## SIMATS ENGINEERING

## DSA0201-Computer Vision with OpenCV

## for Robotics and Automation

**NAME: U.Jaya Krishna**

**Register number:192324060**

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 to Convert an Image to Grayscale using python.

**Program:**

import cv2

from matplotlib import pyplot as plt

filename = '1.PNG'

image = cv2.imread(filename)

if image is None:

print("Error: Image not found. Make sure '1.png' exists in your Colab files.")

else:

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

# Step 6: Display the images

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

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.subplot(1, 2, 2)

plt.title("Grayscale Image")

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

plt.axis('off')

plt.tight\_layout()

plt.show()

****

1. **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 image operations to convert an image to blur using python.

**Program:**

import cv2

from matplotlib import pyplot as plt

image = cv2.imread('Annotation 2025-04-29 144420.png')

if image is None:

print("Error: Image not found or couldn't be loaded.")

else:

blurred\_image = cv2.GaussianBlur(image, (15, 15), 0) # (15,15) is the kernel size

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

plt.subplot(1, 2, 1)

plt.title('Original Image')

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.subplot(1, 2, 2)

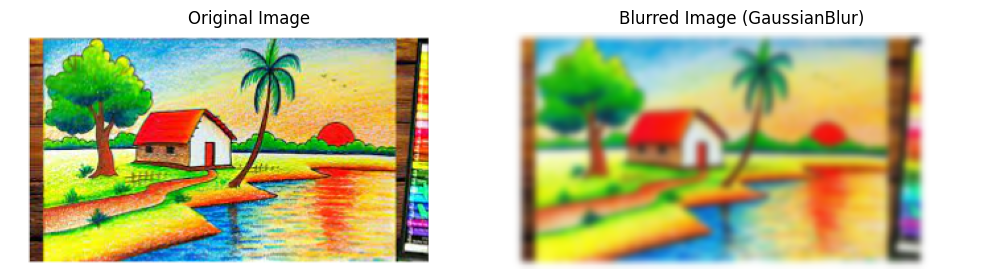
plt.title('Blurred Image (GaussianBlur)')

plt.imshow(cv2.cvtColor(blurred\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.tight\_layout()

plt.show()

****

**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 image operations to convert an image to canny edge using python.

**Program:**

import cv2

from matplotlib import pyplot as plt

image = cv2.imread('3.png')

if image is None:

print("Error: Image not found or couldn't be loaded.")

else:

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

edges = cv2.Canny(gray, threshold1=100, threshold2=200)

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

plt.subplot(1, 2, 1)

plt.title('Original Image')

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.subplot(1, 2, 2)

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

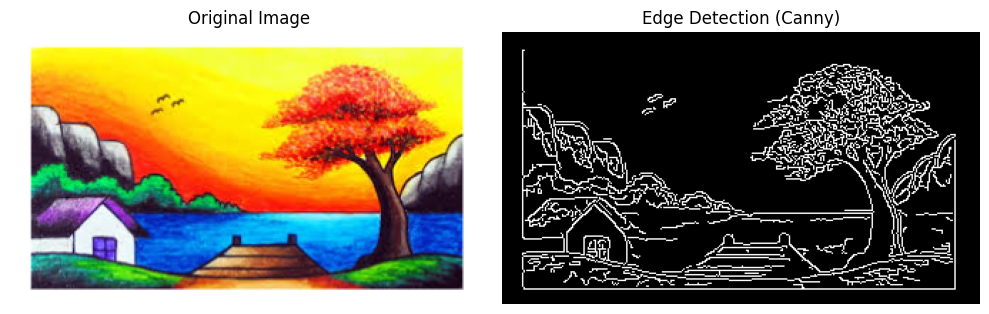
plt.imshow(edges, cmap='gray')

plt.axis('off')

plt.tight\_layout()

plt.show()

**Output:**



**4.Implement histogram equalization on the given image and compare it with the original image using Open CV.**

**AIM:**

To perform Histogram Equalization on an image using OpenCV in Python and compare it with the original image.

**PROGRAM:**

import cv2

import numpy as np

image = cv2.imread("sample.jpg") # Replace with your image file

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

equalized\_image = cv2.equalizeHist(gray\_image)

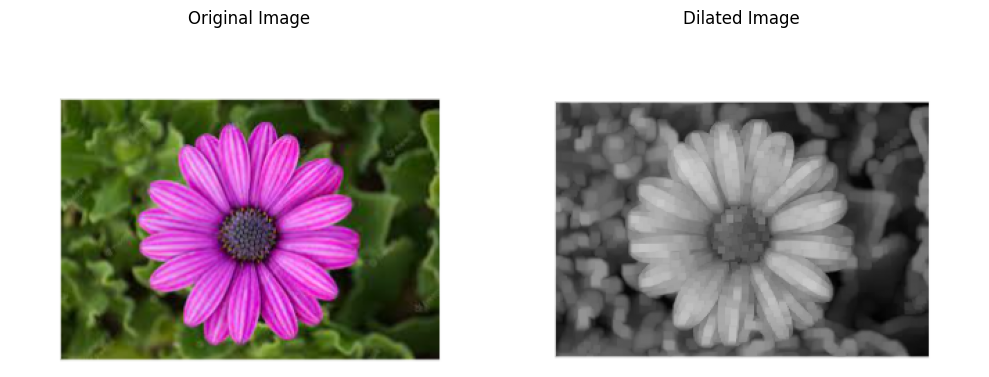
cv2.imshow("Original Grayscale Image", gray\_image)

cv2.imshow("Histogram Equalized Image", equalized\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**OUTPUT:**



**5. Write a Python function to analyze the histogram of the given input image based on color levels using Open CV.**

**AIM:**

To write a Python function to analyze and display the histogram of an image based on color levels using OpenCV.

