Implementation of Histogram Equalization for Image Enhancement

Aradhya Dhruv (G2303518F - Masters in Artificial Intelligence)

The presented code is a Python script designed to enhance digital images through the application of histogram equalization. The code imports several essential libraries for image processing and analysis, including OpenCV (cv2), PyWavelets (pywt), SEWAR (sewar), pandas, and scikit-image. It also employs functions for image quality assessment metrics such as structural similarity and peak signal-to-noise ratio.

- **Import Necessary Libraries:** The script starts by importing essential libraries, including image processing tools (OpenCV), wavelet analysis (PyWavelets), image quality assessment (SEWAR, scikit-image), and data manipulation (pandas)
- **Image Loading:** Using the 'glob' function, the code reads image files from a specified directory and stores them in a list called 'image_lst.' Each image is read using OpenCV and appended to the list.
- **Histogram Equalization:** The key functionality of this code lies in the **'histogram_equalizers'** class, which is called with the **'image_lst' as input**. The class implments different types of histogram equalization techniques such as Global, Adpative, Luminicance, Contrast limiting etc and also calculates the psnr and ssim score of all the images along with their histogram vs intensity graphs
- There is one more class **"improve_histogram_equalization"** in the end which has **explored possible improvements** over the existing histogram equalization techniques.

In [2]:

```
#Importing Relevant Libraries
import cv2 as cv
import pywt as pwt
import sewar
import pandas as pd
from skimage.metrics import structural_similarity
import numpy as np
import glob
import matplotlib.pyplot as plt
from skimage import data, img_as_float
from skimage.restoration import denoise_nl_means, estimate_sigma
from skimage.metrics import peak_signal_noise_ratio
from skimage.util import random_noise
```

In [55]:

```
# Unoptimized code:
class histogram_equalizers():
    """
    The class has been broken down into multiple methods to make the code structure organized
    """
    def __init__(self, image_lst):
        """
        The constructor only prints the intensity vs frequency graph before performing any equalizations
        """
        self.image_lst = image_lst
        self.psnr_score_withoutOpenCV = []
        self.ssim_score_withoutOpenCV = []
        self.psnr_scores_global = []
        self.ssim_scores_global = []
        self.psnr_scores_luminance = []
        self.ssim_scores_luminance = []
        self.psnr_scores_clahe = []
        self.ssim_scores_clahe = []
        self.psnr_scores_adaptive = []
        self.ssim_scores_adaptive = []
        self.ssim_scores_denoised_adaptive = []
        self.ssim_scores_denoised_adaptive = []
        self.plot_original_histogram()

    def initialize_subplots(self, graph_title):
        """
        Initializes the subplots, before plotting histogram
        """
        fig, self.axs = plt.subplots(2, 4, figsize=(10, 10))
        fig.tight_layout(pad=3)
        fig.suptitle(graph_title, fontsize=15, y=1.08)

    def histogram_equalization_bgr(self, image_lst):
        """
        The function splits the image in b,g,r components and passes the channel values to "equalize_channels"
        which performs HE without the use of OpenCV library
        """
        self.initialize_subplots("After Histogram EqualizationL: Without using CV2 Library")
        equalized_image_lst = []
        for count, item in enumerate(image_lst):
            blue_channel, green_channel, red_channel = cv.split(item)
            blue_equalized = self.equalize_channels(blue_channel)
            green_equalized = self.equalize_channels(green_channel)
            red_equalized = self.equalize_channels(red_channel)
            equalized_image = cv.merge([blue_equalized, green_equalized, red_equalized])
            channels = [blue_equalized, green_equalized, red_equalized]
```

```
def equalize_channels(self, channel):
    """
    The function equalizes the b,g,r channnels without the use if cv2 library
    """
    hist, _ = np.histogram(channel, bins=256, range=(0, 256))
    cdf = hist.cumsum()
    cdf_normalized = (cdf - cdf.min()) * 255 / (cdf.max() - cdf.min())
    equalized_channel = np.interp(channel, range(256), cdf_normalized).astype(np.uint8)
    return equalized_channel

def wavelet_transformation(self, image_lst):
    """
    The function performs wavelet transformation on the luminance channel of the image
    before the histogram equalization is implemented
    """
    self.initialize_subplots("After Wavelet Transformation HSE:")
    wavelet_he_images = []
    print('**Metrics for Wavelet Histogram Equalization**\n')
    for count, item in enumerate(image_lst):
        lab_image = cv.cvtColor(item, cv.COLOR_BGR2LAB)
        l,a,b = cv.split(item)
        l_coeff = pwt.wavedec(l, 'sym5', level=2)
        l_coeff[0] = cv.equalizeHist(np.uint8(np.abs(l_coeff[0])))
        enhanced_l = pwt.waverec(l_coeff, 'sym5')
        enhanced_l = cv.resize(enhanced_l, (item.shape[1], item.shape[0]))
        enhanced_image = cv.merge([np.uint8(enhanced_l), a, b])
        b,g,r = cv.split(enhanced_image)
        channels = [b,g,r]
        self.plot_histogram(channels, count)
        wavelet_he_images.append(enhanced_image)
    self.calculate_mse(image_lst, wavelet_he_images, "MSE Scores of Wavelet Transfomration HSE")
    self.calculate_psnr(image_lst, wavelet_he_images, "PSNR Scores of Wavelet Transfomration HSE")
    self.calculate_ssim(image_lst, wavelet_he_images, "SSIM Scores of Wavelet Transfomration HSE")
    self.display_equalized_image(image_lst, wavelet_he_images, "Wavelet Transfomration Histogram Equalized Images (2nd Row)")

def global_HE(self, image_lst):
    """
    The function implements Global HSE
    """
    self.initialize_subplots("After Global HSE:")
    print('**Metrics for Global Histogram Equalization**\n')
    global_he_images = []
    for count, item in enumerate(image_lst):
        b,g,r = cv.split(item)
        equalized_b = cv.equalizeHist(b)
        equalized_g = cv.equalizeHist(g)
        equalized_r = cv.equalizeHist(r)
        channels = [equalized_b, equalized_g, equalized_r]
        self.plot_histogram(channels, count)
        equalized_image = cv.merge([equalized_b, equalized_g, equalized_r])
        global_he_images.append(equalized_image)
    self.calculate_mse(image_lst, global_he_images, "MSE Scores of Global HSE")
    self.calculate_psnr(image_lst, global_he_images, "PSNR Scores of Global HSE")
    self.calculate_ssim(image_lst, global_he_images, "SSIM Scores of Global HSE")
    self.display_equalized_image(image_lst, global_he_images, "Global Histogram Equalized Images (2nd Row)")


def luminance_HE(self, image_lst):
    """
    The function implements Luminance HSE
    """
    self.initialize_subplots("After Luminance HSE")
    luminance_he_images = []
    print('**Metrics for Luminance Histogram Equalization**\n')
    for count, item in enumerate(image_lst):
        hsv_image = cv.cvtColor(item, cv.COLOR_BGR2HSV)
        h,s,v = cv.split(hsv_image)
        equalized_v = cv.equalizeHist(v)
        merged_hsv = cv.merge([h, s, equalized_v])
        equalized_image = cv.cvtColor(merged_hsv, cv.COLOR_HSV2BGR)
        b,g,r = cv.split(equalized_image)
        channels = [b,g,r]
        self.plot_histogram(channels, count)
        luminance_he_images.append(equalized_image)
    self.calculate_mse(image_lst, luminance_he_images, "MSE Scores of luminance HSE")
    self.calculate_psnr(image_lst, luminance_he_images, "PSNR Scores of luminance HSE")
    self.calculate_ssim(image_lst, luminance_he_images, "SSIM Scores of luminance HSE")
    self.display_equalized_image(self.image_lst, luminance_he_images, "Luminance Histogram Equalized Images (2nd Row)")

def adaptive_HE(self, image_lst):
    """
    The function implements Adaptive HSE
    """
    self.initialize_subplots("After Adaptive HSE")
    adaptive_he_images = []
    print('**Metrics for Adaptive Histogram Equalization**\n')
    for count,item in enumerate(image_lst):
        clahe_hse = cv.createCLAHE(clipLimit = 2, tileGridSize = (4,4))
        labimage = cv.cvtColor(item, cv.COLOR_BGR2LAB)
        labimage[:,:,0 ] = clahe_hse.apply(labimage[:,:,0 ])
        equalized_img_bgr = cv.cvtColor(labimage, cv.COLOR_LAB2BGR)
        b,g,r = cv.split(equalized_img_bgr)
```

```
        self.calculate_ssim(image_lst, adaptive_he_images, "SSIM Scores of Adaptive HSE")
        self.display_equalized_image(self.image_lst, adaptive_he_images, "Adaptive Histogram Equalized Images (2nd Row)")

    def contrast_limiting_HE(self,image_lst):
        """
        The function implements contrast limiting HSE
        """
        self.initialize_subplots("After CLAHE HSE")
        contrast_limit_he_images = []
        print('**Metrics for Contrast Limiting Histogram Equalization**\n')
        for count,item in enumerate(image_lst):
            clahe_hse = cv.createCLAHE(clipLimit = 5)
            labimage = cv.cvtColor(item, cv.COLOR_BGR2LAB)
            labimage[:,:,0 ] = clahe_hse.apply(labimage[:,:,0 ])
            equalized_img_bgr = cv.cvtColor(labimage, cv.COLOR_LAB2BGR)
            b,g,r = cv.split(equalized_img_bgr)
            channels = [b,g,r]
            self.plot_histogram(channels, count)
            contrast_limit_he_images.append(equalized_img_bgr)
        self.calculate_mse(image_lst, contrast_limit_he_images, "MSE Scores of CLAHE HSE")
        self.calculate_psnr(image_lst, contrast_limit_he_images, "PSNR Scores of CLAHE HSE")
        self.calculate_ssim(image_lst, contrast_limit_he_images, "SSIM Scores of CLAHE HSE")
        self.display_equalized_image(self.image_lst, contrast_limit_he_images, "CLAHE Histogram Equalized Images (2nd Row)")

    def calculate_mse(self, image_lst, equalized_image_set, title):
        """
        The function calculates the mean squared error on the equalized images
        """
        original_images = []
        equalized_images = []
        total_error = 0
        for count, (original_image, equalized_image) in enumerate(zip(image_lst, equalized_image_set)):
            error = sewar.mse(original_image,equalized_image)
            total_error +=error
        print("\nMSE of All Images: ", round(total_error/len(image_lst)))

    def calculate_psnr(self, image_lst, equalized_image_set, title):
        """
        The function calculates peak-signal-to-noise ratio on the equalized images
        """
        total_error = 0
        for count, (original_image, equalized_image) in enumerate(zip(image_lst, equalized_image_set)):
            error = sewar.psnr(original_image, equalized_image)
            total_error +=error
            if ('luminance' in title.lower()):
                self.psnr_scores_luminance.append(error)
            elif('clahe' in title.lower()):
                self.psnr_scores_clahe.append(error)
            elif('global' in title.lower()):
                self.psnr_scores_global.append(error)
            elif('without' in title.lower()):
                self.psnr_score_withoutOpenCV.append(error)
            else:
                self.psnr_scores_adaptive.append(error)
        print("\nPSNR of All Images: ", round(total_error/len(image_lst)))

    def calculate_ssim(self, image_lst, equalized_image_set, title):
        """
        The function calculates Structure-Similarity-Index on the equalized images
        """
        total_error = 0
        for count, (original_image, equalized_image) in enumerate(zip(image_lst, equalized_image_set)):
            error_b = structural_similarity(original_image[:,:,0], equalized_image[:,:,0])
            error_g = structural_similarity(original_image[:,:,1], equalized_image[:,:,1])
            error_r = structural_similarity(original_image[:,:,2], equalized_image[:,:,2])
            error = (error_b+error_g+error_r)/3
            total_error+=error
            if ('luminance' in title.lower()):
                self.ssim_scores_luminance.append(error)
            elif ('clahe' in title.lower()):
                self.ssim_scores_clahe.append(error)
            elif('global' in title.lower()):
                self.ssim_scores_global.append(error)
            elif('without' in title.lower()):
                self.ssim_score_withoutOpenCV.append(error)
            else:
                self.ssim_scores_adaptive.append(error)
        print("\nSSIM of All Images: ", total_error/len(image_lst))

    def plot_errors(self):
        """
        The function creates a dataframe out of the metric scores(psnr, ssim metrics) and visualizes them for an easy
        comparison
        """
        samples = ["sample1", "sample2", "sample3", "sample4", "sample5", "sample6", "sample7", "sample8"]

        df = pd.DataFrame(list(zip(samples, self.psnr_scores_luminance, self.ssim_scores_luminance,
                                   self.psnr_scores_clahe, self.ssim_scores_clahe,
                                   self.psnr_scores_global, self.ssim_scores_global,
                                   self.psnr_scores_adaptive, self.ssim_scores_adaptive,
                                   self.psnr_score_withoutOpenCV, self.ssim_score_withoutOpenCV)),
                          columns=['sample','psnr_score_luminance', 'ssim_score_luminance',
                                   'psnr_score_clahe', 'ssim_score_clahe',
```

