

## 7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.

import cv2

import numpy as np

# Read the image

img = cv2.imread(image\_pat)

# Get the height and width of the image

height, width = img.shape[:2]

# Split the image into four quadrants

quad1 = img[:height//2, :width//2]

quad2 = img[:height//2, width//2:]

quad3 = img[height//2:, :width//2]

quad4 = img[height//2:, width//2:]

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

plt.subplot(1, 2, 1)

plt.imshow(quad1)

plt.title("1")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(quad2)

plt.title("2")

plt.axis("off")

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

plt.subplot(1, 2, 1)

plt.imshow(quad3)

plt.title("3")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(quad4)

plt.title("4")

plt.axis("off")

plt.show()

**output**





## Additional pgm

# Up- down

import cv2

import numpy as np

# Read the image

img = cv2.imread(image\_path)

# Get the height and width of the image

height, width = img.shape[:2]

up = img[:height//2,:]

down = img[height//2:,:]

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

plt.subplot(1, 2, 1)

plt.imshow(up)

plt.title("Up")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(down)

plt.title("down")

plt.axis("off")

plt.show()



# left- right

import cv2

import numpy as np

# Read the image

img = cv2.imread('/content/3.PNG')

# Get the height and width of the image

height, width = img.shape[:2]

left = img[:, :width//2]

right = img[:, width//2:]

up = img[:height//2,:]

down = img[height//2:,:]

quad1 = img[:height//2, :width//2]

quad2 = img[:height//2, width//2:]

quad3 = img[height//2:, :width//2]

quad4 = img[height//2:, width//2:]

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

plt.subplot(1, 2, 1)

plt.imshow(left)

plt.title("left")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(right)

plt.title("right")

plt.axis("off")

plt.show()



## 8. Write a program to show rotation, scaling, and translation on an image.

#Rotation and scaling of image

import cv2

def translate\_image(image, dx, dy):

    rows, cols = image.shape[:2]

    translation\_matrix = np.float32([[1, 0, dx], [0, 1, dy]])

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

    return translated\_image

# Read the image

image = cv2.imread('/content/sample\_data/3.png')

# Get image dimensions

height, width = image.shape[:2]

# Calculate the center coordinates of the image

center = (width // 2, height // 2)

rotation\_value = int(input("Enter the degree of Rotation:"))

scaling\_value = int(input("Enter the zooming factor:"))

# Create the 2D rotation matrix

rotated = cv2.getRotationMatrix2D(center=center, angle=rotation\_value, scale=1)

rotated\_image = cv2.warpAffine(src=image, M=rotated, dsize=(width, height))

scaled = cv2.getRotationMatrix2D(center=center, angle=0, scale=scaling\_value)

scaled\_image = cv2.warpAffine(src=rotated\_image, M=scaled, dsize=(width, height))

h = int(input("How many pixels you want the image to be translated horizontally? "))

v = int(input("How many pixels you want the image to be translated vertically? "))

translated\_image = translate\_image(scaled\_image, dx=h, dy=v)

cv2.imwrite('Final\_image.png', translated\_image)

## 9. Read an image and extract and display low-level features such as edges, textures using filtering techniques.

import cv2

import numpy as np

# Load the image

image\_path = "image/atc.jpg" # Replace with the path to your image

img = cv2.imread(image\_path)

# Convert the image to grayscale

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

# Edge detection

edges = cv2.Canny(gray, 100, 200) # Use Canny edge detector

# Texture extraction

kernel = np.ones((5, 5), np.float32) / 25 # Define a 5x5 averaging kernel

texture = cv2.filter2D(gray, -1, kernel) # Apply the averaging filter for texture extraction

# Display the original image, edges, and texture

cv2.imshow("Original Image", img)

cv2.imshow("Edges", edges)

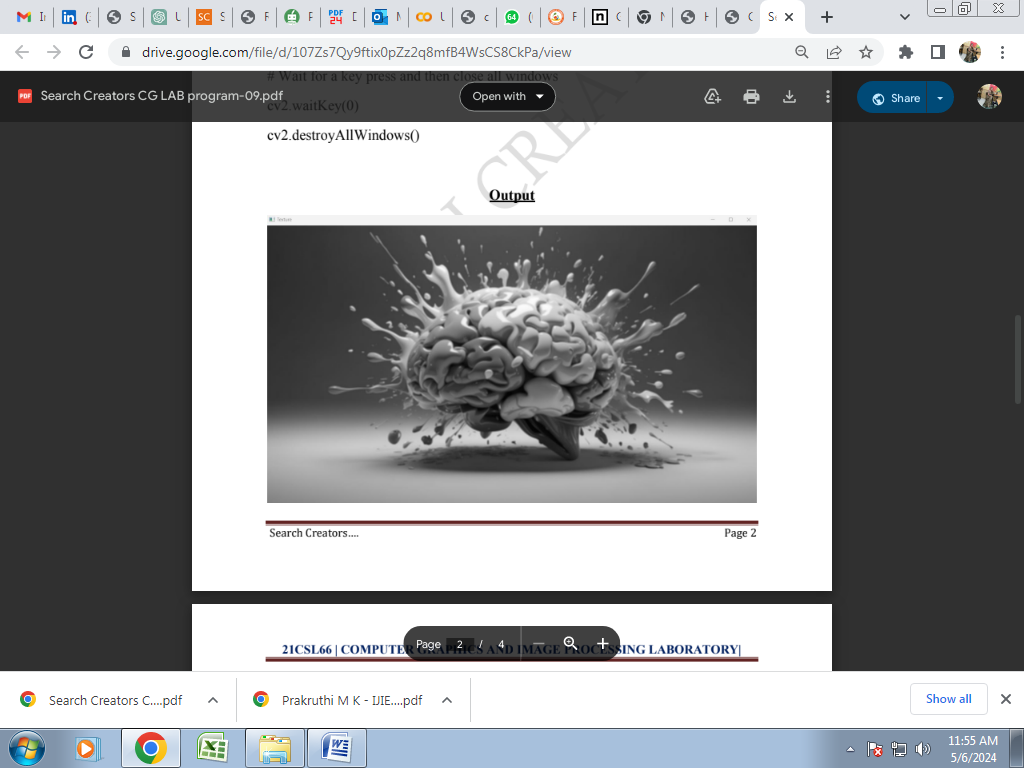
cv2.imshow("Texture", texture)

