image, 1 + amount, blurred, -amount, 0)

# Clip to valid range

sharpened = np.clip(sharpened, 0, 255).astype(np.uint8)

# Display results

plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)

plt.title("Original Image")

plt.imshow(image, cmap='gray')

plt.subplot(1, 3, 2)

plt.title("Blurred Image")

plt.imshow(blurred, cmap='gray')

plt.subplot(1, 3, 3)

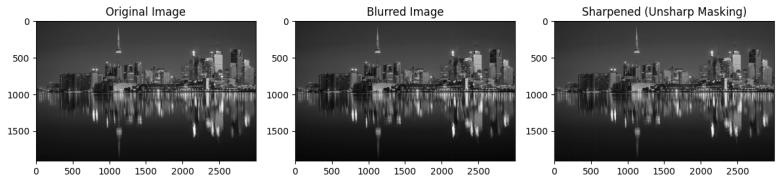
plt.title("Sharpened (Unsharp Masking)")

plt.imshow(sharpened, cmap='gray')

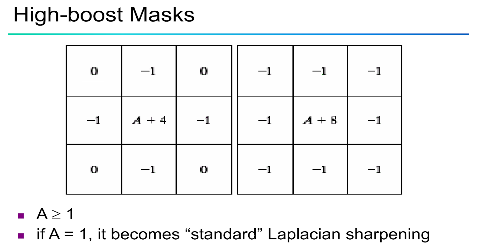
plt.tight\_layout()

plt.show()

**INPUT & OUTPUT:**

****

**24. PERFORM SHARPENING OF IMAGE USING HIGH-BOOST MASKS.**



**AIM:** To Perform Sharpening of Image using High-Boost Masks.

**PROGRAM:**

Import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load image in grayscale

image = cv2.imread('/content/pexels-pixabay-36729.jpg', cv2.IMREAD\_GRAYSCALE)

# Blur the image (low-pass filter)

blurred = cv2.GaussianBlur(image, (5, 5), sigmaX=1.0)

# Set high-boost scaling factor (A > 1)

A = 1.5

# Apply high-boost filter

high\_boost = cv2.addWeighted(image, A, blurred, -(A - 1), 0)

# Clip the result

high\_boost = np.clip(high\_boost, 0, 255).astype(np.uint8)

# Display results

plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)

plt.title("Original Image")

plt.imshow(image, cmap='gray')

plt.subplot(1, 3, 2)

plt.title("Blurred Image")

plt.imshow(blurred, cmap='gray')

plt.subplot(1, 3, 3)

plt.title(f"High-Boost Sharpened (A={A})")

plt.imshow(high\_boost, cmap='gray')

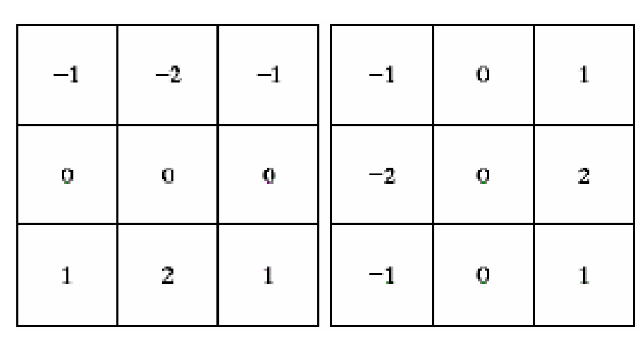
plt.tight\_layout()

plt.show()

**INPUT & OUTPUT:**

****

**25. PERFORM SHARPENING OF IMAGE USING GRADIENT MASKING.**

****

**AIM:** To Perform Sharpening of Image using Gradient masking.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

# Load the image in grayscale

image = cv2.imread('/content/pexels-frans-van-heerden-201846-2375010.jpg', cv2.IMREAD\_GRAYSCALE)

# Compute gradients using Sobel operator

grad\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=3)

grad\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=3)

# Compute gradient magnitude

gradient\_magnitude = np.sqrt(grad\_x\*\*2 + grad\_y\*\*2)

gradient\_magnitude = np.clip(gradient\_magnitude, 0, 255).astype(np.uint8)

