

Problem_2

April 15, 2018

```
In [1]: import cv2 # tested with 3.1.0 version
import numpy as np
import matplotlib.pyplot as plt
import os
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

1 2. Image Enhancement

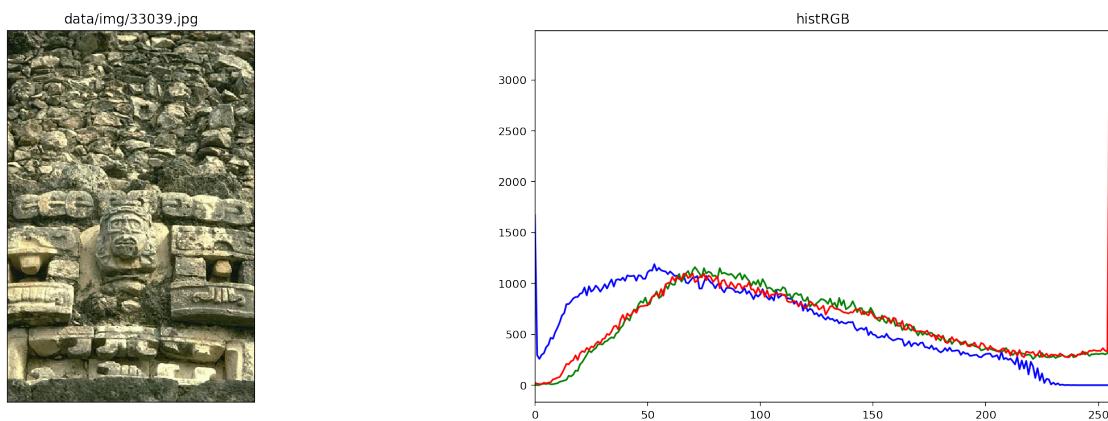
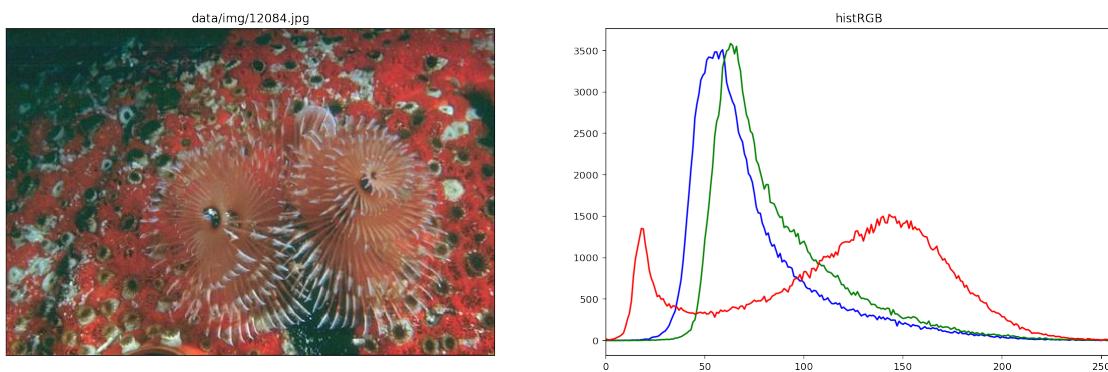
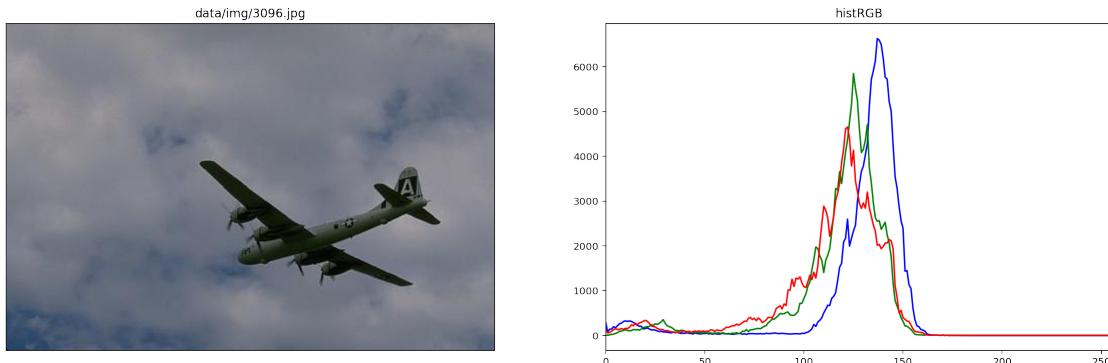
1.1 2.1 Histogram

