4TN4 Assignment 2

Adam Bujak (400113347)

January 31, 2022

1 Theory

1.1 Histogram Equalization

Consider the histogram (2, 2, 4, 8, 16, 32, 64, 128) where the number of gray levels is 8 (0-255 intensity levels). What is the output histogram of histogram equalization?

To simplify this problem, I will ignore the fact that there are 0-255 intensity levels, and simply take each bin of the histogram as a gray level. The given histogram is shown in Figure 1.

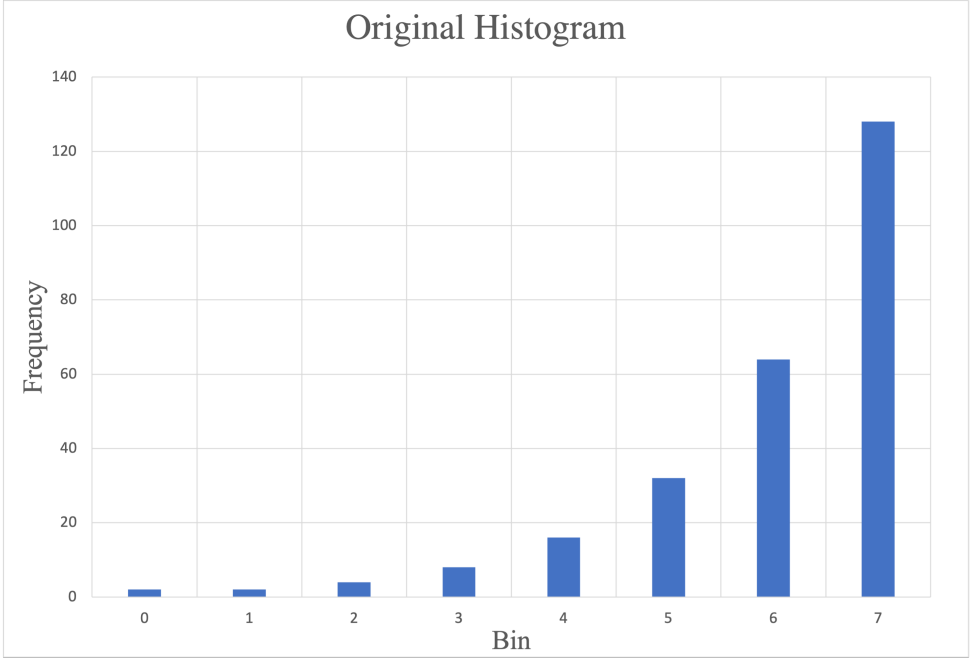


Figure 1: Given Histogram

In Table 1 I show the calculated values for the CDF, normalized CDF, and the equalization values.

The equalization values are calculated given the formula below, where y is the normalized CDF value for the gray level, y' is the equalization value, and $L = 8$ (the number of gray levels).

$$y' = y \cdot (L - 1)$$

Gray Level	Frequency	CDF	CDF - Normalized	Equalization
0	2	2	0.0078125	0
1	2	4	0.015625	0
2	4	8	0.03125	0
3	8	16	0.0625	0
4	16	32	0.125	1
5	32	64	0.25	2
6	64	128	0.5	4
7	128	256	1	7

Table 1: Calculations

The calculated equalization values now each represent a new gray level or bin in the new histogram.

By summing the values in the frequency column for each new gray level we can determine the equalized histogram values. For example, for the new gray level of 0, we see that $2 + 2 + 4 + 8 = 16$, and similarly, for the new gray level of 1, there is only one frequency value which is 16.

Doing this for each new gray level we get the data in Table 2:

Gray Level	Frequency
0	16
1	16
2	32
3	0
4	64
5	0
6	0
7	128

Table 2: New Histogram Values

In Table 1, we see there are no equalization values with values 3, 5, or 6, therefore, the frequencies in those bins of the new histogram are inherently 0.

We can see in Figure 2 that there is a more equalized distribution of gray levels in the new histogram.

