PROBLEM STATEMENT:

Write a program to do image format conversion i.e., from RGB to gray, gray to binary, RGB to binary.

THEORY:

Grayscale images are monochrome images, Means they have only one color. Grayscale images do not contain any information about color. Each pixel determines available different grey levels. A normal grayscale image contains 8 bits/pixel data, which has 256 different grey levels. In medical images and astronomy, 12 or 16 bits/pixel images are used.

Color images are formed by a combination of individual 2-D images. The RGB color system, a color image consists of three (red, green and blue) individual component images. For this reason many of the techniques developed for monochrome images can be extended to color images by processing the three component images individually.

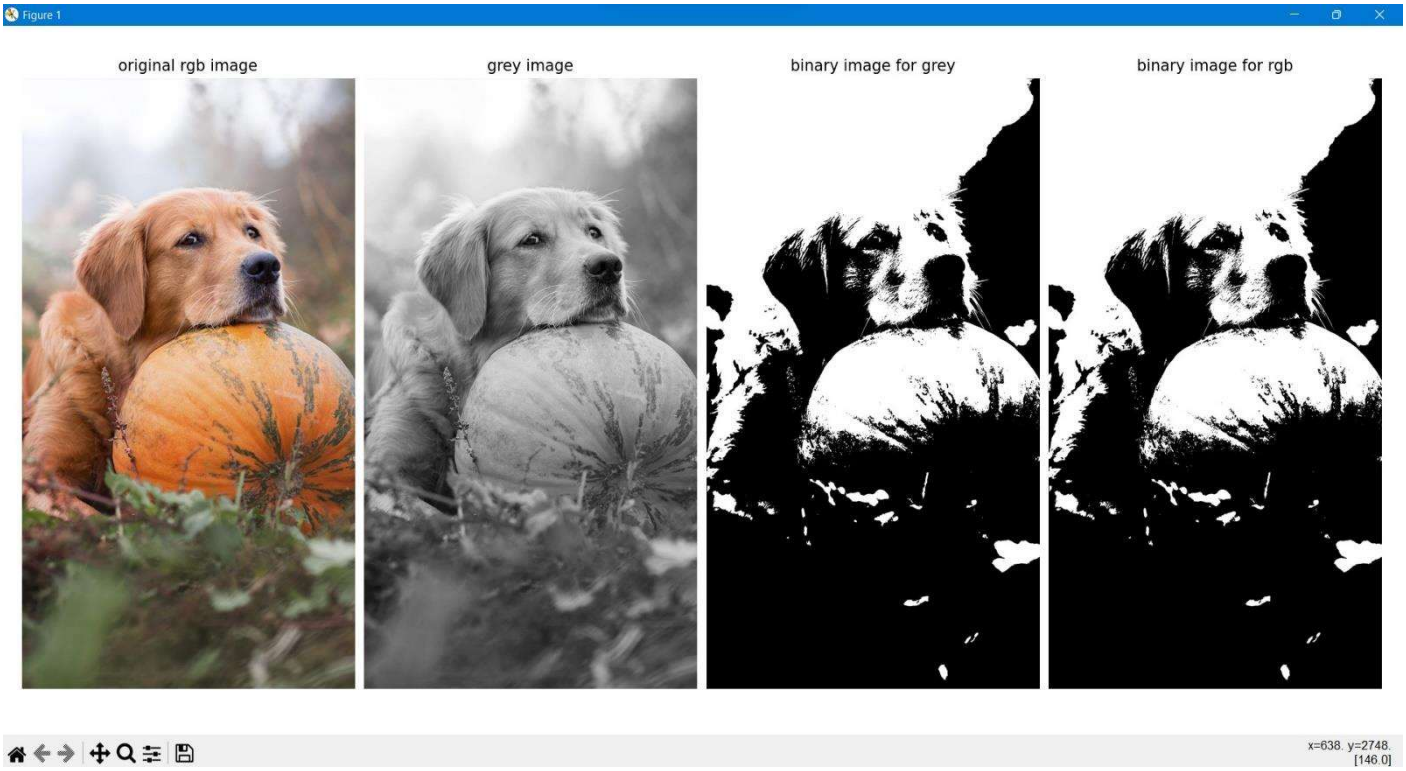
Binary images are images whose pixels have only two possible intensity values. They are normally displayed as black and white. Numerically, the two values are often 0 for black, and either 1 or 255 for white.

CODE:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mpimg


def channelExtraction(img):
    row, col, chan = img.shape

    rchan = np.zeros((row, col), dtype=np.uint8)
    gchan = np.zeros((row, col), dtype=np.uint8)
    bchan = np.zeros((row, col), dtype=np.uint8)

    for i, row in enumerate(img):
        for j, col in enumerate(row):
            r, g, b = col
            rchan[i, j] = r
            gchan[i, j] = g
            bchan[i, j] = b

    return bchan, gchan, rchan


def rgb_to_grey(b, g, r):
    row, col = r.shape
    grey_chan = np.zeros((row, col), dtype=np.uint8)

    for i in range(row):
        for j in range(col):
            grey_chan[i, j] = r[i, j] * 0.30 + g[i, j] * 0.59 + b[i, j] * 0.11
```

```
    return grey_chan


def grey_to_binary(grey_image):
    row, col = grey_image.shape
    binary = np.zeros((row, col), dtype=np.uint8)
    for i in range(row):
        for j in range(col):
            if grey_image[i, j] > 150:
                binary[i, j] = 255
            else:
                binary[i, j] = 0
    return binary


def rgb_to_binary(b, g, r):
    row, col = r.shape
    grey_image = np.zeros((row, col), dtype=np.uint8)

    for i in range(row):
        for j in range(col):
            grey_image[i, j] = r[i, j] * 0.30 + g[i, j] * 0.59 + b[i, j] * 0.11
    row1, col1 = grey_image.shape
    binary = np.zeros((row1, col1), dtype=np.uint8)
    for i1 in range(row1):
        for j1 in range(col1):
            if grey_image[i1, j1] > 150:
                binary[i1, j1] = 255
            else:
                binary[i1, j1] = 0
    return binary


img = mpimg.imread('retriever.jpg')
b, g, r = channelExtraction(img)
greyimg = rgb_to_grey(b, g, r)
binary = grey_to_binary(greyimg)
binary1 = rgb_to_binary(b, g, r)

fig = plt.figure(figsize=(16, 10))
plt_x = 1
plt_y = 4
fig.add_subplot(plt_x, plt_y, 1)
plt.imshow(img)
plt.axis("off")
plt.title("original rgb image")

fig.add_subplot(plt_x, plt_y, 2)
plt.imshow(greyimg, cmap="gray")
plt.axis("off")
plt.title("grey image")

fig.add_subplot(plt_x, plt_y, 3)
plt.imshow(binary, cmap="gray")
plt.axis("off")
plt.title("binary image for grey")

fig.add_subplot(plt_x, plt_y, 4)
plt.imshow(binary1, cmap="gray")
plt.axis("off")
plt.title("binary image for rgb")