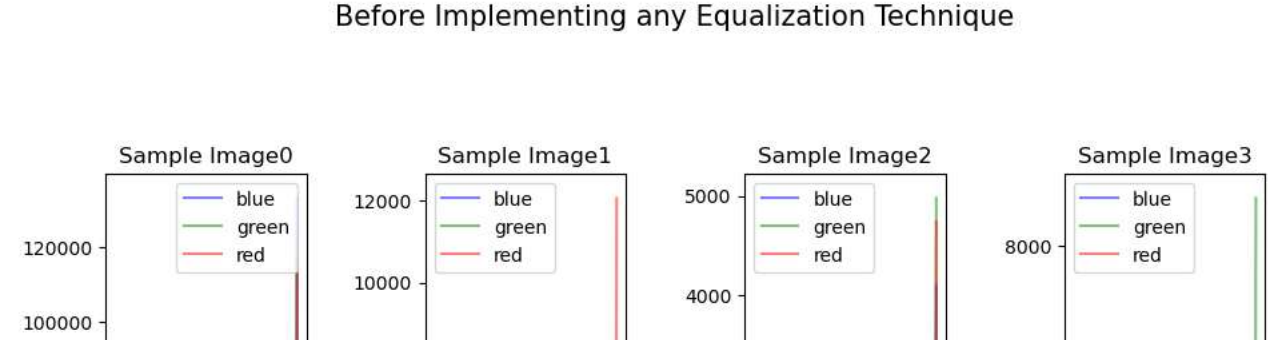
```
                                            df['psnr_score_global'].mean(), df['psnr_score_adaptive'].mean(),
                                           df['psnr_score_withoutOpenCv'].mean()]],
                                    columns=['luminance_psnr', 'Clahe_psnr', 'Global_psnr', 'Adaptive_psnr','withoutCV_psnr'])

        df_avg_ssim_scores = pd.DataFrame(data=[[df['ssim_score_luminance'].mean(), df['ssim_score_clahe'].mean(),
                                            df['ssim_score_global'].mean(),df['ssim_score_adaptive'].mean(),
                                           df['ssim_score_withoutOpenCv'].mean()]],
                                    columns=['luminance_ssim', 'Clahe_ssim', 'Global_ssim', 'Adaptive_ssim', 'withoutCV_ssim'])

        df_avg_psnr_scores.plot(kind='bar', title='Average PSNR Scores').legend(loc='upper right')
        df_avg_ssim_scores.plot(kind='bar', title='Average SSIM Scores').legend(loc='upper right')

        df.plot(x="sample", y=['psnr_score_luminance', 'psnr_score_clahe',
                               'psnr_score_global', 'psnr_score_adaptive','psnr_score_withoutOpenCv'], kind="line", figsize=(10, 4), li
width=2.0,
                title='PSNR Scores of All Four Equalizations').legend(loc='upper right')

        df.plot(x="sample", y=['ssim_score_luminance', 'ssim_score_clahe',
                               'ssim_score_global','ssim_score_adaptive','ssim_score_withoutOpenCv'], kind="line", figsize=(10, 4), lin
idth=2.0,
                title='SSIM Scores of All Four Equalizations').legend(loc='upper right')

    def display_equalized_image(self, image_lst, equalized_image_set, frame_title):
        """
        The function displays original vs equalized image together for comparison
        (1st row = Orginal Image, 2nd row = Equalized Image)
        """
        stacked_image_set = None
        stacked_equalized_img_set = None
        for i, image in enumerate(image_lst):
            x = cv.resize(image, (190,250))
            stacked_image_set = cv.hconcat([stacked_image_set, x]) if stacked_image_set is not None else x
        for j, image in enumerate(equalized_image_set):
            y = cv.resize(image, (190,250))
            stacked_equalized_img_set = cv.hconcat([stacked_equalized_img_set, y]) if stacked_equalized_img_set is not None else y
        stack_all_images = cv.vconcat([stacked_image_set, stacked_equalized_img_set])
        cv.imshow(frame_title, stack_all_images)
        cv.waitKey(0)


    def plot_original_histogram(self):
        """
        This function is specifically used to plot orginal BGR values before any histogram equalization was implemented
        """
        self.initialize_subplots("Before Implementing any Equalization Technique")
        for count, item in enumerate(self.image_lst):
            b,g,r = cv.split(item)
            channels = [b,g,r]
            self.plot_histogram(channels, count)

    def plot_histogram(self, channels, count):
        """
        This function only takes the individual color channels and then plots a histogram for each color
        """
        colors = ['blue','green', 'red']
        for color_itr,channel_val in enumerate(channels):
            standardize_channel = cv.calcHist([channel_val], [0], None, [256], [0,256]).flatten()
            row = count//4
            col = count%4
            axs = self.axs[row,col]
            axs.plot(standardize_channel, color = colors[color_itr], label=colors[color_itr], alpha=0.5)
            axs.legend()
            axs.set_title('Sample Image' + str(count))
```