```
In [2]: IMG_NAMES = ['3096.jpg', '12084.jpg', '33039.jpg'] # Take any 3 images from the database

def show_histRGB(img, imagepath):
    color = ('b', 'g', 'r')
    plt.figure(figsize=(20, 6)) # Figure size can be adjusted.
    plt.subplot(121), plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)) # show the corr
    plt.xticks([])
    plt.yticks([])
    plt.title(imagepath)
    for i, col in enumerate(color):
        # a) Calculate the histograms for this image.
        histr = cv2.calcHist([img],[i],None,[256],[0,256])
        plt.subplot(122), plt.plot(histr, color = col) # plot histogram with the b g r
        plt.xlim([0, 256])
        plt.title("histRGB")
    plt.show()

for imagename in IMG_NAMES:
    imagepath = os.path.join('data/img/', imagename)
    img = cv2.imread(imagepath,1) # read the image from image path using opencv.
    show_histRGB(img, imagepath)
```

```
E:\Softwares\Anaconda3\envs\py2\lib\site-packages\matplotlib\cbook\deprecation.py:107: MatplotlibDeprecationWarning: This feature will be removed in a future version.
warnings.warn(message, mplDeprecation, stacklevel=1)
```



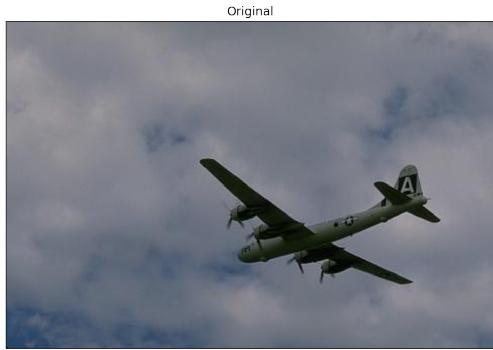
1.1.1 b) What can you say about the spread of the color intensities for these images? (Please briefly answer this question in this block)

The picture which is more complex or more colorful tend to have wide spread of color densities in the histogram.

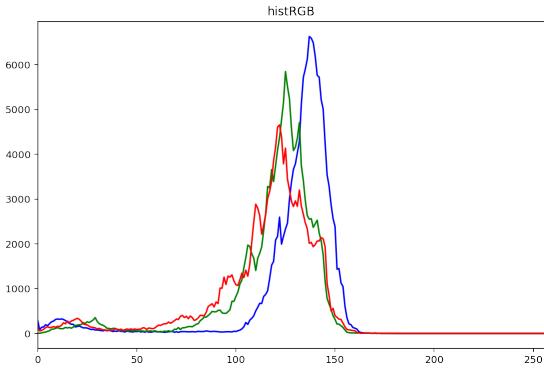
1.2 2.2 Global Histogram Equalization

```
In [3]: def histogram_equalization(imagepath):
    img = cv2.imread(imagepath,1) # Read the image in color
    img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV) # Convert the image to YUV
    img_yuv = np.stack((cv2.equalizeHist(img_yuv[:, :, 0]),img_yuv[:, :, 1],img_yuv[:, :, 2]))
    equ = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR) # Convert back to GBR
    show_histRGB(img, "Original")
    show_histRGB(equ, "Equalized (Global)")

