**DIGITAL IMAGE PROCESSING LAB**

**WEEK - 1**

**1a) Aim :** Write a program to display Grayscale image.

**Program :**

import matplotlib.pyplot as plt

import matplotlib.image as img

image = img.imread('sample.jpg')

plt.imshow(image[:,:,1], cmap='gray', vmin = 0,

vmax = 256,interpolation='none')

plt.show()

**Output :**

****

**1b) Aim :** Write a program to convert a 2D array into a Grayscale image.

**Program :**

import matplotlib.pyplot as plt

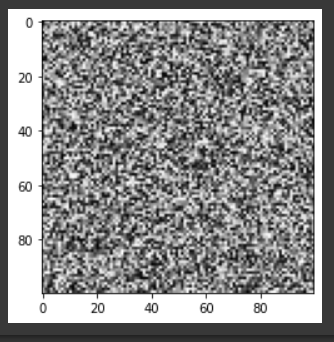
import numpy as np

X = np.random.random((100, 100)) # sample 2D array

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

plt.show()

**Output :**

****

**1c) Aim :** Write a program to convert gray images into an array of numbers.

**Program :**

from PIL import Image

from numpy import asarray

# load the image

image = Image.open('sample.jpg')

# convert image to numpy array

data = asarray(image)

print(type(data))

# summarize shape

print(data.shape)

# create Pillow image

image2 = Image.fromarray(data)

print(type(image2))

# summarize image details

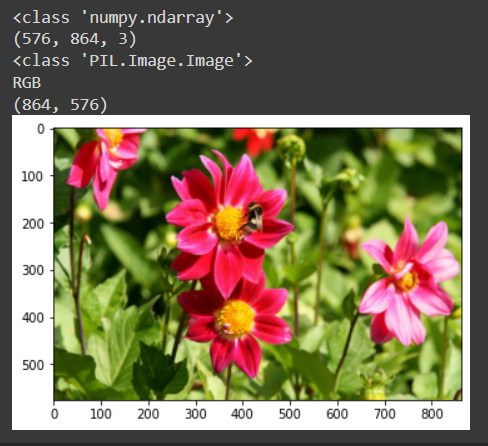
print(image2.mode)

print(image2.size)

plt.imshow(image2, cmap='gray')

plt.show()

**Output :**

****

**1d) Aim :** write a program to convert grey image into Binary image.

**Program :**

import cv2

img = cv2.imread('sample.jpg', 2)

ret, bw\_img = cv2.threshold(img, 127, 255,

cv2.THRESH\_BINARY)

# converting to its binary form

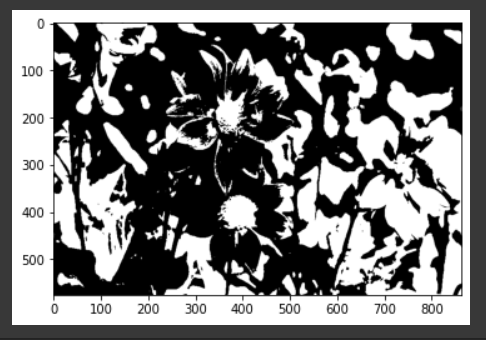
bw = cv2.threshold(img, 127, 255,

cv2.THRESH\_BINARY)

plt.imshow(bw\_img, cmap='gray')

plt.show()

**Output :**

****

**WEEK - 2**

**Aim :** Write a program to find histogram value and display histograph of a grayscale and color image

**Program :**

# importing required libraries of opencv

import cv2

# importing library for plotting

from matplotlib import pyplot as plt

# reads an input image

img = cv2.imread('sample.jpg',0)

# find frequency of pixels in range 0-255

histr = cv2.calcHist([img],[0],None,[256],[0,256])