# Sharpen the image by adding gradient

alpha = 1.0 # controls sharpening strength

sharpened = cv2.addWeighted(image, 1.0, gradient\_magnitude, alpha, 0)

sharpened = np.clip(sharpened, 0, 255).astype(np.uint8)

# Display the images

plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)

plt.title("Original Image")

plt.imshow(image, cmap='gray')

plt.subplot(1, 3, 2)

plt.title("Gradient Magnitude")

plt.imshow(gradient\_magnitude, cmap='gray')

plt.subplot(1, 3, 3)

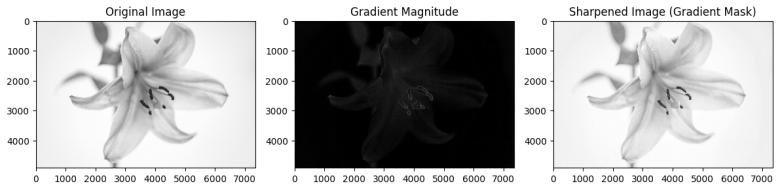
plt.title("Sharpened Image (Gradient Mask)")

plt.imshow(sharpened, cmap='gray')

plt.tight\_layout()

plt.show()

**INPUT & OUTPUT**

****

**26. INSERT WATER MARKING TO THE IMAGE USING OPENCV.**

**AIM:** To Insert water marking to the image using OpenCV.

**PROGRAM:**

!pip install -q opencv-python-headless

image\_path = "/content/sample\_image.jpg" # Change this to your image file path

print(f"Using image: {image\_path}")

# Step 3: Import libraries

import cv2

import numpy as np

from matplotlib import pyplot as plt

# Step 4: Load and process image

image = cv2.imread(image\_path)

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Step 5: Watermark settings

watermarked = image.copy()

cv2.putText(watermarked, "Watermark", (10, image.shape[0] - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE\_AA)

# Step 6: Display original and watermarked images

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(image)

plt.title("Original Image")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(watermarked)

plt.title("Watermarked Image")

plt.axis("off")

plt.show()

**INPUT & OUTPUT:**



**27. DO CROPPING, COPYING AND PASTING IMAGE INSIDE ANOTHER IMAGE USING OPENCV.**

**AIM:** To Do Cropping, Copying and pasting image inside another image using OpenCV.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from google.colab import files

# Step 1: Upload source image

print("Upload source image (from which to crop):")

uploaded = files.upload()

src\_path = list(uploaded.keys())[0]

# Step 2: Upload destination image

print("Upload destination image (where to paste):")

uploaded = files.upload()

dst\_path = list(uploaded.keys())[0]

# Step 3: Load images

src\_img = cv2.imread(src\_path)

dst\_img = cv2.imread(dst\_path)

# Optional: Resize destination image to match source image

dst\_img = cv2.resize(dst\_img, (src\_img.shape[1], src\_img.shape[0]))

# Step 4: Crop region from source image

x, y, w, h = 20, 20, 50, 50  # Coordinates and size of crop

cropped\_region = src\_img[y:y+h, x:x+w]

# Step 5: Paste cropped region into destination image

paste\_x, paste\_y = 100, 100

dst\_img[paste\_y:paste\_y+h, paste\_x:paste\_x+w] = cropped\_region

# Step 6: Convert BGR to RGB for Matplotlib display

dst\_img\_rgb = cv2.cvtColor(dst\_img, cv2.COLOR\_BGR2RGB)

# Step 7: Display using matplotlib

plt.figure(figsize=(8, 6))

plt.imshow(dst\_img\_rgb)

plt.title("Image After Cropping and Pasting")

plt.axis('off')

plt.show()

**OUTPUT:**



28. **FIND THE BOUNDARY OF THE IMAGE USING CONVOLUTION KERNEL FOR THE GIVEN IMAGE.**

**AIM:** To Find the boundary of the image using Convolution kernel for the given image.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from google.colab import files

# Step 1: Upload image

print("Upload the image to detect boundaries:")

uploaded = files.upload()

img\_path = list(uploaded.keys())[0]

# Step 2: Read the image and convert to grayscale

img = cv2.imread(img\_path)

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

# Step 3: Define a simple Laplacian kernel

laplacian\_kernel = np.array([[0, 1, 0],

                             [1, -4, 1],

                             [0, 1, 0]])