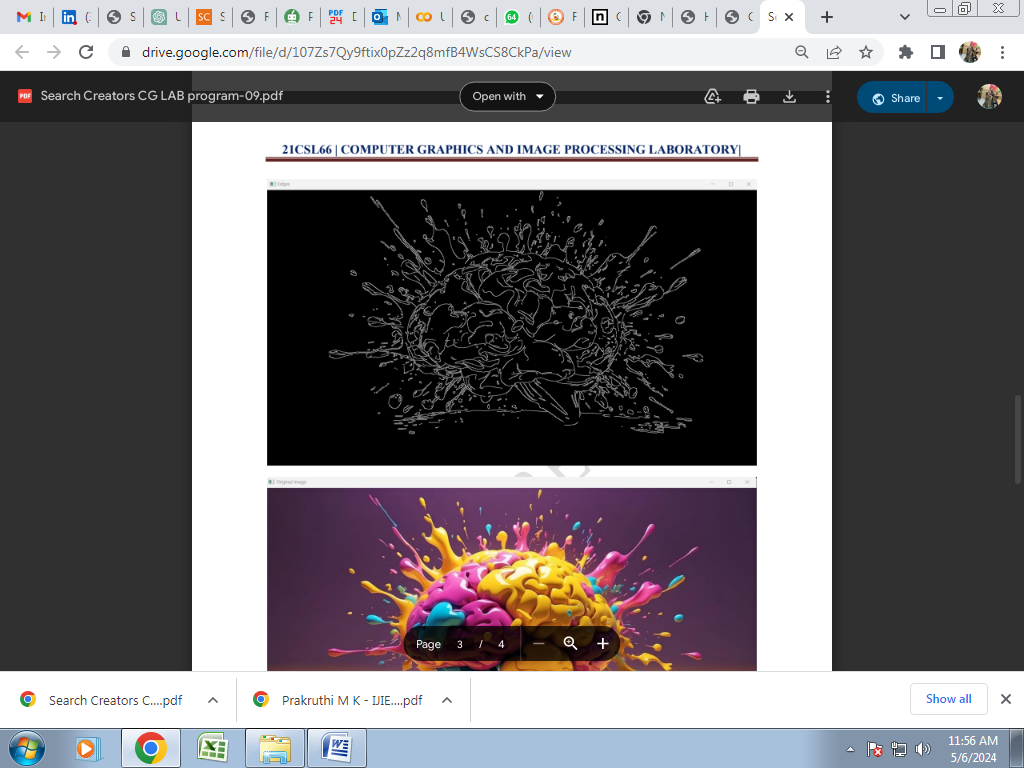
# Wait for a key press and then close all windows

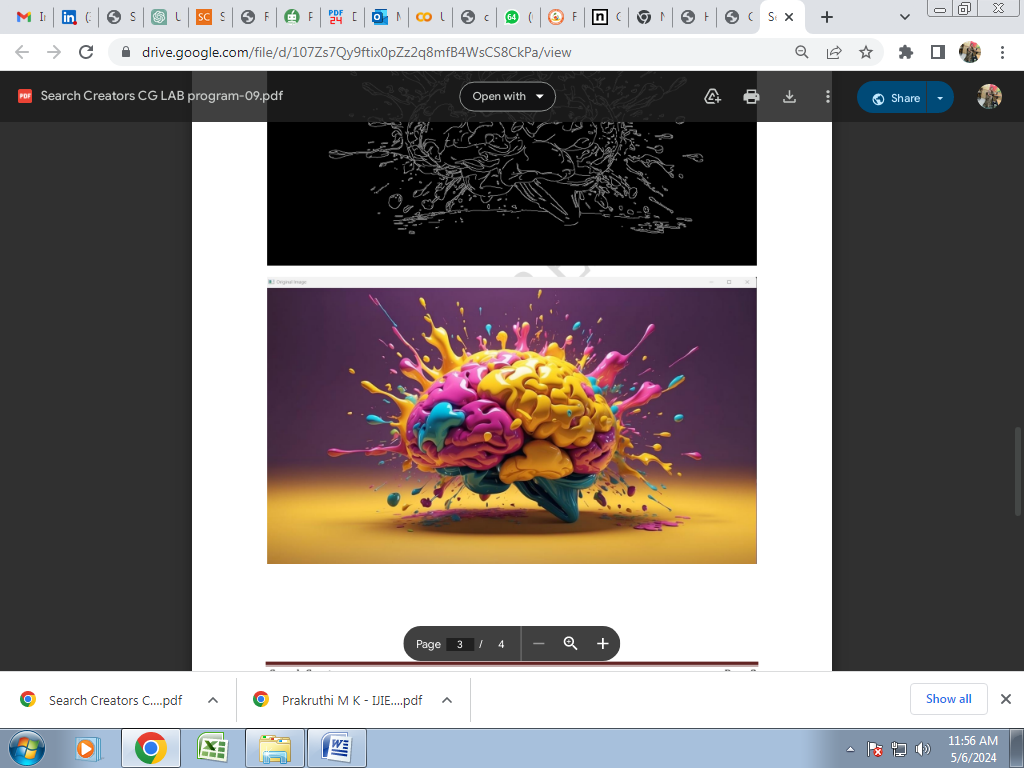
cv2.waitKey(0)