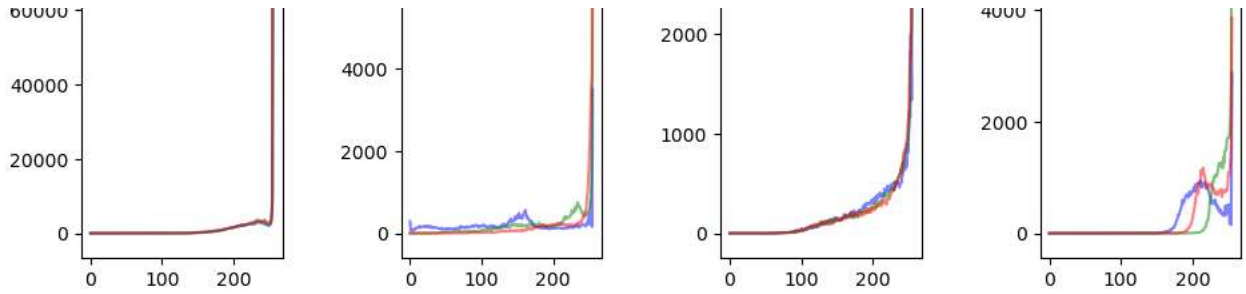
In [56]:

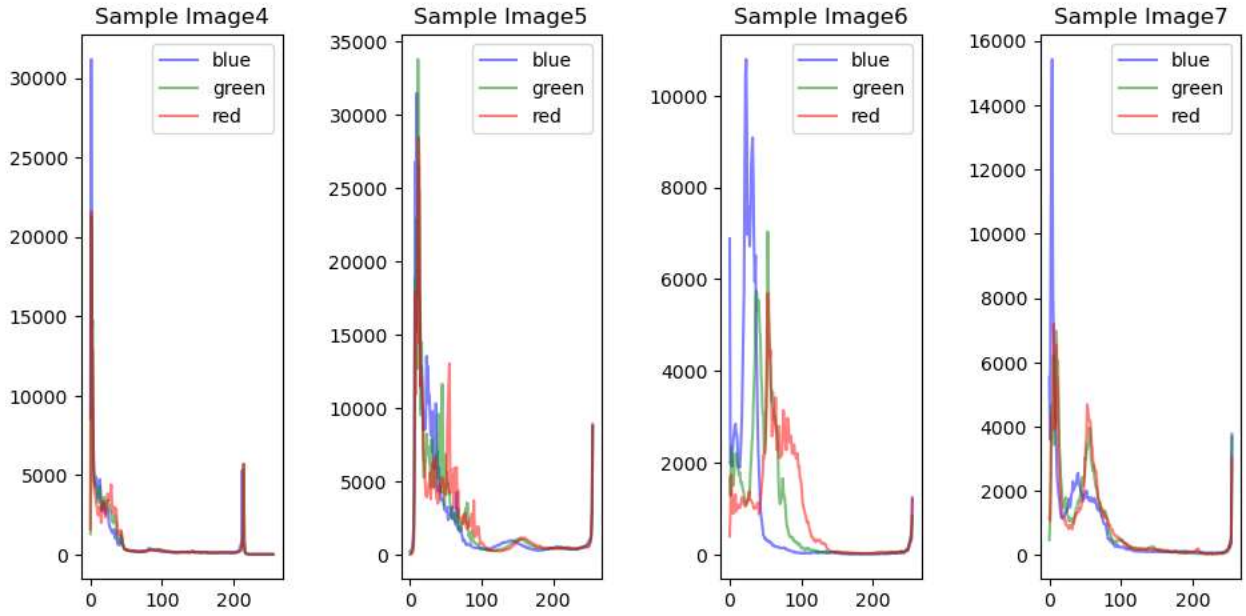
```
### Reading all image files from the folder structure using glob function

path = "D:/MSAI Coursework/sample_images/*.*"
image_lst=[]
for file in glob.glob(path):
  image_read = cv.imread(file)
  image_lst.append(image_read)
```

In [57]:

```
x = histogram_equalizers(image_lst)
```





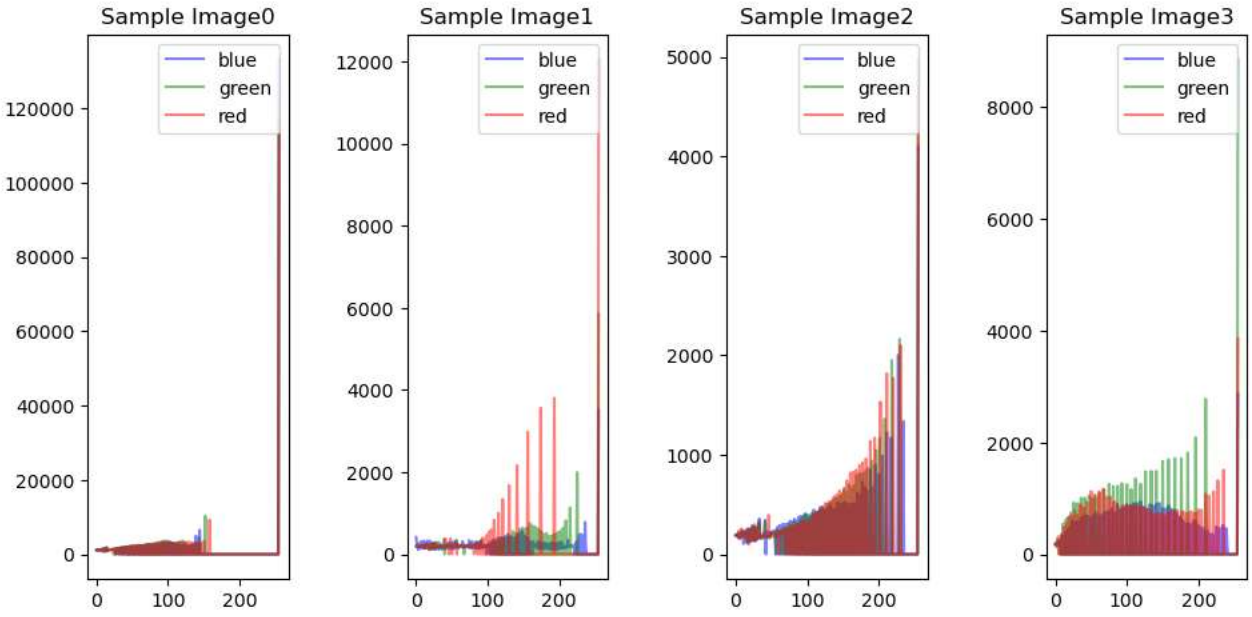


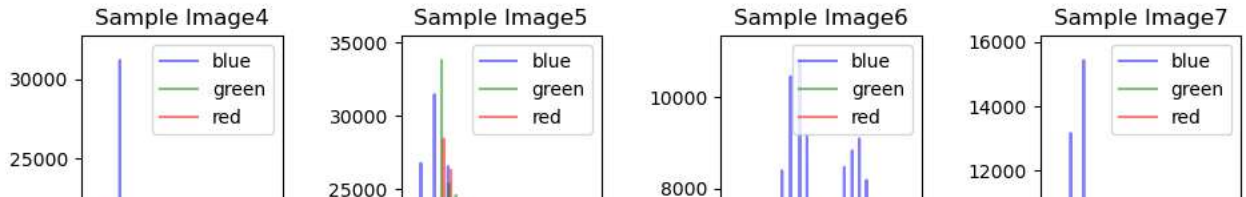
In [58]:

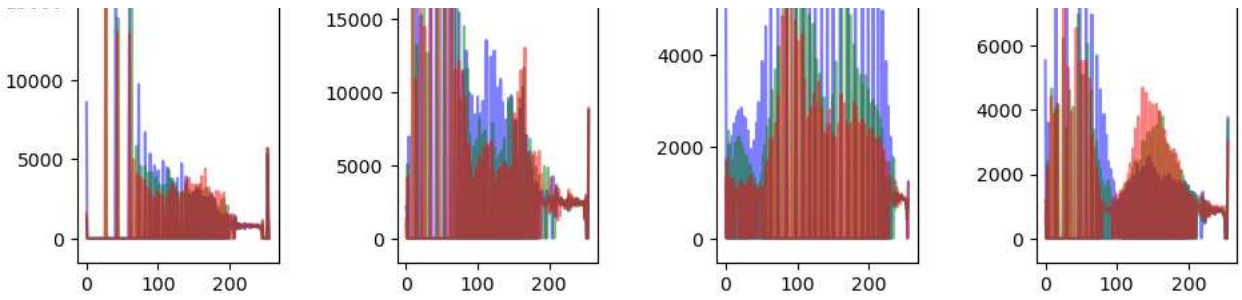
```
x.histogram_equalization_bgr(image_lst)
```

```
PSNR of All Images:  8

SSIM of All Images:  0.4540376841034637
```