# Step 4: Apply convolution using filter2D

boundary = cv2.filter2D(gray, -1, laplacian\_kernel)

# Step 5: Display original and boundary-detected images

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.imshow(gray, cmap='gray')

plt.title("Original Grayscale Image")

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(boundary, cmap='gray')

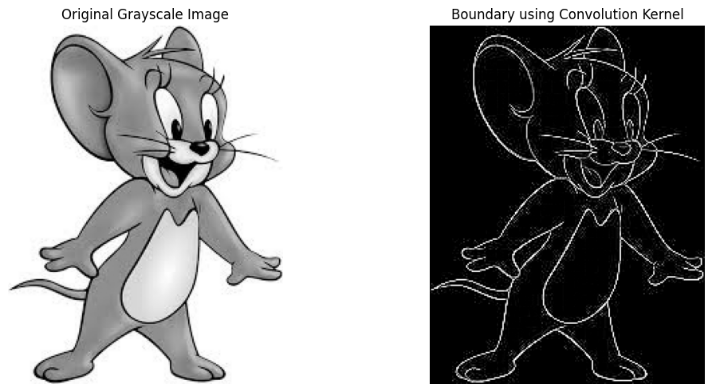
plt.title("Boundary using Convolution Kernel")

plt.axis('off')

plt.tight\_layout()

plt.show()

**INPUT & OUTPUT:**



**29. MORPHOLOGICAL OPERATIONS BASED ON OPENCV USING EROSION TECHNIQUE.**

**AIM:** To Morphological operations based on OpenCV using Erosion technique.

**PROGRAM:**

!pip install opencv-python-headless

# Step 2: Import libraries

import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab import files

# Step 3: Upload an image

print("Upload a binary or grayscale image:")

uploaded = files.upload()

image\_path = next(iter(uploaded))

# Step 4: Read the image

img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

# Step 5: Threshold (if needed) to make binary

# Convert grayscale to binary (0 or 255)

\_, binary = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

# Step 6: Define kernel (structuring element)

kernel = np.ones((5, 5), np.uint8) # 5x5 square kernel

# Step 7: Apply erosion

eroded = cv2.erode(binary, kernel, iterations=1)

# Step 8: Show results

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(binary, cmap='gray')

plt.title('Original Binary Image')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(eroded, cmap='gray')

plt.title('Eroded Image')

plt.axis('off')

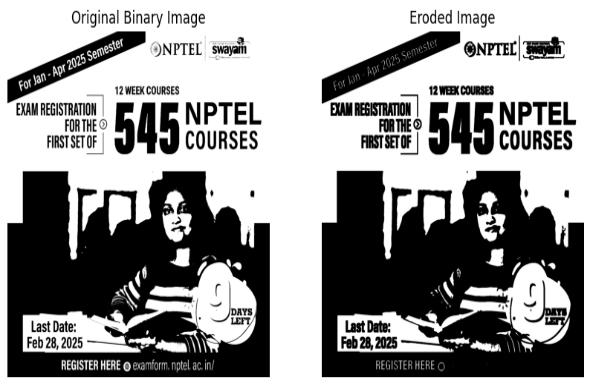
plt.show()

# Step 9: Save the output

cv2.imwrite("eroded\_output.jpg", eroded)

print("Saved as eroded\_output.jpg")

**INPUT & OUTPUT:**

****

**30. MORPHOLOGICAL OPERATIONS BASED ON OPENCV USING DILATION TECHNIQUE.**

**AIM:** To Morphological operations based on OpenCV using Dilation technique.

**PROGRAM:**

!pip install opencv-python-headless

# Step 2: Import required libraries

import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab import files

# Step 3: Upload an image

print("Upload a binary or grayscale image:")

uploaded = files.upload()

image\_path = next(iter(uploaded))

# Step 4: Read the image in grayscale

img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

# Step 5: Convert to binary (thresholding)

\_, binary = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

# Step 6: Define the structuring element (kernel)

kernel = np.ones((5, 5), np.uint8) # Square 5x5 kernel

# Step 7: Apply dilation

dilated = cv2.dilate(binary, kernel, iterations=1)

