EXPERIMENT : 01

Aim:

"Write a Program to display the negative of a digital image."

Theory:

An image negative is obtained by inverting the pixel values. In a typical 8-bit image, pixel values range from 0 to 255. The negative of an image is computed by inverting these values such that the darker areas of the image become lighter, and the lighter areas become darker. For an RGB image, the negative is computed for each of the three channels (Red, Green, Blue) separately.
The formula for converting an image to its negative is:

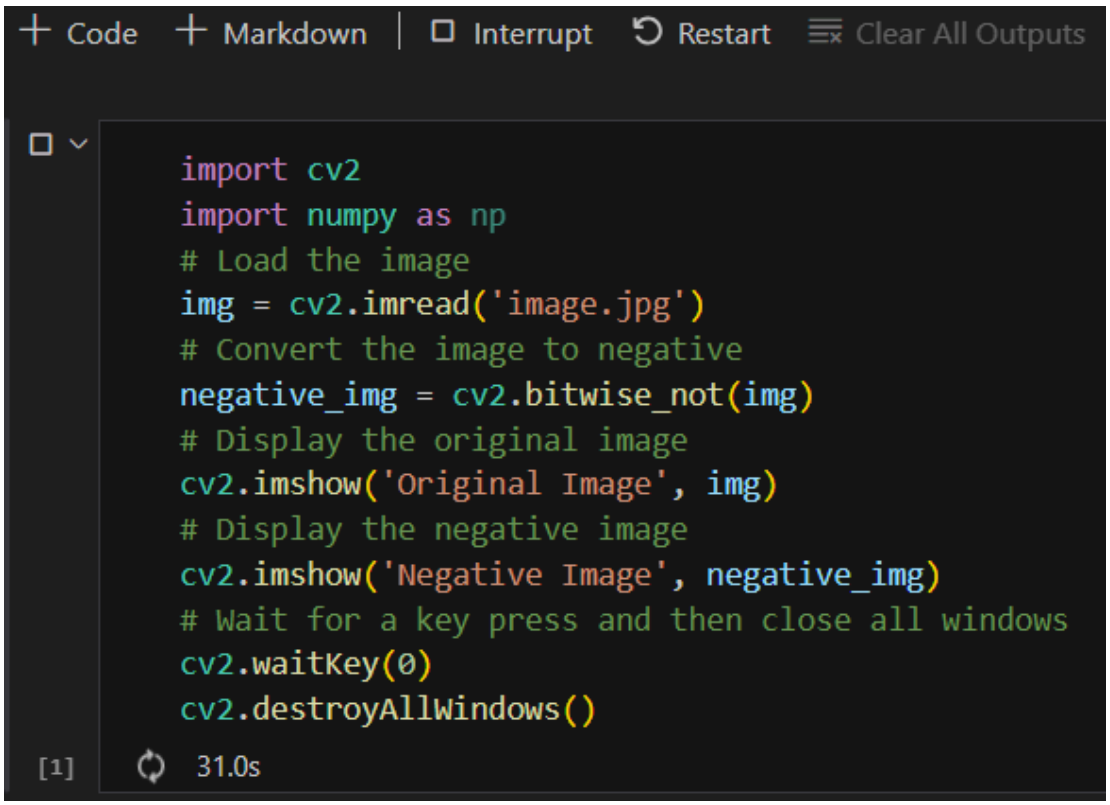
$$I_{neg}(x,y)=255-I(x,y)$$

Where:

- $I(x,y)$ is the pixel value at position (x,y).
- $I_{neg}(x,y)$ is the pixel value in the negative image.