After Histogram EqualizationL: Without using CV2 Library







In [59]:

```
x.global_HE(image_lst)
```

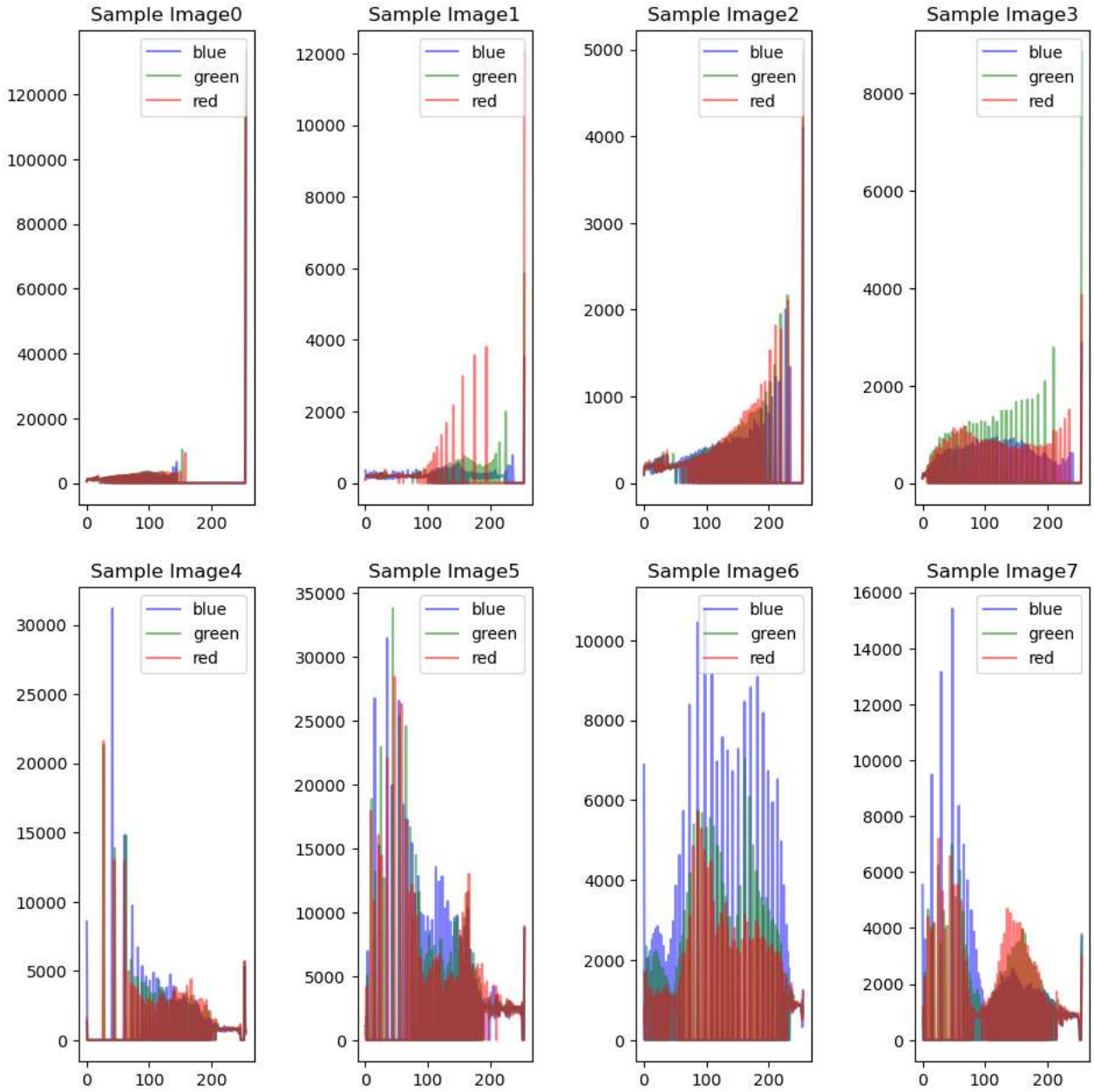
Metrics for Global Histogram Equalization

MSE of All Images: 9536

PSNR of All Images: 8

SSIM of All Images: 0.45407956285285156

After Global HSE:

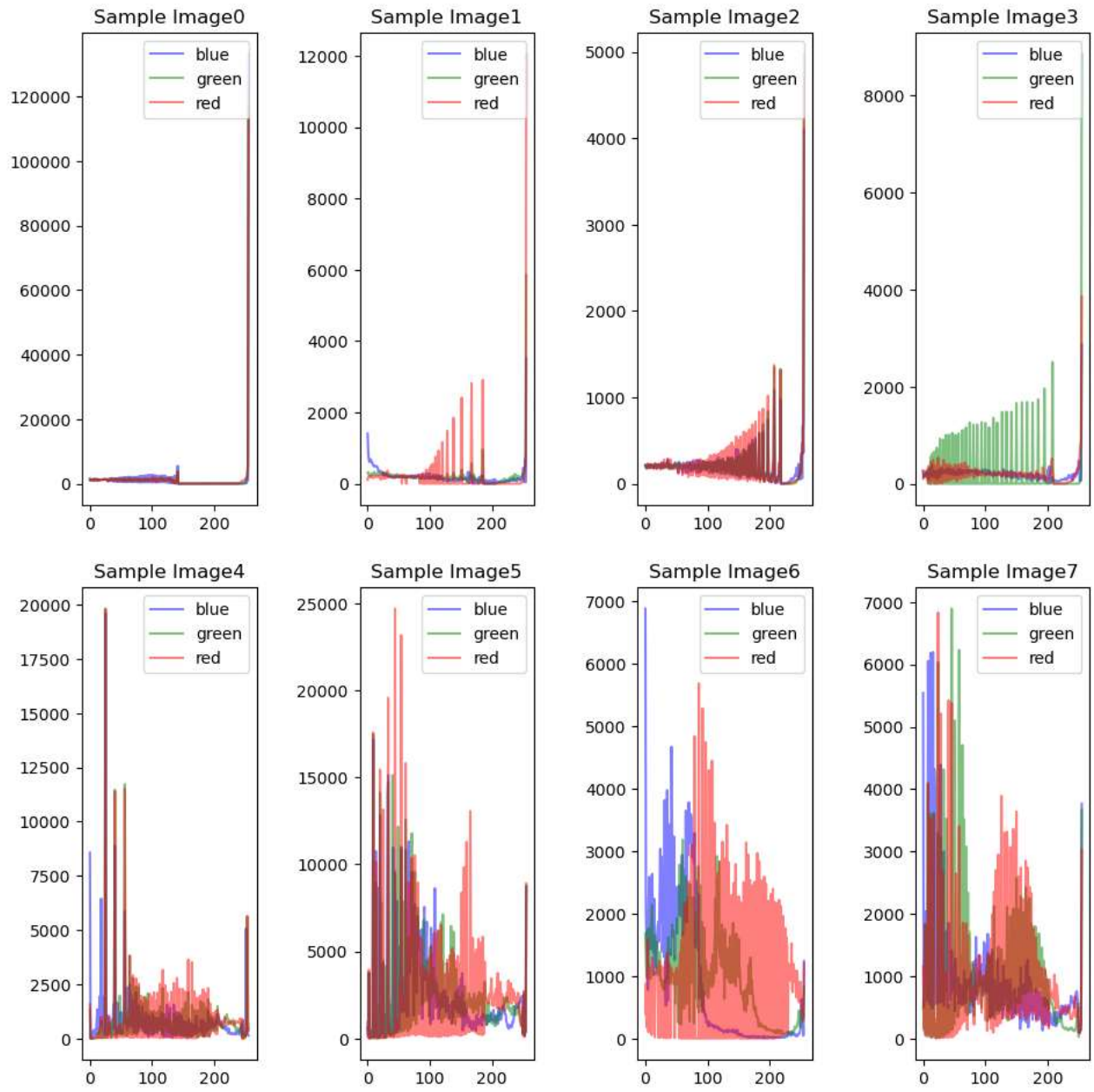


In [60]:

```
x.luminance_HE(image_lst)
```

PSNR of All Images: 10

SSIM of All Images: 0.4499255922409561



In [61]:

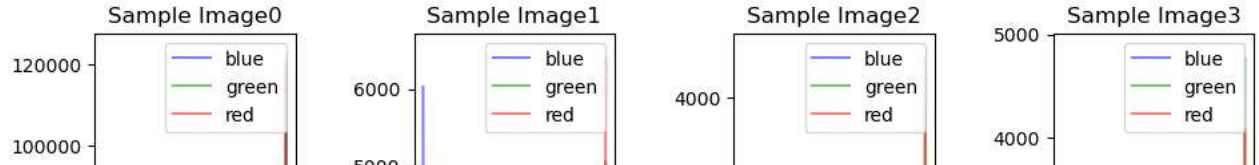
```
x.contrast_limiting_HE(image_lst)
```

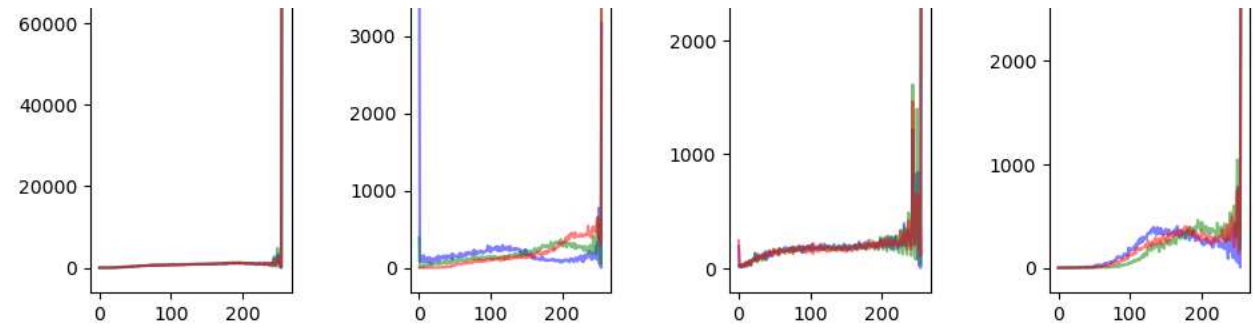
Metrics for Contrast Limiting Histogram Equalization

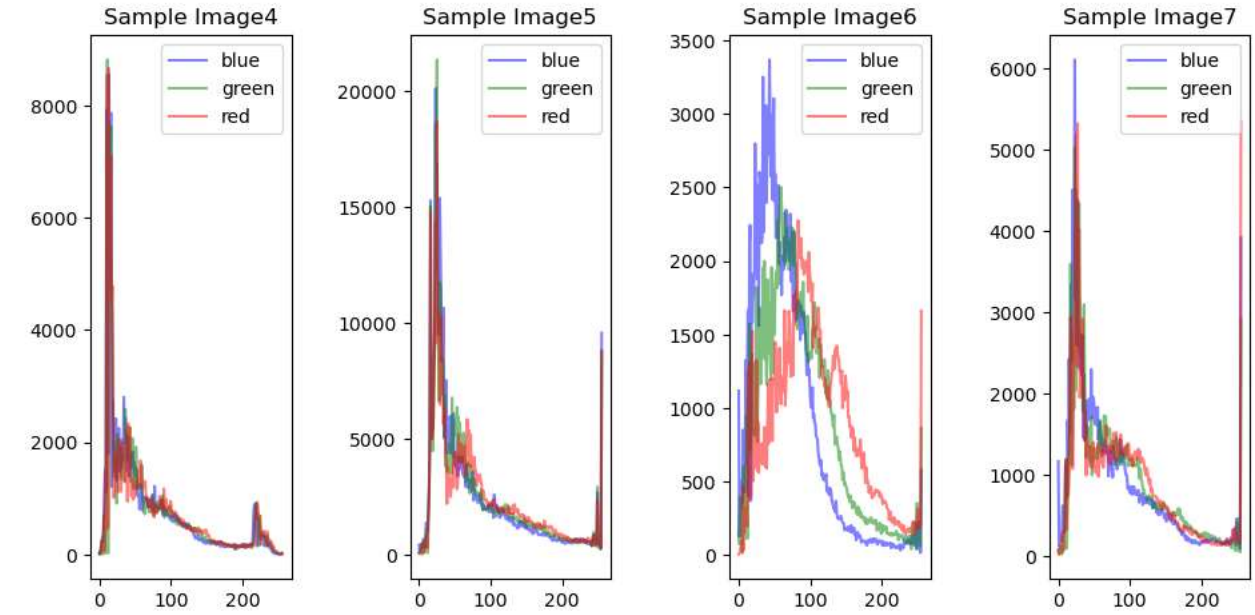
MSE of All Images: 2413

PSNR of All Images: 15

SSIM of All Images: 0.6117754121699437







In [62]:

```
x.adaptive_HE(image_lst)
```

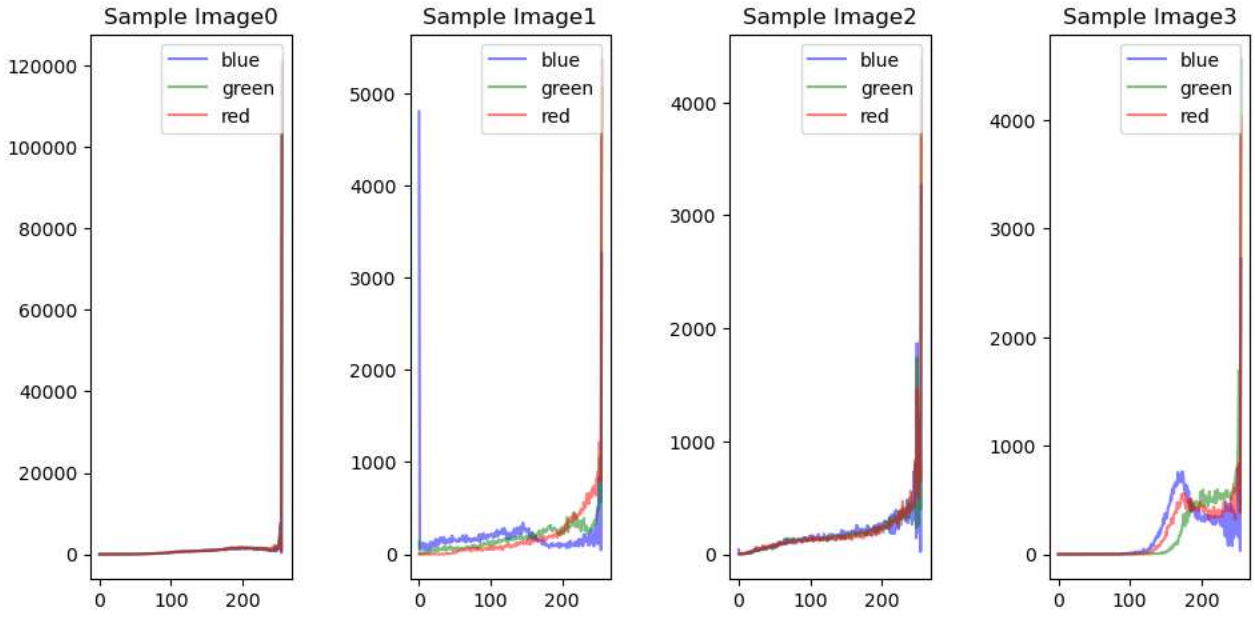
Metrics for Adaptive Histogram Equalization

