Assignment #2

Johnathan Spinelli (spinellj)

January 30, 2022

1 Theory

1.1 Histogram Equalization

Given the histogram (2,2,4,8,16,32,64,128), we want to equalize it to make an even distribution of pixel values. Using the equations:

$$p_i = \frac{n_i}{n}$$

$$T(k) = floor((7) * \sum_{n=0}^{k} p_n)$$

where T(k) is the upper bound for the equalized bin. The computed p_i values are: ($\frac{1}{128}, \frac{1}{64}, \frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, \frac{1}{1}$).

The p values were then used to compute the T values (the re-mapped bins), which are: [0,0,0,0,0,1,3,7]. This maps out the old bins to new bins, producing the new histogram: [32,32,0,64,0,0,0,128]

1.2 Transfer Function

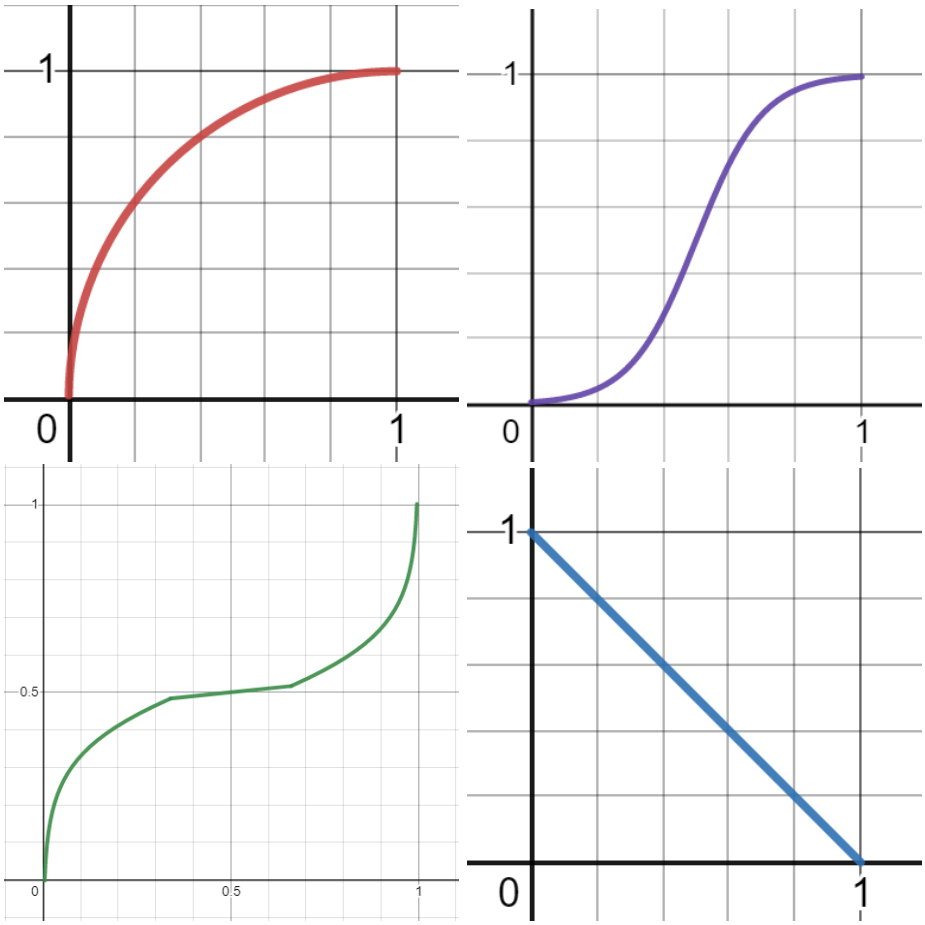

Figure 1: Intensity Transfer Functions of given image. Top row: a, b. Bottom row: c, d.

1.3 Filtering in Spatial Domain

Assuming that the 0's included in zero padding influences the values along the border of the image, the image filtered using the masks would look like this:

a) $\begin{bmatrix} 0 & 2 & 2 & 0 \\ 3 & 5 & 5 & 3 \\ 3 & 5 & 5 & 3 \\ 0 & 4 & 4 & 0 \end{bmatrix}$ b) $\begin{bmatrix} 3 & 3 & 4 & 4 \\ 3 & 5 & 7 & 3 \\ 4 & 6 & 5 & 4 \\ 3 & 5 & 5 & 3 \end{bmatrix}$

1.4 Fourier Domain

a) H. This picture has a lot of noise and small features. Since there are no obvious patterns, we would expect this image to consist of a wide variety of frequencies with little pattern to them.

b) A. The three dots correspond to horizontal rows, as seen in lecture. The further apart the dots, the higher the frequency of rows.

c) G. This photo is generally much brighter than the others, which can account for the intensity of low frequency near the centre of the frequency domain image. The frequency domain image also has rows on a slight tile, which matches with the angle of the honeycomb rows.