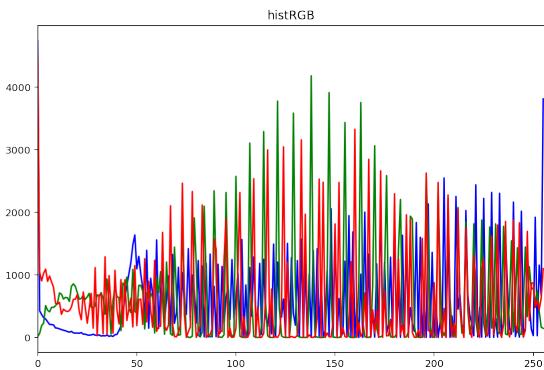
for imagename in IMG_NAMES:
    imagepath = os.path.join('data/img/', imagename)
    histogram_equalization(imagepath)
```

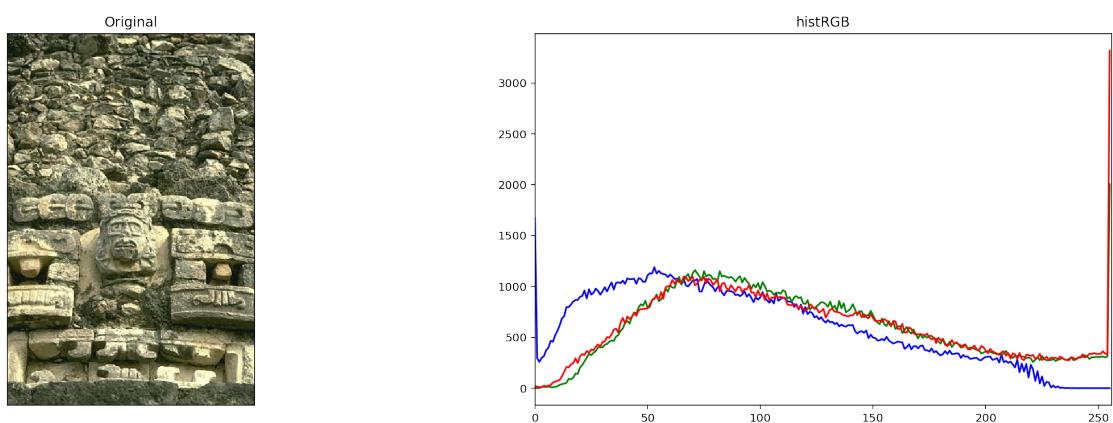
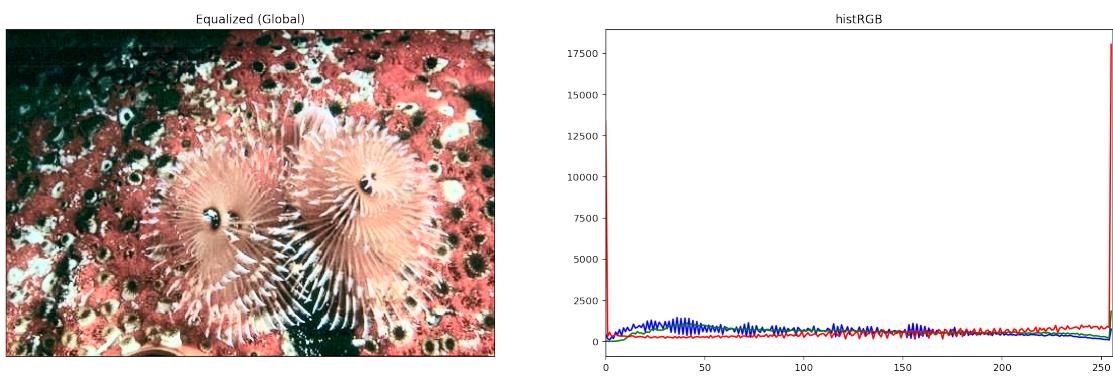