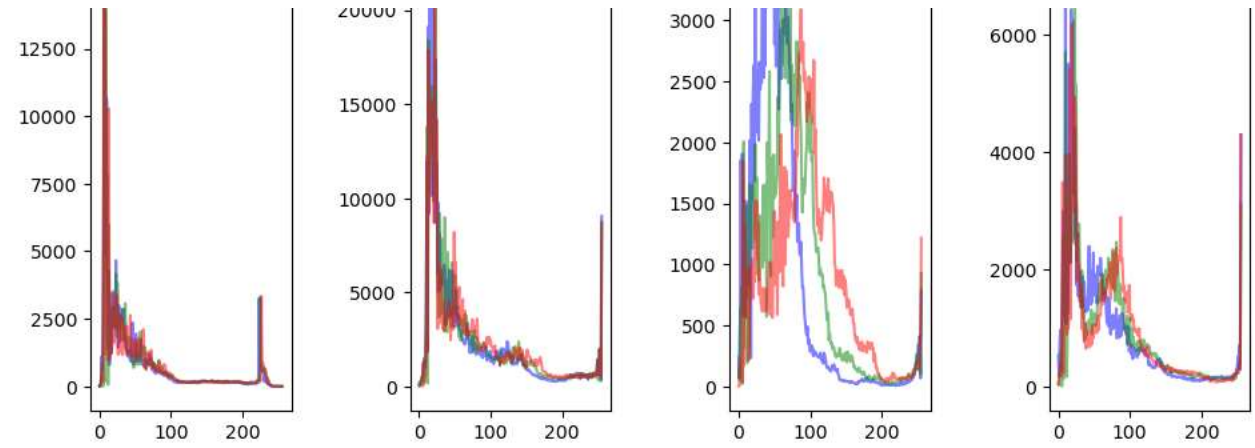
MSE of All Images: 906

PSNR of All Images: 19

SSIM of All Images: 0.8038868010069306

After Adaptive HSE





In [63]:

```
x.wavelet_transformation(image_lst)
```

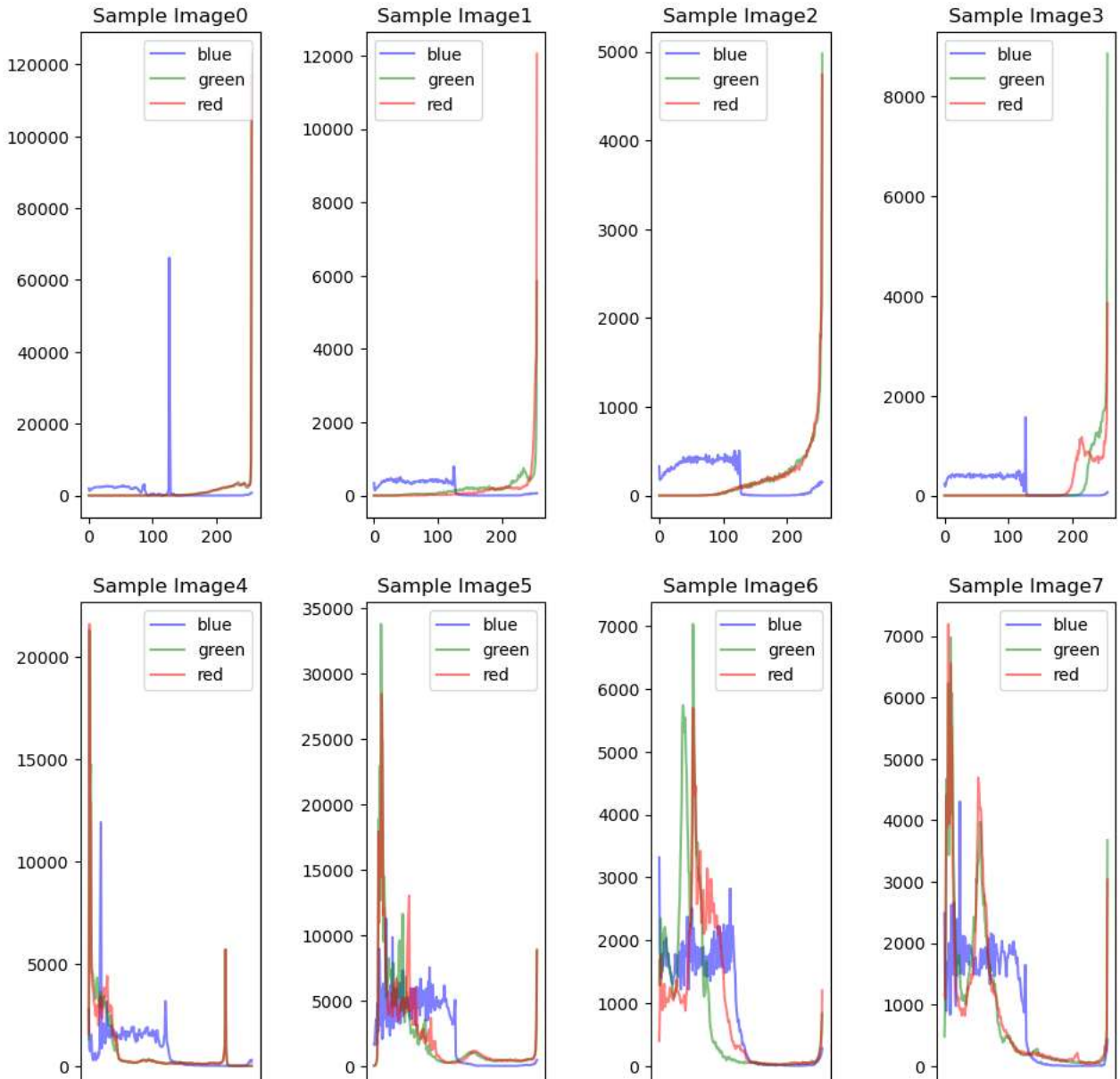
Metrics for Wavelet Histogram Equalization

MSE of All Images: 3883

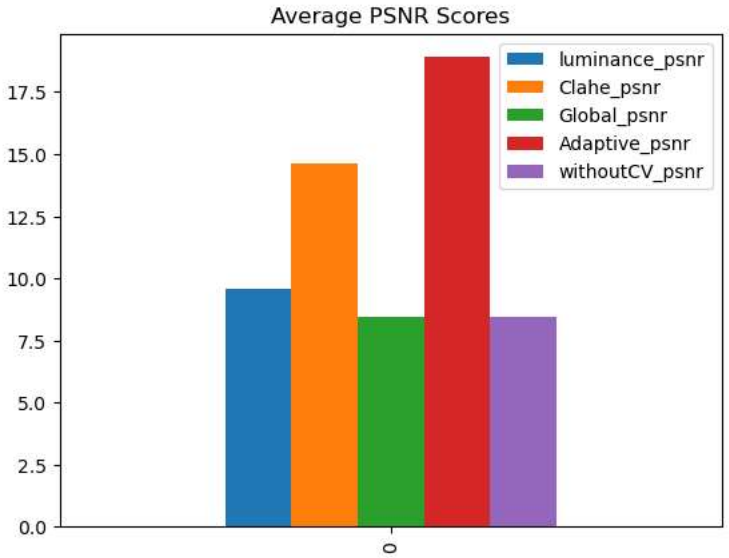
PSNR of All Images: 14

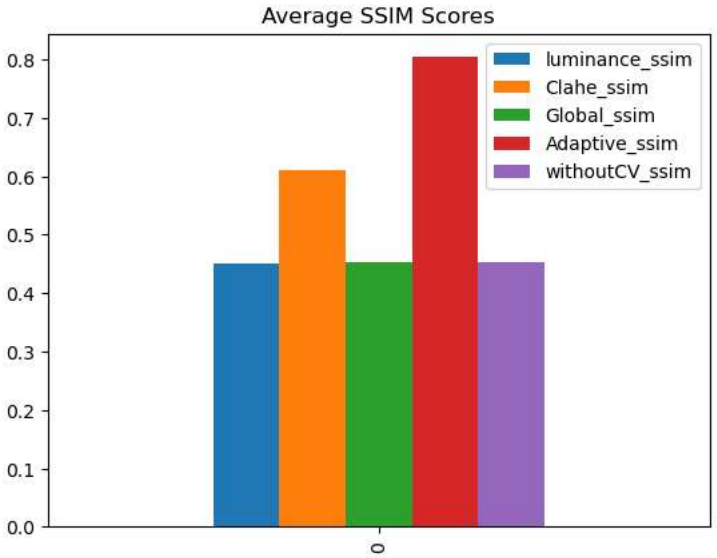
SSIM of All Images: 0.8147748448486718

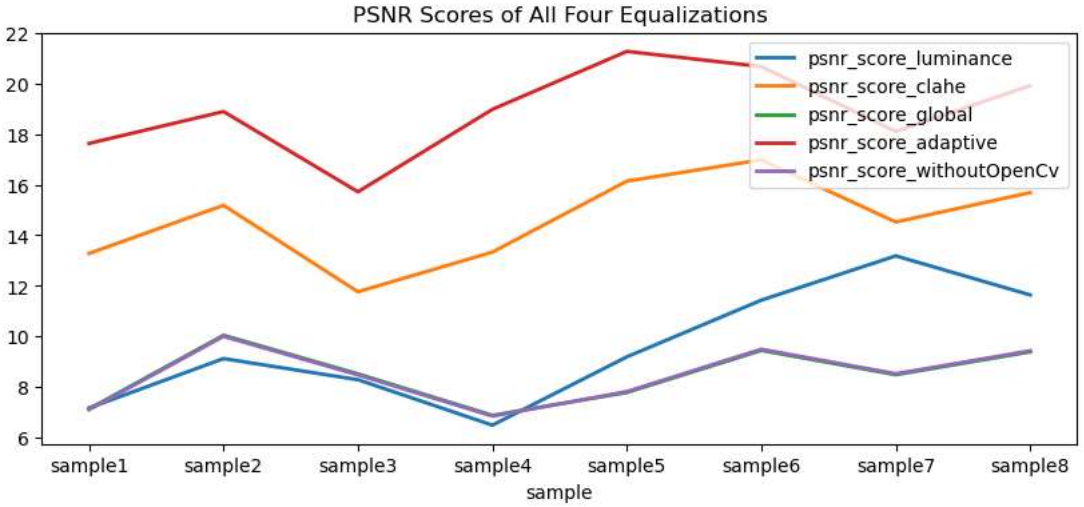
After Wavelet Transformation HSE:



```
x.plot_errors()
```







Experimentation:

In [65]:

```
class improve_histogram_equalization():
    def __init__(self, image_lst):
        """
        The constructor defines empty lists for storing metrics scores of each image
        """
        self.image_lst = image_lst
        self.denoised_image_lst = []
        self.gaussian_images = []
        self.ssim_scores_adaptive_denoised = []
        self.psnr_scores_adaptive_denoised = []
        self.ssim_scores_gaussian_adaptive = []
        self.psnr_scores_gaussian_adaptive = []

    def initialize_subplots(self, graph_title):
        """
        The function initializes the subplots, before plotting histogram
        """
        fig, self.axs = plt.subplots(2, 4, figsize=(10, 10))
        fig.tight_layout(pad=3)
        fig.suptitle(graph_title, fontsize=15, y=1.08)

    def NLM_denoising(self, image_lst):
        """
        The function implements NLM denoising on the images
        """
        for count, item in enumerate(image_lst):
            item = cv.cvtColor(item, cv.COLOR_BGR2LAB)
            sigma_est_l = np.mean(estimate_sigma(item[:,:,0]))

            patch_kw = dict(patch_size=2,      # 5x5 patches
                            patch_distance=4)  # 13x13 search area

            denoised_l = denoise_nl_means(item[:,:,0], h=10, fast_mode=True, **patch_kw)
            denoised_l = cv.convertScaleAbs(denoised_l)
            denoised_lab = cv.merge([denoised_l, item[:,:,1], item[:,:,2]])
            denoised_image = cv.cvtColor(denoised_lab, cv.COLOR_LAB2BGR)
            self.denoised_image_lst.append(denoised_image)

        return self.denoised_image_lst

    def adaptive_HE_denoised(self, image_lst):
        """
        The function implements Adaptive HSE
        """
        self.denoised_image_lst = self.NLM_denoising(image_lst)
        self.initialize_subplots("After Adaptive HSE")
        adaptive_he_images_denoised = []
        print('**Metrics for Adaptive Histogram Equalization after NLM Denoising**\n')
        for count,item in enumerate(self.denoised_image_lst):
            clahe_hse = cv.createCLAHE(clipLimit = 2, tileGridSize = (4,4))
            labimage = cv.cvtColor(item, cv.COLOR_BGR2LAB)
            labimage[:,:,0 ] = clahe_hse.apply(labimage[:,:,0 ])
            equalized_img_bgr = cv.cvtColor(labimage, cv.COLOR_LAB2BGR)
            b,g,r = cv.split(equalized_img_bgr)
            channels = [b,g,r]
            self.plot_histogram(channels, count)
            adaptive_he_images_denoised.append(equalized_img_bgr)
        self.calculate_psnr(image_lst, adaptive_he_images_denoised, "PSNR Scores of Denoised Adaptive HSE")
        self.calculate_ssim(image_lst, adaptive_he_images_denoised, "SSIM Scores of Denoised Adaptive HSE")
        self.display_equalized_image(image_lst, adaptive_he_images_denoised, "After NLM Denoising and Adaptive Histogram Equalization
(2nd Row)")

    def gaussian_blur(self, image_lst):
        """
        The function implements Gaussian Blur, before equalization is performed on the image set
        """
        gaussian_images = []
        for count, item in enumerate(image_lst):
            denoised_color_image_bgr = cv.GaussianBlur(item, (5, 5), -1)  # Adjust the kernel size as needed
            # Perform Histogram Equalization on the denoised image (if needed)
            self.gaussian_images.append(denoised_color_image_bgr)
        return self.gaussian_images

    def adaptive_HE_gaussian(self, image_lst):
        """
        The function implements Adaptive HSE
        """
        self.gaussian_img_lst= self.gaussian_blur(image_lst)
        self.initialize_subplots("After Gaussian Blurring and Adaptive HSE")
        adaptive_he_gaussian_images = []
        print('**Metrics for Adaptive Histogram Equalization after Gaussian smoothing**\n')
        for count,item in enumerate(self.gaussian_img_lst):
            clahe_hse = cv.createCLAHE(clipLimit = 2, tileGridSize = (4,4))
            labimage = cv.cvtColor(item, cv.COLOR_BGR2LAB)
            labimage[:,:,0 ] = clahe_hse.apply(labimage[:,:,0 ])
            equalized_img_bgr = cv.cvtColor(labimage, cv.COLOR_LAB2BGR)
```

```
        self.calculate_ssim(image_lst, adaptive_he_gaussian_images, "SSIM Scores of Gaussian Smoothing Adaptive HSE")
        self.display_equalized_image(image_lst, adaptive_he_gaussian_images, "After Gaussian smoothing and Adaptive Histogram
Equalization (2nd Row)")

    def plot_histogram(self, channels, count):
        """
        This function only takes the individual color channels and then plots a histogram for each color
        """
        colors = ['blue','green', 'red']
        for color_itr,channel_val in enumerate(channels):
            standardize_channel = cv.calcHist([channel_val], [0], None, [256], [0,256]).flatten()
            row = count//4
            col = count%4
            axs = self.axs[row,col]
            axs.plot(standardize_channel, color = colors[color_itr], label=colors[color_itr], alpha=0.5)
            axs.legend()
            axs.set_title('Sample Image' + str(count))

    def calculate_psnr(self, image_lst, equalized_image_set, title):
        """
        The function calculates the Peak-Signal-To-Noise Ratio after HSE algorithm is implemented
        """
        total_error = 0
        for count, (original_image, equalized_image) in enumerate(zip(image_lst, equalized_image_set)):
            error = sewar.psnr(original_image, equalized_image)
            total_error +=error
            if ('denoised' in title.lower()):
                self.psnr_scores_adaptive_denoised.append(error)
            else:
                self.psnr_scores_gaussian_adaptive.append(error)
        print("PSNR of All Images: ", round(total_error/len(image_lst)))

    def calculate_ssim(self, image_lst, equalized_image_set, title):
        """
        The function calculates the Structure-Similarity-Index after HSE algorithm is implemented
        """
        total_error = 0
        for count, (original_image, equalized_image) in enumerate(zip(image_lst, equalized_image_set)):
            error_b = structural_similarity(original_image[:,:,0], equalized_image[:,:,0])
            error_g = structural_similarity(original_image[:,:,1], equalized_image[:,:,1])
            error_r = structural_similarity(original_image[:,:,2], equalized_image[:,:,2])
            error = (error_b+error_g+error_r)/3
            total_error+=error
            if('denoised' in title.lower()):
                self.ssim_scores_adaptive_denoised.append(error)
            else:
                self.ssim_scores_gaussian_adaptive.append(error)
        print("SSIM of All Images: ", total_error/len(image_lst))

    def display_equalized_image(self, image_lst, equalized_image_set, frame_title):
        """
        The function displays original vs equalized image together for comparison
        (1st row = Orginal Image, 2nd row = Equalized Image)
        """
        stacked_image_set = None
        stacked_equalized_img_set = None
        for i, image in enumerate(image_lst):
            x = cv.resize(image, (190,250))
            stacked_image_set = cv.hconcat([stacked_image_set, x]) if stacked_image_set is not None else x
        for j, image in enumerate(equalized_image_set):
            y = cv.resize(image, (190,250))
            stacked_equalized_img_set = cv.hconcat([stacked_equalized_img_set, y]) if stacked_equalized_img_set is not None else y
        # Make sure all images have the same number of rows before concatenation
        stack_all_images = cv.vconcat([stacked_image_set, stacked_equalized_img_set])
        cv.imshow(frame_title, stack_all_images)
        cv.waitKey(0)
```

In [66]:

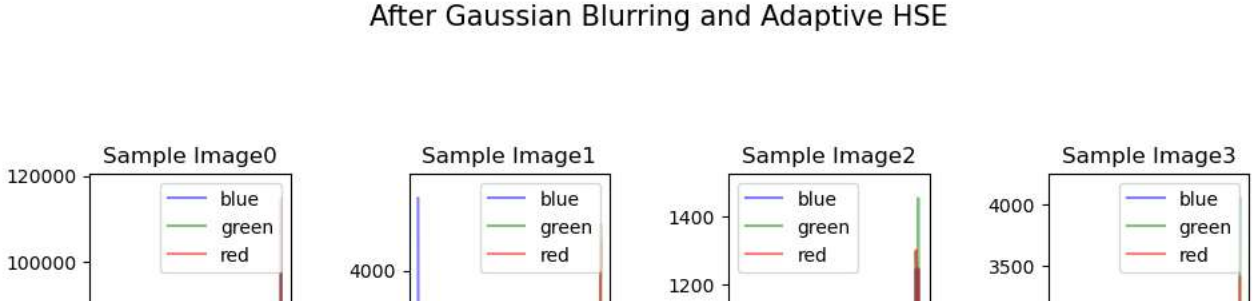
```
y = improve_histogram_equalization(image_lst)
```

