Manthan Dhole
201080020

Digital Image Processing(DIP)Lab
Experiment 6

Aim: Take TWO input RGB images. Perform the following task.

1. Display the color histogram.
2. Display the grayscale histogram.
3. Compare the histograms and analyze them.
4. Apply histogram equalization and display new generated images with histogram.(for input image)
5. Separate the RGB channels

Theory:

The aim of this experiment is to analyze the color and grayscale histograms of two input RGB images, and apply histogram equalization to generate new images with adjusted histograms. Additionally, the experiment involves separating the RGB channels of the input images to manipulate individual color components. Histograms are graphical representations of the distribution of pixel intensities in an image. In color images, the histogram is generated for each color channel (red, green, and blue) separately, while in grayscale images, the histogram is based on the intensity values.

The color histogram provides information about the distribution of colors in the image, while the grayscale histogram shows the distribution of the intensity values. Histogram equalization is a technique used to adjust the contrast of an image by redistributing the pixel values to increase the number of pixels with intermediate intensities. The separation of RGB channels allows for the isolation and manipulation of individual color components. This can be useful for various image processing tasks, such as color correction and enhancing certain color features in the image.

Steps:

1. Convert the input images from BGR color space to grayscale using cv2.cvtColor() function.
2. Extract the individual color channels (Red, Green, and Blue) from each input image.
3. Plot a histogram for each color channel (R, G, B) for both input images using plt.hist() function and display them using plt.show() function.

4. Plot a histogram for the grayscale version of each input image and display them using plt.show() function.
5. Combine the histograms for each color channel (R, G, B) of both input images into a single plot and display them using plt.show() function.
6. Combine the histograms for the grayscale version of both input images into a single plot and display them using plt.show() function.
7. Apply histogram equalization to the grayscale version of both input images using cv2.equalizeHist() function.
8. Combine the original and equalized grayscale images of each input image into a single plot and display them using plt.show() function.
9. Extract the individual color channels (R, G, B) of each input image and set the other two channels to zero.
10. Combine the individual color channels of each input image into a single plot and display them using cv2.imshow() function and cv2.waitKey() function.

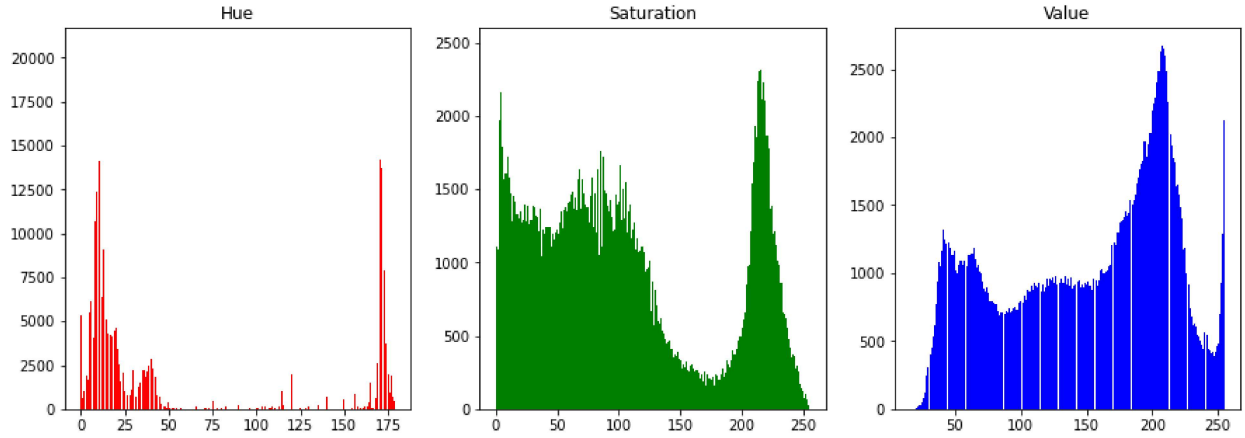
Code:

1.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
img1 = cv2.imread('image1.png', cv2.IMREAD_COLOR)
img1 = cv2.resize(img1, (500,500))
img2 = cv2.imread('image2.png', cv2.IMREAD_COLOR)
img2 = cv2.resize(img2, (700,500))
hsv_img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2HSV)
h, s, v = cv2.split(hsv_img1)
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].hist(h.flatten(), bins=256, color='r')
axs[1].hist(s.flatten(), bins=256, color='g')
axs[2].hist(v.flatten(), bins=256, color='b')
axs[0].set_title('Hue')
axs[1].set_title('Saturation')
axs[2].set_title('Value')
plt.show()
hsv_img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2HSV)
```