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

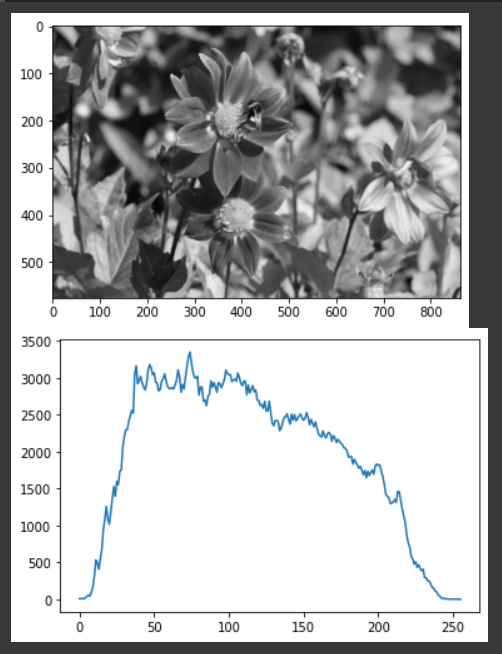
plt.show()

# show the plotting graph of an image

plt.plot(histr)

plt.show()

**Output :**

****

**WEEK - 3**

**Aim :** Program to implement histogram equalization in image enhancement.

**Program :**

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt

img\_filename = '/content/sample\_data/sample.jpg'

save\_filename = '/content/sample\_data/output2.jpg'

######################################

# READ IMAGE FROM FILE

######################################

#load file as pillow Image

img = Image.open(img\_filename)

# convert to grayscale

imgray = img.convert(mode='L')

#convert to NumPy array

img\_array = np.asarray(imgray)

######################################

# PERFORM HISTOGRAM EQUALIZATION

######################################

"""

STEP 1: Normalized cumulative histogram

"""

#flatten image array and calculate histogram via binning

histogram\_array = np.bincount(img\_array.flatten(), minlength=256)

#normalize

num\_pixels = np.sum(histogram\_array)

histogram\_array = histogram\_array/num\_pixels

#normalized cumulative histogram

chistogram\_array = np.cumsum(histogram\_array)

"""

STEP 2: Pixel mapping lookup table

"""

transform\_map = np.floor(255 \* chistogram\_array).astype(np.uint8)

"""

STEP 3: Transformation

"""

# flatten image array into 1D list

img\_list = list(img\_array.flatten())

6+

# transform pixel values to equalize

eq\_img\_list = [transform\_map[p] for p in img\_list]

# reshape and write back into img\_array

eq\_img\_array = np.reshape(np.asarray(eq\_img\_list), img\_array.shape)

# Let's plot the histograms

#histogram and cumulative histogram of original image has been calculated above

ori\_cdf = chistogram\_array

ori\_pdf = histogram\_array

#calculate histogram and cumulative histogram of equalized image

eq\_histogram\_array = np.bincount(eq\_img\_array.flatten(), minlength=256)

num\_pixels = np.sum(eq\_histogram\_array)

eq\_pdf = eq\_histogram\_array/num\_pixels

eq\_cdf = np.cumsum(eq\_pdf)

#plot

plt.figure()

plt.plot(ori\_pdf)

plt.plot(eq\_pdf)

plt.xlabel('Pixel intensity')

plt.ylabel('Distribution')

plt.legend(['Original','Equalized'])

plt.figure()

plt.plot(ori\_cdf)

plt.plot(eq\_cdf)

plt.xlabel('Pixel intensity')

plt.ylabel('Distribution')

plt.legend(['Original','Equalized'])

######################################

# WRITE EQUALIZED IMAGE TO FILE

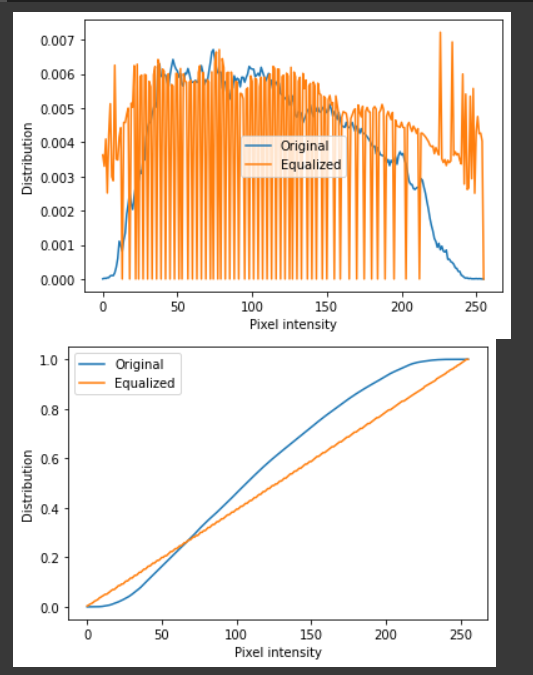
######################################

#convert NumPy array to pillow Image and write to file

eq\_img = Image.fromarray(eq\_img\_array, mode='L')

eq\_img.save(save\_filename)