**PROGRAM:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def analyze\_histogram(image\_path):

image = cv2.imread(image\_path)

color\_channels = ('b', 'g', 'r')

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

for i, color in enumerate(color\_channels):

histogram = cv2.calcHist([image], [i], None, [256], [0, 256])

plt.plot(histogram, color=color, label=f"{color.upper()} Channel")

plt.xlim([0, 256]) # Pixel intensity range

plt.title("Color Histogram Analysis")

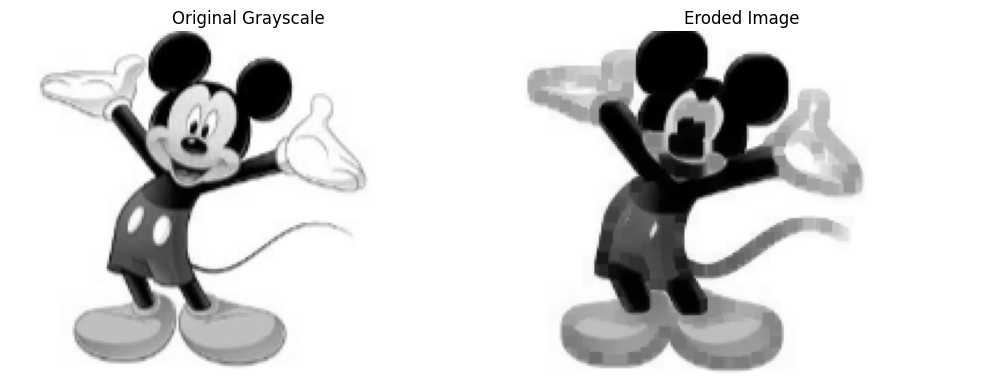
plt.xlabel("Pixel Intensity")

plt.ylabel("Frequency")

plt.legend()

plt.show()

analyze\_histogram("sample.jpg") # Replace with your image file

**OUTPUT:**

**6. Perform basic Image Handling and processing operations on the image is to read an image in python and Convert an Image to erode using Erode function.**

**AIM:**

To read an image and convert it to grayscale using OpenCV in Python.

**PROGRAM:**

import cv2

image = cv2.imread("sample.jpg") # Replace 'sample.jpg' with your image path

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

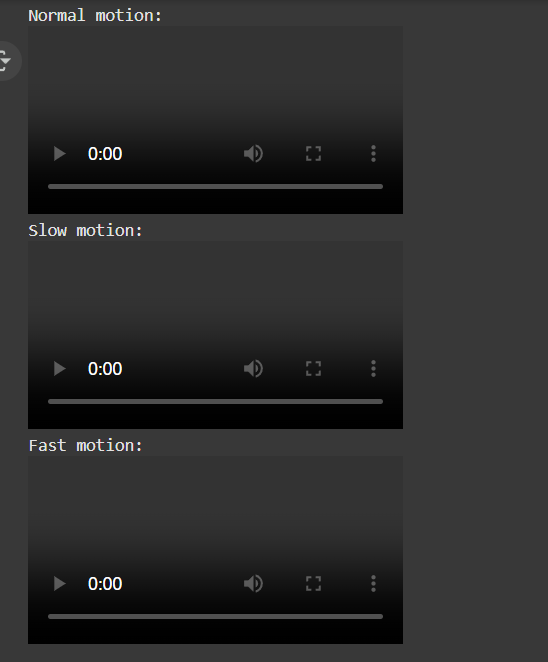
cv2.imshow("Original Image", image)

cv2.imshow("Grayscale Image", gray\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output:**



**7.** **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 an image and convert it to grayscale using OpenCV in Python.

**PROGRAM:**

import cv2

image = cv2.imread("sample.jpg") # Replace 'sample.jpg' with your image path

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

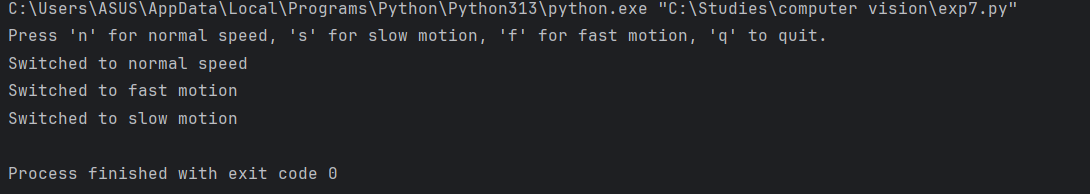
cv2.imshow("Original Image", image)

cv2.imshow("Grayscale Image", gray\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**output:**