plt.show()
```

OUTPUT:



DISCUSSION:

Here an RGB image is taken and converted that image into gray image using format conversion, then we have converted that same gray image to binary image and at last the original RGB image is converted to binary image.

PROBLEM STATEMENT:

Write a program to do image format conversion i.e., from RGB to YCbCr.

THEORY:

Image consumes a lot of data. One of the reasons is because they are represented in the RGB format. However, is not worth to store or transmit information in this color space representaion, once it has a large bandwidth. Thus all the pixels should be converted to YCbCr to accomplish that.

YCbCr is also same like RGB.

Here Y is the luma component of the color. Luma component is the brightness of the color. That means the light intensity of the color. The human eye is more sensitive to this component.

Cb and Cr is the blue component and red component related to the chroma component. That means **"Cb is the blue component relative to the green component. Cr is the red component relative to the green component."** These components are less sensitive to the human eyes.

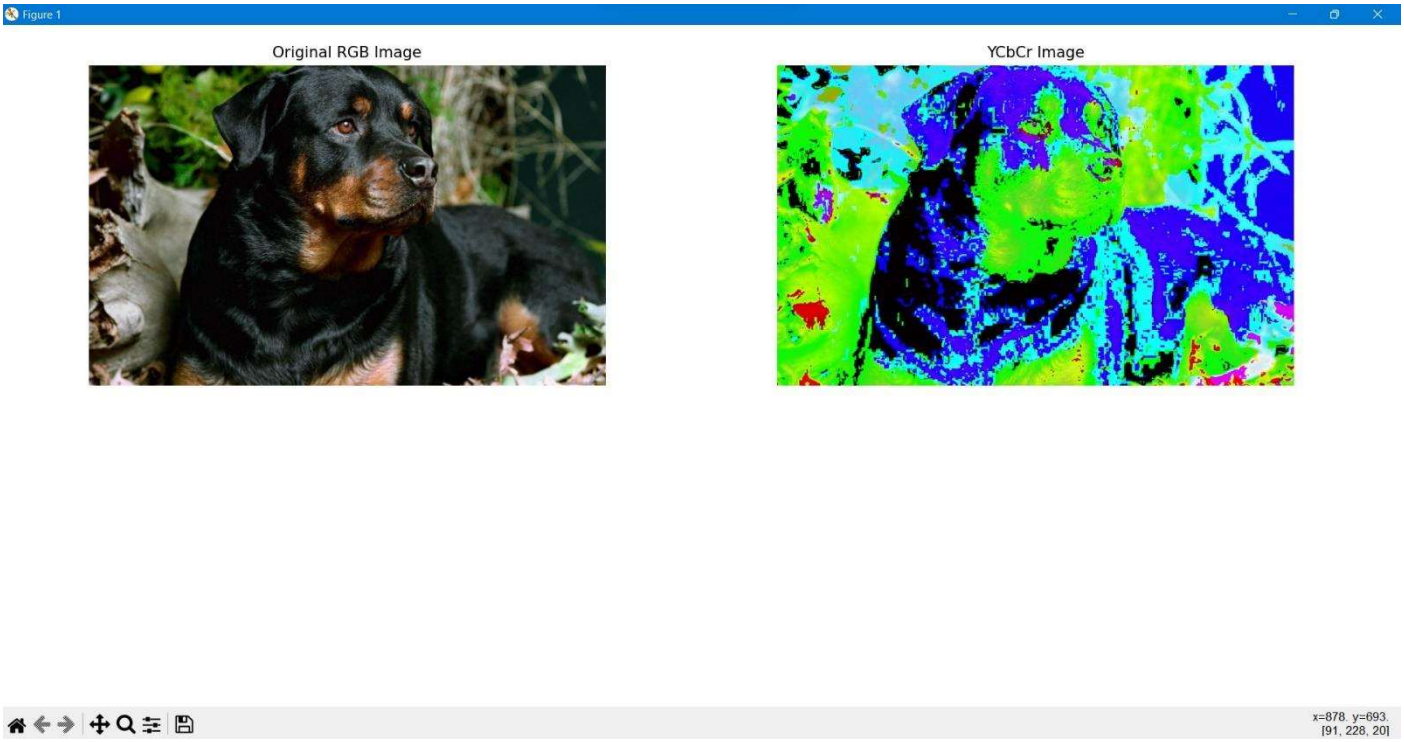
Since the Y component is more sensitive to the human eye, it needs to be more correct and Cb and Cr is less sensitive to the human eye. Therefore it needs not to be more accurate.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg


def channelExtract(img):
    row, col, channel = img.shape

    r_chan = np.zeros((row, col), dtype=np.uint8)
    g_chan = np.zeros((row, col), dtype=np.uint8)
    b_chan = np.zeros((row, col), dtype=np.uint8)

    for i, row in enumerate(img):
        for j, col in enumerate(row):
            r, g, b = col

            r_chan[i, j] = r
            g_chan[i, j] = g
            b_chan[i, j] = b

    return [b_chan, g_chan, r_chan]


def rgbToYCbCr(b, g, r, img):
    row, col, channel = img.shape
```

```
    ybr = np.zeros((row, col, channel), dtype=np.uint8)

    for i in range(row):
        for j in range(col):
            ybr[i, j, 0] = 0.299 * r[i, j] + 0.587 * g[i, j] + 0.114 * b[i, j]
            ybr[i, j, 1] = -0.168736 * r[i, j] - 0.331264 * g[i, j] + 0.500 * b[i, j]
            ybr[i, j, 2] = 0.500 * r[i, j] - 0.418688 * g[i, j] - 0.081312 * b[i, j]

    return ybr


img = mpimg.imread('weiler.jpg')
b, g, r = channelExtract(img)
ycbcr = rgbToYCbCr(b, g, r, img)

fig = plt.figure(figsize=(16, 10))
fig.add_subplot(2, 2, 1)
plt.imshow(img)
plt.axis("off")
plt.title("original rgb image")

fig.add_subplot(2, 2, 2)
plt.imshow(ycbcr, cmap="gray")
plt.axis("off")
plt.title("YCbCr image")

plt.show()
```

OUTPUT:



DISCUSSION:

Here a RGB image is taken and, then the image is converted to YCbCr using suitable formula.

PROBLEM STATEMENT:

Write a program to perform color separation into R, G, and B from an color image.

THEORY:

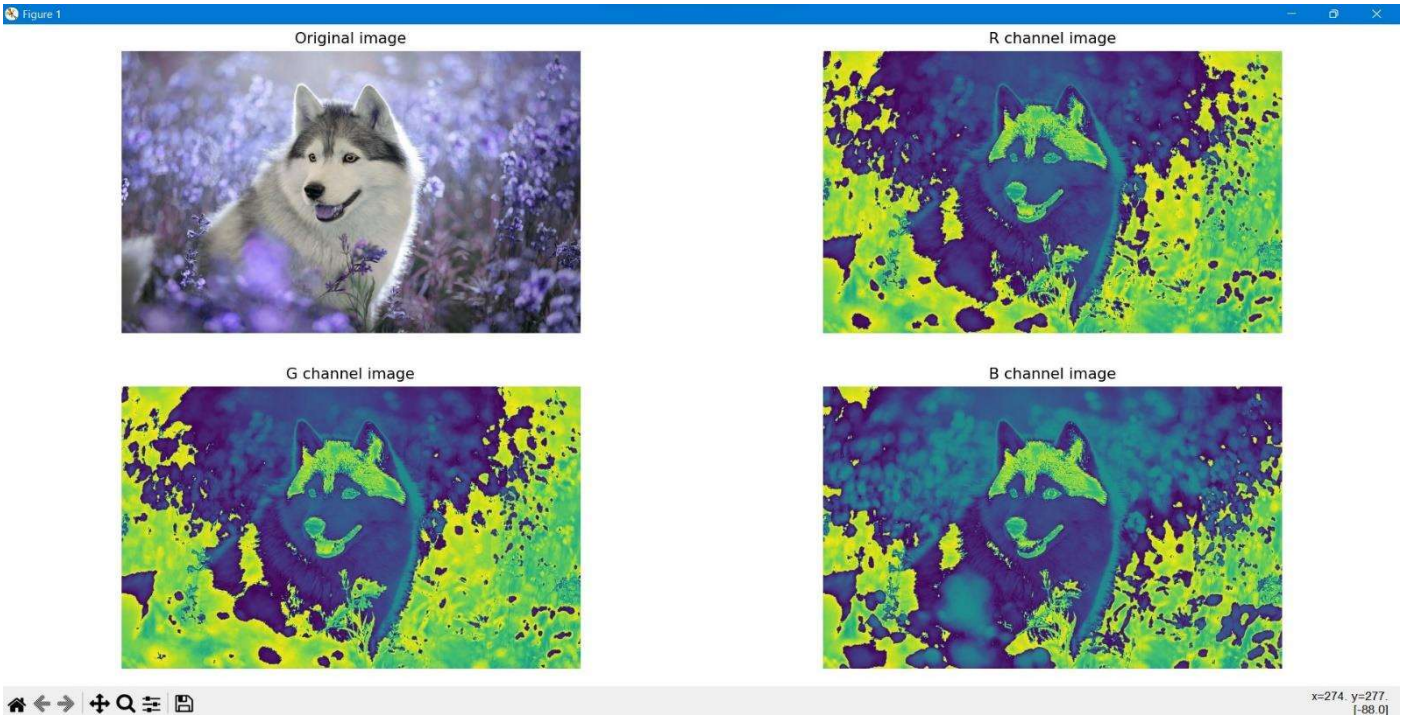
Colour images are three band monochrome images in which, each band contains a different color and the actual information is stored in the digital image. The color images contain gray level information in each spectral band. The images are represented as red, green and blue (RGB images). And each color image has 24 bits/pixel means 8 bits for each of the three color band(RGB). 8-bit color is used for storing image information in a computer's memory or in a file of an image. In this format, each pixel represents one 8 bit byte. It has 0-255 range of colors, in which 0 is used for black, 255 for white and 127 for gray color. The 8-bit color format is also known as a grayscale image. Initially, it was used by the UNIX operating system.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

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

row, col, channel = img.shape

rCha = np.zeros((row, col), dtype=np.int8)
gCha = np.zeros((row, col), dtype=np.int8)
bCha = np.zeros((row, col), dtype=np.int8)

for i, row in enumerate(img):
    for j, col in enumerate(row):
        r, g, b = col
        rCha[i, j] = r
        gCha[i, j] = g
        bCha[i, j] = b

fig = plt.figure(figsize=(17, 12))
fig.add_subplot(2, 2, 1)
plt.imshow(img)
plt.axis('off')
plt.title('Original image')

fig.add_subplot(2, 2, 2)
plt.imshow(rCha)
plt.axis('off')
plt.title('R channel image')

fig.add_subplot(2, 2, 3)
plt.imshow(gCha)
plt.axis('off')
plt.title('G channel image')
```