```
h, s, v = cv2.split(hsv_img2)
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].hist(h.flatten(), bins=256, color='r')
axs[1].hist(s.flatten(), bins=256, color='g')
axs[2].hist(v.flatten(), bins=256, color='b')
axs[0].set_title('Hue')
axs[1].set_title('Saturation')
axs[2].set_title('Value')
plt.show()
```

Output:



2. Code:

```
gray_img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
fig, axs = plt.subplots(1, 1, figsize=(5, 5))
axs.hist(gray_img1.flatten(), bins=256, color='k')
axs.set_title('Grayscale')
plt.show()
```

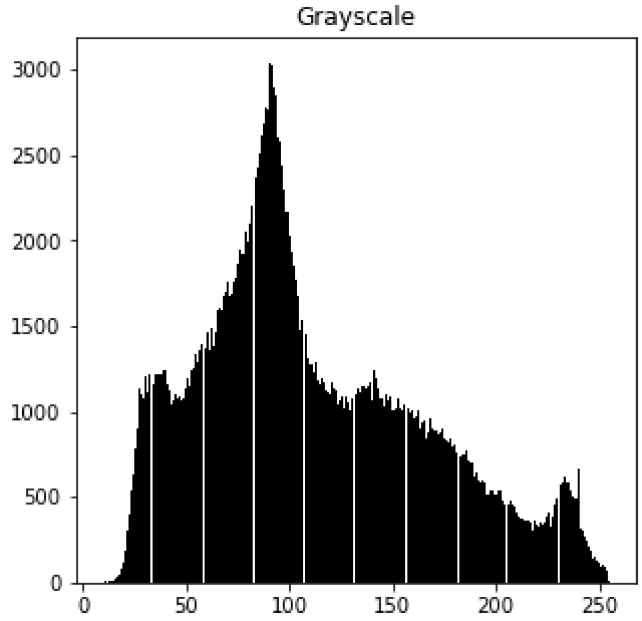
Output:

Manthan Dhole

201080020

Digital Image Processing(DIP)Lab

Experiment 6



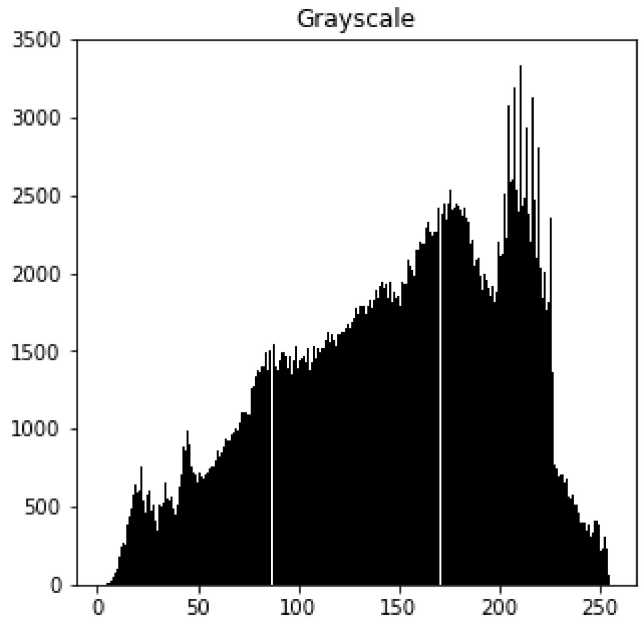

```
gray_img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
fig, axs = plt.subplots(1, 1, figsize=(5, 5))
axs.hist(gray_img2.flatten(), bins=256, color='k')
axs.set_title('Grayscale')
plt.show()
```

Output:

Manthan Dhole
201080020

Digital Image Processing(DIP)Lab

Experiment 6



3.

Code:

```
hist1 = cv2.calcHist([img1], [0, 1, 2], None, [256, 256, 256], [0, 255, 0, 255, 0, 255])
hist2 = cv2.calcHist([img2], [0, 1, 2], None, [256, 256, 256], [0, 255, 0, 255, 0, 255])
diff = cv2.compareHist(hist1, hist2, cv2.HISTCMP_BHATTACHARYYA)
print(diff)
```

Output:

0.934629146345223 10

4.

Code:

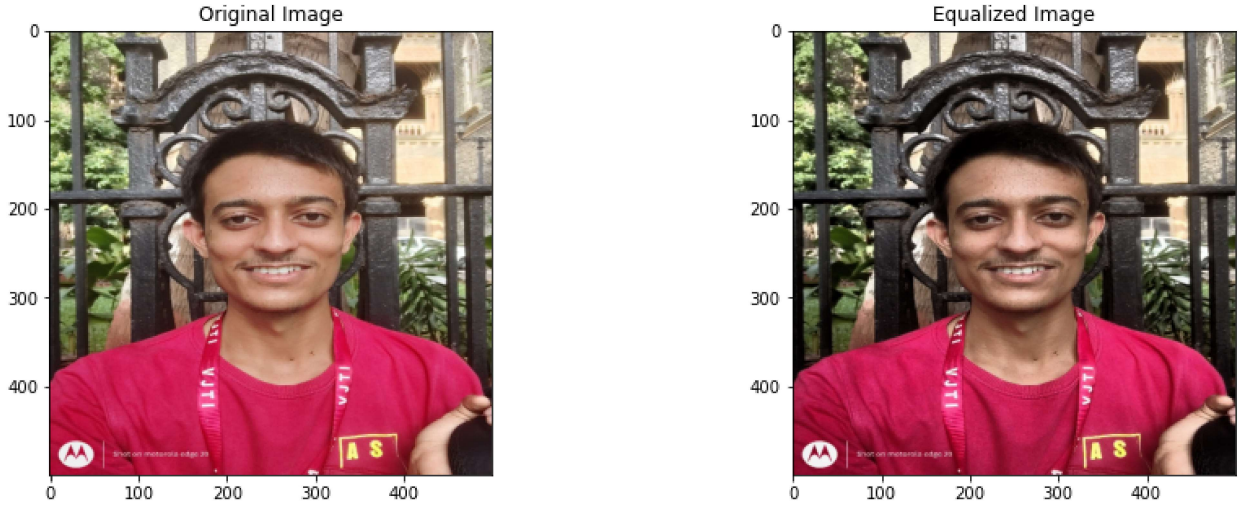
```
hsv_img1[..., 2] = cv2.equalizeHist(hsv_img1[..., 2])
img1_equalized = cv2.cvtColor(hsv_img1, cv2.COLOR_HSV2BGR)
fig, axs = plt.subplots(1, 2, figsize=(15, 5))
```

Manthan Dhole
201080020

Digital Image Processing(DIP)Lab

Experiment 6

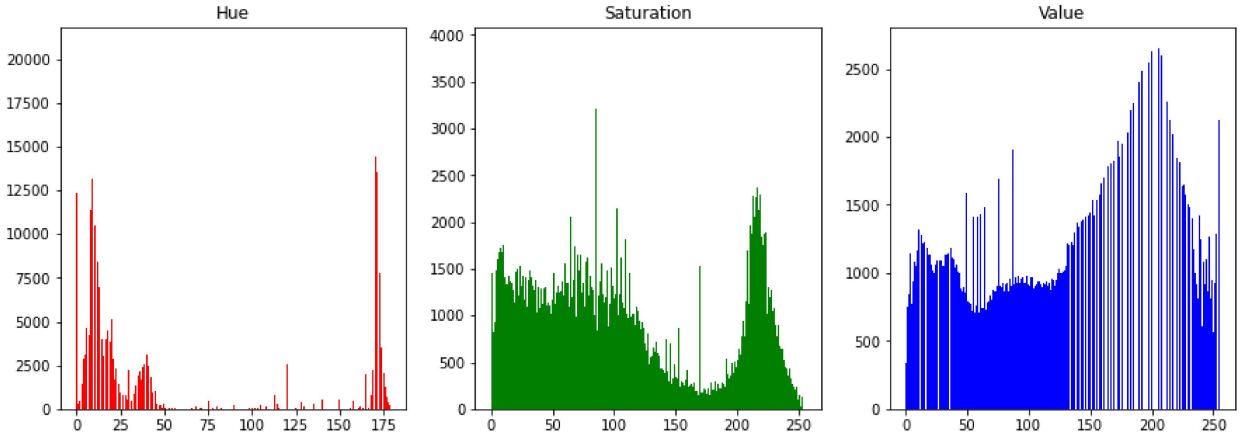
```
axs[0].imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
axs[0].set_title('Original Image')
axs[1].imshow(cv2.cvtColor(img1_equalized, cv2.COLOR_BGR2RGB))
axs[1].set_title('Equalized Image')
plt.show()
```