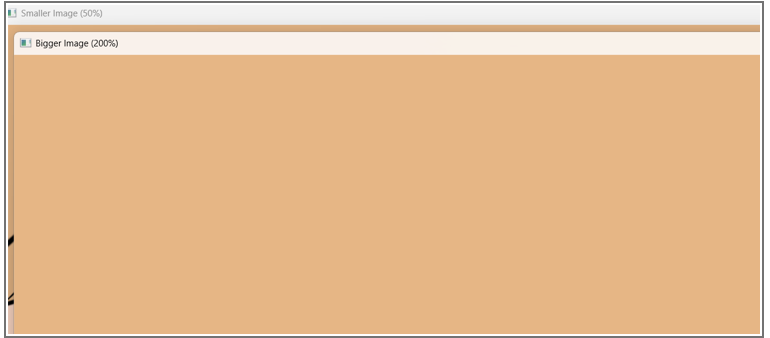
**8.** **Scaling an image to its Bigger and Smaller sizes.**

**AIM:** Scaling an image to its Bigger and Smaller sizes.

**Program:**

import cv2  
image = cv2.imread('1299888.png')  
  
if image is None:  
 print("Error: Could not load image.")  
 exit()  
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 (50%)', smaller)  
cv2.imshow('Bigger Image (200%)', bigger)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**output:**



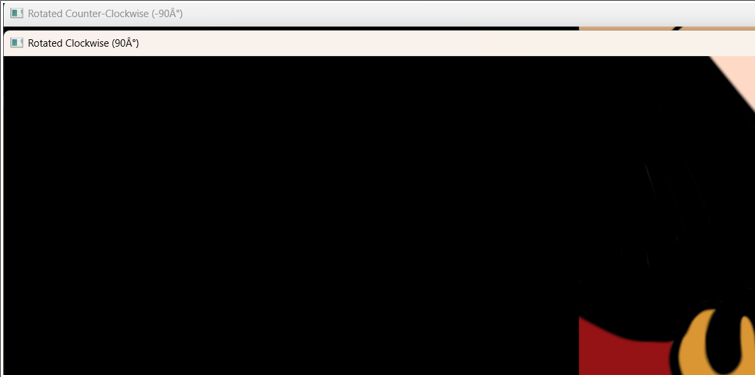
**9. Perform Rotation of an image to clockwise and counter clockwise direction.**

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

**Program:**

import cv2  
image = cv2.imread('1299888.png')  
  
if image is None:  
 print("Error: Could not load image.")  
 exit()  
(h, w) = image.shape[:2]  
center = (w // 2, h // 2)  
M\_clockwise = cv2.getRotationMatrix2D(center, -90, 1.0)  
rotated\_clockwise = cv2.warpAffine(image, M\_clockwise, (w, h))  
M\_counter = cv2.getRotationMatrix2D(center, 90, 1.0)  
rotated\_counter = cv2.warpAffine(image, M\_counter, (w, h))  
cv2.imshow("Original Image", image)  
cv2.imshow("Rotated Clockwise (90°)", rotated\_clockwise)  
cv2.imshow("Rotated Counter-Clockwise (-90°)", rotated\_counter)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**Output:**

****

**10. Perform moving of an image from one place to another.**

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

**Program:**

import cv2  
import numpy as np  
image = cv2.imread('1299888.png')  
  
if image is None:  
 print("Error: Could not load image.")  
 exit()  
tx = 200  
ty = 150  
translation\_matrix = np.float32([[1, 0, tx], [0, 1, ty]])  
moved\_image = cv2.warpAffine(image, translation\_matrix, (image.shape[1], image.shape[0]))  
cv2.imshow("Original Image", image)  
cv2.imshow("Moved Image", moved\_image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()

**Output:**

****

**11. Perform Affine Transformation on the image.**

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

**Program:**

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

image = cv2.imread(r"pexels-souvenirpixels-414612.jpg")

rows, cols, ch = image.shape

pts1 = np.float32([[50, 50], [200, 50], [50, 200]])

pts2 = np.float32([[10, 100], [200, 50], [100, 250]])

matrix = cv2.getAffineTransform(pts1, pts2)

transformed\_image = cv2.warpAffine(image, matrix, (cols, rows))

cv2\_imshow(image)

cv2\_imshow(transformed\_image)

**output:**

****

**12. Perform Perspective Transformation on the image.**

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

**Program:**

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

image = cv2.imread("pexels-souvenirpixels-414612.jpg")

if image is None:

print("Error: Image not found!")

exit()

pts1 = np.float32([[100, 100], [400, 100], [100, 400], [400, 400]])

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

matrix = cv2.getPerspectiveTransform(pts1, pts2)

output = cv2.warpPerspective(image, matrix, (300, 300))

cv2\_imshow(image)

cv2\_imshow(output)

cv2.imwrite("perspective\_transformed.jpg", output)

**output:**

****

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

**AIM:** Perform Perspective Transformation on the Video.

**Program:**

import cv2

import numpy as np

cap = cv2.VideoCapture("nature\_video.mp4")

out = cv2.VideoWriter("transformed\_video.mp4", cv2.VideoWriter\_fourcc(\*'mp4v'),

cap.get(cv2.CAP\_PROP\_FPS), (300, 300))

w = int(cap.get(3))

h = int(cap.get(4))

pts1 = np.float32([[50, 50], [w-50, 50], [50, h-50], [w-50, h-50]])