d) F. The fingerprint has a circular pattern, but not a perfect circle. This can be noticed in the slanted/oblong shape in the frequency domain. The inconsistency within the temporal domain image can be seen as noise in the frequency domain image as well.

e) B. As seen in class, diagonal dots in the frequency domain correspond to a diagonal pattern in the temporal domain. The angle of the dots matches with the angle of the diagonal lines.

f) E. The frequency domain image contains rows and a column of dots, indicating some sort of horizontal and vertical patterns in the image, with varying frequencies. This pattern can be observed in the brick wall. Additionally, the noise between the dots helps to differentiate the bricks from the checkerboard and other patterned images.

g) C. The obvious and consistent pattern in the frequency domain image immediately corresponds to the checkerboard pattern. A combination of vertical and horizontal rows combining to make a checkerboard image.

h) D. The symmetry of the frequency domain image suggests a perfectly symmetrical image that does not contain rows and columns. This can be seen in the target/concentric circles image.

2 Implementation

2.1 Background Removal

The background of the video can be estimated by examining the pixels throughout the duration of the video. One can compare the values for a single pixel throughout time and select the median pixel value. This median pixel value can be treated as the estimate for the background of the image. We can assume that objects passing in front of the image will be both brighter and darker than the background, or the objects will not be blocking the background for a long period of time.

Figure 2: Background of video with cars removed.

```
import numpy as np
import cv2
import time

# Open Video
cap = cv2.VideoCapture('video.mp4')
# Randomly select 25 frames
frameIds = cap.get(cv2.CAP_PROP_FRAME_COUNT) * np.random.uniform(size=20)

# Store selected frames in an array
frames = []
for fid in frameIds:
    # Go to each selected frame
    cap.set(cv2.CAP_PROP_POS_FRAMES, fid)
    ret, frame = cap.read()
    frames.append(frame)

# Calculate median across the selected frames, convert to gray
medianFrame = np.median(frames, axis=0).astype(dtype=np.uint8)
medianGray = cv2.cvtColor(medianFrame, cv2.COLOR_BGR2GRAY)


# Display median frame
cv2.imshow('frame', medianFrame)
#cv2.imshow('gray', medianGray)
cv2.imwrite('1_background.png',medianFrame)
```

```
cap.release()

# Replay video to extract cars
cap = cv2.VideoCapture('video.mp4')
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

out = cv2.VideoWriter('1_cars.avi',cv2.VideoWriter_fourcc('M','J','P','G'),
                      20, (frame_width,frame_height))

# Remove the background from video
ret = True
while(ret):
    ret, frame = cap.read()

    if ret == True:
        # Grayscale difference to detect car
        frameGray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        diff = cv2.absdiff(frameGray, medianGray)
        th, diffFrame = cv2.threshold(diff, 15, 255, cv2.THRESH_BINARY)
        # Extract the cars from mask
        result = cv2.bitwise_and(frame, frame, mask=diffFrame)
        out.write(result)
    else:
        break

cap.release()
out.release()
```

2.2 Image Manipulation in FFT Domain

Pictured below are two images. The first image uses the Messi's magnitude and Ronaldo's phase. The second image uses Ronaldo's magnitude and Messi's phase. The two images look very different from each other. The more noticeable component from each image comes from the phase - the image with Messi's magnitude bears no resemblance to his photo, however, Ronaldo can clearly be seen. This is because the phase contains information about the locations of features.

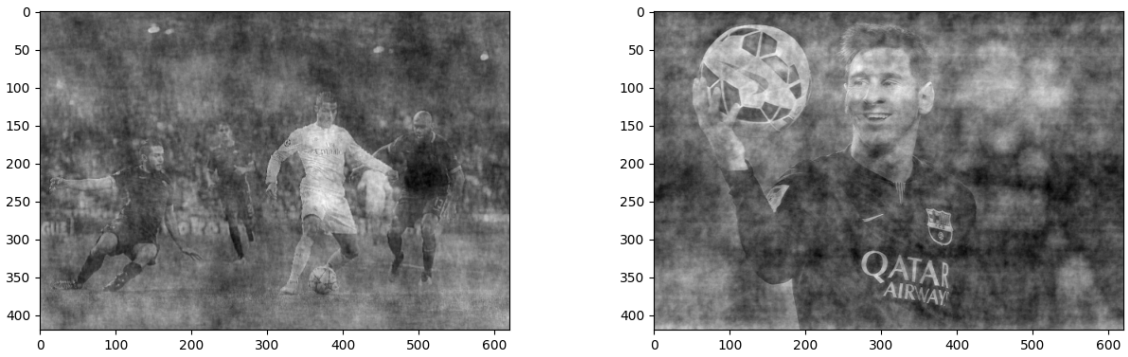

Figure 3: Left: Messi's magnitude with Ronaldo's phase. Right: Ronaldo's magnitude with Messi's phase.