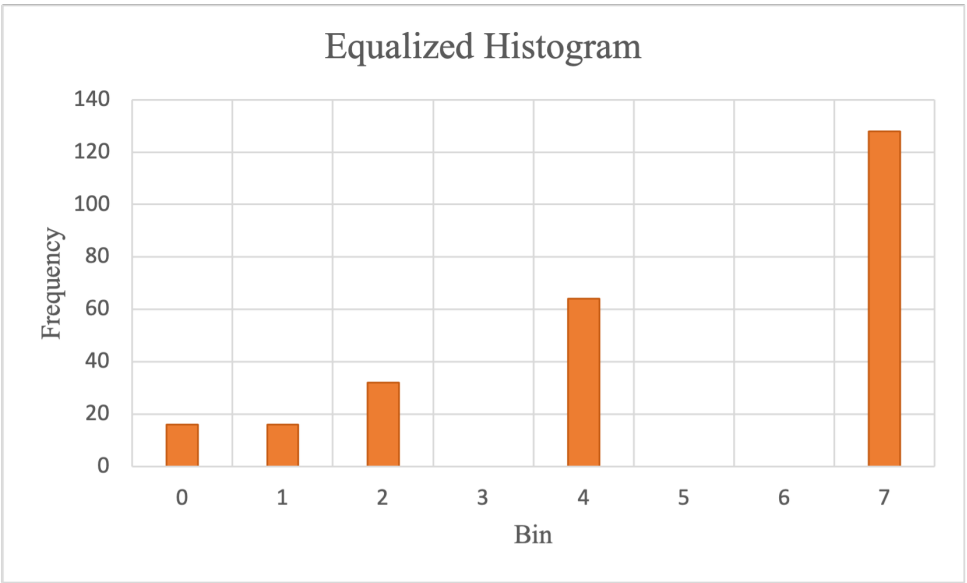
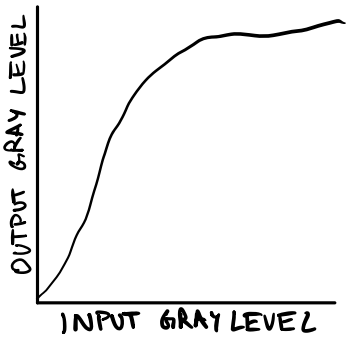


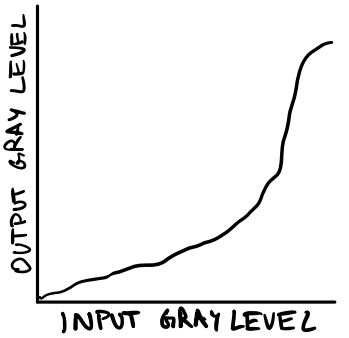
Figure 2: Equalized Histogram

1.2 Transfer Function

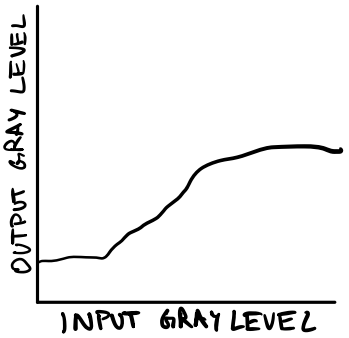
a)


OUTPUT GRAY LEVEL
INPUT GRAY LEVEL

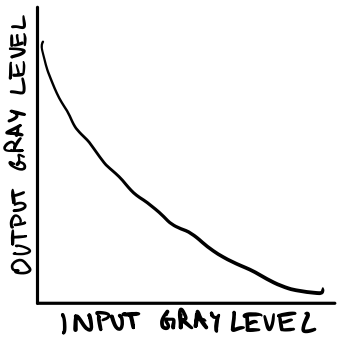
b)


OUTPUT GRAY LEVEL
INPUT GRAY LEVEL

c)


OUTPUT GRAY LEVEL
INPUT GRAY LEVEL

d)


OUTPUT GRAY LEVEL
INPUT GRAY LEVEL

1.3 Filtering in Spatial Domain

Given the following image, we can execute median filtering using the following kernels:

$$\begin{bmatrix} 3 & 5 & 8 & 4 \\ 9 & 1 & 2 & 9 \\ 4 & 6 & 7 & 3 \\ 3 & 8 & 5 & 4 \end{bmatrix}$$

1.3.1 a) Kernel 1

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Given the above kernel and image I will show an example calculation for pixel [1,1] (with value 1).

To find the pixel value we find the median of the set $3, 5, 8, 9, 1, 2, 4, 6, 7$. When sorted, the set becomes: $1, 2, 3, 4, 5, 6, 7, 8, 9$. And we can easily see the median value is 5.