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

matrix = cv2.getPerspectiveTransform(pts1, pts2)

while True:

ret, frame = cap.read()

if not ret:

break

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

out.write(warped)

cap.release()

out.release()

**Output:**

Video

**14. Perform transformation using Homography matrix.**

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

**Program:**

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

image = cv2.imread('cv 14.jpg')

if image is None:

print("Error: Image not found!")

else:

src\_pts = np.float32([[50, 50], [300, 50], [50, 300], [300, 300]])

dst\_pts = np.float32([[0, 0], [400, 0], [0, 400], [400, 400]])

H, \_ = cv2.findHomography(src\_pts, dst\_pts)

transformed\_image = cv2.warpPerspective(image, H, (400, 400))

cv2\_imshow(image)

cv2\_imshow(transformed\_image)

cv2.imwrite('transformed\_image.jpg', transformed\_image)

Output:

****

**15. Perform transformation using Direct Linear Transformation.**

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

**Program:**

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

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

if image is None:

print("Error: Image not found!")

else:

src\_pts = np.float32([[50, 50], [300, 50], [50, 300], [300, 300]])

dst\_pts = np.float32([[0, 0], [400, 0], [0, 400], [400, 400]])

A = []

for i in range(4):

x1, y1 = src\_pts[i]

x2, y2 = dst\_pts[i]

A.append([x1, y1, 1, 0, 0, 0, -x2\*x1, -x2\*y1, -x2])

A.append([0, 0, 0, x1, y1, 1, -y2\*x1, -y2\*y1, -y2])

A = np.array(A)

\_, \_, Vt = np.linalg.svd(A)

H = Vt[-1].reshape(3, 3)

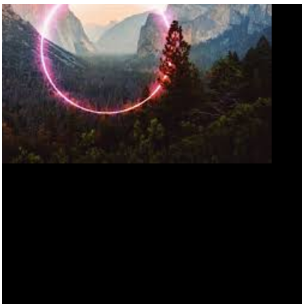
transformed\_image = cv2.warpPerspective(image, H, (400, 400))

cv2\_imshow(image)

cv2\_imshow(transformed\_image)

cv2.imwrite('transformed\_image\_dlt.jpg', transformed\_image)

**Output:**



**16. Perform Edge detection using canny method**

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

**Program:**

import cv2

import matplotlib.pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

blurred = cv2.GaussianBlur(image, (5, 5), 1.4)

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

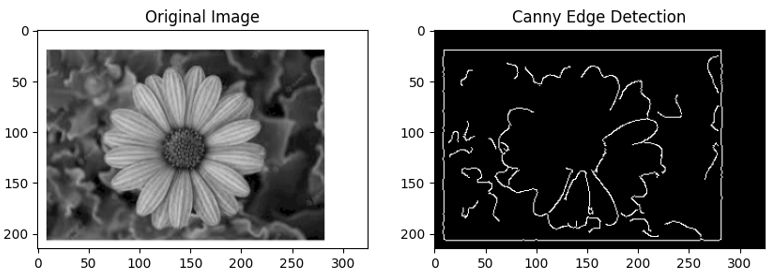
plt.figure(figsize=(10,5))

plt.subplot(1,2,1), plt.imshow(image, cmap='gray'), plt.title('Original Image')

plt.subplot(1,2,2), plt.imshow(edges, cmap='gray'), plt.title('Canny Edge Detection')

plt.show()

**Ouput:**



**17. Perform Edge detection using Sobel Matrix along X axis**

**Aim:** Perform Edge detection using Sobel Matrix along X axis

**Program:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

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

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

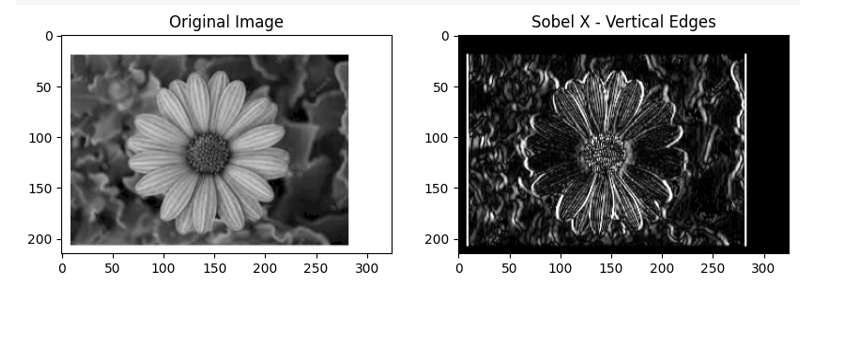
plt.figure(figsize=(10,5))

plt.subplot(1,2,1), plt.imshow(image, cmap='gray'), plt.title('Original Image')

plt.subplot(1,2,2), plt.imshow(sobel\_x, cmap='gray'), plt.title('Sobel X - Vertical Edges')

plt.show()