2.2.1 Filters

The images below show the process of filtering the two images. After applying the filters, Messi's image becomes blurred once the high frequency features are filtered out, and Ronaldo's image appears to be mainly countours - the remaining high frequency features. After combining the images, there is a visual illusion similar to the example seen

in class. If you focus on the image from close-up, you can see Ronaldo's contours. From afar or with blurry vision, you can only see Messi. In the smaller version of the image, a lot of detail is lost or can't be properly seen, so it looks almost like the original image of Messi.

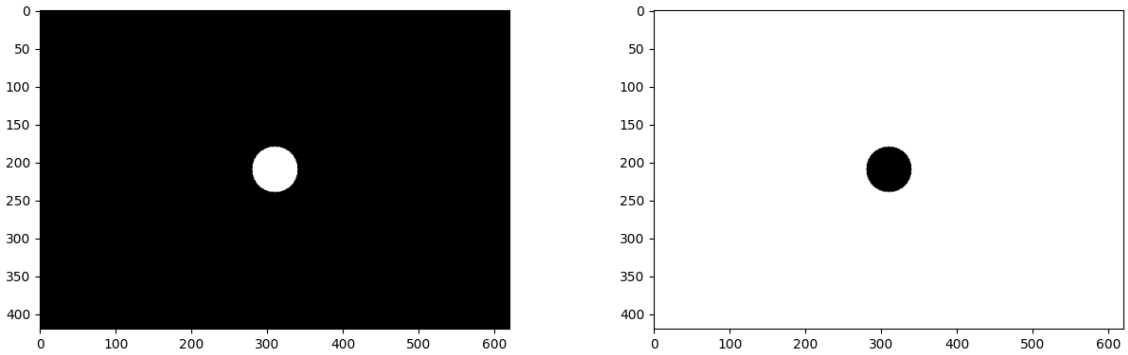

Figure 4: Left: Low pass filter in frequency domain. Right: High pass filter in frequency domain.

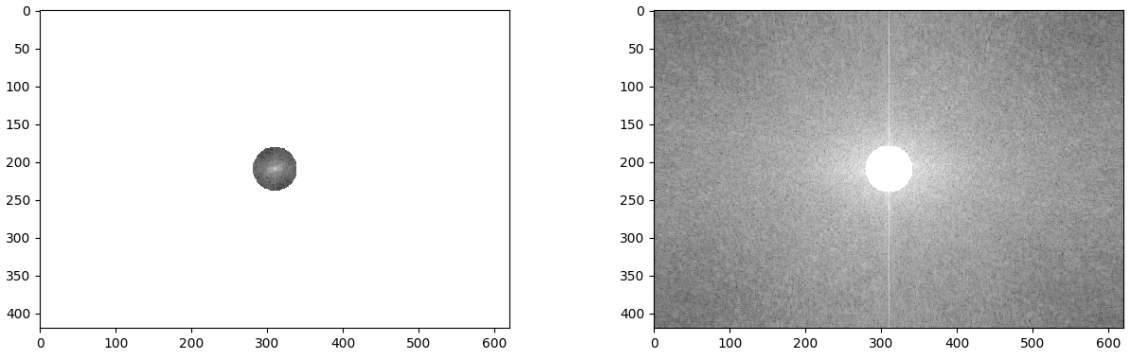

Figure 5: Left: Low pass filter applied to Messi. Right: High pass filter applied to Ronaldo.

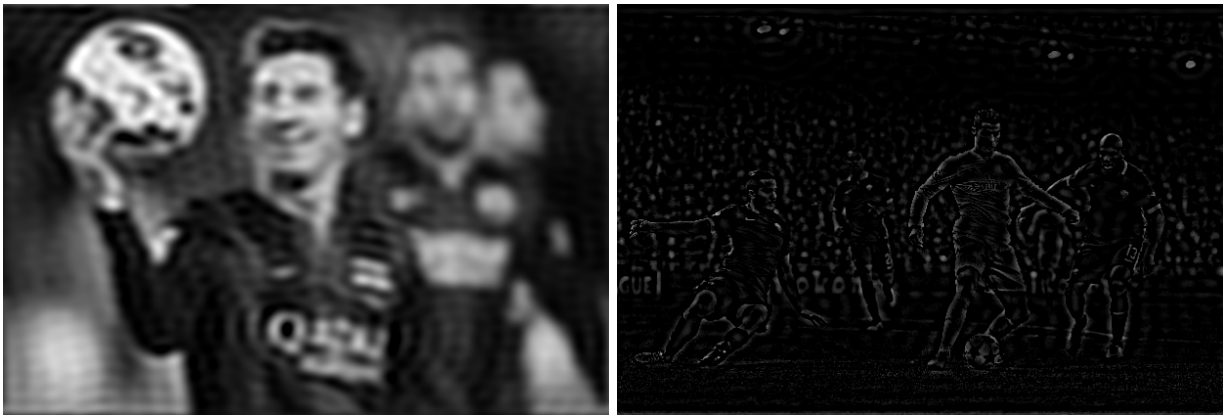

Figure 6: Left: Low pass result. Right: High pass result.