Doing this for each pixel in the image with 0 padding, we get the following image:

$$\begin{bmatrix} 0 & 2 & 2 & 0 \\ 3 & 5 & 5 & 3 \\ 3 & 5 & 5 & 3 \\ 0 & 4 & 4 & 0 \end{bmatrix}$$

1.3.2 b) Kernel 2

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Given the above kernel and image I will show an example calculation for pixel [1,1] (with value 1).

To find the pixel value we find the median of the set:

$$[3 \cdot 0], [5 \cdot 1], [8 \cdot 0], [9 \cdot 1], [1 \cdot 1], [2 \cdot 1], [4 \cdot 0], [6 \cdot 1], [7 \cdot 0]$$

which becomes the set: $5, 9, 1, 2, 6$.

When sorted, the set becomes: $1, 2, 5, 6, 9$. And we can easily see the median value is 5.

Doing this for each pixel in the image with 0 padding, we get the following image:

$$\begin{bmatrix} 3 & 3 & 4 & 4 \\ 3 & 5 & 7 & 3 \\ 4 & 6 & 5 & 4 \\ 3 & 5 & 5 & 3 \end{bmatrix}$$

1.4 Fourier Domain

$$a) \rightarrow H.$$

The randomness and noisiness of the bees in the temporal domain are seen by the noisiness and randomness of the frequencies in the Fourier domain.

$$b) \rightarrow A.$$

A single pair of dots on the y-axis in the Fourier domain is equivalent to horizontal stripes in the temporal domain, and the dot at the origin corresponds to a DC offset.

$$c) \rightarrow G.$$

As we know, a single pair of diagonal dots centered on the axis of the frequency domain corresponds to a diagonal striped pattern (we know this because it's a rotation of the horizontal stripes). The additional information of the bees and the hexagons of the beehive create the noise we see in the frequency domain and the repeating dots in the higher frequency areas (likely due to the edges of the honeycomb changing quickly). The diagonal grid of dots in the frequency domain corresponds to the hexagonal grid of the beehive.

$$d) \rightarrow F.$$

There is clearly a low frequency darker area on the fingerprint along approximately 30° line which is evident by looking at the bright spot in the middle of the plot in the frequency domain. Then the random high frequency edges that make up the ridges of the fingerprint by the outer hazy area in the high frequency region of the frequency plot.

$$e) \rightarrow B.$$

As we know, a single pair of diagonal dots centered on the axis of the frequency domain corresponds to a diagonal striped pattern (we know this because it's a rotation of the horizontal stripes). So we can clearly tell that these dots correspond to the diagonal stripes of B.

$$f) \rightarrow E.$$

As we know from the analysis of the checkered pattern of C, a checkered pattern is represented by a pattern like we see in (g). The pattern seen in (f) is similar to that of (g) except with some noise. This indicates that the temporal representation of (f) will be similar to a checkered pattern. The wall is very similar to a checkered pattern, and the noise is just due to the other changes in the picture since it isn't a perfect checkered pattern. We also see a strong vertical line in the frequency domain which is due to the horizontal lines present in the temporal domain at various frequencies.

$$g) \rightarrow D.$$

We can picture a checkered pattern simply as a combination of a few horizontal and vertical lines at various different frequencies. This is captured perfectly by (g).

$$h) \rightarrow D.$$

We can imagine making a circle in the temporal domain by rotating a line 360°. This would cause a circle to be made in the frequency domain as well from what we know about vertical and horizontal lines. Therefore, the frequency domain plot of D. must have a circle in it and (h) is the only one that does.

2 Implementation

2.1 Background Removal

By getting the median of a every frame of a video with a static background, the moving objects will be removed. If we subtract the remaining image from each frame of the video, only the moving objects will remain.

Figure 3 shows the background of the video extracted using the median technique described above.