**Output :**

****

**WEEK - 4**

**Aim :** Program to implement smoothing and sharpening filters in spatial domain.

**Program :**

spatial smoothing using averaging filter of kernal size 5\*5

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('/content/sample\_data/sample.jpg',0)

kernel = np.ones((5,5),np.float32)/25

dst = cv2.filter2D(img,-1,kernel,borderType=cv2.BORDER\_CONSTANT)

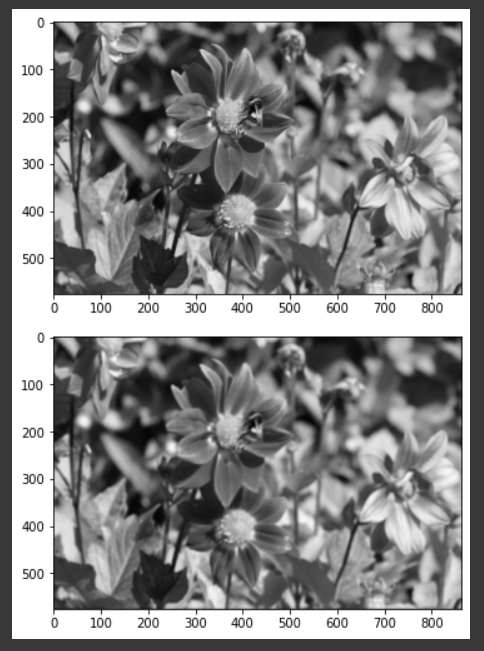
plt.imshow(img, cmap="gray")

plt.show()

plt.imshow(dst, cmap="gray")

plt.show()

**Output :**

****

spatial smoothing using median and Gaussian filter of kernal size 5\*5

import cv2

from matplotlib import pyplot as plt

img = cv2.imread('/content/sample\_data/sample.jpg',0)

median = cv2.medianBlur(img, 5)

gauss = cv2.GaussianBlur(img, (5,5), 0)

images = np.concatenate((median, gauss), axis=1)

plt.imshow(images, cmap="gray")

plt.show()

**Output :**

****

spatial sharpening using filter of kernal size 3\*3

import numpy as np

import matplotlib.pyplot as plt

from skimage.io import imshow, imread

from skimage.color import rgb2yuv, rgb2hsv, rgb2gray, yuv2rgb, hsv2rgb

from scipy.signal import convolve2d

# Sharpen

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

[-1, 5, -1],

[0, -1, 0]])

# Gaussian Blur

gaussian = (1 / 16.0) \* np.array([[1., 2., 1.],

[2., 4., 2.],

[1., 2., 1.]])

fig, ax = plt.subplots(1,2, figsize = (4,2))

ax[0].imshow(sharpen, cmap='gray')

ax[0].set\_title(f'Sharpen', fontsize = 18)

ax[1].imshow(gaussian, cmap='gray')

ax[1].set\_title(f'Gaussian Blur', fontsize = 18)

[axi.set\_axis\_off() for axi in ax.ravel()];

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

photo\_grey = rgb2gray(img)

plt.figure(num=None, figsize=(4, 2), dpi=80)

imshow(photo\_grey);

def multi\_convolver(image, kernel, iterations):

for i in range(iterations):

image = convolve2d(image, kernel, 'same', boundary = 'fill',

fillvalue = 0)

return image

omp1 = multi\_convolver(photo\_grey, gaussian, 2)

plt.figure(num=None, figsize=(4, 2), dpi=80)