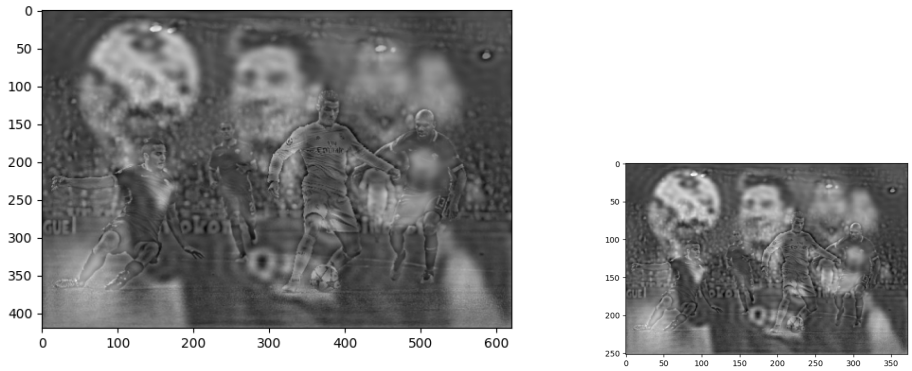

Figure 7: Left: Low pass filter in frequency domain. Right: High pass filter in frequency domain.

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

# Read and display original images in grayscale
img = cv2.imread('messi.jpg')
img2 = cv2.imread('ronaldo.jpg')
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray_img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
cv2.imshow('Messi',gray_img)
cv2.imshow('Ronaldo',gray_img2)

# Fourier transform images
gray_fourier = np.fft.fft2(gray_img)
shifted_gf = np.fft.fftshift(gray_fourier)

gray_fourier2 = np.fft.fft2(gray_img2)
shifted_gf2 = np.fft.fftshift(gray_fourier2)

plt.imshow(np.log(abs(shifted_gf)), cmap='gray')
#plt.show()
plt.imshow(np.angle(shifted_gf), cmap = 'gray')
#plt.show()

# Create Low Pass
rows, cols  = gray_img.shape
crow, ccol = int(rows / 2), int(cols / 2)

LP = np.zeros((rows,cols), np.uint8)
r = 30
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r*r
LP[mask_area] = 1

# Create High Pass
HP = np.ones((rows, cols), np.uint8)
mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r*r
HP[mask_area] = 0

plt.imshow(LP, cmap = 'gray')
```

```
plt.show()
cv2.imwrite
plt.imshow(HP, cmap = 'gray')
plt.show()

# Apply filters to TF images
LP_mes = LP*shifted_gf
HP_ron = HP*shifted_gf2

plt.imshow(np.log(abs(LP_mes)), cmap = 'gray')
plt.show()
plt.imshow(np.log(abs(HP_ron)), cmap = 'gray')
plt.show()

# Switch magnitudes/phases of TF images
mes_ron = abs(shifted_gf) * np.exp(1j*np.angle(shifted_gf2))
ron_mes = abs(shifted_gf2) * np.exp(1j*np.angle(shifted_gf))

# Transform Images back
LP_mes = np.fft.ifftshift(LP_mes)
LP_mes_final = np.fft.ifft2(LP_mes).real
HP_ron = np.fft.ifftshift(HP_ron)
HP_ron_final = np.fft.ifft2(HP_ron).real

blended_img = cv2.addWeighted(LP_mes_final,0.5,HP_ron_final,0.5,0)

mes_ron = np.fft.ifftshift(mes_ron)
mes_ron_final = np.fft.ifft2(mes_ron).real
ron_mes = np.fft.ifftshift(ron_mes)
ron_mes_final = np.fft.ifft2(ron_mes).real

# Plot and save Images
plt.imshow(LP_mes_final, cmap = 'gray')
plt.show()
plt.imshow(HP_ron_final, cmap = 'gray')
plt.show()
plt.imshow(blended_img, cmap = 'gray')
plt.show()

# Scale down blended for a smaller version
scale = 60
width = int(blended_img.shape[1] * scale / 100)
height = int(blended_img.shape[0] * scale / 100)
dim = width, height
smaller_blend = cv2.resize(blended_img, dim, interpolation = cv2.INTER_AREA)

# Continue plotting
plt.imshow(smaller_blend, cmap = 'gray')
plt.show()
plt.imshow(mes_ron_final, cmap = 'gray')
plt.show()
plt.imshow(ron_mes_final, cmap = 'gray')
plt.show()
```

2.3 Removing Noise

The median filter does a better job at removing the noise from the image. The low pass filter doesn't completely remove the noise because the mask is weighted, including the noise values in the resulting image. The median filter, however, only selects one value from its neighbours to represent the pixel, which is beneficial for removing small

instances of noise like in the given image.

Figure 8: Left: Median Filter. Right: Low Pass Filter.