Figure 3: Static Background of Video

```python
import numpy as np
import cv2
import time
import statistics

def open_video():
    # import video
    vid = cv2.VideoCapture('video.mp4')
    if (vid.isOpened() == False):
        print("error")
        exit(0)
    return vid

def get_background_img(vid):

    width  = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
    frames = int(vid.get(cv2.CAP_PROP_FRAME_COUNT))
    fps = vid.get(cv2.CAP_PROP_FPS)  # float `height`

    video = np.zeros((frames,height,width,3)) # Array filled with zeros
    background = np.zeros((height,width,3)) # Array filled with zeros

    cnt = 0
    while(vid.isOpened()):
        ret, frame = vid.read()
        if ret == False:
            break
        video[cnt] = frame
        cnt+=1

    # for each channel of each pixel, calculate the median across all frames
    for i in range(height):
        for j in range(width):
            for channel in range(3):
                values = []
                for frame in range(frames):
                    values += [video[frame][i][j][channel]]
                median = statistics.median(values)
                background[i][j][channel] = median

    # save image
    cv2.imwrite('back.png', background)
    vid.release()

def subtract_background():
    vid = open_video()
    width  = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))

    # import background image
    background = cv2.imread('back.png')
    out = cv2.VideoWriter('outpy.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 24, (width,height))

    while(vid.isOpened()):
        ret, frame = vid.read()
        if ret == False:
            break
        subtracted_image = cv2.subtract(frame, background)
        out.write(subtracted_image)

    vid.release()
    out.release()

vid = open_video()
get_background_img(vid)
subtract_background()
```


Figure 4: Code for question 1

2.2 Image Manipulation in FFT Domain

From the figures of the soccer players below, we see that the content of the image is mainly contained in the phase of the image. Since we can still clearly see Ronaldo and the other players from that image, and none of Messi in Figure 5, we know that the phase information contains the meaning of the image. The magnitude in the frequency domain contains information about how bright the image should be at every point in the image.

2.2.1 Phase and Magnitude information

Figure 5: Fourier magnitude of Messi combined with phase of Ronaldo

Figure 6: Fourier magnitude of Ronaldo combined with phase of Messi

```
import numpy as np
import cv2
import time

# import messi image
messi = cv2.imread('messi.jpg', 0)
# import ronaldo image
ronaldo = cv2.imread('ronaldo.jpg', 0)

messi_fft = np.fft.fft2(messi)
ronaldo_fft = np.fft.fft2(ronaldo)

magnitude_messi = np.abs(messi_fft)
phase_messi = np.angle(messi_fft)

magnitude_ronaldo = np.abs(ronaldo_fft)
phase_ronaldo = np.angle(ronaldo_fft)

messi_ronaldo = np.multiply(magnitude_messi, np.exp(1j*phase_ronaldo))
ronaldo_messi = np.multiply(magnitude_ronaldo, np.exp(1j*phase_messi))

messi_mag = np.fft.ifft2(messi_ronaldo)
ronaldo_mag = np.fft.ifft2(ronaldo_messi)

messi_mag = np.abs(messi_mag)
ronaldo_mag = np.abs(ronaldo_mag)
#save img
cv2.imwrite('messi_mag_ronaldo_phase.png', messi_mag)
cv2.imwrite('ronaldo_mag_messi_phase.png', ronaldo_mag)
```

Figure 7: Phase and Magnitude Separation Code

Humans are better at processing low frequency visual information from farther away and high frequency visual information from closer range. When we combine the low-pass filtered image of Messi, with the high-pass filtered image of Ronaldo, in Figure 8 we see Ronaldo more easily, and in Figure 9 we see Messi more easily.

2.2.2 Fourier Domain Filtering

Figure 8: Hybrid Image Close Up

Figure 9: Hybrid Image Zoomed Out

```python
import numpy as np
import cv2
import time

def apply_mask_in_frequency_domain(img, mask):
    img_fft = np.fft.fft2(img)
    img_shift = np.fft.fftshift(img_fft)

    masked_fft = img_shift * mask
    new_img = np.fft.ifftshift(masked_fft)

    new_img = np.fft.ifft2(new_img)
    new_img = np.abs(new_img)
    return new_img


# import messi image
messi = cv2.imread('messi.jpg', 0)
# import ronaldo image
ronaldo = cv2.imread('ronaldo.jpg', 0)

image = np.zeros((len(messi), len(messi[0])), np.uint8)
center = (int(len(messi[0])/2), int(len(messi)/2))
circle = cv2.circle(image, center, 30, 1, -1)

messi_new = np.multiply(apply_mask_in_frequency_domain(messi, circle), 0.5)
inv_circle = np.divide(cv2.bitwise_not(circle), 255)
ronaldo_new = np.multiply(apply_mask_in_frequency_domain(ronaldo, inv_circle), 0.5)

#save img
cv2.imwrite('messi_circle.png', cv2.add(messi_new, ronaldo_new))
```