```
hsv_img1e = cv2.cvtColor(img1_equalized, cv2.COLOR_BGR2HSV)
h, s, v = cv2.split(hsv_img1e)
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].hist(h.flatten(), bins=256, color='r')
axs[1].hist(s.flatten(), bins=256, color='g')
axs[2].hist(v.flatten(), bins=256, color='b')
axs[0].set_title('Hue')
axs[1].set_title('Saturation')
axs[2].set_title('Value')
plt.show()
```

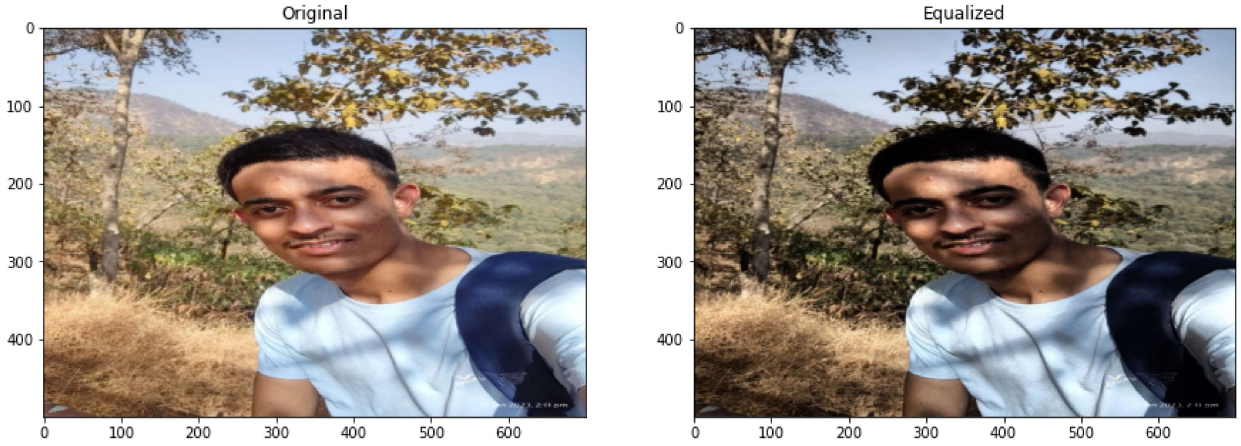
Manthan Dhole **Digital Image Processing(DIP)Lab**

201080020 **Experiment 6**



2 = equalizeHist hsv_img2 ... 2

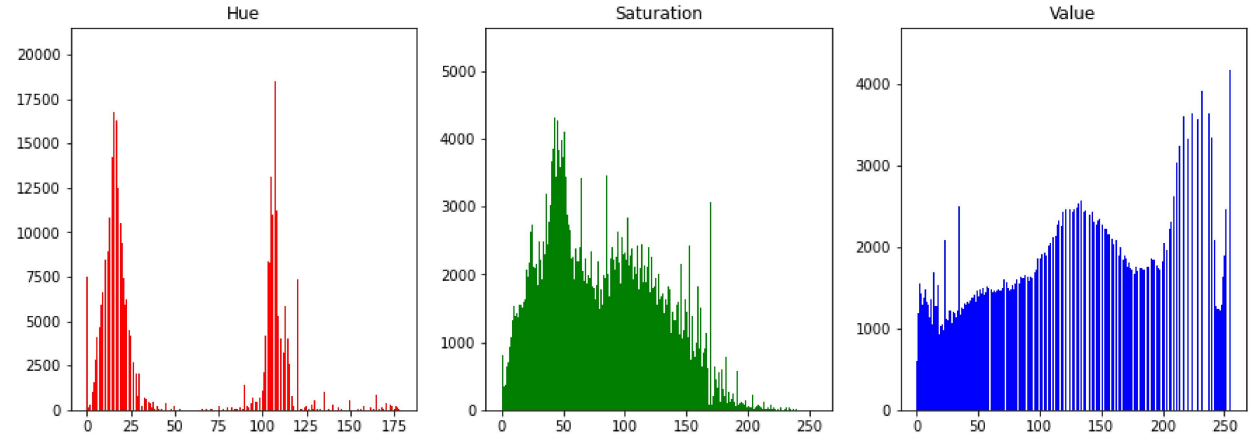
```
img2_equalized = cv2.cvtColor(hsv_img2, cv2.COLOR_HSV2BGR)
fig, axs = plt.subplots(1, 2, figsize=(15, 5))
axs[0].imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
axs[0].set_title('Original')
axs[1].imshow(cv2.cvtColor(img2_equalized, cv2.COLOR_BGR2RGB))
axs[1].set_title('Equalized')
plt.show()
```



= cvtColor img2_equalized cv2 COLOR_BGR2HSV

```
h, s, v = cv2.split(hsv_img2e)
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].hist(h.flatten(), bins=256, color='r')
axs[1].hist(s.flatten(), bins=256, color='g')
```