```
fig.add_subplot(2, 2, 4)
plt.imshow(bCha)
plt.axis('off')
plt.title('B channel image')

plt.show()
```

OUTPUT:



DISCUSSION:

Here a RGB image is taken and then each channel was separated from that image using channel extraction method.

PROBLEM STATEMENT:

Write a program to enhance the image in spatial domain using –

a. Image negative
b. Log transformation
c. Power law transformation
d. Piecewise linear transformation

THEORY:

Image negative: The negative of an image is achieved by replacing the intensity 'i' in the original image by 'i-1', i.e. the darkest pixels will become the brightest and the brightest pixels will become the darkest. Image negative is produced by subtracting each pixel from the maximum intensity value. For example in an 8-bit grayscale image, the max intensity value is 255, thus each pixel is subtracted from 255 to produce the output image. The transformation function used in image negative is :

$s = T(r) = (L – 1) – r$

Where L - 1 is the max intensity value, s is the output pixel value and r is the input pixel value.

Log transformation: The log transformations can be defined by this formula:

$s = c \log(r + 1)$.

Where s and r are the pixel values of the output and the input image and c is a constant. The value 1 is added to each of the pixel value of the input image because if there is a pixel intensity of 0 in the image, then log (0) is equal to infinity. So 1 is added, to make the minimum value at least 1.

During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation. This result in following image enhancement.The value of c in the log transform adjust the kind of enhancement you are looking for.

Power law transformation: There are further two transformation is power law transformations, that include nth power and nth root transformation. These transformations can be given by the expression:

$s=cr^{\gamma}$

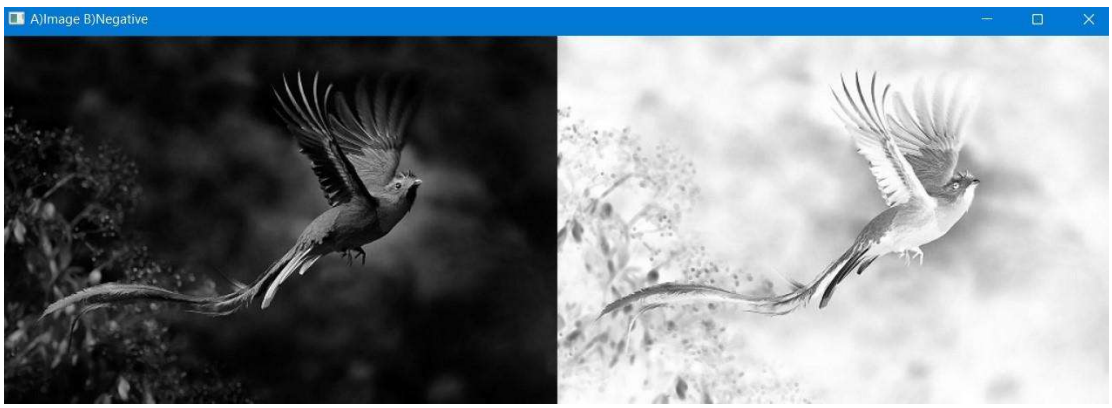
This symbol γ is called gamma, due to which this transformation is also known as gamma transformation.Variation in the value of γ varies the enhancement of the images. Different display devices / monitors have their own gamma correction, that's why they display their image at different intensity.This type of transformation is used for enhancing images for different type of display devices. The gamma of different display devices is different

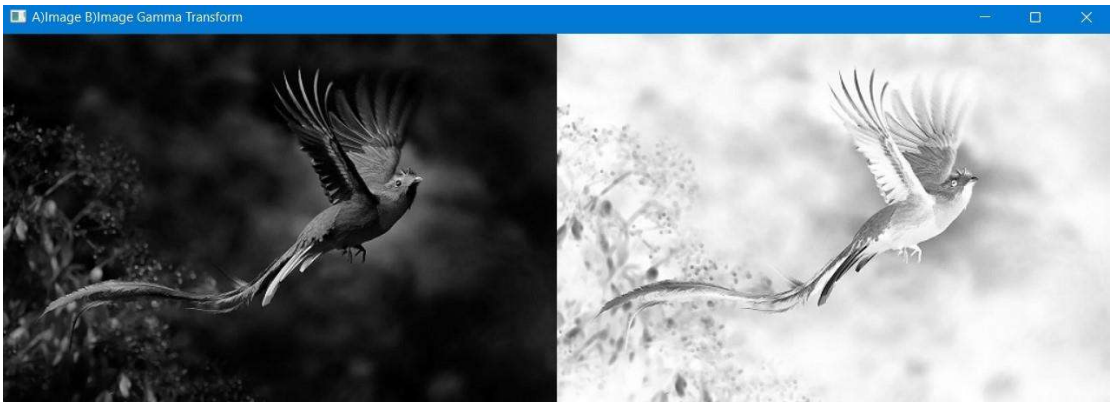
Piecewise linear transformation: It is a type of gray level transformation that is used for image enhancement. It is a spatial domain method. It is used for manipulation of an image so that the result is more suitable than the original for a specific application.

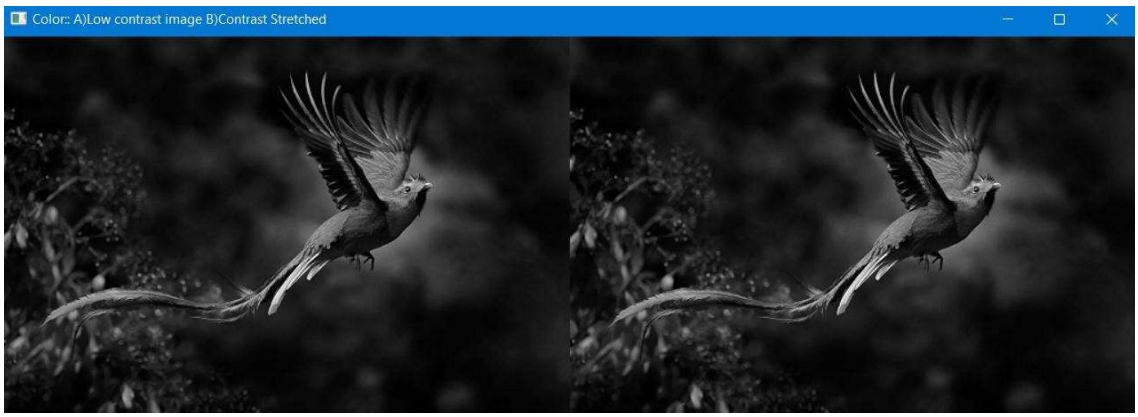
CODE:

```
import cv2 as cv
import numpy as np

filepath = "paradise1.jpg"
img = cv.imread(filepath, 0)

img2 = 255-img
imgs1 = np.hstack((img, img2))
cv.imshow("A)Image B)Negative", imgs1)

img3 = img.copy()
img3 = np.log2(img3)
img3 = 255*(img3/np.max(img3))
img3 = img3.astype(np.uint8)

imgs2 = np.hstack((img, img3))
cv.imshow("A)Image B)Image Log Transform", imgs2)

img4 = img.copy().astype(np.float)
img4 = img4/np.max(img4)
img4 = np.power(img4+1, 2)
img4 = 255*(img4/np.max(img4))
img4 = img4.astype(np.uint8)
imgs3 = np.hstack((img, img2))
cv.imshow("A)Image B)Image Gamma Transform", imgs3)

img5 = img.copy()
img_min = np.min(img5)
img_max = np.max(img5)
band_min = 0
band_max = 255
nimg = (img5-img_min)*((band_max-band_min)/(img_max-img_min))+band_min
print(np.min(nimg), np.max(nimg))
nimg = nimg.astype(np.uint8)
imgs4 = np.hstack((img, nimg))
cv.imshow("Color:: A)Low contrast image B)Contrast Stretched ", imgs4)