cv2.destroyAllWindows()

**Output**







## 10. Write a program to blur and smoothing an image.

img = cv2.imread("/content/sample\_data/smaple.jpg",cv2.IMREAD\_GRAYSCALE)

image\_array = np.array(img)

print(image\_array)

def sharpen():

return np.array([[1,1,1],[1,1,1],[1,1,1]])

def filtering(image, kernel):

m, n = kernel.shape

if (m == n):

y, x = image.shape

y = y - m + 1 # shape of image - shape of kernel + 1

x = x - m + 1

new\_image = np.zeros((y,x))

for i in range(y):

for j in range(x):

new\_image[i][j] = np.sum(image[i:i+m, j:j+m]\*kernel)

return new\_image

# Display the original and sharpened images

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale Image")

plt.axis("off")

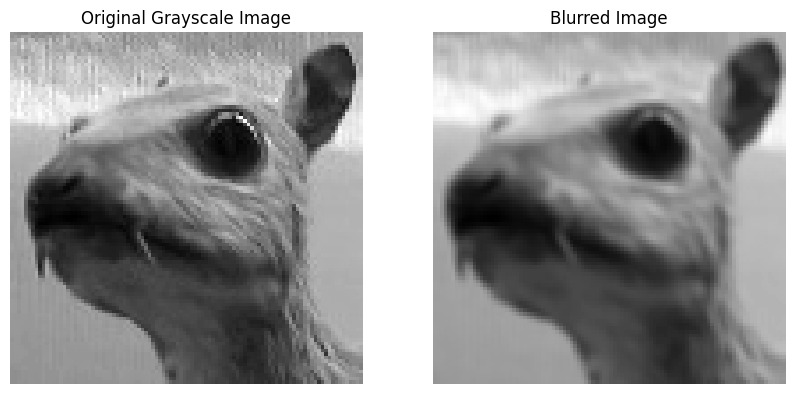
plt.subplot(1, 2, 2)

plt.imshow(filtering(image\_array, sharpen()),cmap='gray')

plt.title("Blurred Image")

plt.axis("off")

plt.show()





**Extra programs:**

1. #blur

import cv2

# Read the input image (replace 'your\_image.jpg' with the actual image path)

image\_path = '1.png'

image = cv2.imread(image\_path)

# Apply average blur (simple box filter)

average\_blur = cv2.blur(image, (5, 5)) # Adjust the kernel size as needed

# Apply Gaussian blur

gaussian\_blur = cv2.GaussianBlur(image, (5, 5), 0) # Adjust the kernel size and sigma as needed

# Display the results

cv2.imshow('Original Image', image)

cv2.waitKey(0)

cv2.imshow('Average Blurred Image', average\_blur)

cv2.waitKey(0)

cv2.imshow('Gaussian Blurred Image', gaussian\_blur)

cv2.waitKey(0)

cv2.destroyAllWindows()

## 11. Write a program to contour an image.

import cv2

import numpy as np

image\_path = '1.png'

image = cv2.imread(image\_path)

# Convert the image to grayscale (contours work best on binary images)

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

# Apply thresholding (you can use other techniques like Sobel edges)

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

# Find contours

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

# Draw all contours on the original image

cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

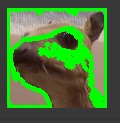
# Display the result

cv2.imshow('Contours', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

## output



## 12. Write a program to detect a face/s in an image.

import cv2

# Load the pre-trained Haar Cascade classifier for face detection

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

eye\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_eye.xml')

# Read the input image (replace 'your\_image.jpg' with the actual image path)

image\_path = 'face.jpeg'

image = cv2.imread(image\_path)

# Convert the image to grayscale

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

# Detect faces in the image

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

# Draw rectangles around detected faces

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

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

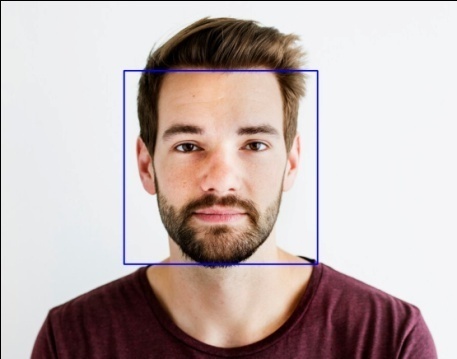
# Save or display the result

cv2.imwrite('detected\_faces.jpg', image) # Save the result

cv2.imshow('Detected Faces', image) # Display the result

cv2.waitKey(0)

cv2.destroyAllWindows()