**Output:**

****

**18. Perform Edge detection using Sobel Matrix along Y axis**

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

**Program:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

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

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

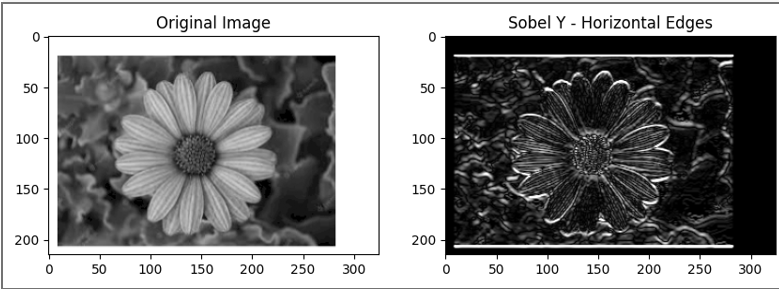
plt.figure(figsize=(10,5))

plt.subplot(1,2,1), plt.imshow(image, cmap='gray'), plt.title('Original Image')

plt.subplot(1,2,2), plt.imshow(sobel\_y, cmap='gray'), plt.title('Sobel Y - Horizontal Edges')

plt.show()

**Output:**



**19. Perform Edge detection using Sobel Matrix along XY axis**

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

**Program:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

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

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

sobel\_combined = cv2.magnitude(sobel\_x, sobel\_y)

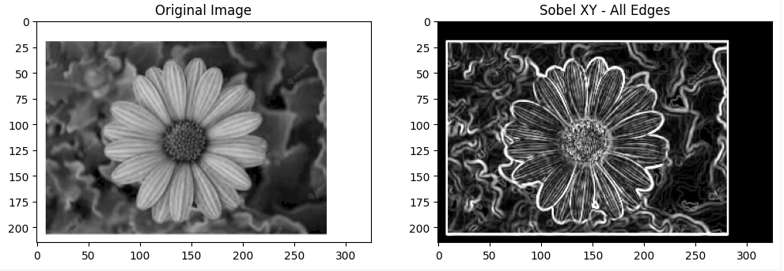
sobel\_combined = cv2.convertScaleAbs(sobel\_combined)

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

plt.subplot(1,2,1), plt.imshow(image, cmap='gray'), plt.title('Original Image')

plt.subplot(1,2,2), plt.imshow(sobel\_combined, cmap='gray'), plt.title('Sobel XY - All Edges')

plt.show()  
**output:**



**20. Perform Sharpening of Image using Laplacian mask with negative center coefficient.**

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

**Program:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('1.png')

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

kernel = np.array([[0, -1, 0],

[-1, 5, -1],

[0, -1, 0]])

sharpened = cv2.filter2D(image\_gray, -1, kernel)

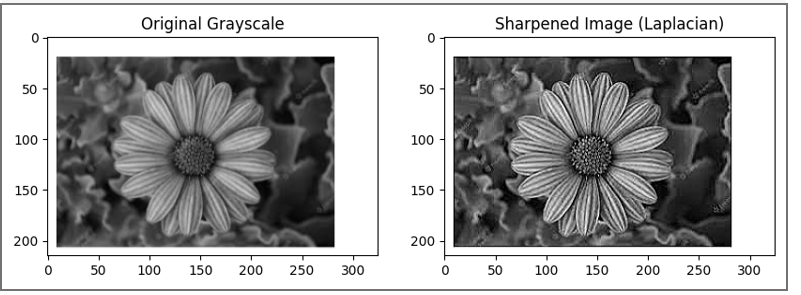
plt.figure(figsize=(10,5))

plt.subplot(1,2,1), plt.imshow(image\_gray, cmap='gray'), plt.title('Original Grayscale')

plt.subplot(1,2,2), plt.imshow(sharpened, cmap='gray'), plt.title('Sharpened Image (Laplacian)')

plt.show()

**Output:**

****

**21. Perform Sharpening of Image using Laplacian mask implemented with an extension of**

**diagonal neighbors,**

**AIM:** 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

image = cv2.imread('1.png')

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

laplacian\_kernel\_8 = np.array([[1, 1, 1],

[1, -8, 1],

[1, 1, 1]])

laplacian = cv2.filter2D(gray, cv2.CV\_64F, laplacian\_kernel\_8)

sharpened = cv2.convertScaleAbs(gray - laplacian)

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale")

plt.axis('off')

plt.subplot(1, 2, 2)

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

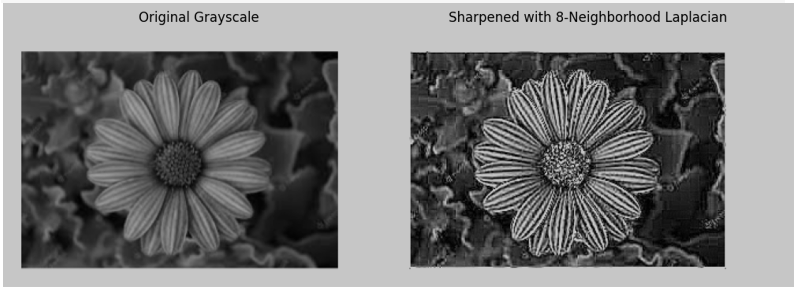
plt.title("Sharpened with 8-Neighborhood Laplacian")

plt.axis('off')

plt.tight\_layout()

plt.show()

**output:**

****

**22. Perform Sharpening of Image using Laplacian mask with positive center coefficient.**

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

**Program:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('1.png')

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

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

[-1, 4, -1],

[0, -1, 0]])

laplacian = cv2.filter2D(gray, cv2.CV\_64F, laplacian\_kernel\_pos)

sharpened = cv2.convertScaleAbs(gray + laplacian)