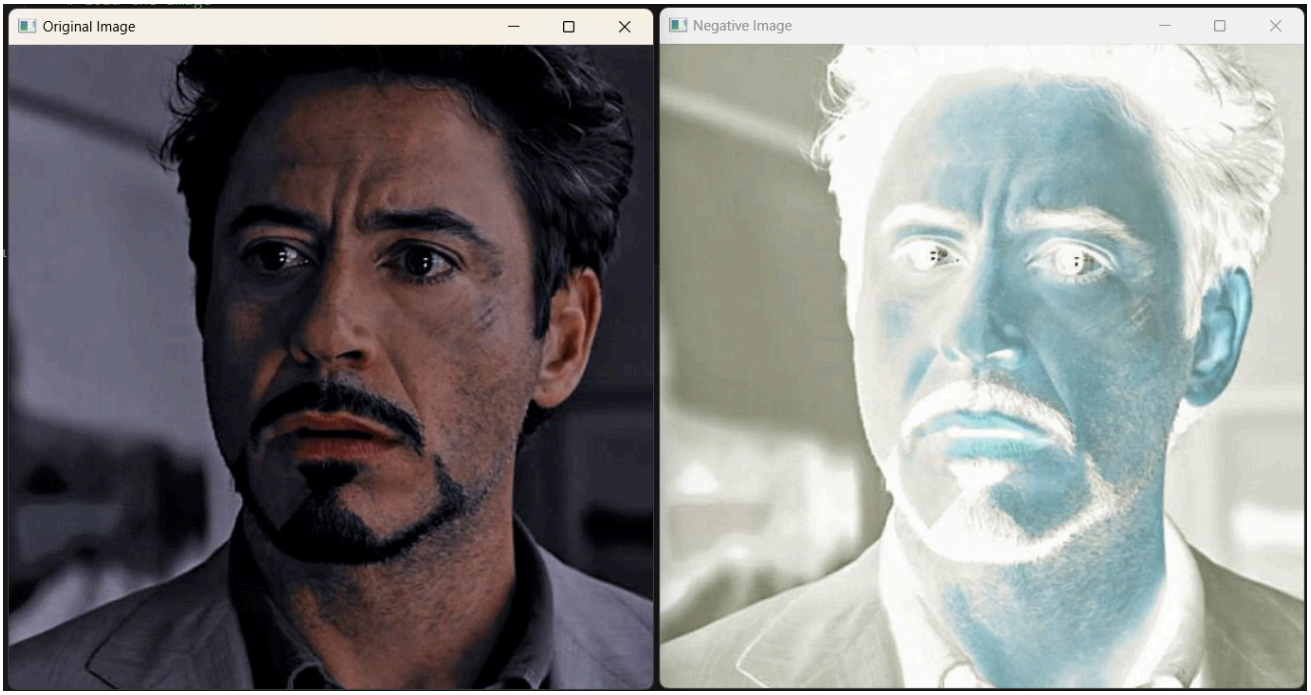
In this practical, we use OpenCV's `bitwise_not()` function, which efficiently inverts pixel values to produce a negative image.

Code :



Output:

Negative of a Digital Image.



EXPERIMENT : 02

Aim:

Write a Program to perform thresholding on an input image.

Theory:

Thresholding is a fundamental image segmentation technique that converts a grayscale image into a binary image. It works by setting a pixel intensity threshold. Pixels with intensity values below the threshold are set to 0 (black), while those above are set to 255 (white). This simplifies image data, making it easier to detect objects or regions of interest, such as in OCR (Optical Character Recognition) or medical imaging. Thresholding is particularly effective when there is a clear distinction between foreground and background intensities, allowing for quick isolation of objects from the background in an image.