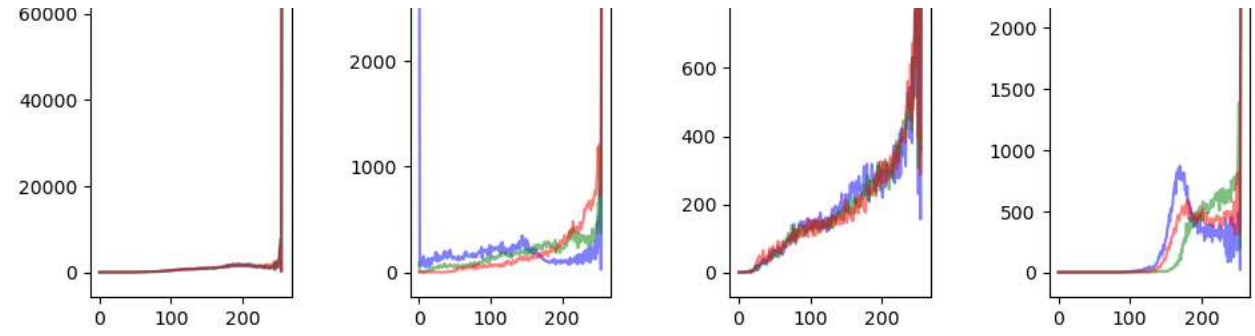
In [67]:

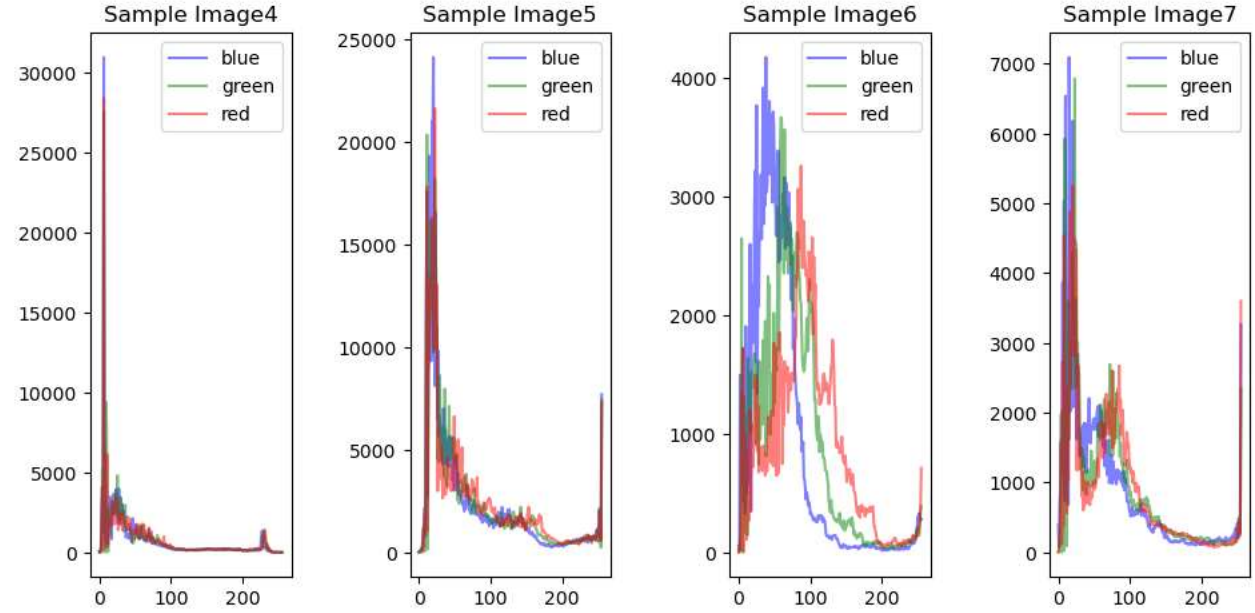
```
y.adaptive_HE_gaussian(image_lst)
```

Metrics for Adaptive Histogram Equalization after Gaussian smoothing

```
PSNR of All Images:  19
SSIM of All Images:  0.8067327605569145
```







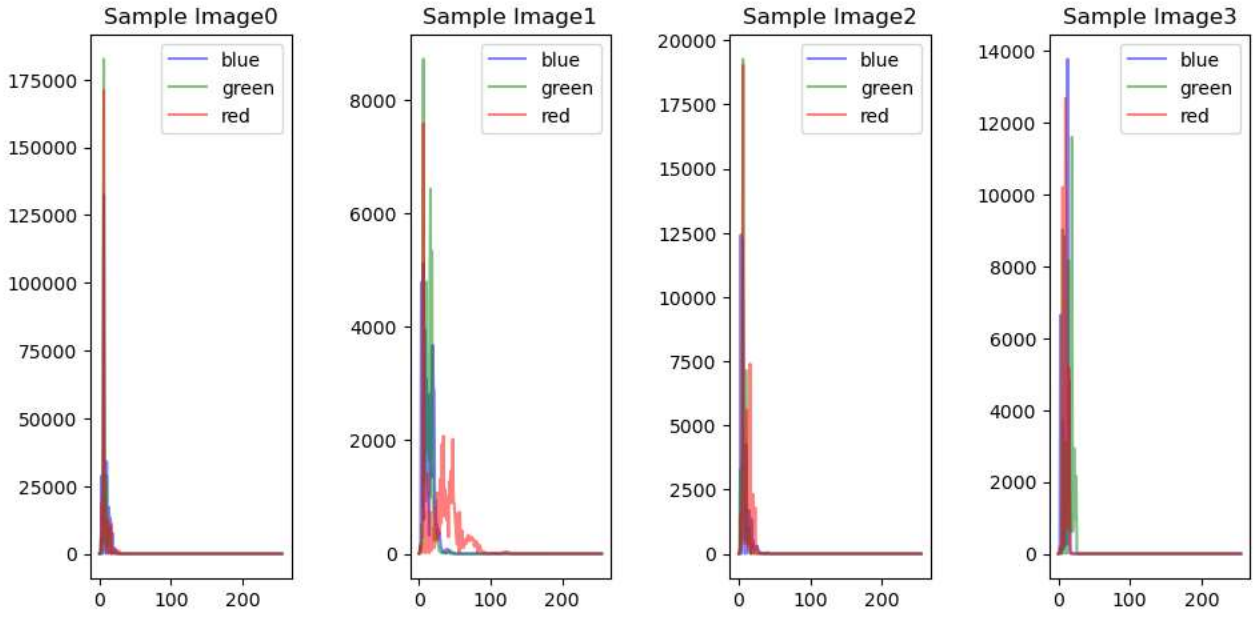
In [68]:

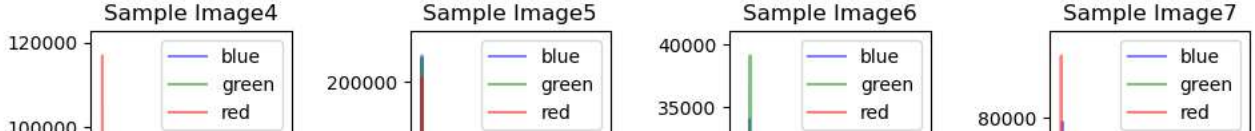
```
y.adaptive_HE_denoised(image_lst)
```

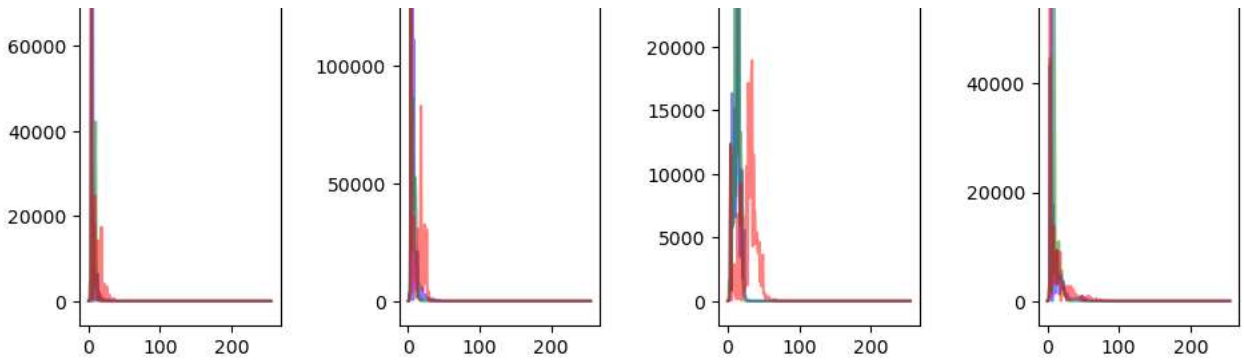
Metrics for Adaptive Histogram Equalization after NLM Denoising

PSNR of All Images: 7
SSIM of All Images: 0.26782371566925617

After Adaptive HSE






60000
40000
20000
0
0
100
200
100000
50000
0
0
100
200
20000
15000
10000
5000
0
0
100
200
40000
20000
0
0
100
200