1. another one

import cv2

import matplotlib.pyplot as plt

from IPython.display import display, clear\_output

# Initialize the webcam

video\_capture = cv2.VideoCapture(0)

while True:

# Capture frame-by-frame

ret, frame = video\_capture.read()

# Perform face detection (you can use any pre-trained face detection model)

# For example, using Haar Cascade classifier:

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

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

# Draw rectangles around detected faces

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

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

# Display the frame in the notebook

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

plt.axis('off')

plt.show()

clear\_output(wait=True)

# Press 'q' to exit the loop

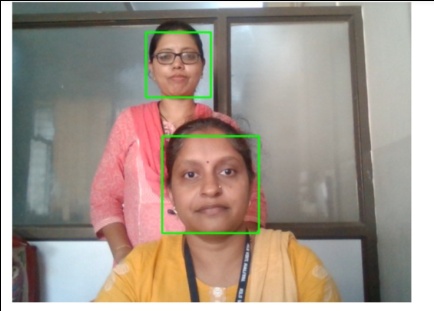
if cv2.waitKey(0) & 0xFF == ord('q'):

break

# Release the webcam

video\_capture.release()

cv2.destroyAllWindows()



#face detection with emotions

import cv2

from deepface import DeepFace

# Read an image (replace 'your\_image.jpg' with the actual image path)

image\_path = 'Angry.jpg'

image = cv2.imread(image\_path)

# Detect faces in the image

faces = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml').detectMultiScale(image, scaleFactor=1.1, minNeighbors=5)

# Predict emotions for each detected face

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

face\_roi = image[y:y + h, x:x + w]

result = DeepFace.analyze(face\_roi)

emotion = result[0]['emotion']

emotion = dict(sorted(emotion.items(), key=lambda item: item[1]))

emotion = (list(emotion.keys())[-1])

cv2.putText(image, emotion, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)

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

# Save or display the result

cv2.imwrite('emotion\_detected.jpg', image) # Save the result

cv2.imshow('Emotion Detection', image) # Display the result

cv2.waitKey(0)

cv2.destroyAllWindows()

## output



## Extra programs in IP

1. import cv2

# Read the input image

image = cv2.imread(image\_pat)

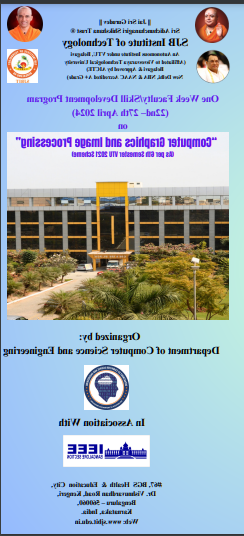
# Flip the image horizontally

flipped\_image = cv2.flip(image,-1 )

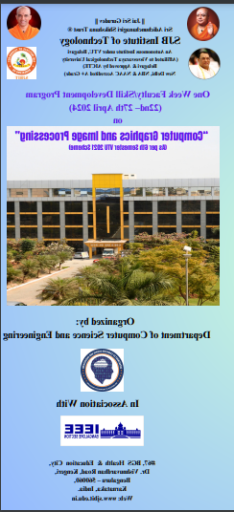
# Save the flipped image

cv2.imwrite('flipped\_image.png', flipped\_image)

**output**



flipped\_image = cv2.flip(image,1 )



flipped\_image = cv2.flip(image,0 )



2. # Thresholding

import cv2

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

#here 0 means that the image is loaded in gray scale format

gray\_image = cv2.imread('/content/3.PNG',0)

ret,thresh\_binary = cv2.threshold(gray\_image,127,255,cv2.THRESH\_BINARY)

ret,thresh\_binary\_inv = cv2.threshold(gray\_image,127,255,cv2.THRESH\_BINARY\_INV)

ret,thresh\_trunc = cv2.threshold(gray\_image,127,255,cv2.THRESH\_TRUNC)

ret,thresh\_tozero = cv2.threshold(gray\_image,127,255,cv2.THRESH\_TOZERO)

ret,thresh\_tozero\_inv = cv2.threshold(gray\_image,127,255,cv2.THRESH\_TOZERO\_INV)

#DISPLAYING THE DIFFERENT THRESHOLDING STYLES

names = ['Oiriginal Image','BINARY','THRESH\_BINARY\_INV','THRESH\_TRUNC','THRESH\_TOZERO','THRESH\_TOZERO\_INV']

images = gray\_image,thresh\_binary,thresh\_binary\_inv,thresh\_trunc,thresh\_tozero,thresh\_tozero\_inv

for i in range(6):

    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')

    plt.title(names[i])

    plt.xticks([]),plt.yticks([])

plt.show()

output



**Extra Pgms to run in google colab**

Pgm1:

img = cv2.imread("/content/sample\_data/1.png",cv2.IMREAD\_GRAYSCALE)

image\_array = np.array(img)

print(image\_array)

def sharpen():

  return np.array([

[0,-1,0],[-1,5,-1],[0,-1,0]

  ])

def filtering(image, kernel):

    m, n = kernel.shape

    if (m == n):

        y, x = image.shape

        y = y - m + 1 # shape of image - shape of kernel + 1

        x = x - m + 1

        new\_image = np.zeros((y,x))

        for i in range(y):

            for j in range(x):

                new\_image[i][j] = np.sum(image[i:i+m, j:j+m]\*kernel)

    return new\_image

# Display the original and sharpened images

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale Image")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(filtering(image\_array, sharpen()),cmap='gray')

plt.title("Blurred Image")

plt.axis("off")

plt.show()