cv.waitKey(0)
```

OUTPUT:







DISCUSSION:

A RGB image was taken and various spatial domain image enhancement was performed using suitable formulas for each.

PROBLEM STATEMENT:

Write a program to perform the following image arithmetic.

a. Image addition.
b. Image subtraction.
c. Image multiplication.
d. Image division.

THEORY:

- **Image addition:** It takes two identically sized images as input and outputs a third image of the same size as the first two, with each pixel value being the sum of the values of the corresponding pixel in each of the two input images. More advanced versions allow the combination of multiple images in a single operation.
- **Image subtraction:** The pixel subtraction operator takes two images as input and outputs the third image with pixel values that are simply the first image's pixel values minus the corresponding pixel values from the second image. Using a single image as input is common and subtracting a constant value from all the pixels is common. Instead of the straightforward signed output, some versions of the operator will simply output the absolute difference between pixel values.
- **Image multiplication:** It takes two identically sized images as input and outputs a third image of the same size as the first two, with each pixel value being the multiplied values of the corresponding pixel in each of the two input images.
- **Image division:** Two identically sized images were taken as input and outputs a third image of the same size as the first two, with each pixel value being the divided values of the corresponding pixel in each of the two input images.

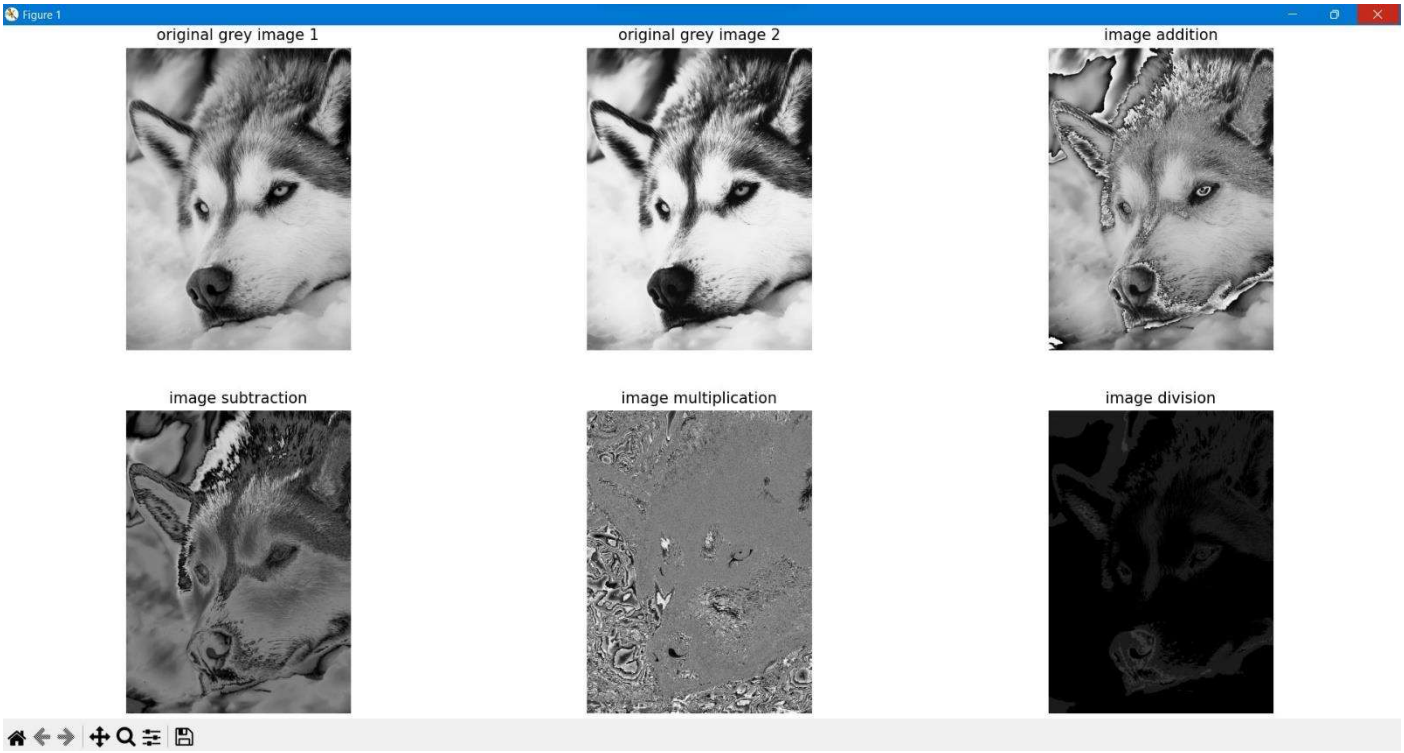
CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


def normalisation(img):
    row, col = img.shape
    normalized_img = np.zeros((row, col), dtype=np.uint8)
    m = np.min(img)
    n = np.max(img)

    for i in range(row):
        for j in range(col):
            c = (255 * (img[i, j] - m) / (n - m)).astype(int)
            normalized_img[i, j] = c

    return normalized_img


def addition(img1, img2):
```

```
    img3 = np.zeros(img2.shape, dtype=np.uint8)
    for i in range(len(img1)):
        for j in range(len(img1[0])):
            img3[i, j] = img1[i][j] + img2[i][j]

    return normalisation(img3)


def subtraction(img1, img2):
    img3 = np.zeros(img2.shape, dtype=np.uint8)
    for i in range(len(img1)):
        for j in range(len(img1[0])):
            if img1[i, j] > img2[i, j]:
                img3[i, j] = img1[i, j] - img2[i, j]
            else:
                img3[i, j] = img2[i, j] - img1[i, j]
    return normalisation(img3)


def multiplication(img1, img2):
    row, col = img1.shape
    img3 = np.zeros((row, col), dtype=np.uint8)
    for i in range(row):
        for j in range(col):
            img3[i, j] = img1[i, j] * img2[i, j]
    return normalisation(img3)


def division(img1, img2):
    row, col = img1.shape
    img3 = np.zeros((row, col), dtype=np.uint8)
    for i in range(row):
        for j in range(col):
            img3[i, j] = img1[i, j] / img2[i, j]
    return normalisation(img3)


img1 = cv2.imread('husky.jpg', 0)
img2 = cv2.imread('husky1.jpg', 0)
add = addition(img1, img2)
subtract = subtraction(img1, img2)
multiply = multiplication(img1, img2)
divide = division(img1, img2)

fig = plt.figure(figsize=(16, 10))
plt_x = 2
plt_y = 3
fig.add_subplot(plt_x, plt_y, 1)
plt.imshow(img1, cmap="gray")
plt.axis("off")
plt.title("original grey image 1")

fig.add_subplot(plt_x, plt_y, 2)
plt.imshow(img2, cmap="gray")
plt.axis("off")
plt.title("original grey image 2")

fig.add_subplot(plt_x, plt_y, 3)
plt.imshow(add, cmap="gray")
plt.axis("off")
plt.title("image addition")

fig.add_subplot(plt_x, plt_y, 4)
plt.imshow(subtract, cmap="gray")
plt.axis("off")
plt.title("image subtraction")
```

```
fig.add_subplot(plt_x, plt_y, 5)
plt.imshow(multiply, cmap="gray")
plt.axis("off")
plt.title("image multiplication")

fig.add_subplot(plt_x, plt_y, 6)
plt.imshow(divide, cmap="gray")
plt.axis("off")
plt.title("image division")

plt.show()
```

OUTPUT:



DISCUSSION:

Two Grey image was taken and various arithmetic operations such as addition, subtraction, multiplication and division were performed to achieve the output image by taking each pixel value from both images and performing the respective operations.

PROBLEM STATEMENT:

Write a program to average two images together into a single image. Display the new image.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img1 = cv2.imread('berries.jpg', 0)
img2 = cv2.imread('berries1.jpg', 0)
h, w = img1.shape
img3 = np.zeros((h, w), dtype=np.uint8)
for i in range(len(img1)):
    for j in range(len(img1[0])):
        img3[i, j] = (img1[i, j] + img2[i, j]) / 2

fig = plt.figure(figsize=(16, 10))
plt_x = 1
plt_y = 3
fig.add_subplot(plt_x, plt_y, 1)
plt.imshow(img1, cmap="gray")
plt.axis("off")
plt.title("original grey image1")

fig.add_subplot(plt_x, plt_y, 2)
plt.imshow(img2, cmap="gray")
plt.axis("off")
plt.title("original grey image2")