# Step 8: Display the original and dilated images

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(binary, cmap='gray')

plt.title('Original Binary Image')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(dilated, cmap='gray')

plt.title('Dilated Image')

plt.axis('off')

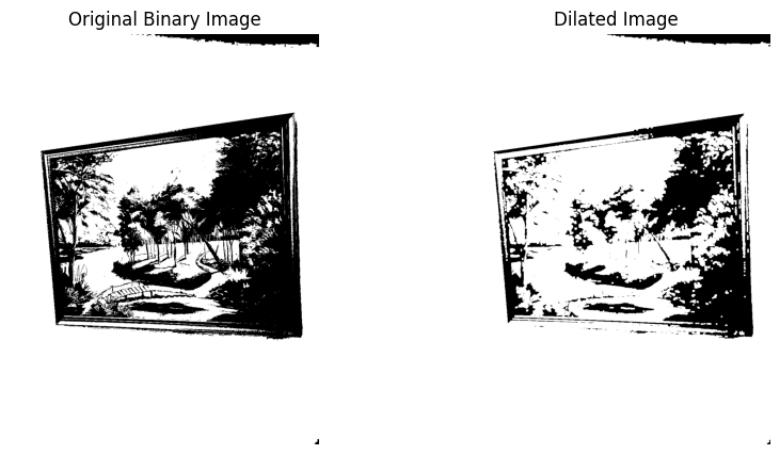
plt.show()

# Step 9: Save the dilated image

cv2.imwrite("dilated\_output.jpg", dilated)

print("Saved as dilated\_output.jpg")

**INPUT & OUTPUT:**

****

**31. MORPHOLOGICAL OPERATIONS BASED ON OPENCV USING OPENING TECHNIQUE.**

**Aim:** To Morphological operations based on OpenCV using Opening technique.

**PROGRAM:**

!pip install opencv-python-headless

# Step 2: Import necessary libraries

import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab import files

# Step 3: Upload an image

print("Upload a binary or grayscale image:")

uploaded = files.upload()

image\_path = next(iter(uploaded))

# Step 4: Read the image in grayscale

img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

# Step 5: Convert to binary image using thresholding

\_, binary = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

# Step 6: Define the structuring element (kernel)

kernel = np.ones((5, 5), np.uint8) # You can adjust the size

# Step 7: Apply Opening (Erosion followed by Dilation)

opened = cv2.morphologyEx(binary, cv2.MORPH\_OPEN, kernel)

# Step 8: Display the original and opened images

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(binary, cmap='gray')

plt.title('Original Binary Image')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(opened, cmap='gray')

plt.title('After Opening')

plt.axis('off')

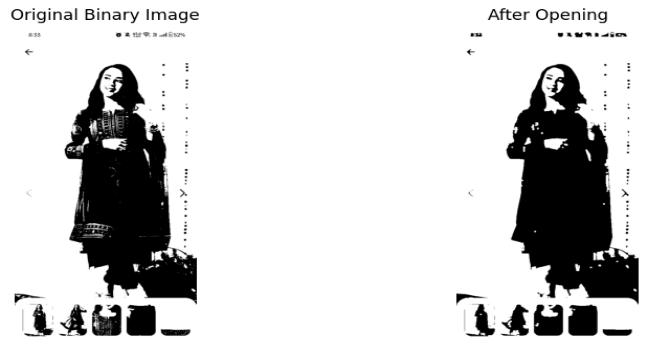
plt.show()

# Step 9: Save the opened image

cv2.imwrite("opened\_output.jpg", opened)

print("Saved as opened\_output.jpg")

**INPUT & OUTPUT:**

****

**32. MORPHOLOGICAL OPERATIONS BASED ON OPENCV USING CLOSING TECHNIQUE.**

**AIM:** To Morphological operations based on OpenCV using Closing technique.

**PROGRAM:**

!pip install opencv-python-headless

# Step 2: Import necessary libraries

import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab import files

# Step 3: Upload an image

print("Upload a binary or grayscale image:")

uploaded = files.upload()

image\_path = next(iter(uploaded))

# Step 4: Read image in grayscale

img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

# Step 5: Convert to binary image using thresholding

\_, binary = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

# Step 6: Define the structuring element (kernel)

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

# Step 7: Apply Closing (Dilation followed by Erosion)

closed = cv2.morphologyEx(binary, cv2.MORPH\_CLOSE, kernel)