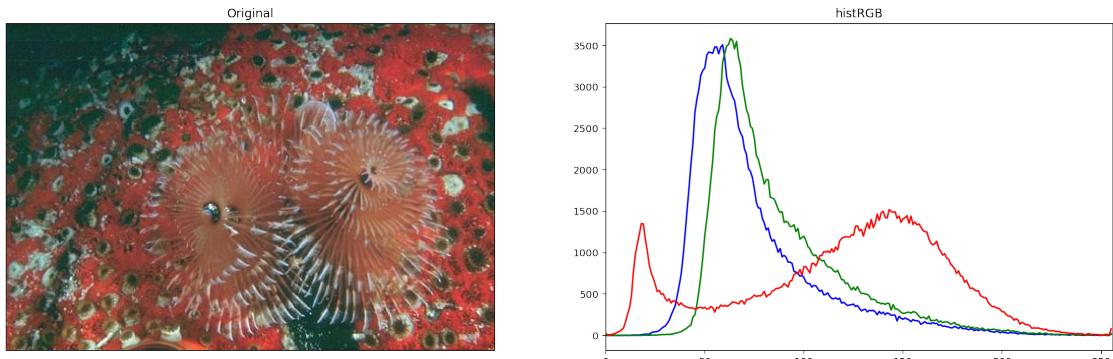


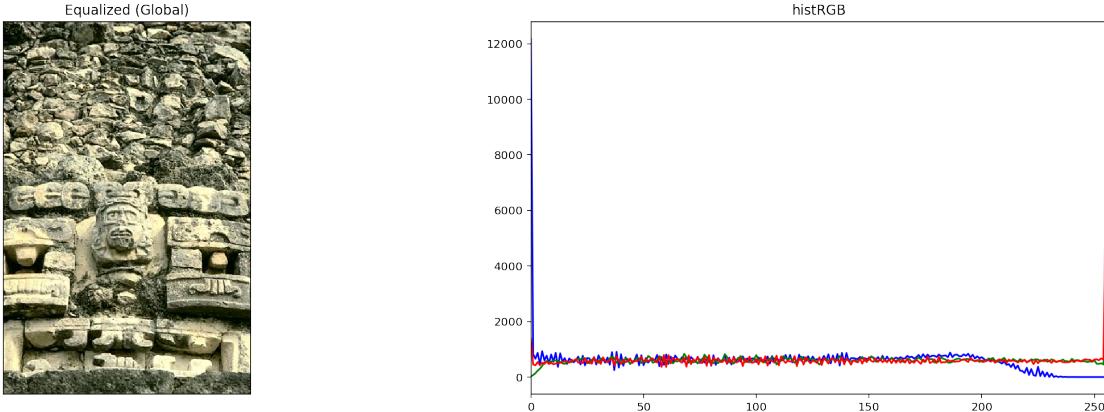
Original



Equalized (Global)





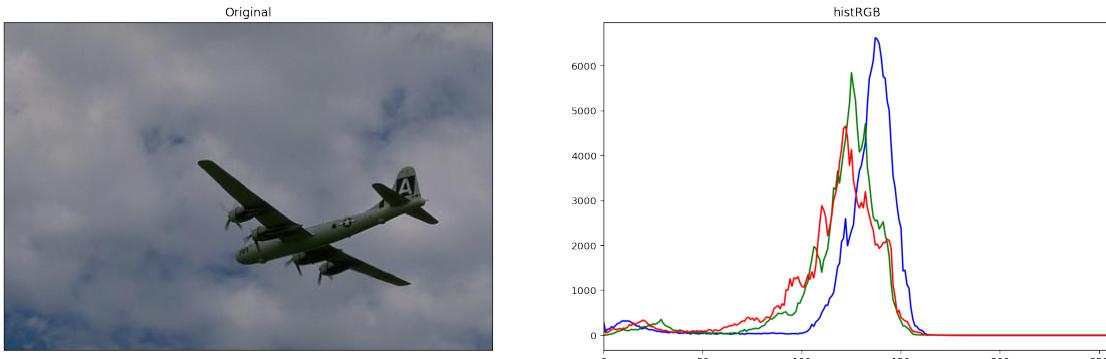


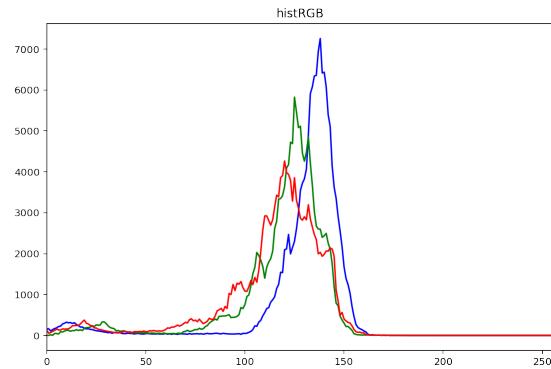
1.3 2.3 Adaptive Histogram Equalization (CLAHE)