fig.add_subplot(plt_x, plt_y, 3)
plt.imshow(img3, cmap="gray")
plt.axis("off")
plt.title("average image")
plt.show()
```

OUTPUT:

DISCUSSION:

Two Grey image was taken and average of two image is calculated by suitable formula.

PROBLEM STATEMENT:

Write a program to compare two images using image substraction.

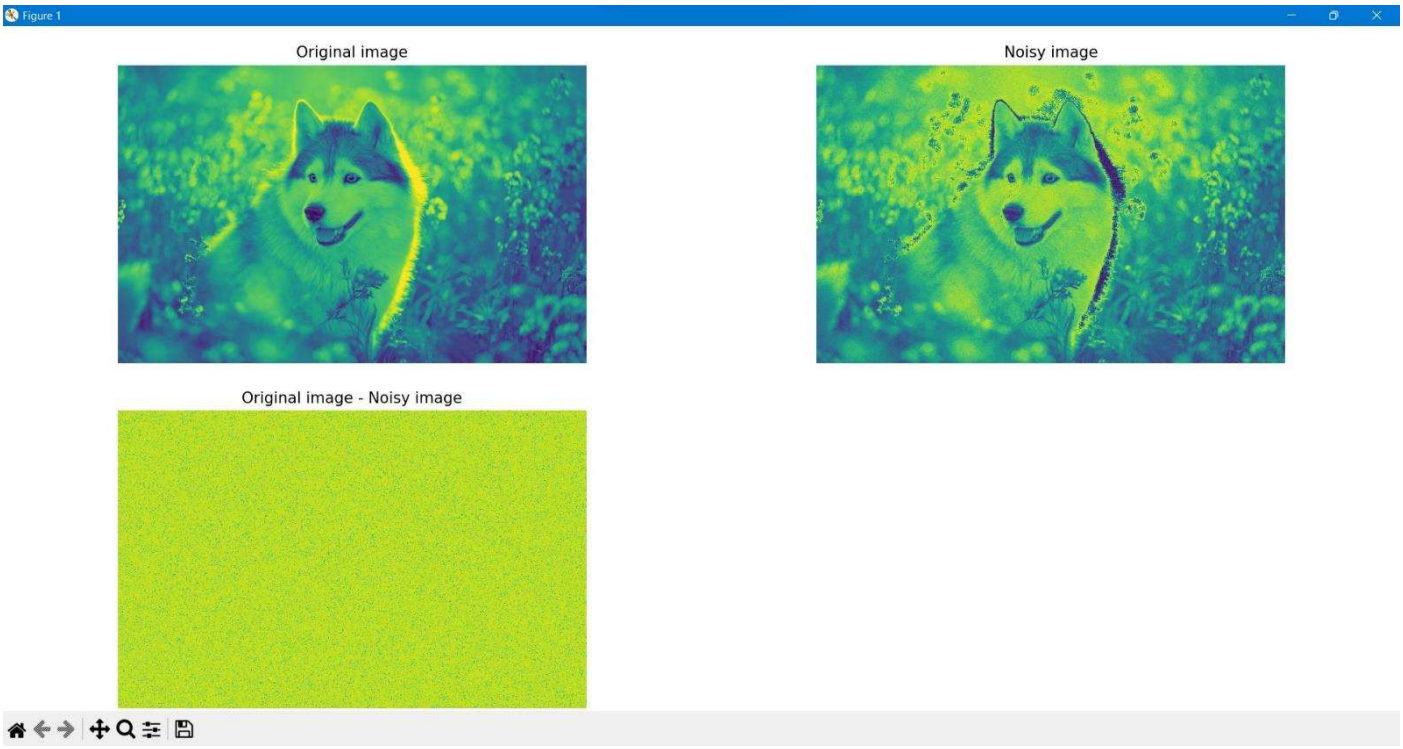
THEORY:

Image subtraction is generally performed between two images that have significant similarities between them. The purpose of image subtraction is to enhance the differences between two images. Images that are not captured under the same or very similar conditions may need to be registered . This may be the case if the images have been acquired at different times or under different settings. The output image may have a very small dynamic range and may need to be rescaled to the available display range.

CODE:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

filepath = "malamute.jpg"
img = cv.imread(filepath, 0)

r, c = img.shape
noise = np.random.randint(0, 50, [r, c], np.uint8)
img2 = img+noise
diff = np.abs(img-img2)

fig = plt.figure(figsize=(17, 12))

fig.add_subplot(2, 2, 1)
plt.imshow(img)
plt.axis('off')
plt.title('Original image')

fig.add_subplot(2, 2, 2)
plt.imshow(img2)
plt.axis('off')
plt.title('Noisy image')

fig.add_subplot(2, 2, 3)
plt.imshow(diff)
plt.axis('off')
plt.title('Original image - Noisy image')

plt.show()
```

OUTPUT:



DISCUSSION:

A Grey image was taken and that same image was taken with noise added in it and then image subtraction was performed to compare two images.

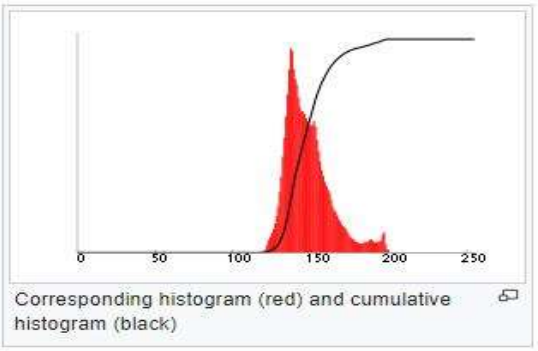
PROBLEM STATEMENT:

Write a program to find the histogram of a gray image and display the histogram.

THEORY:

Histogram is a graphical representation of the intensity distribution of an image. In simple terms, it represents the number of pixels for each intensity value considered.

Before Histogram Equalization



Corresponding histogram (red) and cumulative histogram (black)

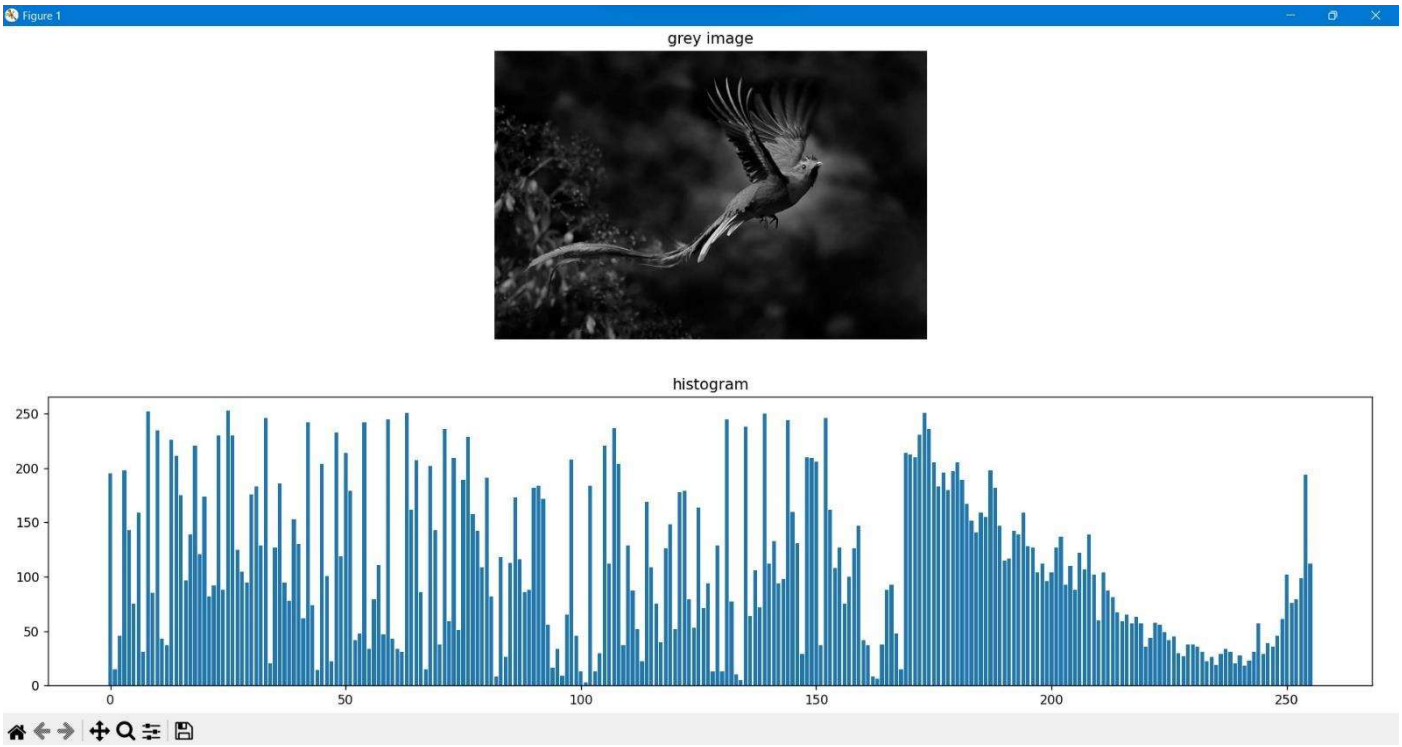
In the above figure, X-axis represents the tonal scale (black at the left and white at the right), and Y-axis represents the number of pixels in an image. Here, the histogram shows the number of pixels for each brightness level (from black to white), and when there are more pixels, the peak at the certain brightness level is higher.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


def channelExtract(img):
    row, col, channel = img.shape

    r_chan = np.zeros((row, col), dtype=np.uint8)
    g_chan = np.zeros((row, col), dtype=np.uint8)
    b_chan = np.zeros((row, col), dtype=np.uint8)

    for i, row in enumerate(img):
        for j, col in enumerate(row):
            b, g, r = col

            r_chan[i, j] = r
            g_chan[i, j] = g
            b_chan[i, j] = b

    return [b_chan, g_chan, r_chan]


def greyConversion(b, g, r):
    row, col = b.shape
    grey_chan = np.zeros((row, col), dtype=np.uint8)
```