Figure 10: Hybrid Image Code

2.3 Removing Noise

Figure 11: Gaussian Blur

Figure 12: Gaussian Blur

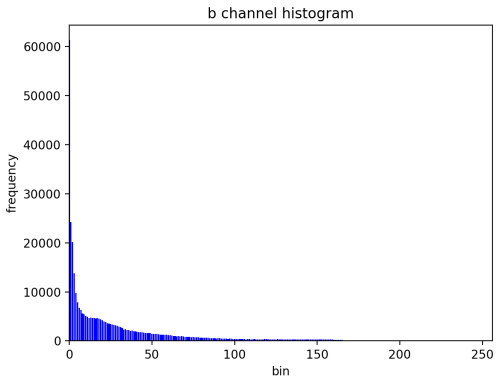
As we can clearly see, the median blur was far more effective at removing the noise from the image than the Gaussian blur. This is because the Gaussian blur acts as a sort of average, so big outliers still affect the final value, whereas with the median filter, a single big outlier will be discarded completely.

```python
import numpy as np
import cv2
import time

# import image
img = cv2.imread('img3.png')

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

#save img
cv2.imwrite('img3_median.png', median)
cv2.imwrite('img3_gauss.png', gauss)
```

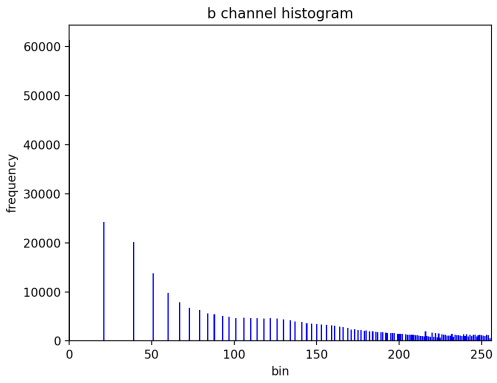
Figure 13: Median and Gaussian Blur Code