```
In [5]: def adaptive_histogram_equalization(imagepath):
    img = cv2.imread(imagepath,1) # Read the image in color
    img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV) # Convert the image to YUV
    clahe = cv2.createCLAHE(clipLimit=5.0, tileGridSize=(8,8)) # a) Create clahe
    print('clipLimit = 5.0    tileGridSize = (8,8)') # b) Print all the hyper parameters
    cl1 = np.stack((clahe.apply(img_yuv[:, :, 0]),img_yuv[:, :, 1],img_yuv[:, :, 2]),axis=2)
    equ = cv2.cvtColor(img_yuv, cv2.COLOR_YUV2BGR) # Convert back to GBR
    show_histRGB(img, "Original")
    show_histRGB(equ, "Equalized (Adaptive)")

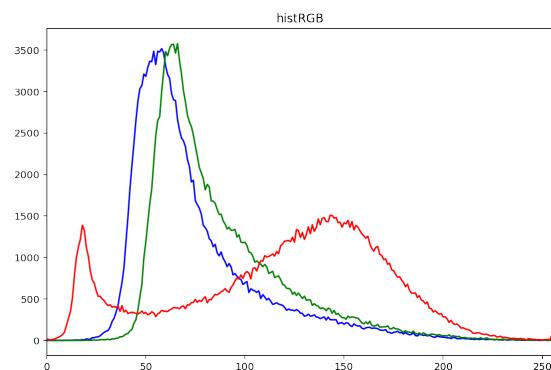
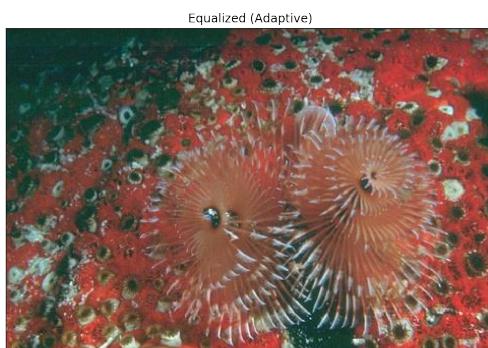
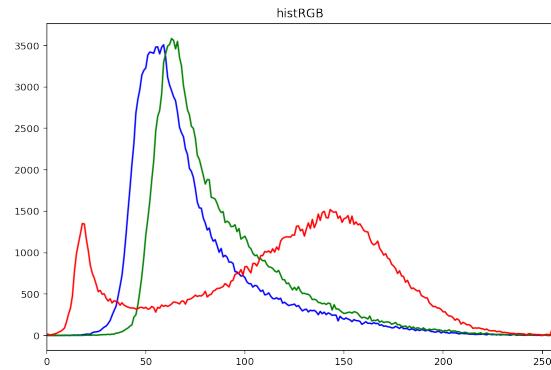
for imagename in IMG_NAMES:
    imagepath = os.path.join('data/img/', imagename)
    adaptive_histogram_equalization(imagepath)