```
    for i in range(row):
        for j in range(col):
            grey_chan[i, j] = r[i, j] * 0.3 + g[i, j] * 0.59 + b[i, j] * 0.11

    return grey_chan


def histogram(grey_image):
    row, col = grey_image.shape
    hist = np.zeros(256, dtype=np.uint8)
    for i in range(row):
        for j in range(col):
            hist[grey_image[i, j]] += 1
    return hist


img = cv2.imread("paradise.jpg")
b, g, r = channelExtract(img)
greyImage = greyConversion(b, g, r)
hist = histogram(greyImage)

fig = plt.figure(figsize=(17, 12))
pltX = 2
pltY = 1

fig.add_subplot(pltX, pltY, 1)
plt.imshow(greyImage, cmap="gray")
plt.axis("off")
plt.title("grey image")

fig.add_subplot(pltX, pltY, 2)
plt.bar(np.arange(len(hist)), hist)
plt.title("histogram")
plt.show()
```

OUTPUT:



PROBLEM STATEMENT:

Write a program to enhance the image in spatial domain using histogram equalization method.

THEORY:

Histogram Equalization is an image processing technique used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. This method usually increases the global contrast of images when its usable data is represented by close contrast values. This allows for areas of lower local contrast to gain a higher contrast.

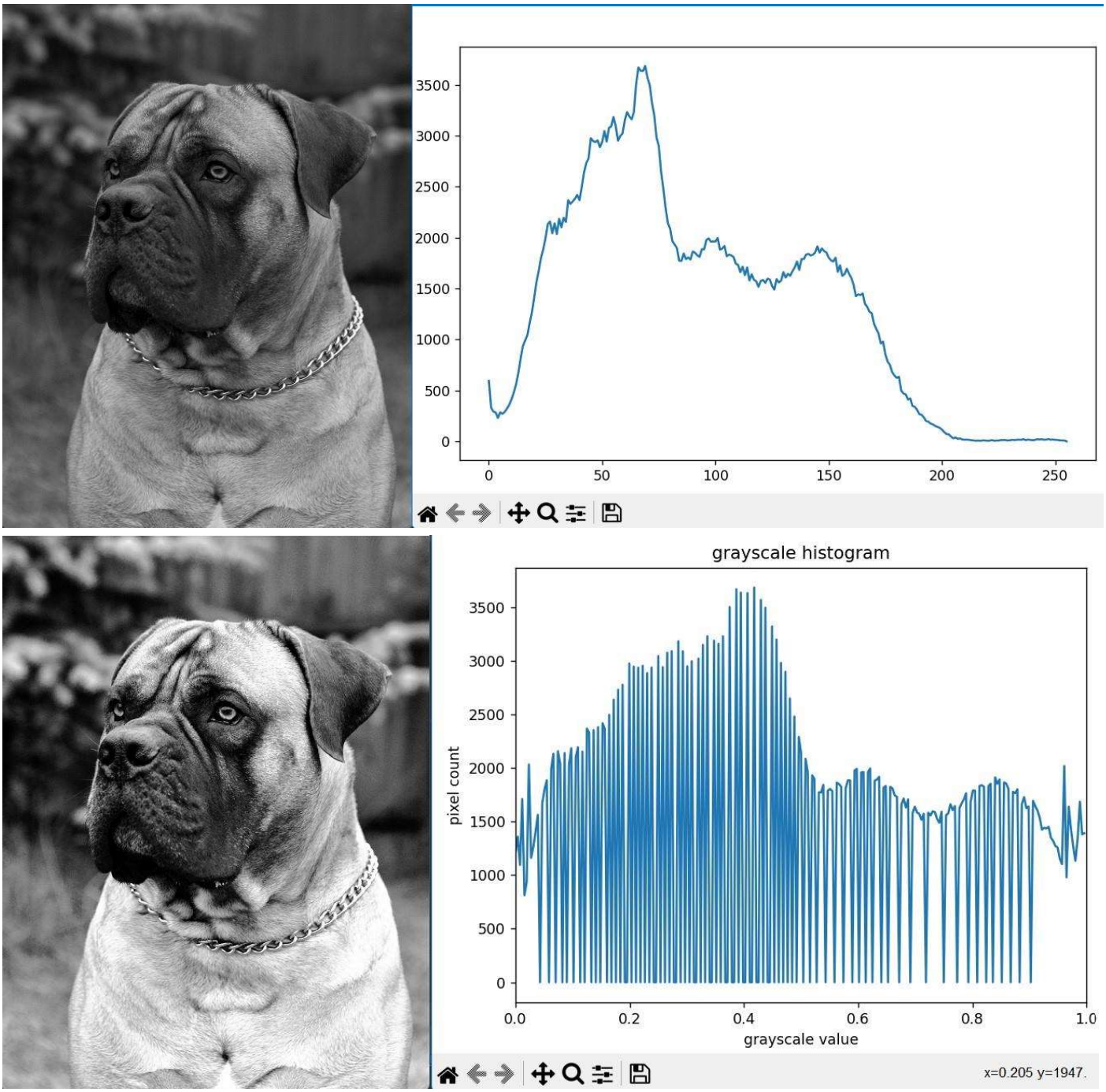
A color histogram of an image represents the number of pixels in each type of color component. Histogram equalization cannot be applied separately to the Red, Green and Blue components of the image as it leads to dramatic changes in the image's color balance. However, if the image is first converted to another color space, like HSL/HSV color space, then the algorithm can be applied to the luminance or value channel without resulting in changes to the hue and saturation of the image.

CODE:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('boerboel1.jpg', 0)
cv2.imshow("original image", img)


histr = cv2.calcHist([img], [0], None, [256], [0, 255])
plt.plot(histr)
plt.show()

h, w = img.shape[:2]
intensity_count = [0] * 256
new_img = np.zeros(img.shape)
for i in range(0, h):
    for j in range(0, w):
        intensity_count[img[i, j]] += 1
l = 256

res = []
for val in intensity_count:
    res.append(val / img.size)

cum_list = []
k = 0
for x in range(0, len(res)):
    k += res[x]
    cum_list.append(k)

intensity = cum_list * (l - 1)

for x in range(0, h):
    for y in range(0, w):
        new_img[x, y] = intensity[img[x, y]]
```

```
cv2.imshow("new image", new_img)

histo, bin_edges = np.histogram(new_img, bins=256, range=(0, 1))
plt.figure()
plt.title("grayscale histogram")
plt.xlabel("grayscale value")
plt.ylabel("pixel count")
plt.xlim([0.0, 1.0])
plt.plot(bin_edges[0:-1], histo)
plt.show()
```

OUTPUT:



DISCUSSION:

In this case a grey image was taken and histogram equilisation was performed to equilised output.

PROBLEM STATEMENT:

Write a program to enhance an image using mean filtering, weighted average filtering, median filtering, max/min filtering.

THEORY:

- **Mean filter:** Mean filtering is a simple, intuitive and easy to implement method of *smoothing* images, *i.e.* reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images. The idea of mean filtering is simply to replace each pixel value in an image with the mean (`average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean.
- **Median filter:** The median filter is normally used to reduce noise in an image, somewhat like the mean filter. However, it often does a better job than the mean filter of preserving useful detail in the image. Like the mean filter, the median filter considers each pixel in the image in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the *mean* of neighboring pixel values, it replaces it with the *median* of those values.
- **Weighted-average filter:** Terminology used to indicate that pixels are multiplied by different coefficients, thus giving more importance (weight) to some pixels at the expense of others. In the mask the pixel at the center of the mask is multiplied by a higher value than any other, thus giving this pixel more importance in the calculation of the average.

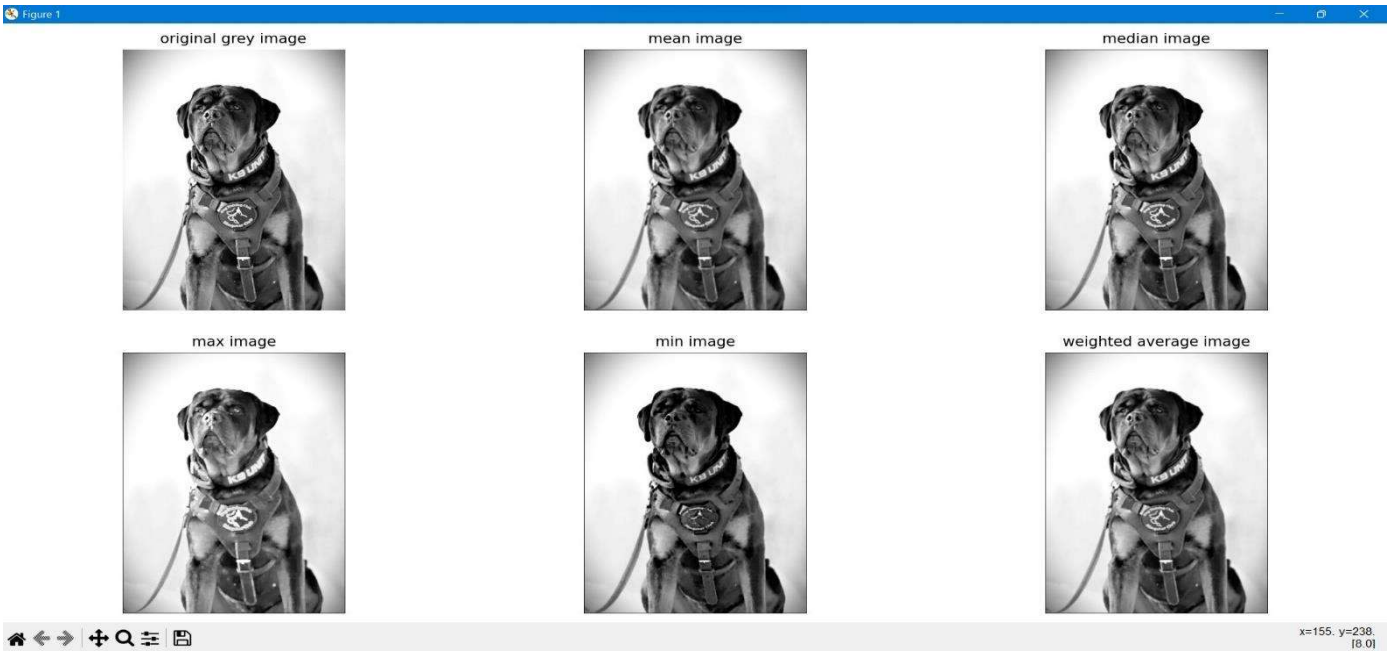
- **Max filter:** The maximum and minimum filters are shift-invariant. Whereas the minimum filter replaces the central pixel with the darkest one in the running window, the maximum filter replaces it with the lightest one. For example, if you have a text string drawn with a thick pen, you can make the sign skinnier.