# Step 8: Display original and closed image

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(binary, cmap='gray')

plt.title('Original Binary Image')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(closed, cmap='gray')

plt.title('After Closing')

plt.axis('off')

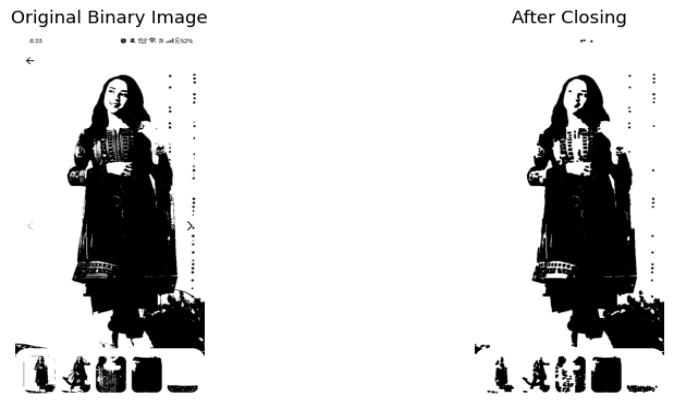
plt.show()

# Step 9: Save the result

cv2.imwrite("closed\_output.jpg", closed)

print("Saved as closed\_output.jpg")

**INPUT & OUTPUT:**

****

**33. MORPHOLOGICAL OPERATIONS BASED ON OPENCV USING MORPHOLOGICAL GRADIENT TECHNIQUE.**

**AIM:** To Morphological operations based on OpenCV using Morphological Gradient technique.

**PROGRAM:**

!pip install opencv-python-headless

# Step 2: Import required libraries

import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab import files

# Step 3: Upload a binary or grayscale image

print("Upload a binary or grayscale image:")

uploaded = files.upload()

image\_path = next(iter(uploaded))

# Step 4: Read the image

img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

# Step 5: Threshold to ensure binary format (optional for clean input)

\_, binary = cv2.threshold(img, 127, 255, cv2.THRESH\_BINARY)

# Step 6: Define kernel

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

# Step 7: Apply Morphological Gradient

gradient = cv2.morphologyEx(binary, cv2.MORPH\_GRADIENT, kernel)

# Step 8: Show original and gradient images

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(binary, cmap='gray')

plt.title('Original Binary Image')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(gradient, cmap='gray')

plt.title('Morphological Gradient')

plt.axis('off')

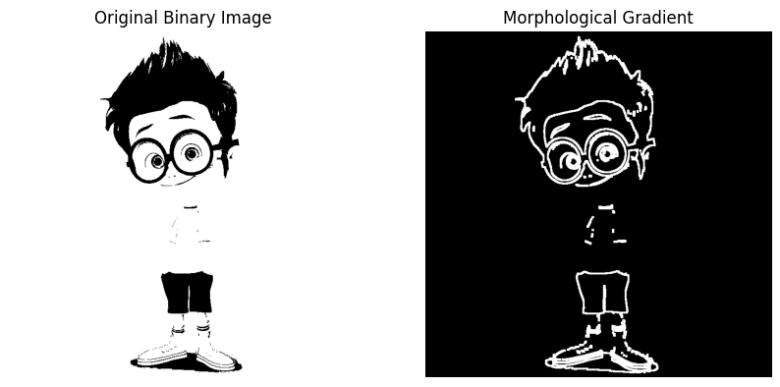
plt.show()

# Step 9: Save the result

cv2.imwrite("gradient\_output.jpg", gradient)

print("Saved as gradient\_output.jpg")

**INPUT & OUTPUT:**

****

**34. MORPHOLOGICAL OPERATIONS BASED ON OPENCV USING TOP HAT TECHNIQUE.**

**AIM**: To Morphological operations based on OpenCV using Top hat technique.

**PROGRAM:**

!pip install opencv-python-headless

# Step 2: Import necessary libraries

import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab import files

# Step 3: Upload an image

print("Upload a grayscale image:")

uploaded = files.upload()

image\_path = next(iter(uploaded))