clipLimit = 5.0    tileGridSize = (8,8)
```

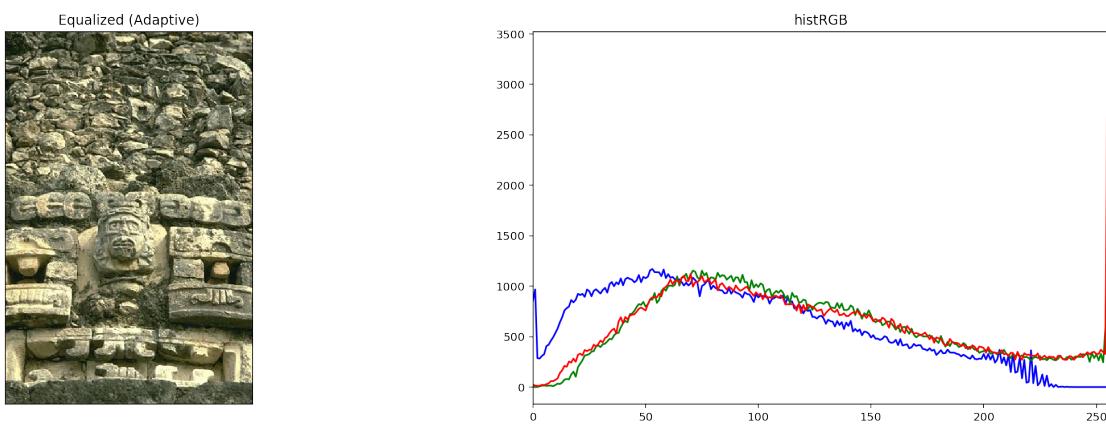
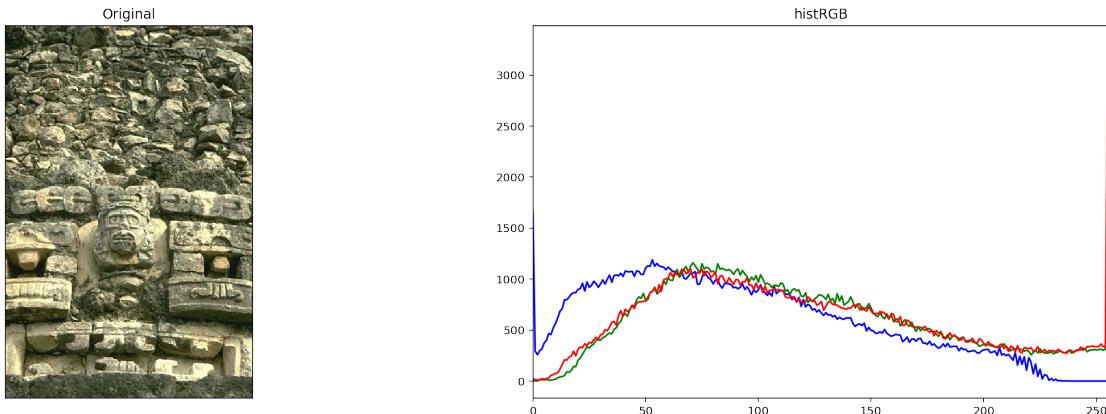




```
clipLimit = 5.0      tileSize = (8,8)
```



```
clipLimit = 5.0      tileSize = (8,8)
```



1.3.1 d) Did you find any difference between the two techniques? (Please briefly answer this question in this block)

1. The global histogram equalization usually will make the picture lost some information and make the brightness inappropriate.
2. The Adaptive Histogram Equalization (CLAHE) is better than the previous one since it does not lose much information and the brightness almost stay the same.