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale")

plt.axis('off')

plt.subplot(1, 2, 2)

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

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

plt.axis('off')

plt.tight\_layout()

plt.show()

**output:**

****

**23. Perform Sharpening of Image using unsharp masking.**

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

**Program:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread('1.png')

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

blurred = cv2.GaussianBlur(gray, (5, 5), 0)

unsharp\_mask = cv2.addWeighted(gray, 1.5, blurred, -0.5, 0)

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale")

plt.axis('off')

plt.subplot(1, 2, 2)

plt.imshow(unsharp\_mask, cmap='gray')

plt.title("Sharpened using Unsharp Masking")

plt.axis('off')

plt.tight\_layout()

plt.show()

**Output:**

****

**24. Perform Sharpening of Image using High-Boost Masks.**

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

**Program:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

A = 1.5

mask\_4 = np.array([[0, -1, 0],

[-1, A + 4, -1],

[0, -1, 0]])

mask\_8 = np.array([[-1, -1, -1],

[-1, A + 8, -1],

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

sharpened\_4 = cv2.filter2D(image, -1, mask\_4)

sharpened\_8 = cv2.filter2D(image, -1, mask\_8)

plt.figure(figsize=(12, 4))

plt.subplot(1, 3, 1)

plt.title("Original")

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

plt.subplot(1, 3, 2)

plt.title("High-Boost (4-neighbor)")

plt.imshow(sharpened\_4, cmap='gray')

plt.subplot(1, 3, 3)

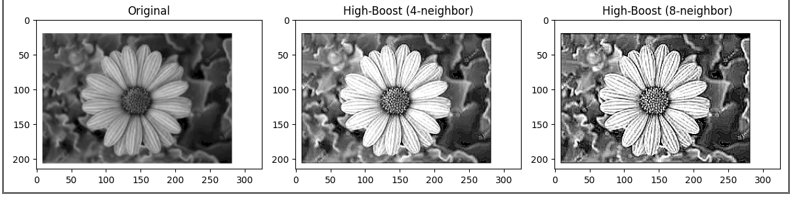
plt.title("High-Boost (8-neighbor)")

plt.imshow(sharpened\_8, cmap='gray')

plt.tight\_layout()

plt.show()

**Output:**



**25. Perform Sharpening of Image using Gradient masking.**

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

**Program:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

gradient\_mask1 = np.array([[-1, -2, -1],

[ 0, 0, 0],

[ 1, 2, 1]])

gradient\_mask2 = np.array([[-1, 0, 1],

[-2, 0, 2],

[-1, 0, 1]])

grad1 = cv2.filter2D(image, -1, gradient\_mask1)

grad2 = cv2.filter2D(image, -1, gradient\_mask2)

gradient\_magnitude = cv2.addWeighted(grad1, 0.5, grad2, 0.5, 0)

sharpened = cv2.add(image, gradient\_magnitude)

plt.figure(figsize=(12, 4))

plt.subplot(1, 4, 1)

plt.title("Original")

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

plt.subplot(1, 4, 2)

plt.title("Gradient Mask 1")

plt.imshow(grad1, cmap='gray')

plt.subplot(1, 4, 3)

plt.title("Gradient Mask 2")

plt.imshow(grad2, cmap='gray')

plt.subplot(1, 4, 4)

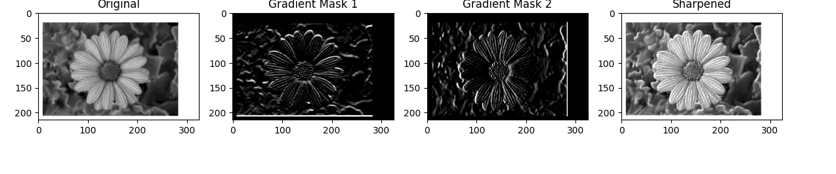
plt.title("Sharpened")

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

plt.tight\_layout()

plt.show()

**Output:**

****

**26. Insert water marking to the image using OpenCV.**

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

**Program:**

import cv2

import numpy as np

image = cv2.imread('1.png')

watermark\_text = "MyWatermark"

font = cv2.FONT\_HERSHEY\_SIMPLEX

font\_scale = 1

thickness = 2

color = (255, 255, 255)

opacity = 0.5

(h, w) = image.shape[:2]

(text\_width, text\_height), \_ = cv2.getTextSize(watermark\_text, font, font\_scale, thickness)

x = w - text\_width - 10

y = h - 10

overlay = image.copy()

cv2.putText(overlay, watermark\_text, (x, y), font, font\_scale, color, thickness)

watermarked = cv2.addWeighted(overlay, opacity, image, 1 - opacity, 0)

cv2.imwrite('2.png', watermarked)

**Output:**

****

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

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

**Program:**

import cv2

image = cv2.imread('1.png')

crop\_y1, crop\_y2 = 50, 150

crop\_x1, crop\_x2 = 100, 200

cropped\_region = image[crop\_y1:crop\_y2, crop\_x1:crop\_x2]

paste\_y, paste\_x = 10, 10

paste\_y2 = paste\_y + cropped\_region.shape[0]

paste\_x2 = paste\_x + cropped\_region.shape[1]

image[paste\_y:paste\_y2, paste\_x:paste\_x2] = cropped\_region

cv2.imwrite('3.png', image)