- **Min filter:** The transformation replaces the central pixel with the darkest one in the running window.For example, if you have text that is lightly printed, the minimum filter makes letters thicker.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


def mean(img):
    h, w = img.shape
    kernel = np.ones([3, 3], dtype=int)
    kernel = kernel/9
    img_new = np.zeros([h, w], dtype=np.uint8)

    for i in range(1, h-1):
        for j in range(1, w-1):
            temp = (img[i-1][j-1]*kernel[0, 0]+img[i-1][j]*kernel[0, 1]+img[i-
1][j+1]*kernel[0, 2]+img[i][j-1]*kernel[1, 0]+img[i][j]*kernel[1,
1]+img[i][j+1]*kernel[1, 2]+img[i+1][j-1]*kernel[2, 0]+img[i+1][j]*kernel[2,
1]+img[i+1][j+1]*kernel[2, 2])
            img_new[i][j] = temp

    return img_new


def median(img):
    h, w = img.shape
    img_new = np.zeros([h, w], dtype=np.uint8)
    for i in range(1, h-1):
        for j in range(1, w-1):
            temp = [img[i-1, j-1], img[i-1, j], img[i-1, j+1], img[i, j-1], img[i, j],
img[i, j+1], img[i+1, j-1], img[i + 1, j], img[i+1, j+1]]
            temp = sorted(temp)
            img_new[i, j] = temp[4]
    return img_new


def max1(img):
    h, w = img.shape
    img_new = np.zeros([h, w], dtype=np.uint8)
    for i in range(1, h-1):
        for j in range(1, w-1):
            temp = [img[i-1, j-1], img[i-1, j], img[i-1, j+1], img[i, j-1], img[i, j],
img[i, j+1], img[i+1, j-1], img[i+1, j], img[i+1, j+1]]
            temp = max(temp)
            img_new[i, j] = temp
    return img_new


def min1(img):
    h, w = img.shape
    img_new = np.zeros([h, w], dtype=np.uint8)
    for i in range(1, h-1):
        for j in range(1, w-1):
            temp = [img[i-1, j-1], img[i-1, j], img[i-1, j+1], img[i, j-1], img[i, j],
img[i, j+1], img[i+1, j-1], img[i+1, j], img[i+1, j+1]]
            temp = min(temp)                        # use min(temp) for minimum filter
            img_new[i, j] = temp
    return img_new


def weighted_avg(kernal, img):
    m = int(np.floor(len(kernal) / 2))
    n = int(np.floor(len(list(zip(*kernal)))) / 2)
    k = np.sum(kernal)
    #print(k)
    rows, cols = img.shape
    blurred_img = np.zeros((rows, cols), np.uint8)
```

```
    for i in range(m, rows - m):
        for j in range(n, cols - n):
            sum = 0
            for x in range(-m, m + 1, 1):
                for y in range(-n, n + 1, 1):
                    sum = sum + (img[i + x, j + y] * kernal[m - x][n - y])
            sum = sum / k
            blurred_img[i, j] = sum
    return blurred_img


img = cv2.imread('rott.jpg', 0)
kernal1 = [[1, 2, 1], [2, 4, 2], [1, 2, 1]]

weighted_avg = weighted_avg(kernal1, img)
mean = mean(img)
median = median(img)
max1 = max1(img)
min1 = min1(img)

fig = plt.figure(figsize=(16, 10))
plt_x = 2
plt_y = 3
fig.add_subplot(plt_x, plt_y, 1)
plt.imshow(img, cmap="gray")
plt.axis("off")
plt.title("original grey image")

fig.add_subplot(plt_x, plt_y, 2)
plt.imshow(mean, cmap="gray")
plt.axis("off")
plt.title("mean image")

fig.add_subplot(plt_x, plt_y, 3)
plt.imshow(median, cmap="gray")
plt.axis("off")
plt.title("median image")

fig.add_subplot(plt_x, plt_y, 4)
plt.imshow(max1, cmap="gray")
plt.axis("off")
plt.title("max image")

fig.add_subplot(plt_x, plt_y, 5)
plt.imshow(min1, cmap="gray")
plt.axis("off")
plt.title("min image")

fig.add_subplot(plt_x, plt_y, 6)
plt.imshow(weighted_avg, cmap="gray")
plt.axis("off")
plt.title("weighted average image")