Code :

```
import cv2
# Loading the image named test.jpg
img = cv2.imread("image.jpg")
# Converting color mode to Grayscale
# as thresholding requires a single channeled image
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('Image', img)
# Thresholding the image placing 127 intensity level as threshold
# Pixel values below 127 would be changed to Black
# Pixel values above 127 would be changed to White (255)
ret, thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
# Displaying the output image
cv2.imshow('Binary Threshold', thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:

EXPERIMENT : 03

Aim:

Write a Program to perform gray level slicing without background.

Theory:

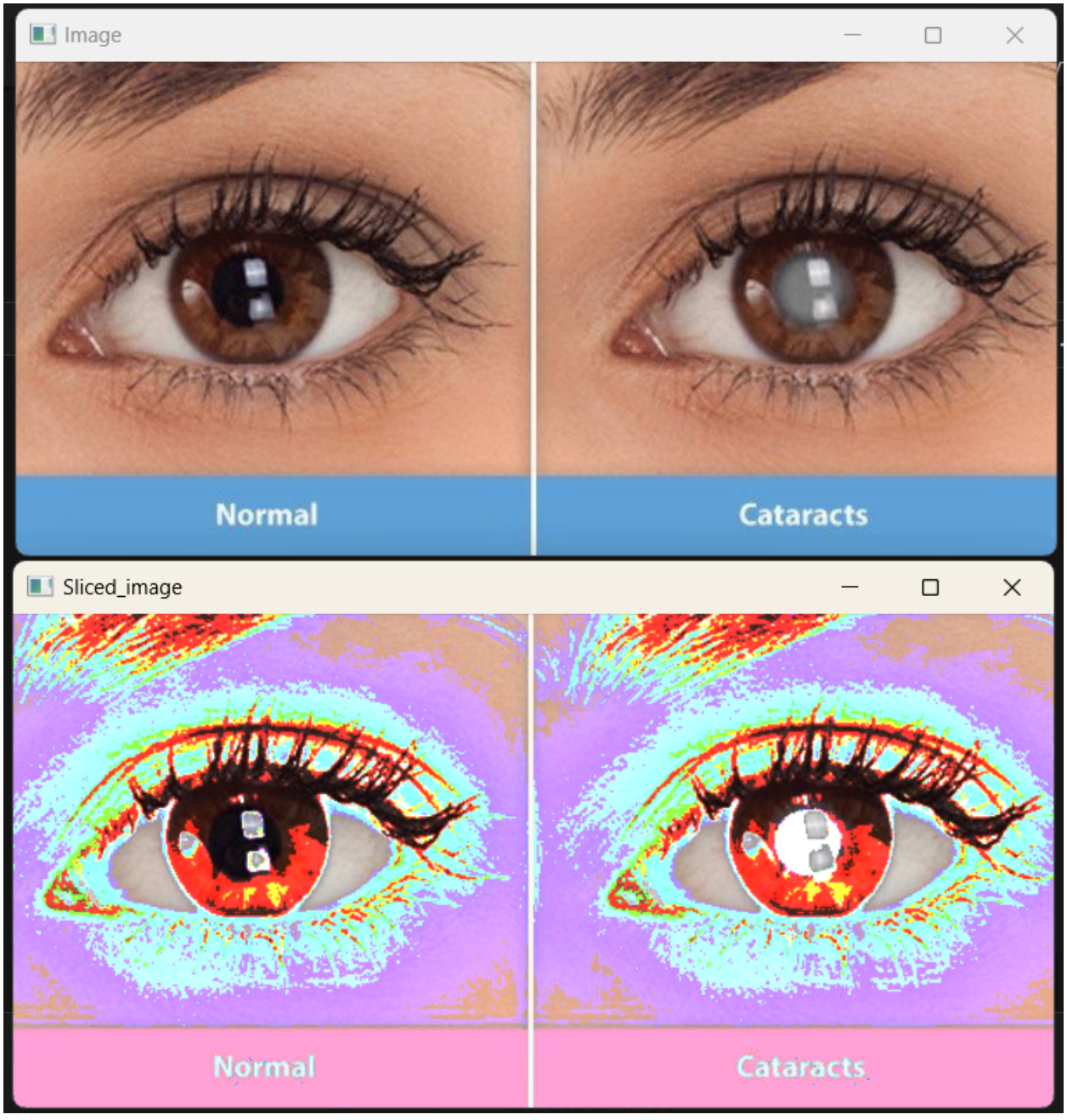
Gray Level Slicing Without Background: Gray level slicing is used to highlight specific intensity ranges in an image, focusing on regions of interest. In slicing without background, pixels outside the specified intensity range are set to black, removing all non-essential details from the image. This technique is useful for emphasizing certain features like abnormalities in medical images, such as tumors or other structures, without distraction from the background. The method is simple yet effective in enhancing the visibility of specific parts of an image, making it easier to analyze the important details while suppressing irrelevant areas.

Code :

```
import numpy as np
import cv2
img = cv2.imread('eye.jpg')
row, column, layer = img.shape
img1 = np.zeros((row,column, layer), dtype = 'uint8')
min_range = 80
max_range = 140
for i in range(row):
    for j in range(column):
        for k in range(layer):
            if img[i][j][k] in range(min_range, max_range):
                img1[i][j][k] = 255
            else:
                img1[i][j][k] = img[i][j][k]
cv2.imshow("Image",img)
cv2.imshow("Sliced_image",img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[2] ✓ 1m 1.4s

Output:



EXPERIMENT : 04

Aim:

Write a Program to perform gray level slicing with the background.