**Output:**

****

**28. Find the boundary of the image using Convolution kernel for the given image.**

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

**Program:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

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

[1, -4, 1],

[0, 1, 0]])

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

cv2.imwrite('4.png', boundary)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.title("Original")

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

plt.axis('off')

plt.subplot(1, 2, 2)

plt.title("Image Boundary")

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

plt.axis('off')

plt.tight\_layout()

plt.show()

**Output:**

****

**29. Morphological operations based on OpenCV using Erosion technique.**

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

**Program:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

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

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

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

cv2.imwrite('5.png', eroded)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.title("Original Binary")

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

plt.axis('off')

plt.subplot(1, 2, 2)

plt.title("Eroded Image")

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

plt.axis('off')

plt.tight\_layout()

plt.show()

**Output:**

****

**30. Morphological operations based on OpenCV using Dilation technique.**

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

**PROGRAM:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

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

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

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

cv2.imwrite('6.png', dilated)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.title("Original Binary")

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

plt.axis('off')

plt.subplot(1, 2, 2)

plt.title("Dilated Image")

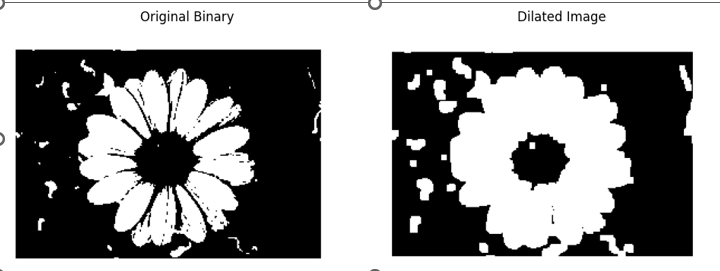
plt.imshow(dilated, cmap='gray')

plt.axis('off')

plt.tight\_layout()

plt.show()

output:

****

**31. Morphological operations based on OpenCV using Opening technique.**

**AIM:** Morphological operations based on OpenCV using Opening technique.

**PROGRAM:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

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

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

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

cv2.imwrite('7.png', opened)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.title("Original Binary")

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

plt.axis('off')

plt.subplot(1, 2, 2)

plt.title("After Opening")

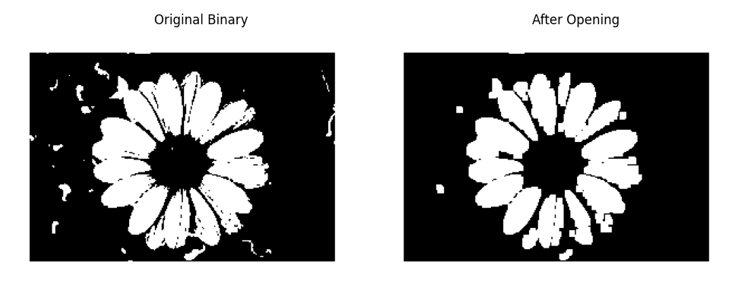
plt.imshow(opened, cmap='gray')

plt.axis('off')

plt.tight\_layout()

plt.show()

**OUTPUT:**



**32. Morphological operations based on OpenCV using Closing technique.**

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

**PROGRAM:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

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

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

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

cv2.imwrite('8.png', closed)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.title("Original Binary")

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

plt.axis('off')

plt.subplot(1, 2, 2)

plt.title("After Closing")

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

plt.axis('off')

plt.tight\_layout()

plt.show()

**OUTPUT:**

**33. Morphological operations based on OpenCV using Morphological Gradient technique.**

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

**PROGRAM:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

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

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

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

cv2.imwrite('9.png', gradient)

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

plt.title("Original Binary")

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

plt.axis('off')

plt.subplot(1, 2, 2)

plt.title("Morphological Gradient")

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

plt.axis('off')

plt.tight\_layout()

plt.show()

**OUTPUT:**



**34. Morphological operations based on OpenCV using Top hat technique.**

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

**PROGRAM:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', 0)

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (15, 15))

top\_hat = cv2.morphologyEx(image, cv2.MORPH\_TOPHAT, kernel)

plt.subplot(1, 2, 1)

plt.title('Original Image')

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

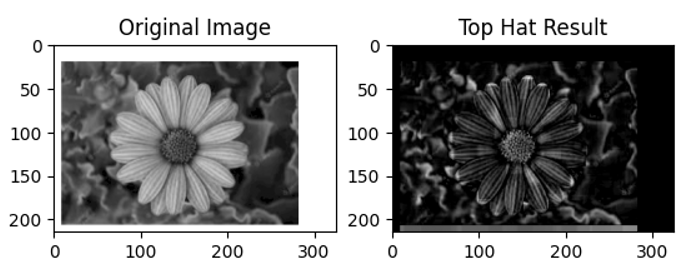
plt.subplot(1, 2, 2)

plt.title('Top Hat Result')

plt.imshow(top\_hat, cmap='gray')

plt.show()

**OUTPUT:**



**35. Morphological operations based on OpenCV using Black hat technique.**

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

**PROGRAM:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('1.png', 0)

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (15, 15))

black\_hat = cv2.morphologyEx(image, cv2.MORPH\_BLACKHAT, kernel)

plt.subplot(1, 2, 1)

plt.title('Original Image')