# Step 4: Read image in grayscale

img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

# Step 5: Define kernel (structuring element)

kernel = np.ones((15, 15), np.uint8) # Larger kernel enhances effect

# Step 6: Apply Top-Hat transformation

tophat = cv2.morphologyEx(img, cv2.MORPH\_TOPHAT, kernel)

# Step 7: Display results

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(img, cmap='gray')

plt.title('Original Image')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(tophat, cmap='gray')

plt.title('Top-Hat Result')

plt.axis('off')

plt.show()

# Step 8: Save the result

cv2.imwrite("tophat\_output.jpg", tophat)

print("Saved as tophat\_output.jpg")

**INPUT & OUTPUT:**

****

**35. MORPHOLOGICAL OPERATIONS BASED ON OPENCV USING BLACK HAT TECHNIQUE.**

**AIM:** To Morphological operations based on OpenCV using Top hat technique.

**PROGRAM:**

!pip install opencv-python-headless

# Step 2: Import required libraries

import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab import files

# Step 3: Upload an image

print("Upload a grayscale image:")

uploaded = files.upload()

image\_path = next(iter(uploaded))

# Step 4: Read the image in grayscale

img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

# Step 5: Define the structuring element (kernel)

kernel = np.ones((15, 15), np.uint8)

# Step 6: Apply Black-Hat transformation

blackhat = cv2.morphologyEx(img, cv2.MORPH\_BLACKHAT, kernel)

# Step 7: Display the results

plt.figure(figsize=(10, 5)

plt.subplot(1, 2, 1)

plt.imshow(img, cmap='gray')

plt.title('Original Image')

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(blackhat, cmap='gray')

plt.title('Black-Hat Result')

plt.axis('off')

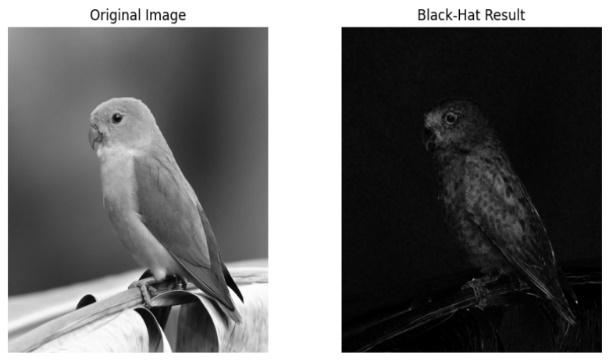
plt.show()

# Step 8: Save the result

cv2.imwrite("blackhat\_output.jpg", blackhat)

print("Saved as blackhat\_output.jpg")

**INPUT & OUTPUT:**

****

**36. Recognise watch from the given image by general Object recognition using OpenCV.**

**AIM:** To recognise watch from the given image by general Object recognition using OpenCV.

PROGRAM:

# STEP 1: Install YOLO

!pip install ultralytics

# STEP 2: Upload Image

from google.colab import files

uploaded = files.upload()

filename = next(iter(uploaded))

# STEP 3: Load YOLO Model and Run Detection

from ultralytics import YOLO

from PIL import Image

import matplotlib.pyplot as plt

model = YOLO("yolov8n.pt")  # Small & fast model

# Load and detect

img = Image.open(filename)

results = model(img)

# STEP 4: Show Image with Detections

results[0].show()

# STEP 5: Optional - Print Detected Classes

print("Detected objects:")

for box in results[0].boxes:

    cls\_id = int(box.cls[0])

    name = model.names[cls\_id]

    print("-", name)

**INPUT AND OUTPUT:**

****

**37. Using Opencv play Video in Reverse mode.**

**AIM:** To use Opencv  to play  theVideo in Reverse mode.

**PROGRAM:**

import cv2

# Path to your video file

video\_path = 'your\_video.mp4'  # Replace with your video path

# Open the video

cap = cv2.VideoCapture(video\_path)

# Get total number of frames

frame\_count = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))

# Get frame rate

fps = cap.get(cv2.CAP\_PROP\_FPS)

# Get video frame dimensions

width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

# Create a window

cv2.namedWindow("Reverse Video", cv2.WINDOW\_NORMAL)

# Loop over the frames in reverse order

for i in range(frame\_count - 1, -1, -1):

    cap.set(cv2.CAP\_PROP\_POS\_FRAMES, i)

    ret, frame = cap.read()

    if not ret:

        continue

    frame = cv2.resize(frame, (width, height))

    cv2.imshow('Reverse Video', frame)