"""# New Section"""

Pgm2:

import numpy as np

import cv2

import matplotlib.pyplot as plt

from google.colab.patches import cv2\_imshow

img = cv2.imread("C:\Users\HP-PC\Pictures\smaple.jpg")

image\_array = np.array(img)

def rgb2gray(image):

    return np.dot(image[..., :3], [0.2989, 0.5870, 0.1140])

grayscale\_image = rgb2gray(image\_array)

print(image\_array.shape)

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

plt.subplot(1, 2, 1)

plt.imshow(image\_array)

plt.title("Original  Image")

plt.axis("off")

plt.subplot(1, 2, 2)

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

plt.title("Grayscale Image")

plt.axis("off")

plt.show()

grayscale\_image[24,8]

def sharpen():

  return np.array([

[0,-1,0],[-1,5,-1],[0,-1,0]

  ])

def filtering(image, kernel):

    m, n = kernel.shape

    if (m == n):

        y, x = image.shape

        y = y - m + 1 # shape of image - shape of kernel + 1

        x = x - m + 1

        new\_image = np.zeros((y,x))

        for i in range(y):

            for j in range(x):

                new\_image[i][j] = np.sum(image[i:i+m, j:j+m]\*kernel)

    return

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

plt.subplot(1, 2, 1)

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

plt.title("Original  Image")

plt.axis("off")

plt.subplot(1, 2, 2)

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

plt.title("Sharpen Image")

plt.axis("off")

plt.show()

pgm3:

#Color image to Gray image

import numpy as np

import cv2

import matplotlib.pyplot as plt

def rgb2gray(image):

    return np.dot(image[..., :3], [0.2989, 0.5870, 0.1140])

    filename = '1.png'

image = cv2.imread("/content/sample\_data/JS pp photo.jpg")

image\_array = np.array(image)

grayscale\_image = rgb2gray(image\_array)

print(image\_array.shape)

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

plt.subplot(1, 2, 1)

plt.imshow(image\_array)

plt.title("Original  Image")

plt.axis("off")

plt.subplot(1, 2, 2)

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

plt.title("Grayscale Image")

plt.axis("off")

plt.show()

pgm4:

#Rotating an image

filename = '/content/sample\_data/JS pp photo.jpg'

image = cv2.imread(filename,cv2.IMREAD\_UNCHANGED)

image\_array = np.array(image)

def get\_rotation(angle):

    angle = np.radians(angle)

    return np.array([

[np.cos(angle), -np.sin(angle), 0],

[np.sin(angle), np.cos(angle), 0],

[0, 0, 1]

])

img\_transformed = np.zeros((400,400,3), dtype=np.uint8)

R1 = get\_rotation(45)

for i, row in enumerate(image\_array):

    for j, col in enumerate(row):

        pixel\_data = image\_array[i, j, :]

        input\_coords = np.array([i, j,1])

        i\_out, j\_out, \_ = (R1 @ input\_coords).astype(int)

        img\_transformed[i\_out+150,j\_out, :] = pixel\_data

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

plt.subplot(1, 2, 1)

plt.imshow(image\_array)

plt.title("Original Grayscale Image")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(img\_transformed)

plt.title("Rotated Image")

plt.axis("off")

plt.show()

cv2.imwrite('/content/sample\_data/rotate.jpg',img\_transformed)

pgm5:

import cv2

from google.colab.patches import cv2\_imshow

img  = cv2.imread('/content/sample\_data/Colors.jpg',-1)

cv2\_imshow(img)

cv2.waitKey(0)

cv2.destroyAllWindows()

#prgm-8

image = cv2.imread('/content/sample\_data/Rainbow.jpg',1)

B, G, R = cv2.split(image)

# Corresponding channels are separated

cv2\_imshow(image)

cv2.waitKey(0)

cv2\_imshow(B)

cv2.waitKey(0)

cv2\_imshow(G)

cv2.waitKey(0)

cv2\_imshow(R)

cv2.waitKey(0)

cv2.destroyAllWindows()

Pgm6:

import cv2

from google.colab.patches import cv2\_imshow

img1 = cv2.imread("/content/sample\_data/do\_not\_copy.png")

#img1=cv2.imread("")

print(img1.shape)

img2 = cv2.imread("/content/sample\_data/3.png")

img2 = cv2.resize(img2,(224,225))

print(img2.shape)

final\_img = cv2.addWeighted(img2,1,img1,0.7,0)

cv2\_imshow(final\_img)

cv2.imwrite('/content/sample\_data/rgbchannels.jpg',image)

filename = '/content/sample\_data/smaple.jpg'

image = cv2.imread(filename,cv2.IMREAD\_GRAYSCALE)

image\_array = np.array(image)

def sharpen():

    return np.array([

[0,-1, 0],

[-1,10, -1],

[0,-1, 0]