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

plt.subplot(1, 2, 2)

plt.title('Black Hat Result')

plt.imshow(black\_hat, cmap='gray')

plt.show()

**OUTPUT:**



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

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

**PROGRAM:**

import cv2

import matplotlib.pyplot as plt

image = cv2.imread('1.png', cv2.IMREAD\_GRAYSCALE)

orb = cv2.ORB\_create(nfeatures=1000)

keypoints, descriptors = orb.detectAndCompute(image, None)

keypoint\_image = cv2.drawKeypoints(image, keypoints, None, color=(0, 255, 0), flags=0)

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

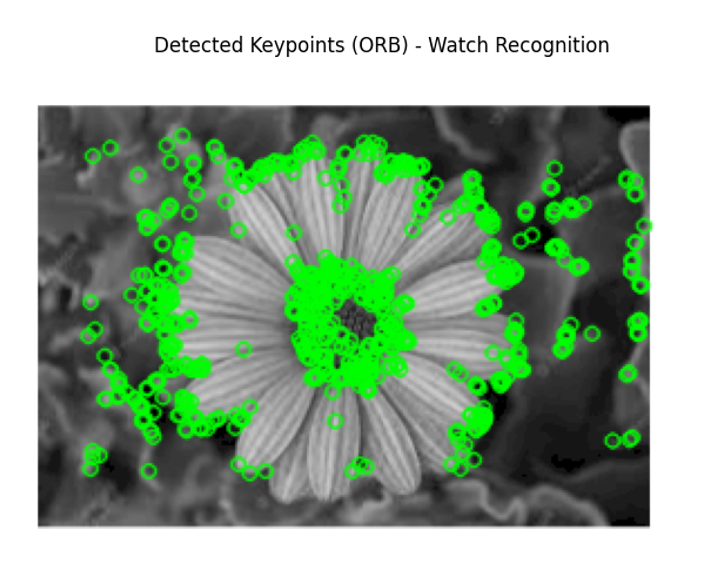
plt.title("Detected Keypoints (ORB) - Watch Recognition")

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

plt.axis('off')

plt.show()

**OUTPUT:**



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

**AIM:** Using Opencv play Video in Reverse mode.

**PROGRAM:**

import cv2

from google.colab.patches import cv2\_imshow

import time

video\_path = 'videoplayback.mp4'

cap = cv2.VideoCapture(video\_path)

if not cap.isOpened():

print("Error: Could not open video.")

exit()

frames = []

while True:

ret, frame = cap.read()

if not ret:

break

frames.append(frame)

cap.release()

for frame in reversed(frames):

cv2\_imshow(frame)

time.sleep(0.03)

**OUTPUT:**



**38. Face Detection using Opencv.**

**AIM:** Face Detection using Opencv.

**PROGRAM:**

import cv2

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

image = cv2.imread('dd.png')

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

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

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

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

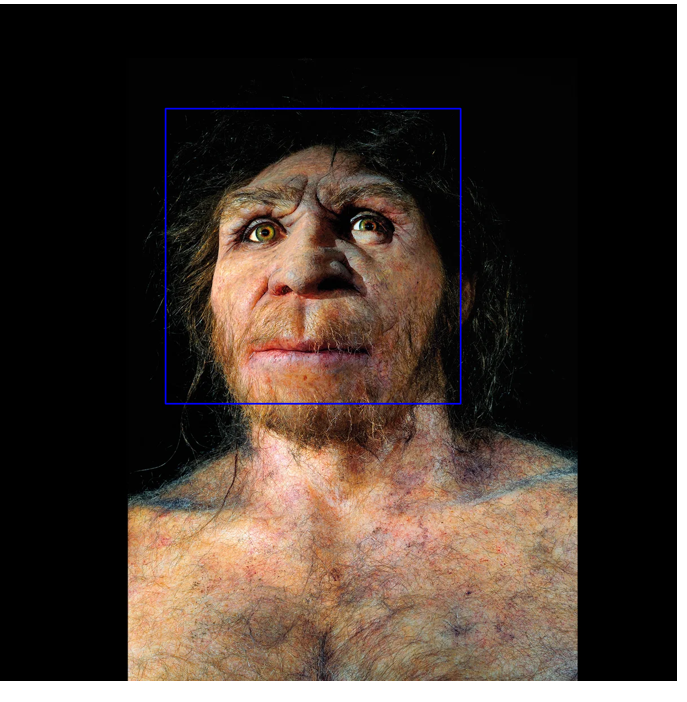
from google.colab.patches import cv2\_imshow

cv2\_imshow(image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**OUTPUT:**



**40. Draw Rectangular shape and extract objects.**

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

**PROGRAM:**

import cv2

from google.colab.patches import cv2\_imshow

image = cv2.imread('12.png')

boxes = {

"Rohit Sharma": (100, 50, 150, 300),

"Virat Kohli": (300, 50, 150, 300)

}

for name, (x, y, w, h) in boxes.items():

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

cv2.putText(image, name, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 0, 0), 2)

cv2\_imshow(image)

for name, (x, y, w, h) in boxes.items():

cropped = image[y:y+h, x:x+w]

print(f"Extracted: {name}")

cv2\_imshow(cropped)

**OUTPUT:**