Theory:

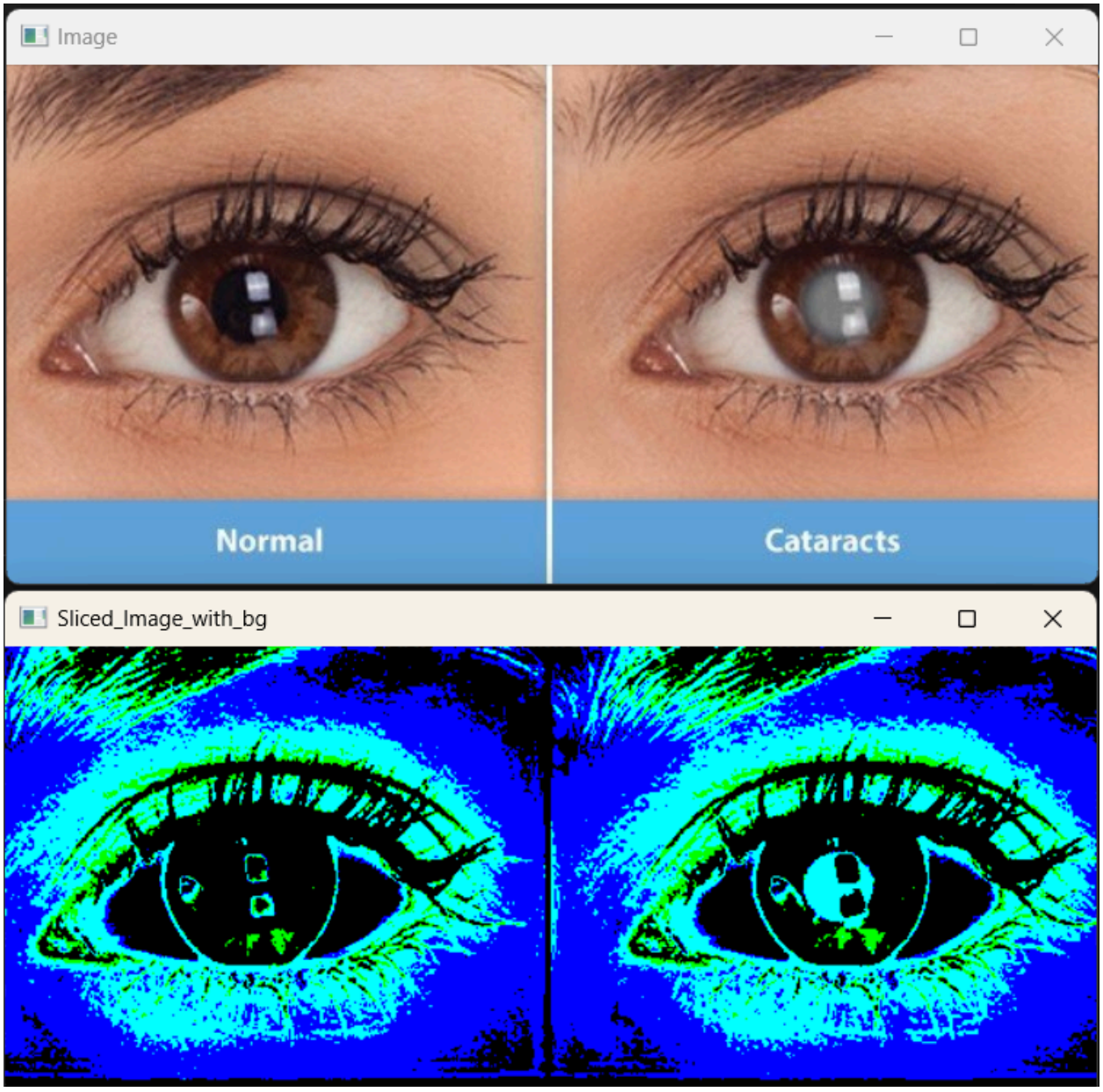
Gray Level Slicing With Background: Gray level slicing with background retains the original image context while enhancing a particular intensity range. Pixels within the defined range are highlighted, often set to the maximum intensity value, while pixels outside the range retain their original intensity. This method preserves the background, making it easier to focus on specific regions of interest without losing the overall structure and context of the image. It's useful for applications like medical imaging, where both the highlighted feature and the surrounding tissues need to be visible for accurate analysis and diagnosis.