```
axs[2].hist(v.flatten(), bins=256, color='b')
axs[0].set_title('Hue')
axs[1].set_title('Saturation')
axs[2].set_title('Value')
plt.show()
```



5.

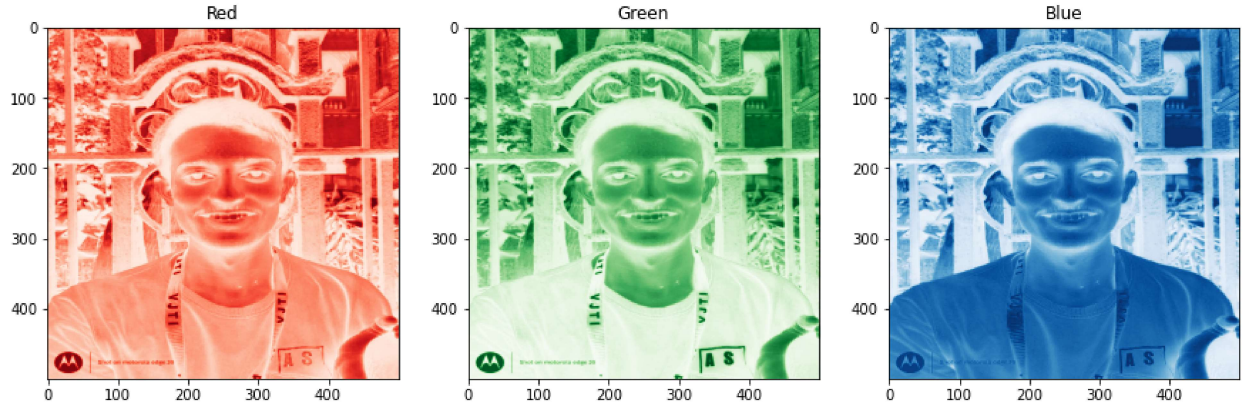
Code:

```
r, g, b = cv2.split(img1)
fig, axs = plt.subplots(1, 3,
figsize=(15, 5))
axs[0].imshow(r, cmap='Reds')
axs[1].imshow(g, cmap='Greens')
axs[2].imshow(b, cmap='Blues')
axs[0].set_title('Red')
axs[1].set_title('Green')
axs[2].set_title('Blue')
plt.show()
```

Output:

Manthan Dhole **Digital Image Processing(DIP)Lab**

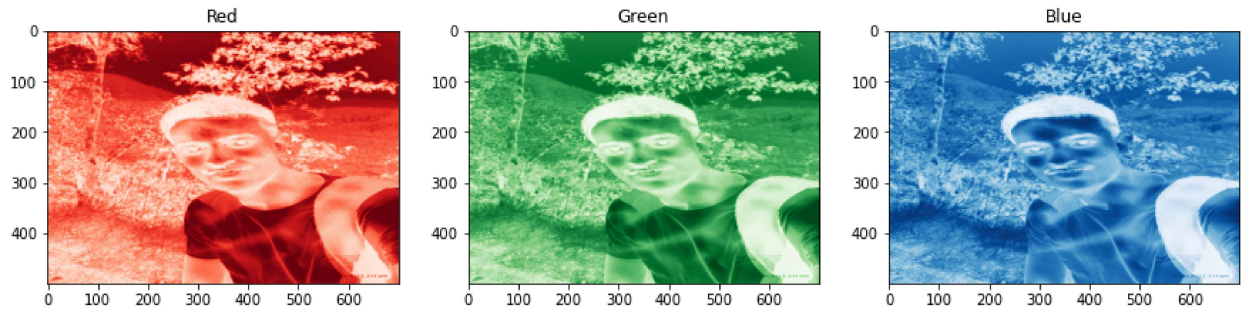
201080020 **Experiment 6**



Code:

```
r, g, b = cv2.split(img2)
fig, axs = plt.subplots(1, 3,
figsize=(15, 5))
axs[0].imshow(r, cmap='Reds')
axs[1].imshow(g, cmap='Greens')
axs[2].imshow(b, cmap='Blues')
axs[0].set_title('Red')
axs[1].set_title('Green')
axs[2].set_title('Blue')
plt.show()
```

Output:



Conclusion:

This experiment involved analyzing and comparing the color and grayscale

histograms of two RGB images, applying histogram equalization to generate new images with adjusted histograms, and separating the RGB channels of the input images. The analysis of the histograms revealed information about the distribution of colors and intensity values in the images. Histogram equalization was used to adjust the contrast of the images, resulting in a new image with a more balanced distribution of pixel values. The separation of RGB channels allowed for the isolation and manipulation of individual color components, which can be useful for various image processing tasks.