2.4 Histogram Equalization of Color Images

2.4.1 RGB Equalization

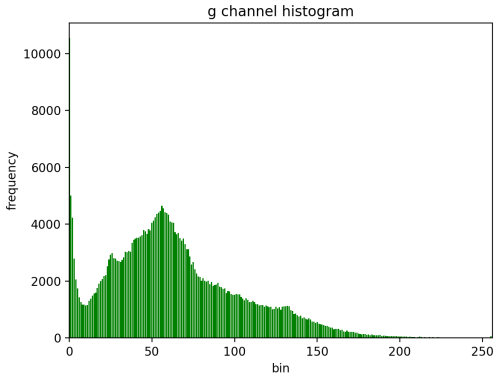


(a) Original Blue Channel Histogram

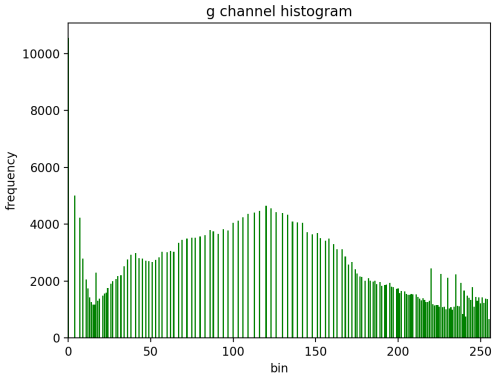


(b) Equalized Blue Channel Histogram

Figure 14: Blue Channel Histogram Before and After Equalization

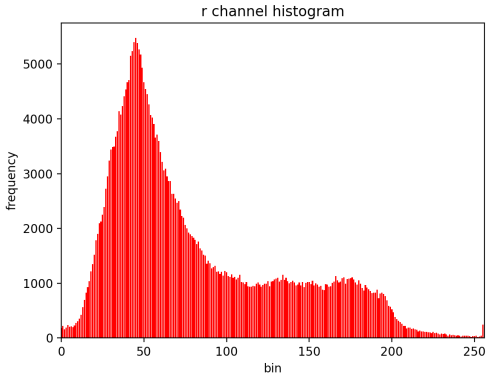


(a) Original Green Channel Histogram

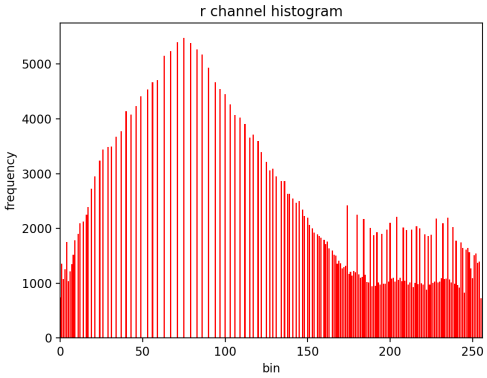


(b) Equalized Green Channel Histogram

Figure 15: Green Channel Histogram Before and After Equalization



(a) Original Red Channel Histogram



(b) Equalized Red Channel Histogram

Figure 16: Red Channel Histogram Before and After Equalization

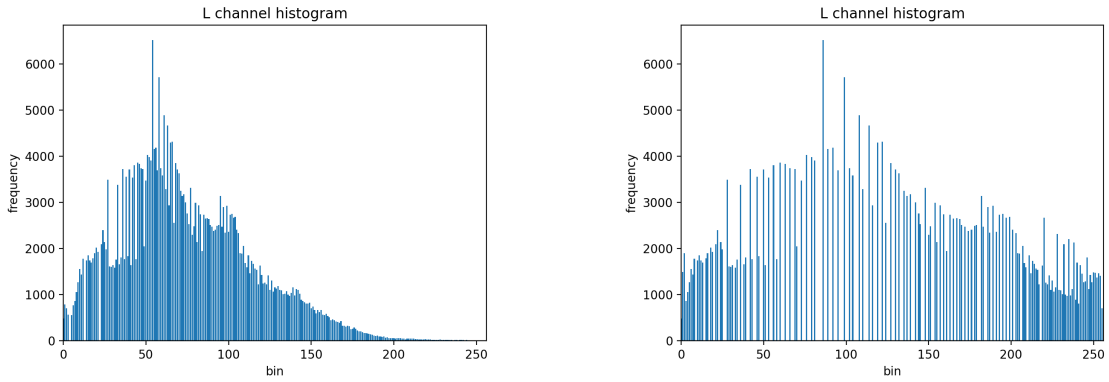
(a) Original Image

(b) Equalized Image

Figure 17: Image Before and After RGB Equalization

2.5 Histogram Equalization of Color Images

2.5.1 LAB Equalization



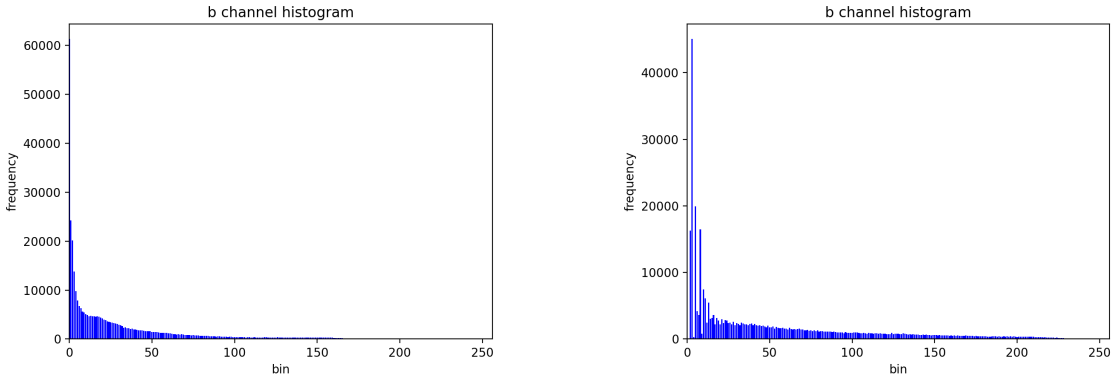
(a) Original L Channel Histogram (b) Equalized L Channel Histogram