])

def filtering(image, kernel):

    m, n = kernel.shape

    if (m == n):

        y, x = image.shape

        y = y - m + 1 # shape of image - shape of kernel + 1

        x = x - m + 1

        new\_image = np.zeros((y,x))

        for i in range(y):

            for j in range(x):

                new\_image[i][j] = np.sum(image[i:i+m, j:j+m]\*kernel)

    return new\_image

# Display the original and sharpened images

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale Image")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(filtering(image\_array, sharpen()),cmap='gray')

plt.title("Sharpened Image")

plt.axis("off")

plt.show()

pgm7:

img = cv2.imread("/content/sample\_data/smaple.jpg",cv2.IMREAD\_GRAYSCALE)

image\_array = np.array(img)

print(image\_array)

def sharpen():

  return np.array([

[1/9,1/9,1/9],[1/9,1/9,1/9],[1/9,1/9,1/9]

  ])

def filtering(image, kernel):

    m, n = kernel.shape

    if (m == n):

        y, x = image.shape

        y = y - m + 1 # shape of image - shape of kernel + 1

        x = x - m + 1

        new\_image = np.zeros((y,x))

        for i in range(y):

            for j in range(x):

                new\_image[i][j] = np.sum(image[i:i+m, j:j+m]\*kernel)

    return new\_image

# Display the original and sharpened images

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale Image")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(filtering(image\_array, sharpen()),cmap='gray')

plt.title("Blurred Image")

plt.axis("off")

plt.show()

pgm8:

#Guassian Blur

img = cv2.imread("/content/sample\_data/smaple.jpg",cv2.IMREAD\_GRAYSCALE)

image\_array = np.array(img)

print(image\_array)

def sharpen():

  return np.array([

[1/16,2/16,1/16],[2/16,4/16,2/16],[1/16,2/16,1/16]

  ])

def filtering(image, kernel):

    m, n = kernel.shape

    if (m == n):

        y, x = image.shape

        y = y - m + 1 # shape of image - shape of kernel + 1

        x = x - m + 1

        new\_image = np.zeros((y,x))

        for i in range(y):

            for j in range(x):

                new\_image[i][j] = np.sum(image[i:i+m, j:j+m]\*kernel)

    return new\_image

# Display the original and sharpened images

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale Image")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(filtering(image\_array, sharpen()),cmap='gray')

plt.title("Guassian Blurred Image")

plt.axis("off")

plt.show()

pgm9:

#Ridge Detection

img = cv2.imread("/content/sample\_data/JS pp photo.jpg",cv2.IMREAD\_GRAYSCALE)

image\_array = np.array(img)

print(image\_array)

def sharpen():

  return np.array([

[0,-1,0],[-1,0,-1],[0,-1,0]

  ])

def filtering(image, kernel):

    m, n = kernel.shape

    if (m == n):

        y, x = image.shape

        y = y - m + 1 # shape of image - shape of kernel + 1

        x = x - m + 1

        new\_image = np.zeros((y,x))

        for i in range(y):

            for j in range(x):

                new\_image[i][j] = np.sum(image[i:i+m, j:j+m]\*kernel)

    return new\_image

# Display the original and sharpened images

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale Image")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(filtering(image\_array, sharpen()),cmap='gray')

plt.title("Ridge detection Image")

plt.axis("off")

plt.show()

pgm10:

#Edge Detection

img = cv2.imread("/content/sample\_data/JS pp photo.jpg",cv2.IMREAD\_GRAYSCALE)

image\_array = np.array(img)

print(image\_array)

def sharpen():

  return np.array([

[-1,-1,-1],[-1,8,-1],[-1,-1,-1]

  ])

def filtering(image, kernel):

    m, n = kernel.shape

    if (m == n):

        y, x = image.shape

        y = y - m + 1 # shape of image - shape of kernel + 1

        x = x - m + 1

        new\_image = np.zeros((y,x))

        for i in range(y):

            for j in range(x):

                new\_image[i][j] = np.sum(image[i:i+m, j:j+m]\*kernel)

    return new\_image

# Display the original and sharpened images

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

plt.subplot(1, 2, 1)

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

plt.title("Original Grayscale Image")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(filtering(image\_array, sharpen()),cmap='gray')

plt.title("edge detection Image")

plt.axis("off")

plt.show()

pgm11:

# comment

import cv2

from google.colab.patches import cv2\_imshow

image = cv2.imread('/content/3.png',1)

B, G, R = cv2.split(image)

# Corresponding channels are separated

cv2\_imshow(image)

cv2.waitKey(0)

cv2\_imshow(B)

cv2.waitKey(0)

cv2\_imshow(G)

cv2.waitKey(0)

cv2\_imshow(R)

cv2.waitKey(0)

cv2.destroyAllWindows()

#Image resizing 1

import cv2

from google.colab.patches import cv2\_imshow

img1 = cv2.imread("/content/sample\_data/do\_not\_copy.png")

print(img1.shape)

img2 = cv2.imread("/content/sample\_data/3.png")

img2 = cv2.resize(img2,(224,225))

print(img2.shape)

final\_img = cv2.addWeighted(img2,1,img1,0.7,0)

cv2\_imshow(final\_img)

#cv2.waitKey(0)

#cv2.destroyAllWindows()

#Image resizing 2

import cv2

from google.colab.patches import cv2\_imshow

img1 = cv2.imread("/content/sample\_data/circle.png")

#img1=cv2.imread("")

print(img1.shape)

img2 = cv2.imread("/content/sample\_data/square.png")

img2 = cv2.resize(img2,(img1.shape[1],img1.shape[0]))

print(img2.shape)

final\_img = cv2.addWeighted(img1,0.7,img2,0.6,0)

cv2\_imshow(final\_img)