Code :

```
import numpy as np
import cv2
img = cv2.imread('eye.jpg')
row, column, layer = img.shape
img1 = np.zeros((row,column, layer), dtype = 'uint8')
min_range= 80
max_range = 140
for i in range(row):
    for j in range(column):
        for k in range(layer):
            if img[i][j][k] in range(min_range, max_range):
                img1[i][j][k] = 255
            else:
                img1[i][j][k]=0
cv2.imshow("Image",img)
cv2.imshow("Sliced_Image_with_bg",img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[4] ✓ 49.4s

Output:



EXPERIMENT : 05

Aim:

Write a Program to perform bit-plane slicing

Theory:

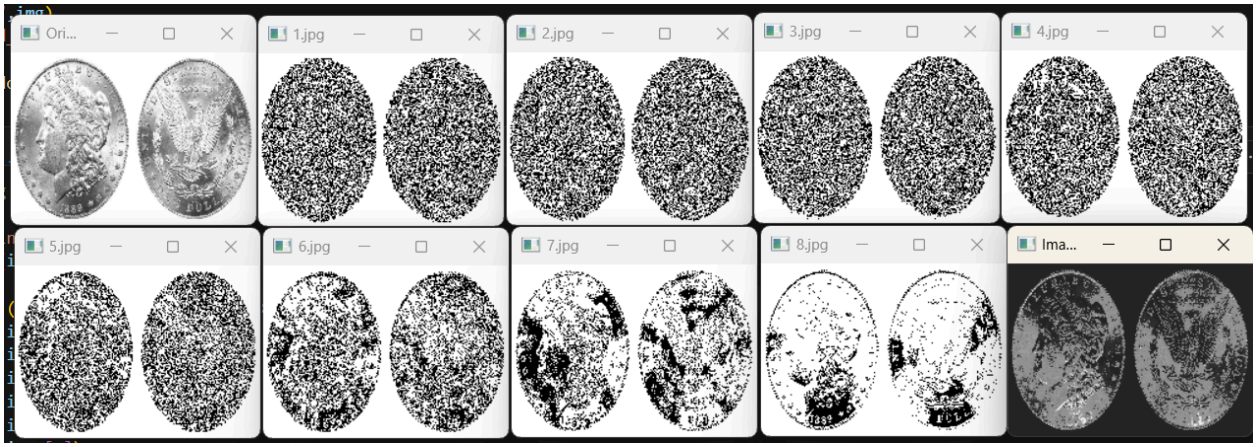
Bit-Plane Slicing: Bit-plane slicing breaks down an image into its constituent binary planes, representing each bit of the pixel's intensity value separately. Each pixel in an 8-bit image is represented by 8 bits, ranging from the most significant bit (which contains coarse details) to the least significant bit (which contains fine details). By isolating individual bit planes, it's possible to analyze which bits carry the most significant visual information. Higher bit planes contain the majority of the structural information, while lower planes contribute finer details. This technique is useful for image compression, watermarking, and noise removal.

Code :