plt.show()
```

OUTPUT:



DISCUSSION:

In this case a Grey image was taken and different filters were applied by applying respective masks to get the desired output image.

PROBLEM STATEMENT:

Write a program to find the edge of a given image with the following operators. a) Difference operator

a. Robert operator.
b. Sobel operator.
c. Prewitt operator.

THEORY:

Robert operator: Robert's cross operator is used to perform 2-D spatial gradient measurement on an image which is simple and quick to compute. In Robert's cross operator, at each point pixel values represents the absolute magnitude of the input image at that point.

Robert's cross operator consists of 2x2 convolution kernels. Gx is a simple kernel and Gy is rotated by 90o

+1	0
0	-1

Gx

0	+1
-1	0

Gy

Sobel operator: The Sobel edge detection operator extracts all the edges of an image, without worrying about the directions. The main advantage of the Sobel operator is that it provides differencing and smoothing effect.Sobel edge detection operator is implemented as the sum of two directional edges. And the resulting image is a unidirectional outline in the original image. Sobel Edge detection operator consists of 3x3 convolution kernels.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

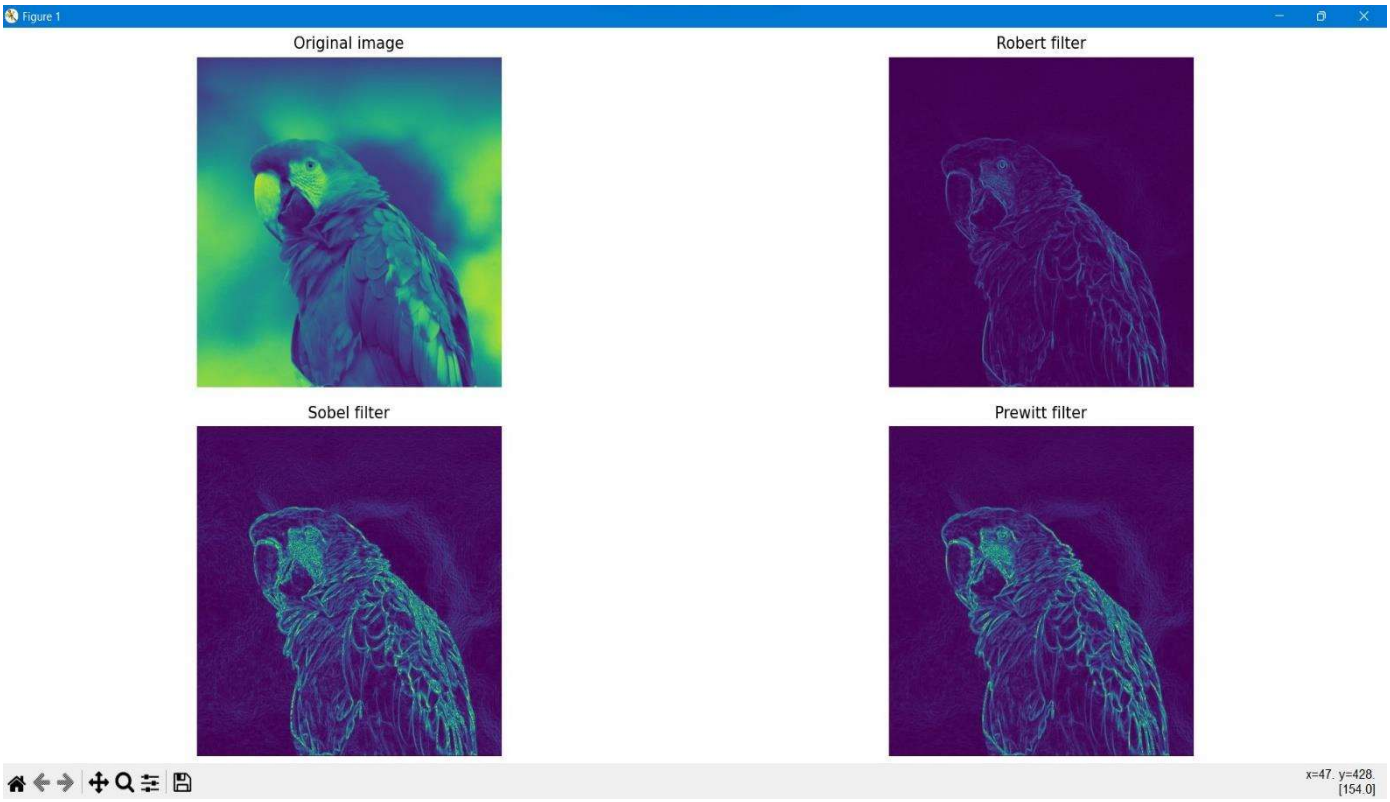
+1	+2	+1
0	0	0
-1	-2	-1

Gy

Prewitt operator: Prewitt operator is a differentiation operator. Prewitt operator is used for calculating the approximate gradient of the image intensity function. In an image, at each point, the Prewitt operator results in gradient vector or normal vector. In Prewitt operator, an image is convolved in the horizontal and vertical direction with small, separable and integer-valued filter. It is inexpensive in terms of computations.

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt


def filter(kx, ky, img):
    m = int(np.floor(len(kx) / 2))
    n = int(np.floor(len(list(zip(*kx)))) / 2)
    rows, cols = img.shape
    filtered_img = np.zeros((rows, cols), np.uint8)
    for i in range(m, rows - m):
        for j in range(n, cols - n):
            sumx = sumy = 0
            for x in range(-m, m + 1, 1):
                for y in range(-n, n + 1, 1):
                    sumx = sumx + (img[i + x, j + y] * kx[m + x][n + y])
                    sumy = sumy + (img[i + x, j + y] * ky[m + x][n + y])
            filtered_img[i, j] = abs(sumx) + abs(sumy)
    return filtered_img


img = cv.imread("scarlett.jpg", 0)

y = [[0, 0, 0], [0, -1, 0], [0, 0, 1]]
x = [[0, 0, 0], [0, 0, -1], [0, 1, 0]]
robert = filter(x, y, img)
y = [[-1, -2, -1], [0, 0, 0], [1, 2, 1]]
x = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]
sobel = filter(x, y, img)
y = [[-1, -1, -1], [0, 0, 0], [1, 1, 1]]
x = [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]
prewitt = filter(x, y, img)

fig = plt.figure(figsize=(17, 12))
fig.add_subplot(2, 2, 1)
plt.imshow(img)
plt.axis('off')
plt.title('Original image')

fig.add_subplot(2, 2, 2)
plt.imshow(robert)
plt.axis('off')
plt.title('Robert filter')

fig.add_subplot(2, 2, 3)
plt.imshow(sobel)
plt.axis('off')
plt.title('Sobel filter')

fig.add_subplot(2, 2, 4)
plt.imshow(prewitt)
plt.axis('off')
plt.title('Prewitt filter')

plt.show()
```

OUTPUT:



DISCUSSION:

A Grey level image was taken and different filters such as Robert, Sobel and Prewitt filters were applied on it by applying their respective masks in the original grey image taken for performing the operation to get the desired output.

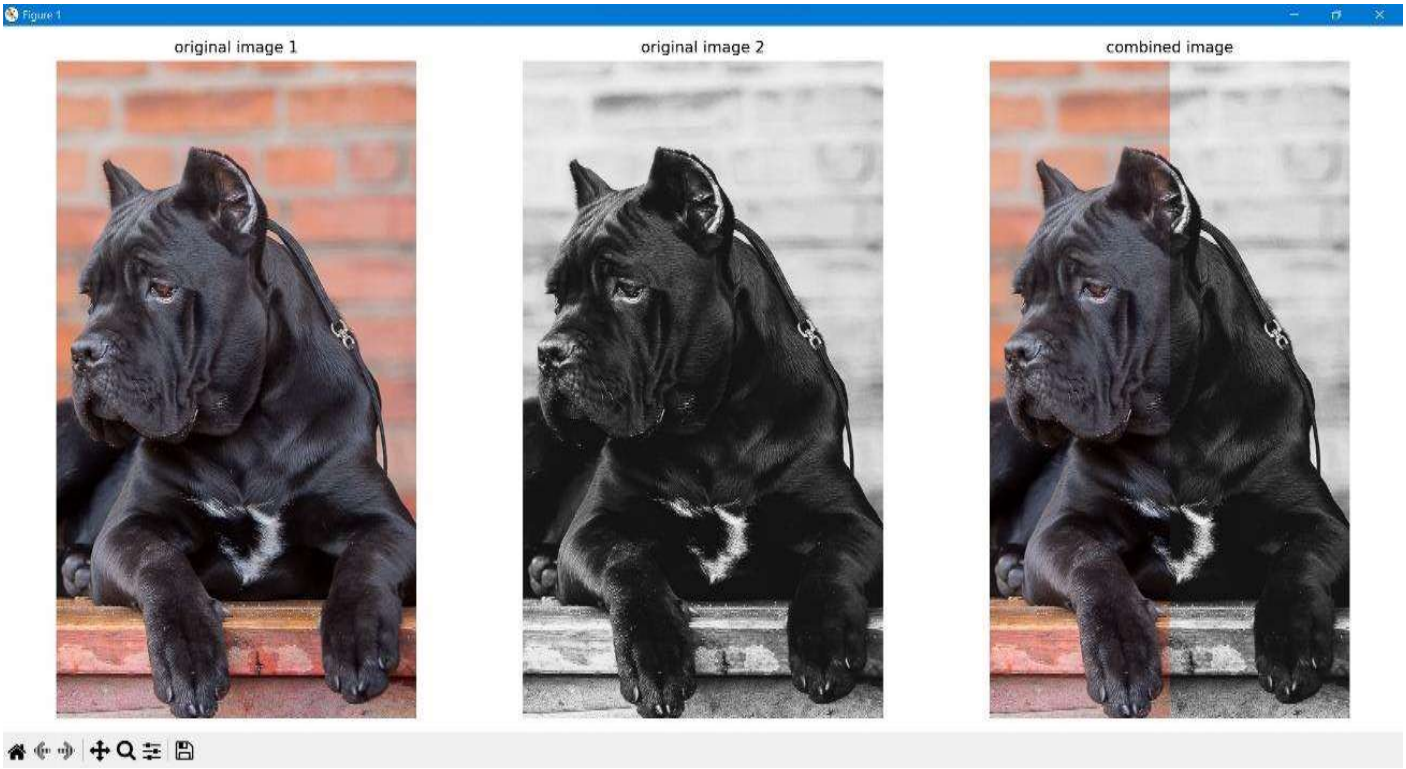
PROBLEM STATEMENT:

Write a program to read two images ‘lena.bin’ and peppers.bin’. define a new 256 x 256 image J as follows : the left half of J i.e., the first 128 columns, should be equal to the left half of lena image and the right half of J, i.e., the 129th column through the 256th column should be equal to the right half of pepper image. Show J image.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

lena = mpimg.imread("corso.jpg")
peppers = cv2.imread("corso1.jpg")

rows, cols, _ = lena.shape

J = np.zeros((rows, cols, _), dtype=np.uint8)
for i in range(rows):
    for j in range(0, int(cols / 2)):
        J[i, j] = lena[i, j]
for i in range(rows):
    for j in range(int(cols / 2), cols):
        J[i, j] = peppers[i, j]

fig = plt.figure(figsize=(16, 10))
plt_x = 1
plt_y = 3
fig.add_subplot(plt_x, plt_y, 1)
plt.imshow(lena)
plt.axis("off")
plt.title("original image 1")

fig.add_subplot(plt_x, plt_y, 2)
plt.imshow(peppers)
plt.axis("off")
plt.title("original image 2")

fig.add_subplot(plt_x, plt_y, 3)
plt.imshow(J)
plt.axis("off")
plt.title("combined image")

plt.show()
```

OUTPUT:



DISCUSSION:

Two images ‘lena.bin’ and ‘peppers.bin’ of dimension 256 x 256 pixels were taken and both images were conbined where the combined image contains the left half of lena.bin image and the right half of peppers.bin image, and ultimately the combined image was displayed.