#Image subtraction

import cv2

from google.colab.patches import cv2\_imshow

img\_1 = cv2.imread('/content/sample\_data/square.png')

print(img\_1.shape)

img\_2 = cv2.imread('/content/sample\_data/circle.png')

print(img\_2.shape)

final\_img = cv2.subtract(img\_2,img\_1)

cv2\_imshow(final\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

#Image subtraction for gray images

img\_1 = cv2.imread('/content/sample\_data/square.png',0)

img\_2 = cv2.imread('/content/sample\_data/circle.png',0)

img\_2  = cv2.resize(img\_2,(img1.shape[1],img\_1.shape[0]))

final\_img = cv2.subtract(img\_2,img\_1)

cv2\_imshow(final\_img)

pgm12:

#Image translation

import cv2

import numpy as np

def translate\_image(image, dx, dy):

    rows, cols = image.shape[:2]

    translation\_matrix = np.float32([[1, 0, dx], [0, 1, dy]])

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

    return translated\_image

# Read the image

image = cv2.imread('/content/sample\_data/circle.png')

# Translate the image by dx=50 pixels and dy=30 pixels

translated\_image = translate\_image(image, dx=20, dy=30)

# Save the translated image to disk

cv2.imwrite('translated\_image.png', translated\_image)

pgm13:

import cv2

import numpy as np

def translate\_image(image, dx, dy):

    rows, cols = image.shape[:2]

    translation\_matrix = np.float32([[1, 0, dx], [0, 1, dy]])

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

    return translated\_image

# Read the image

image = cv2.imread('/content/sample\_data/circle.png')

# Translate the image by dx=20 pixels and dy=0 pixels, translate horizontally by 20px

translated\_image = translate\_image(image, dx=20, dy=0)

# Save the translated image to disk

cv2.imwrite('translated\_image.png', translated\_image)

pgm14:

#Image Zoom in

import cv2

import numpy as np

# Read the image

image = cv2.imread('//content/sample\_data/circle.png')

# Get image dimensions

height, width = image.shape[:2]

# Calculate the center coordinates of the image

center = (width / 2, height / 2)

# Create the 2D rotation matrix

rotate\_matrix = cv2.getRotationMatrix2D(center=center, angle=30, scale=2)

# Rotate the image

rotated\_image = cv2.warpAffine(src=image, M=rotate\_matrix, dsize=(width, height))

# Display the original and rotated images

def translate\_image(image, dx, dy):

    rows, cols = image.shape[:2]

    translation\_matrix = np.float32([[1, 0, dx], [0, 1, dy]])

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

    return translated\_image

# Read the image

#image = cv2.imread('1.jpg')

# Translate the image by dx=50 pixels and dy=30 pixels

translated\_image = translate\_image(rotated\_image, dx=00, dy=20)

# Save the translated image to disk

cv2.imwrite('translated\_image.png', translated\_image)

pgm15:

#Rotation and scaling of image

import cv2

def translate\_image(image, dx, dy):

    rows, cols = image.shape[:2]

    translation\_matrix = np.float32([[1, 0, dx], [0, 1, dy]])

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

    return translated\_image

# Read the image

image = cv2.imread('/content/sample\_data/3.png')

# Get image dimensions

height, width = image.shape[:2]

# Calculate the center coordinates of the image

center = (width // 2, height // 2)

rotation\_value = int(input("Enter the degree of Rotation:"))

scaling\_value = int(input("Enter the zooming factor:"))

# Create the 2D rotation matrix

rotated = cv2.getRotationMatrix2D(center=center, angle=rotation\_value, scale=1)

rotated\_image = cv2.warpAffine(src=image, M=rotated, dsize=(width, height))

scaled = cv2.getRotationMatrix2D(center=center, angle=0, scale=scaling\_value)

scaled\_image = cv2.warpAffine(src=rotated\_image, M=scaled, dsize=(width, height))

h = int(input("How many pixels you want the image to be translated horizontally? "))

v = int(input("How many pixels you want the image to be translated vertically? "))

translated\_image = translate\_image(scaled\_image, dx=h, dy=v)

cv2.imwrite('Final\_image.png', translated\_image)

pgm16:

#Splitting an image into 4 equal quadrants

import cv2

import numpy as np

# Read the image

from google.colab.patches import cv2\_imshow

img = cv2.imread('/content/sample\_data/3.png')

# Get the height and width of the image

height, width = img.shape[:2]

# Split the image into four quadrants

quad1 = img[:height//2, :width//2]

quad2 = img[:height//2, width//2:]

quad3 = img[height//2:, :width//2]

quad4 = img[height//2:, width//2:]

# Display the four quadrants

cv2\_imshow(quad1)

cv2\_imshow(quad2)

cv2\_imshow(quad3)

cv2\_imshow(quad4)

pgm17:

# Commented out IPython magic to ensure Python compatibility.

#displaying an Image with different levels of thresholds

import cv2

import numpy as np

import matplotlib.pyplot as plt

# %matplotlib inline

#here 0 means that the image is loaded in gray scale format

gray\_image = cv2.imread('/content/sample\_data/3.png',0)

ret,thresh\_binary = cv2.threshold(gray\_image,127,255,cv2.THRESH\_BINARY)

ret,thresh\_binary\_inv = cv2.threshold(gray\_image,127,255,cv2.THRESH\_BINARY\_INV)

ret,thresh\_trunc = cv2.threshold(gray\_image,127,255,cv2.THRESH\_TRUNC)

ret,thresh\_tozero = cv2.threshold(gray\_image,127,255,cv2.THRESH\_TOZERO)

ret,thresh\_tozero\_inv = cv2.threshold(gray\_image,127,255,cv2.THRESH\_TOZERO\_INV)