```
#bit plane slicing
import cv2 as cv
img=cv.imread('coin.jpg')
img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
img=cv.resize(img, (200,140))
imgs=[255*((img & (1<<i))>>i) for i in range(8)]
cv.imshow('1.jpg',imgs[0])
cv.imshow('2.jpg',imgs[1])
cv.imshow('3.jpg',imgs[2])
cv.imshow('4.jpg',imgs[3])
cv.imshow('5.jpg',imgs[4])
cv.imshow('6.jpg',imgs[5])
cv.imshow('7.jpg',imgs[6])
cv.imshow('8.jpg',imgs[7])
new_img=(imgs[7]*100)+(imgs[6]*100)+(imgs[5]*10)+(imgs[4]*10)
cv.imshow('Image using 8,7,6,5th bits',new_img)
cv.imshow('Original', img)
cv.waitKey(0)
```

[5] ✓ 2m 57.0s

Output:



EXPERIMENT : 06

Aim:

Write a Program to display a Histogram of an image.

Theory:

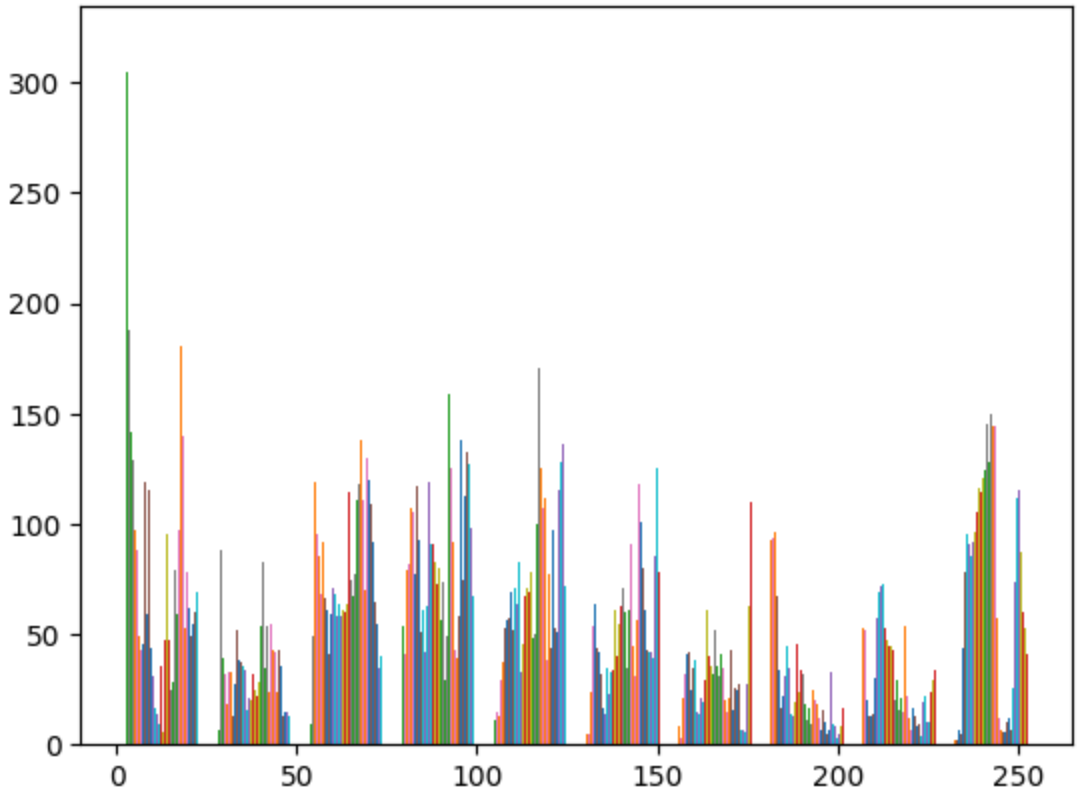
Histogram of an Image: A histogram is a graphical representation of an image's pixel intensity distribution. It plots the number of pixels for each intensity value, ranging from 0 (black) to 255 (white) in an 8-bit image. Analyzing a histogram provides insights into the image's contrast, brightness, and overall dynamic range. Images with a narrow histogram may appear too dark or too bright, while those with a wider histogram have better contrast. Histograms are frequently used in image enhancement techniques, such as histogram equalization, to improve image contrast by redistributing the intensity values for better visualization and detail extraction.

Code :