Figure 18: L Channel Histogram Before and After Equalization

(a) Original Image (b) Equalized Image

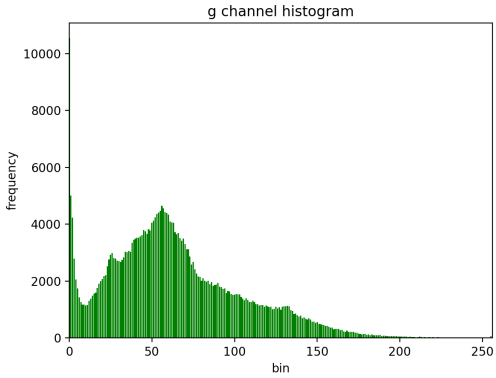
Figure 19: Image Before and After LAB Equalization

2.5.2 RGB Equalization with CLAHE

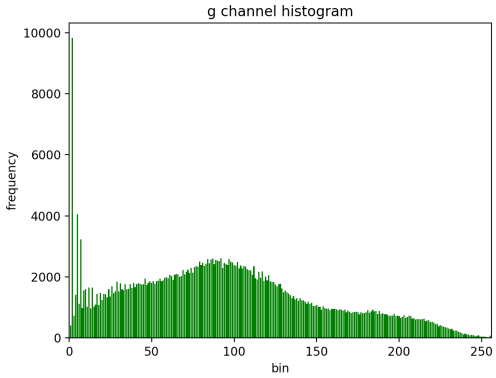


(a) Original Blue Channel Histogram (b) Equalized Blue Channel Histogram

Figure 20: Blue Channel Histogram Before and After CLAHE Equalization

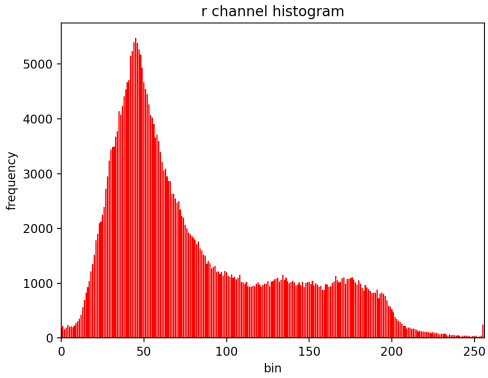


(a) Original Green Channel Histogram

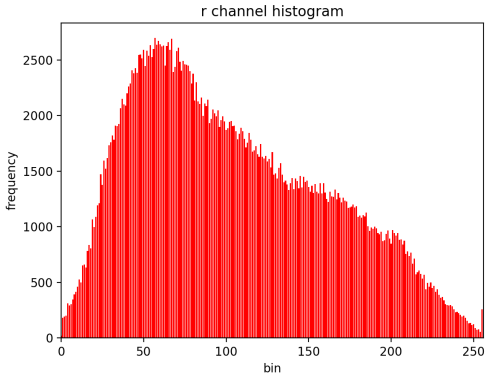


(b) Equalized Green Channel Histogram

Figure 21: Green Channel Histogram Before and After CLAHE Equalization



(a) Original Red Channel Histogram



(b) Equalized Red Channel Histogram

Figure 22: Red Channel Histogram Before and After CLAHE Equalization

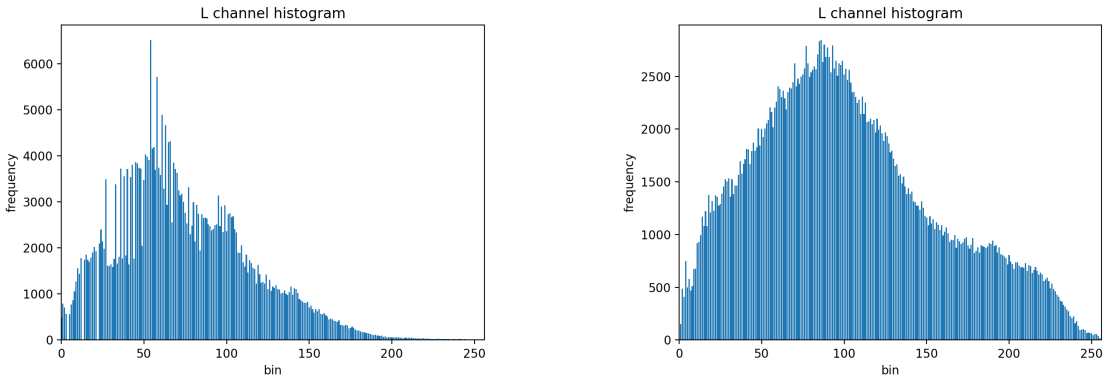
(a) Original Image

(b) Equalized Image

Figure 23: Image Before and After CLAHE RGB Equalization

2.6 Histogram Equalization of Color Images

2.6.1 LAB Equalization with CLAHE



(a) Original L Channel Histogram (b) Equalized L Channel Histogram

Figure 24: L Channel Histogram Before and After CLAHE Equalization

(a) Original Image (b) Equalized Image

Figure 25: Image Before and After CLAHE LAB Equalization

2.6.2 Analysis

We can rank the effectiveness of the equalization from best to worst in the following order:

1. LAB with CLAHE
2. LAB (global)
3. RGB with CLAHE
4. RGB (global)

Both techniques that involve equalizing the RGB channels independently, produce bad results because we are simply equalizing each channel with no regard for the other channels, resulting in the colors changing instead of only the contrast. This is evident in Figures 17 and 23, where we see the saturation of the colors greatly increase.

The LAB color space contains color information in the A and B channels, while the L channel contains only *lightness* information. Therefore, by equalizing the L channel, we can improve the contrast in the image without changing the color information as evident in Figures 19 and 25.

2.6.3 Code

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

def plot_histogram(bins, values, channel_name, color):
    fig = plt.figure()
    plt.ylabel('frequency')
    plt.xlabel('bin')
    plt.xlim([0, 256])
    plt.title("{} channel histogram".format(channel_name))
    plt.bar(bins, values, color=color)
    plt.show()

def get_rgb_histograms(img):
    chans = cv2.split(img)
    colors = ("b", "g", "r")
    for (chan, color) in zip(chans, colors):
        # create a histogram for the current channel and plot it
        hist = cv2.calcHist([chan], [0], None, [256], [0, 256])

        bins = np.arange(0,256,1)
        histvals = []
        for histval in hist:
            histvals += [int(histval[0])]
        plot_histogram(bins, histvals, color, color)

def apply_histogram_equalization_rgb(img, clahe: bool):
    chans = cv2.split(img)
    new_chans = []
    for chan in chans:
        if clahe:
            clahe = cv2.createCLAHE(clipLimit =2.0, tileGridSize=(8,8))
            chan = clahe.apply(chan)
        else:
            chan = cv2.equalizeHist(chan)
        new_chans += [chan]
    img = cv2.merge(new_chans)
    return img

USE_CLAHE = True
# import image
img = cv2.imread('img4.png')

get_rgb_histograms(img)
img = apply_histogram_equalization_rgb(img, USE_CLAHE)
get_rgb_histograms(img)

img_name = "img4_rgb_he"
if USE_CLAHE:
    img_name += "CLAHE"
img_name += ".png"

#save img
cv2.imwrite(img_name, img)
```

Figure 26: RGB Equalization Code

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

def plot_histogram(bins, values, channel_name, color):
    fig = plt.figure()
    plt.ylabel('frequency')
    plt.xlabel('bin')
    plt.xlim([0, 256])
    plt.title("{} channel histogram".format(channel_name))
    plt.bar(bins, values, color=color)
    plt.show()

def get_lab_histogram(img):
    l_channel,a_channel,b_channel = cv2.split(img)
    hist = cv2.calcHist([l_channel], [0], None, [256], [0, 256])

    bins = np.arange(0,256,1)
    histvals = []
    for histval in hist:
        histvals += [int(histval[0])]
    plot_histogram(bins, histvals, 'L', None)

def apply_histogram_equalization_lab(img, clahe: bool):
    L, A, B = cv2.split(img)
    if clahe:
        clahe = cv2.createCLAHE(clipLimit =2.0, tileGridSize=(8,8))
        Le = clahe.apply(L)
    else:
        Le = cv2.equalizeHist(L)
    img = cv2.merge((Le, A, B))
    return img

USE_CLAHE = True
# import image
img = cv2.imread('img4.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)

# show histogram of l channel
get_lab_histogram(img)
# apply histogram equalization on l channel
img = apply_histogram_equalization_lab(img, USE_CLAHE)
# show histogram of l channel
get_lab_histogram(img)
img = cv2.cvtColor(img, cv2.COLOR_LAB2BGR)

img_name = "img4_lab_he"
if USE_CLAHE:
    img_name += "CLAHE"
img_name += ".png"

#save img
cv2.imwrite(img_name, img)
```

Figure 27: LAB Equalization Code