#DISPLAYING THE DIFFERENT THRESHOLDING STYLES

names = ['Oiriginal Image','BINARY','THRESH\_BINARY\_INV','THRESH\_TRUNC','THRESH\_TOZERO','THRESH\_TOZERO\_INV']

images = gray\_image,thresh\_binary,thresh\_binary\_inv,thresh\_trunc,thresh\_tozero,thresh\_tozero\_inv

for i in range(6):

    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')

    plt.title(names[i])

    plt.xticks([]),plt.yticks([])

plt.show()

pgm17:

#Zooming out of an image

import cv2

from google.colab.patches import cv2\_imshow

# Read the input image

original\_image = cv2.imread('/content/sample\_data/3.png')

# Zoom out (reduce size by half)

zoomed\_out\_image = cv2.pyrDown(original\_image)

# Display the original and zoomed-out images

cv2\_imshow(original\_image)

cv2\_imshow(zoomed\_out\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

pgm18:

import cv2

import numpy as np

# Read the image

from google.colab.patches import cv2\_imshow

img = cv2.imread('/content/3.png')

# Get the height and width of the image

height, width = img.shape[:2]

# Split the image into four quadrants

quad1 = img[:height//2, :width//2]

quad2 = img[:height//2, width//2:]

quad3 = img[height//2:, :width//2]

quad4 = img[height//2:, width//2:]

# Display the four quadrants

cv2.imshow('quadrant1',quad1)

cv2.imshow('quadrant2',quad2)

cv2.imshow('quadrant3',quad3)

cv2.imshow('quadrant4',quad4)

cv2.imwrite('quad1.png', quad1)

cv2.imwrite('quad2.png', quad2)

cv2.imwrite('quad3.png', quad3)

cv2.imwrite('quad4.png', quad4)

pgm19:

import cv2

import numpy as np

# Read the image

img = cv2.imread('/content/sample\_data/3.png')

# Get the height and width of the image

height, width = img.shape[:2]

# Split the image into four quadrants

quad1 = img[:height//2, :width//2]

quad2 = img[:height//2, width//2:]

quad3 = img[height//2:, :width//2]

quad4 = img[height//2:, width//2:]

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

plt.subplot(1, 2, 1)

plt.imshow(quad1)

plt.title("1")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(quad2)

plt.title("2")

plt.axis("off")

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

plt.subplot(1, 2, 1)

plt.imshow(quad3)

plt.title("3")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(quad4)

plt.title("4")

plt.axis("off")

plt.show()

pgm20:

# Up- down

import cv2

import numpy as np

# Read the image

img = cv2.imread('/content/3.png')

# Get the height and width of the image

height, width = img.shape[:2]

up = img[:height//2,:]

down = img[height//2:,:]

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

plt.subplot(1, 2, 1)

plt.imshow(up)

plt.title("Up")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(down)

plt.title("down")

plt.axis("off")

plt.show()

pgm21:

# Up- down

import cv2

import numpy as np

# Read the image

img = cv2.imread('/content/3.png')

# Get the height and width of the image

height, width = img.shape[:2]

up = img[:height//2,:]

down = img[height//2:,:]

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

plt.subplot(2, 1, 1)

plt.imshow(up)

plt.title("Up")

plt.axis("off")

plt.subplot(2, 1, 2)

plt.imshow(down)

plt.title("down")

plt.axis("off")

plt.show()

pgm21:

# left right

import cv2

import numpy as np

# Read the image

img = cv2.imread('/content/sample\_data/JS pp photo.jpg')

#height means all rows and

#width means all the columns

# Get the height and width of the image

height, width = img.shape[:2]

left = img[:, :width//2]

right = img[:, width//2:]

up = img[:height//2,:]

down = img[height//2:,:]

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

plt.subplot(2, 1, 1)

plt.imshow(left)

plt.title("Left")

plt.axis("off")

plt.subplot(2, 1, 2)

plt.imshow(right)

plt.title("right")

plt.axis("off")

plt.show()

# Split the image into four quadrants

quad1 = img[:height//2]

quad2 = img[height//2:]

quad3 = img[:, :width//2:]

quad4 = img[:, width//2:]

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

plt.subplot(1, 2, 1)

plt.imshow(quad1)

plt.title("1")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(quad2)

plt.title("2")

plt.axis("off")

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

plt.subplot(1, 2, 1)

plt.imshow(quad3)

plt.title("3")

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(quad4)

plt.title("4")

plt.axis("off")

plt.show()

pgm22:

import cv2

import numpy as np

image = cv2.imread('/content/sample\_data/JS pp photo.jpg')

# Increase the brightness by adding 20 to each pixel value

brightness = 20

# Increase the contrast by scaling the pixel values by 5

contrast = 2

# Apply the brightness and contrast adjustments

image = cv2.addWeighted(image, contrast, np.zeros(image.shape, image.dtype), 0, brightness)

# Save the image

cv2.imwrite('image\_brightened\_and\_contrasted.png', image)