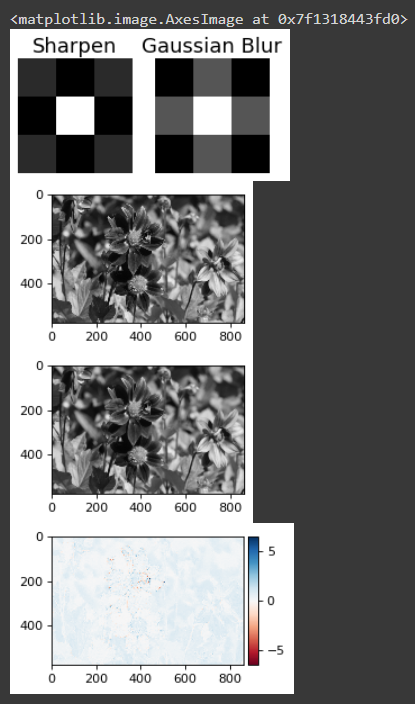
imshow(omp1);

omp2 = multi\_convolver(photo\_grey, sharpen, 2)

plt.figure(num=None, figsize=(4, 2), dpi=80)

imshow(omp2)

**Output :**

****

**WEEK - 5**

**Aim :** Program to fill the region of interest in the given image.

**Program :**

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

image= cv2.imread('sample.jpg',cv2.IMREAD\_UNCHANGED)

face=np.ones((100,100,3))

cv2\_imshow(image)

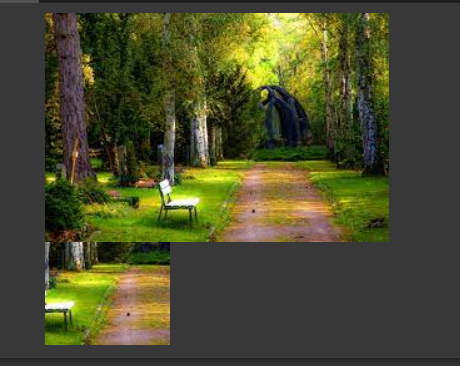
face=image[100:200,100:200]

cv2\_imshow(face)

cv2.waitKey()

cv2.destroyAllWindows()

**Output :**

****

**WEEK - 6**

**Aim :** Program for edge detection algorithms.

**Program :**

import cv2

from google.colab.patches import cv2\_imshow

# Read the original image

img = cv2.imread('sample.jpg')

# Display original image

cv2\_imshow(img)

cv2.waitKey(0)

# Convert to graycsale

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

# Blur the image for better edge detection

img\_blur = cv2.GaussianBlur(img\_gray, (3,3), 0)

# Sobel Edge Detection

sobelx = cv2.Sobel(src=img\_blur, ddepth=cv2.CV\_64F, dx=1, dy=0, ksize=5) # Sobel Edge Detection on the X axis

sobely = cv2.Sobel(src=img\_blur, ddepth=cv2.CV\_64F, dx=0, dy=1, ksize=5) # Sobel Edge Detection on the Y axis

sobelxy = cv2.Sobel(src=img\_blur, ddepth=cv2.CV\_64F, dx=1, dy=1, ksize=5) # Combined X and Y Sobel Edge Detection

# Display Sobel Edge Detection Images

#cv2\_imshow(sobelx)

cv2.waitKey(0)

#cv2\_imshow(sobely)

cv2.waitKey(0)

#cv2\_imshow(sobelxy)

cv2.waitKey(0)

# Canny Edge Detection

edges = cv2.Canny(image=img\_blur, threshold1=100, threshold2=200) # Canny Edge Detection

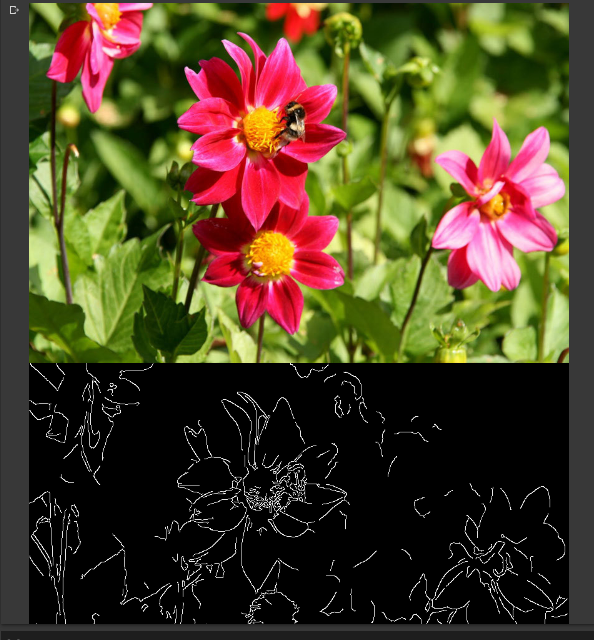
# Display Canny Edge Detection Image

cv2\_imshow(edges)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output :**