```
import numpy as np
import cv2

img = cv2.imread('img3.png')

median = cv2.medianBlur(img,3)
cv2.imshow('median',median)
cv2.imwrite('median_filter.png', median)

low = cv2.GaussianBlur(img,(5,5),0)
cv2.imshow('low pass',low)
cv2.imwrite('lowpass_filter.png', low)
```

2.4 Histogram Equalization of Colour Images

2.4.1 RGB

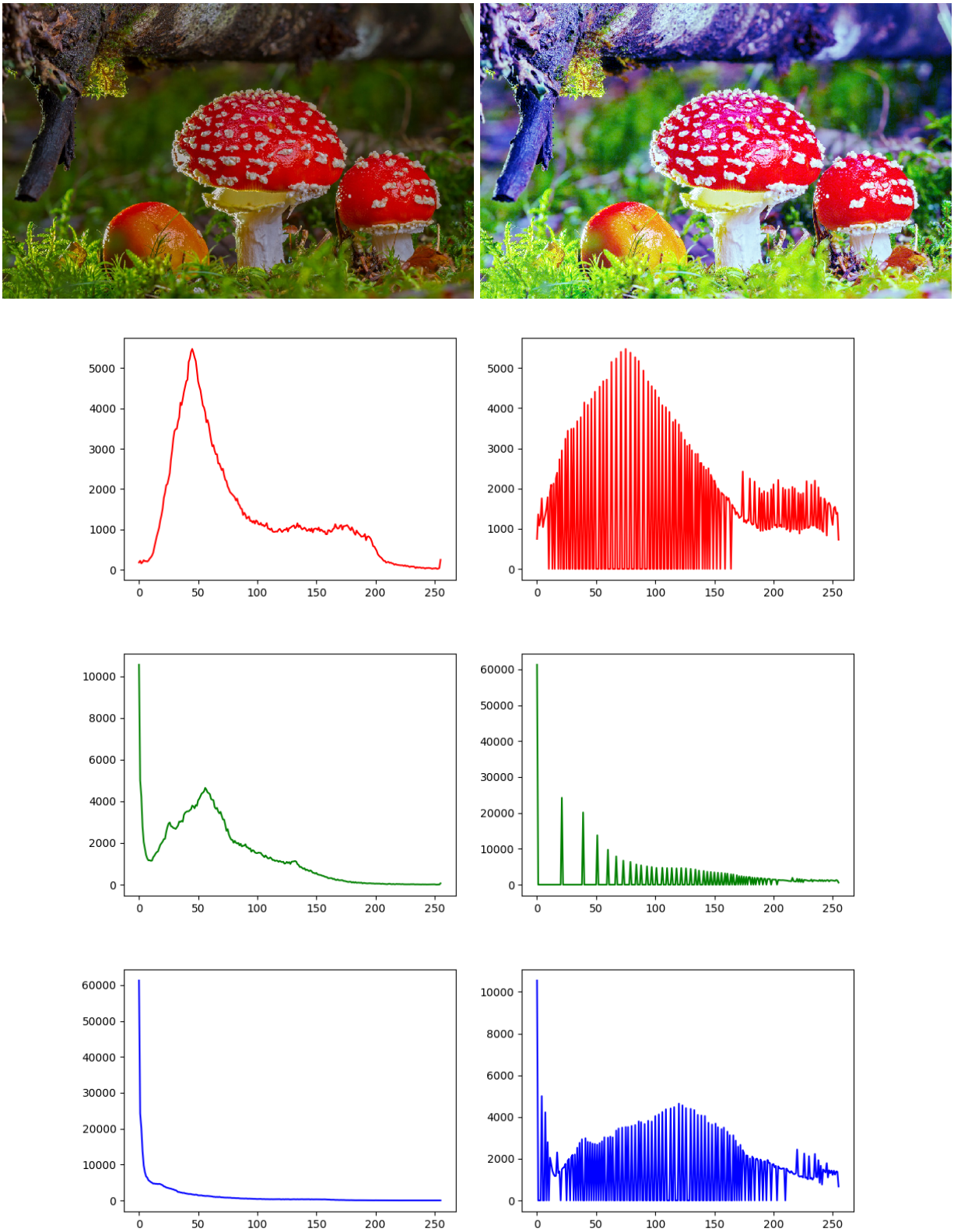

Figure 9: Original image and image after RGB equalization. Histograms before and after equalization.

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

```
# Read Image
img = cv2.imread('img4.png')
cv2.imshow('image',img)