```
import matplotlib.pyplot as plt #importing matplotlib
import numpy as np
import cv2
img = cv2.imread('image.jpg')
row, column, layer = img.shape
print(row,column)
cv2.imshow("Original", img)
a=[[0 for i in range(column)] for i in range(row)]
for i in range(row):
    for j in range(column):
        a[i][j]=img[i][j][0]
plt.hist(a)
plt.show()
cv2.waitKey(0)
```

[6] ✓ 1m 4.5s

Output:



EXPERIMENT 7

Aim:

Write a Program to perform a Log Transformation of an image

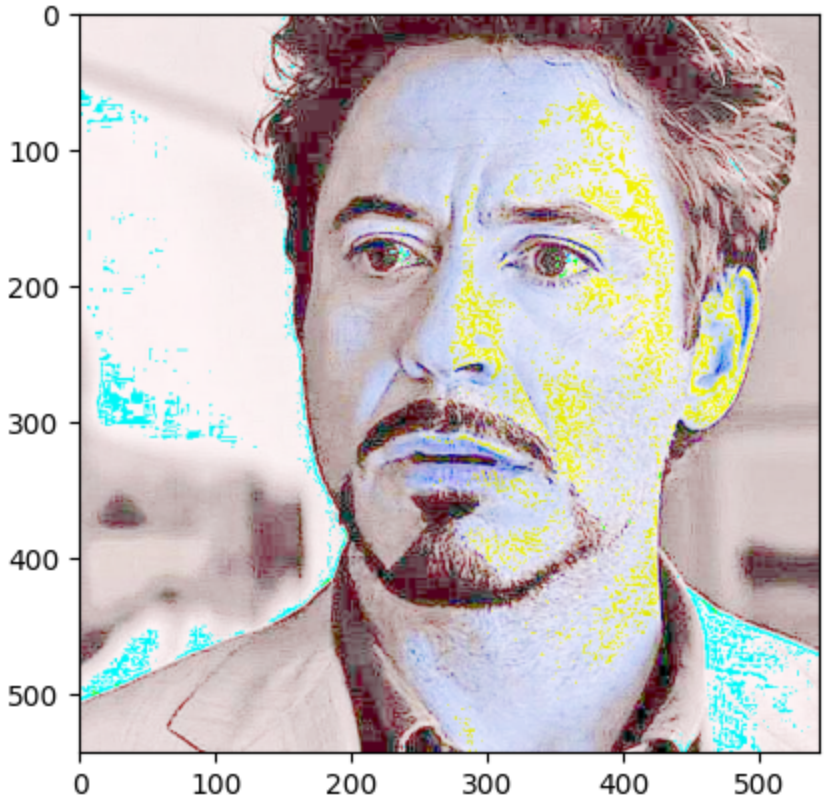
Theory:

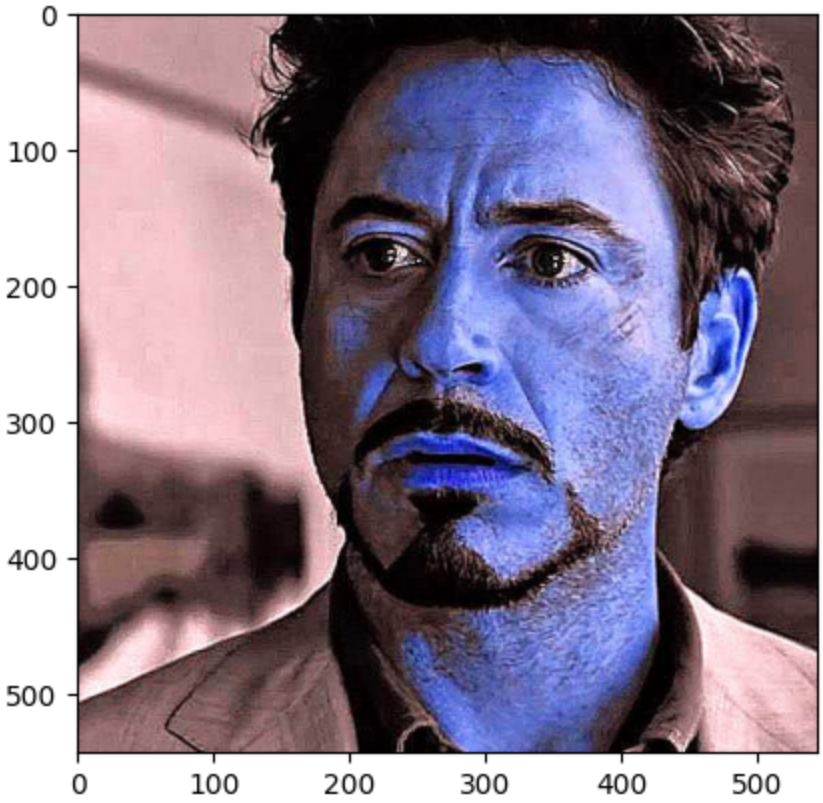
Log Transformation: Log transformation is a technique used to enhance the dynamic range of an image, especially when there is a large variation between dark and bright regions. It applies a logarithmic function to pixel intensities, compressing the higher intensity values while expanding the lower ones. This transformation is particularly useful for highlighting details in darker areas without oversaturating the bright regions, making it ideal for applications in medical imaging or satellite imagery. Log transformation increases the visibility of fine details in low-intensity regions, improving the interpretability of the image, especially when dealing with wide-ranging brightness levels.

Code :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read an image
image = cv2.imread('image.jpg')
# Apply log transformation method
c = 255 / np.log(1 + np.max(image))
log_image = c * (np.log(image + 1))
# Specify the data type so that
# float value will be converted to int
log_image = np.array(log_image, dtype = np.uint8)
# Display both images
plt.imshow(image)
plt.show()
plt.imshow(log_image)
plt.show()
```

Output:


0
100
200
300
400
500
0
100
200
300
400
500


0
100
200
300
400
500
0
100
200
300
400
500