    # Press ESC to break

    if cv2.waitKey(int(1000 / fps)) & 0xFF == 27:

        break

# Release everything

cap.release()

cv2.destroyAllWindows()

**INPUT AND OUPUT:**

****

**38. FACE DETECTION USING OPENCV.**

**AIM:** To Detect Face using Opencv.

**PROGRAM:**

import cv2

from google.colab.patches import cv2\_imshow

from google.colab import files

import numpy as np

# Upload the image

uploaded = files.upload()

file\_name = next(iter(uploaded))

# Load Haar cascade for face detection

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

# Read the original image

original = cv2.imread(file\_name)

# Convert to grayscale for face detection

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

# Detect faces

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

# Make a copy of original image to draw rectangles on

output = original.copy()

# Draw rectangles around faces

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

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

# Resize images if needed (optional, to ensure same height)

# Here we assume images are same size as original and output are same size

# Stack images side by side (horizontal stack)

combined = np.hstack((original, output))

# Display the combined image

cv2\_imshow(combined)

**INPUT AND OUTPUT**



**39. Vehicle Detection in a Video frame using OpenCV .**

**AIM: To Vehicle Detection in a Video frame using OpenCV .**

**PROGRAM:**

pip install opencv-python

import cv2

# Load video

video\_path = 'traffic\_video.mp4'  # Replace with your video path

cap = cv2.VideoCapture(video\_path)

# Create background subtractor

fgbg = cv2.createBackgroundSubtractorMOG2(history=100, varThreshold=40)

# Define kernel for morphological operations

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (5, 5))

while cap.isOpened():

    ret, frame = cap.read()

    if not ret:

        break

    # Resize frame for consistency

    frame = cv2.resize(frame, (800, 450))

    # Apply background subtraction

    fgmask = fgbg.apply(frame)

    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH\_OPEN, kernel)

    # Find contours in the mask

    contours, \_ = cv2.findContours(fgmask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

    for cnt in contours:

        # Filter small contours

        if cv2.contourArea(cnt) < 500:

            continue

        x, y, w, h = cv2.boundingRect(cnt)

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

        cv2.putText(frame, 'Vehicle', (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (36,255,12), 2)

    # Show result

    cv2.imshow('Vehicle Detection', frame)

    if cv2.waitKey(30) & 0xFF == 27:  # Press ESC to exit

        break

cap.release()

cv2.destroyAllWindows**()**

**INPUT AND OUTPUT:**

****

**40. DRAW RECTANGULAR SHAPE AND EXTRACT OBJECTS.**

**AIM:** To Draw Rectangular shape and extract objects.

**PROGRAM:**

from PIL import Image

import numpy as np

import cv2

from google.colab.patches import cv2\_imshow

from google.colab import files

import os

# Upload file

uploaded = files.upload()

file\_name = next(iter(uploaded))

print("Uploaded file:", file\_name)

# Rename file to avoid spaces in filename

new\_name = file\_name.replace(" ", "\_")

if new\_name != file\_name:

    os.rename(file\_name, new\_name)

    print(f"Renamed '{file\_name}' to '{new\_name}'")

else:

    new\_name = file\_name

# Load image with Pillow

pil\_img = Image.open(new\_name)

# Convert PIL image to OpenCV format

image = cv2.cvtColor(np.array(pil\_img), cv2.COLOR\_RGB2BGR)

# Convert to grayscale

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

# Load Haar cascade

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

# Detect faces

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

print(f"Detected {len(faces)} faces.\n")

# Draw rectangles and extract faces

output\_folder = 'extracted\_faces'

os.makedirs(output\_folder, exist\_ok=True)

for i, (x, y, w, h) in enumerate(faces, start=1):

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

    face\_crop = image[y:y+h, x:x+w]

    face\_filename = os.path.join(output\_folder, f'face\_{i}.jpg')

    cv2.imwrite(face\_filename, face\_crop)

    print(f"Extracted face {i} saved as: {face\_filename}")

    cv2\_imshow(face\_crop)

print("Original image with detected faces:")

cv2\_imshow(image)

**INPUT & OUTPUT:**