# Original Histograms
b, g, r = img[:,:,0], img[:,:,1], img[:,:,2]
hist_b = cv2.calcHist([b],[0],None,[256],[0,256])
hist_g = cv2.calcHist([g],[0],None,[256],[0,256])
hist_r = cv2.calcHist([r],[0],None,[256],[0,256])

# Equalized Histogramgs
r2 = cv2.equalizeHist(r)
b2 = cv2.equalizeHist(b)
g2 = cv2.equalizeHist(g)
hist_b2 = cv2.calcHist([b2],[0],None,[256],[0,256])
hist_g2 = cv2.calcHist([g2],[0],None,[256],[0,256])
hist_r2 = cv2.calcHist([r2],[0],None,[256],[0,256])

# Display Result
img[:,:,0], img[:,:,1], img[:,:,2] = b2, g2, r2
cv2.imshow('equalized image',img)
cv2.imwrite('Equalized_Mushroom.png',img)

# Plot histograms
f, arr = plt.subplots(1,2)
arr[0].plot(hist_r, color='r', label="r")
arr[1].plot(hist_r2, color='r', label="r2")
f.set_figheight(4)
f.set_figwidth(12)
plt.show()

f, arr = plt.subplots(1,2)
arr[0].plot(hist_g, color='g', label="g")
arr[1].plot(hist_b2, color='g', label="g2")
f.set_figheight(4)
f.set_figwidth(12)
plt.show()

f, arr = plt.subplots(1,2)
arr[0].plot(hist_b, color='b', label="b")
arr[1].plot(hist_g2, color='b', label="b2")
f.set_figheight(4)
f.set_figwidth(12)
plt.show()
```

2.4.2 LAB

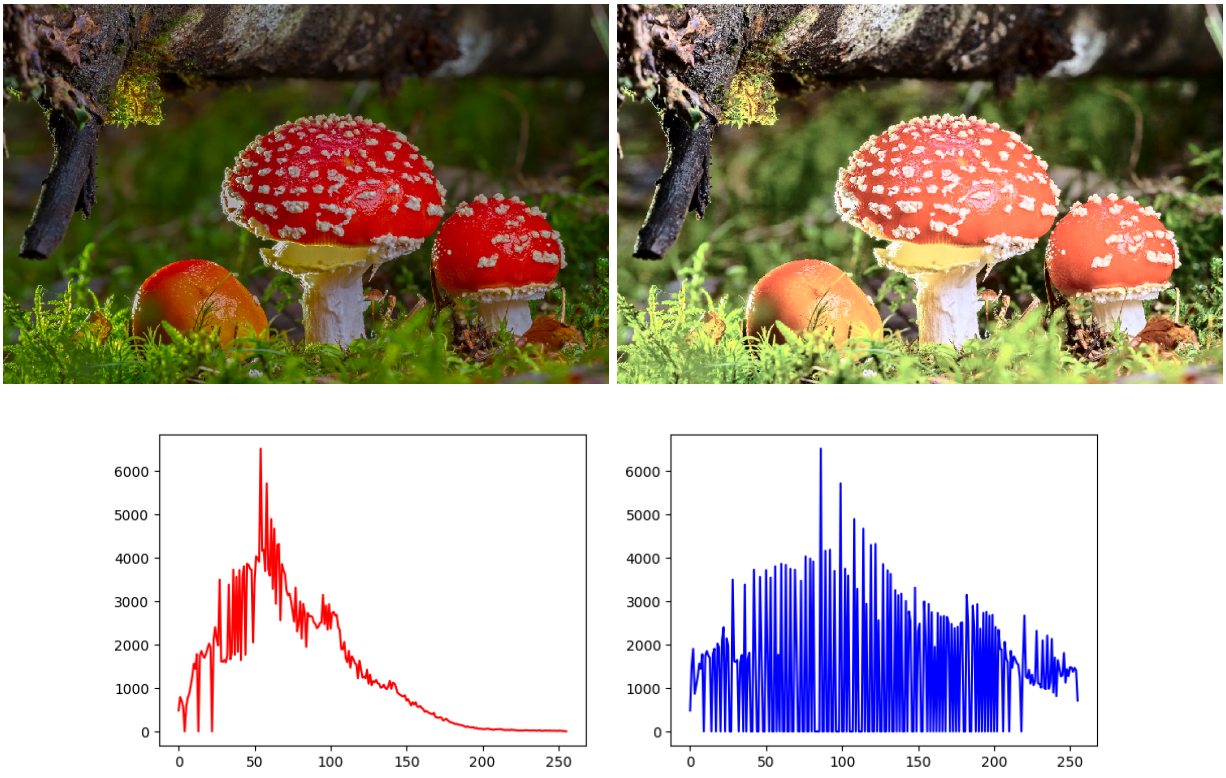

Figure 10: Original image and image after L equalization. Histograms before and after equalization.

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

# Read Image
img = cv2.imread('img4.png')
cv2.imshow('image',img)

img = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
#cv2.imshow('image2',img)

# Original Histograms
L = img[:,:,0]
hist_L = cv2.calcHist([L],[0],None,[256],[0,256])

# Equalized Histogram
L2 = cv2.equalizeHist(L)
norm = np.zeros((800,800))
hist_L2 = cv2.calcHist([L2],[0],None,[256],[0,256])

# Update Image
img[:,:,0] = L2
#cv2.imshow('equalized image',img)

img = cv2.cvtColor(img, cv2.COLOR_LAB2BGR)
cv2.imshow('converted image',img)
cv2.imwrite('Equalized_Mushroom_L.png',img)

# Plot Histograms
f, arr = plt.subplots(1,2)
```

```
arr[0].plot(hist_L, color='r', label="L")
arr[1].plot(hist_L2, color='b', label="L2")
f.set_figheight(4)
f.set_figwidth(12)
plt.show()
```