****

**WEEK - 7**

**Aim :** Program for noise restoration using filters.

**Program :**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('sample2.jpg')

dst = cv2.fastNlMeansDenoisingColored(image, None, 11, 6, 7, 21)

row, col = 1, 2

fig, axs = plt.subplots(row, col, figsize=(15, 10))

fig.tight\_layout()

axs[0].imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

axs[0].set\_title('Original')

axs[1].imshow(cv2.cvtColor(dst, cv2.COLOR\_BGR2RGB))

axs[1].set\_title('Fast Means Denoising')

plt.show()

**Output :**

****

**WEEK - 8**

**Aim :** Program for morphological operation

**Program :**

# Python program to demonstrate erosion and

# dilation of images.

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

# Reading the input image

img = cv2.imread('sample.jpg', 0)

# Taking a matrix of size 5 as the kernel

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

# The first parameter is the original image,

# kernel is the matrix with which image is

# convolved and third parameter is the number

# of iterations, which will determine how much

# you want to erode/dilate a given image.

img\_erosion = cv2.erode(img, kernel, iterations=1)

img\_dilation = cv2.dilate(img, kernel, iterations=1)

cv2\_imshow(img)

cv2\_imshow(img\_erosion)

cv2\_imshow(img\_dilation)

cv2.waitKey(0)

**Output :**

****

**Dilated -**

****

**Erosion -**

****

**WEEK - 9**

**Aim :** Program for object detection (eg: skin color detection)

**Program :**

# Import OpenCV module

import cv2

# Import pyplot from matplotlib as plt

from matplotlib import pyplot as pltd

from google.colab.patches import cv2\_imshow

# Opening the image from files

imaging = cv2.imread("sample1.png")

cv2\_imshow(imaging)

# Altering properties of image with cv2

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

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

# Importing Haar cascade classifier xml data

xml\_data = cv2.CascadeClassifier('stop\_data.xml')

# Detecting object in the image with Haar cascade classifier

detecting = xml\_data.detectMultiScale(imaging\_gray, minSize = (30, 30))

# Amount of object detected

amountDetecting = len(detecting)

# Using if condition to highlight the object detected

if amountDetecting != 0:

for (a, b, width, height) in detecting:

cv2.rectangle(imaging\_rgb, (a, b), # Highlighting detected object with rectangle

(a + height, b + width),

(0, 275, 0), 9)

# Plotting image with subplot() from plt

pltd.subplot(1, 1, 1)

# Displaying image in the output

pltd.imshow(imaging\_rgb)

pltd.show()

**Output :**

****

****

**WEEK - 10**

**Aim :** Program to implement various filters on coloured image (eg: Vector median filter).

**Program :**

**Gaussian filter -**

import cv2

import numpy

from google.colab.patches import cv2\_imshow

# image path

path = r'sample.jpg'

# using imread()

img = cv2.imread(path)

dst = cv2.GaussianBlur(img,(5,5),cv2.BORDER\_DEFAULT)

cv2\_imshow(numpy.hstack((img, dst)))

cv2.waitKey(0);

cv2.destroyAllWindows();

cv2.waitKey(1)

**Output :**

****

**Median filter -**

import cv2

import numpy

from google.colab.patches import cv2\_imshow

# image path

path = r'sample.jpg'

# using imread()

img = cv2.imread(path)

dst = cv2.medianBlur(img,7)

cv2\_imshow(numpy.hstack((img, dst)))

cv2.waitKey(0);

cv2.destroyAllWindows();

cv2.waitKey(1)

**Output :**

****