2.4.3 RGB CLAHE and LAB CLAHE

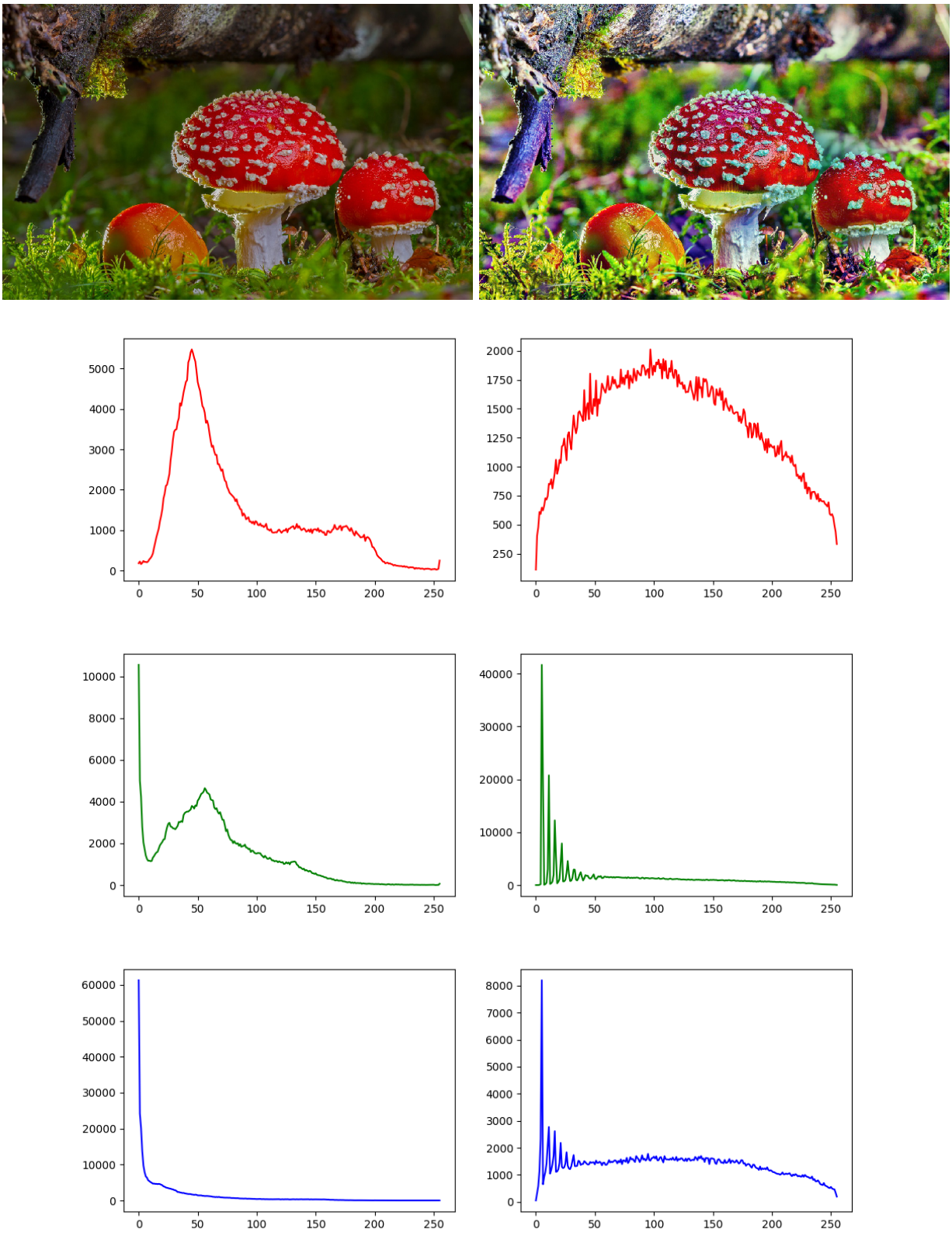

Figure 11: Original image and image after RGB CLAHE function. Histograms before and after CLAHE.

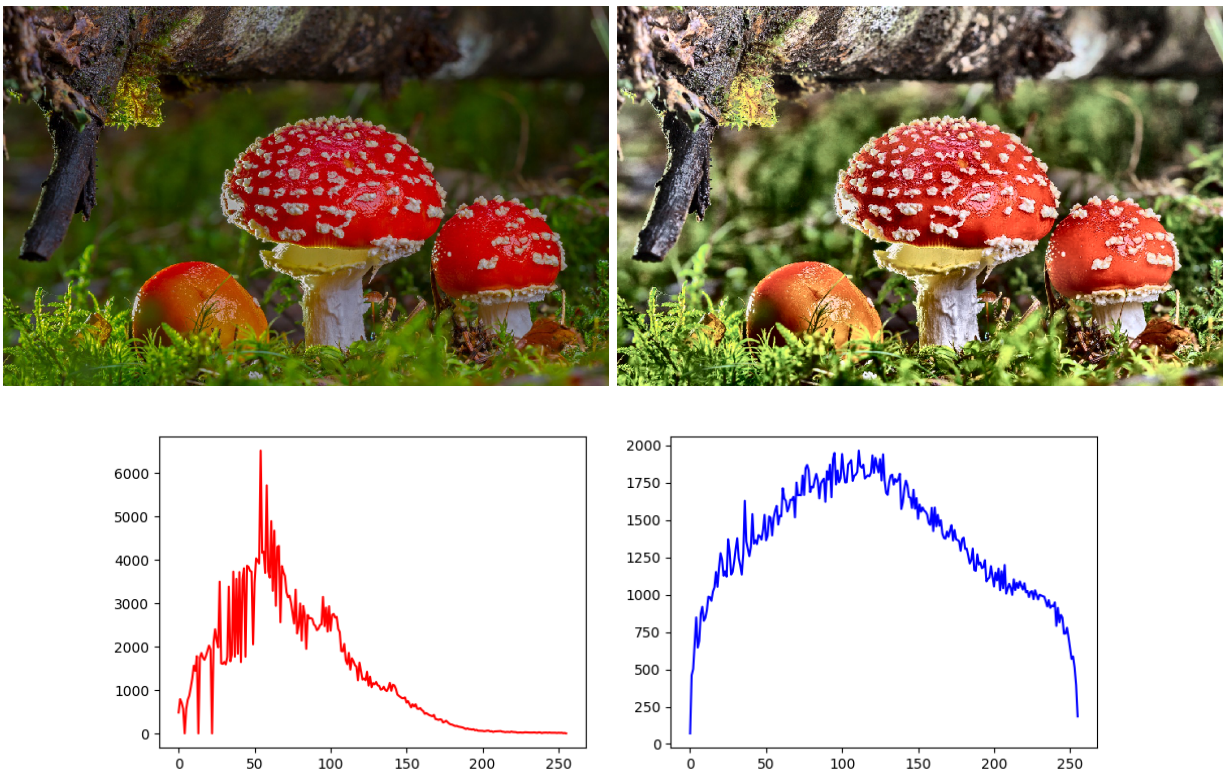

Figure 12: Original image and image after LAB CLAHE function. Histograms before and after CLAHE.

RGB CLAHE (in place of Equalized Histograms):

```
# Use CLAHE
clahe = cv2.createCLAHE(clipLimit = 5)
b2 = clahe.apply(b)
g2 = clahe.apply(g)
r2 = clahe.apply(r)
hist_b2 = cv2.calcHist([b2],[0],None,[256],[0,256])
hist_g2 = cv2.calcHist([g2],[0],None,[256],[0,256])
hist_r2 = cv2.calcHist([r2],[0],None,[256],[0,256])
```

LAB CLAHE (in place of Equalized Histogram):

```
# Use CLAHE
clahe = cv2.createCLAHE(clipLimit = 5)
L2 = clahe.apply(L)

hist_L2 = cv2.calcHist([L2],[0],None,[256],[0,256])
```

2.4.4 Results

The four methods yield very different results from each other. The RGB and the RGB CLAHE methods do not produce great results, as the equalization affects the amount of colour in the image, bringing out colours that otherwise were not visible. As seen in the RGB histogram equalization, the branch in the background turns blue.

The LAB images produce more realistic results, keeping the colours consistent with the original image. This is because only the lightness value is being manipulated to make the image brighter. The CLAHE image is much sharper with more contrast, between features, while the L equalization makes everything appear brighter.

Out of all four options, both of the LAB images look better than the RGB images. The LAB CLAHE image looks very realistic, and the L equalization image looks like there is too much light on the scene. Depending on the goal of the image processing, either LAB image could be a successful result of equalization.