# AI6121: Computer Vision

# Assignment 1
# Histogram Equalization

**Ron Kow Kheng Hui**

**ID: G1903451J**

15 September 2021

**School of Computer Science and Engineering**

**Nanyang Technological University**

## 1 Introduction

In this assignment, we implement the Histogram Equalization (HE) algorithm using Python and apply HE to eight sample images. We will use the same algorithm in the following three ways and compare the results:

- HE applied globally across all three color channels in the RGB color space.
- HE applied individually to each color channel in the RGB color space.
- HE applied to the *value* channel in the HSV color space.

We will also use the OpenCV library to apply CLAHE (Contrast Limited Adaptive Histogram Equalization) and compare the results with the three ways listed above. This report is organized as follows:

- In Section 2 (Assignment Task 1), we present our implementation of the Histogram Equalization algorithm and the results from the three applications of HE: RGB (HE applied globally), RGB (HE applied individually to each color channel) and HSV.
- In Section 3 (Assignment Task 2), we compare the results, discuss the pros and cons of each implementation and the possible causes of any unsatisfactory results.
- In Section 4 (Assignment Task 3), we discuss possible improvements to the basic HE algorithm. We will present the results from OpenCV's implementation of CLAHE and compare them with our earlier results.
- Lastly, in Section 5, we summarize and conclude our findings.

## 2 Implementation and Application of Histogram Equalization (Task 1)

### 2.1 Definitions and Algorithm

For images with poor contrast due to an excess of dark and light color intensity values, we can improve the contrast by brightening the dark areas or darkening the light areas. This can be achieved by spreading the pixel frequencies across the entire range of intensity values (0 to 255 for 8 bit representation of each color channel).

In an image, the distribution of pixels can be visualized by plotting a histogram of pixel frequencies $h(i)$ against intensity values $i$. Histogram Equalization (HE) is an image processing algorithm for improving contrast in images. The goal of HE is to find an intensity mapping function $f(i)$ that will spread the pixel frequencies such that the resulting histogram is flatter.

To find $f(i)$, we use the cumulative distribution $c(i)$. At intensity value $k$, the cumulative distribution is:

$$c(k) = \frac{1}{N} \textstyle\sum_{i=0}^{k} h(i),$$

where $i = 0$ to 255 is the range of intensity values, $N$ is the total number of pixels in the image and $h(i)$ is the pixel frequency for intensity value $i$.

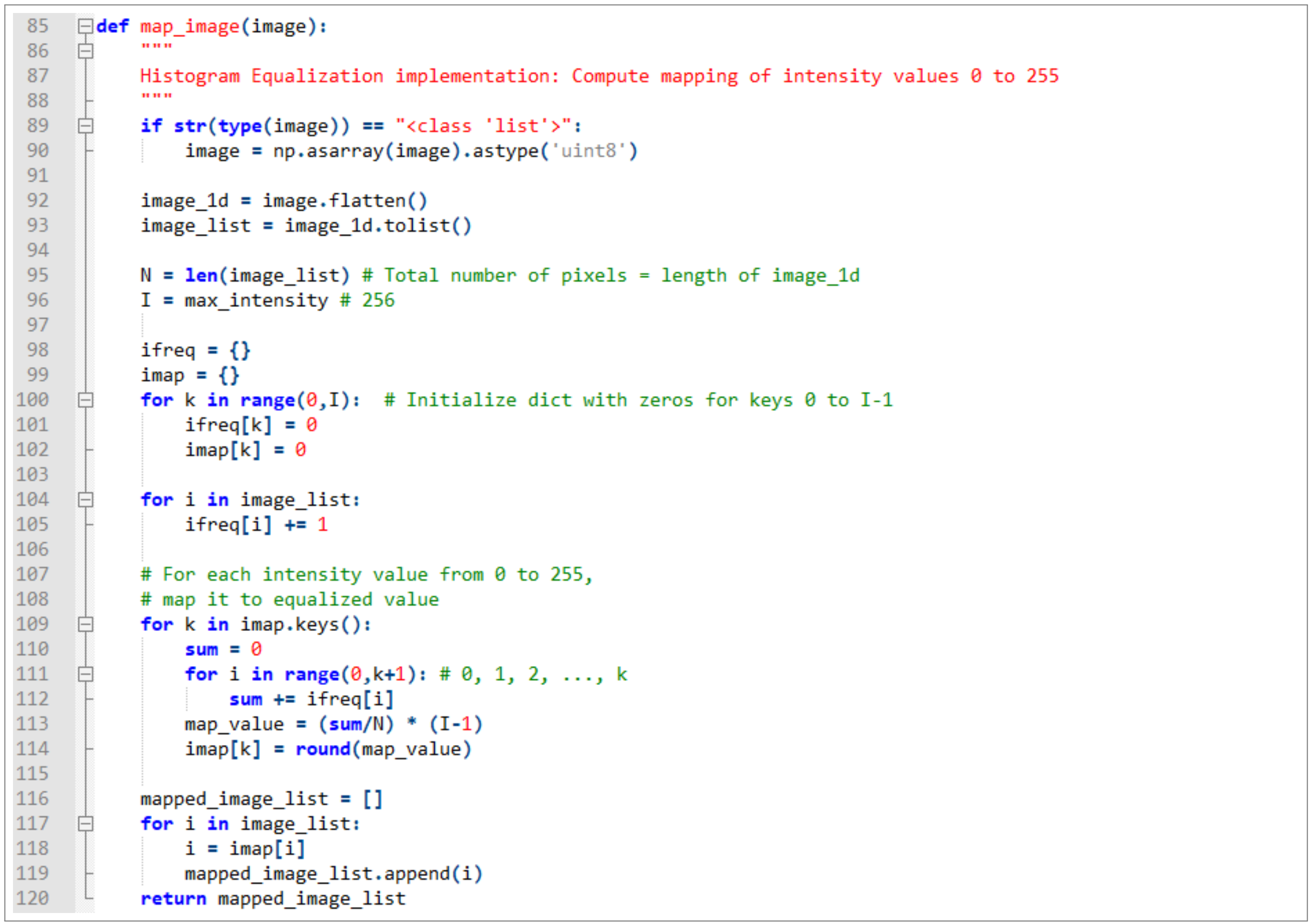
Scale this function to $[0, 255]$ and round the result to the nearest integer to obtain the intensity mapping for $k$:

$$f(k) = c(k) \times (I - 1),$$

where $I = 256$. Each original intensity value will be mapped to a new intensity value.

## 2.2 Implementation

Our implementation of the HE algorithm is shown below. We pass the image (a Numpy array) to the function `map_image`.



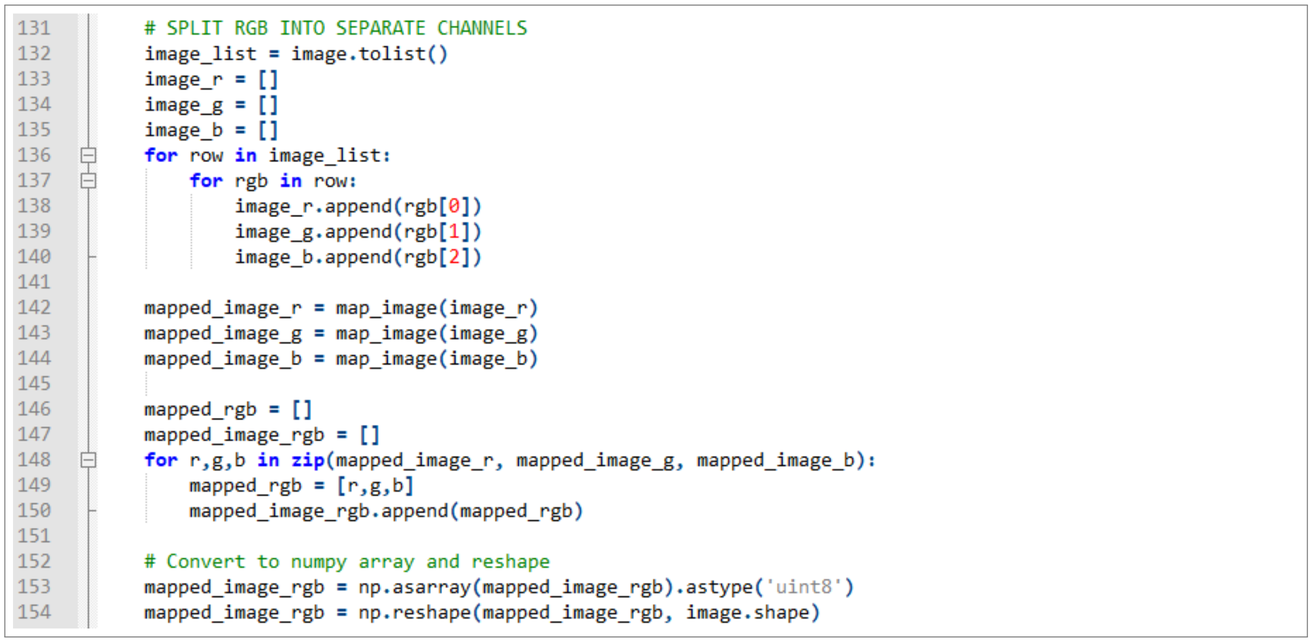
```python
def map_image(image):
    """
    Histogram Equalization implementation: Compute mapping of intensity values 0 to 255
    """
    if str(type(image)) == "<class 'list'>":
        image = np.asarray(image).astype('uint8')

    image_1d = image.flatten()
    image_list = image_1d.tolist()

    N = len(image_list) # Total number of pixels = length of image_1d
    I = max_intensity # 256

    ifreq = {}
    imap = {}
    for k in range(0,I):  # Initialize dict with zeros for keys 0 to I-1
        ifreq[k] = 0
        imap[k] = 0

    for i in image_list:
        ifreq[i] += 1

    # For each intensity value from 0 to 255,
    # map it to equalized value
    for k in imap.keys():
        sum = 0
        for i in range(0,k+1): # 0, 1, 2, ..., k
            sum += ifreq[i]
        map_value = (sum/N) * (I-1)
        imap[k] = round(map_value)

    mapped_image_list = []
    for i in image_list:
        i = imap[i]
        mapped_image_list.append(i)
    return mapped_image_list
```

## 2.3 HE Applied Across All Color Channels (RGB Color Space)

In RGB color space, we can apply the HE algorithm globally across all three color channels red, green and blue. To implement this, we simply pass the entire image to the function `map_image`.

## 2.4 HE Applied Individually to Each Color Channel (RGB Color Space)

In RGB color space, we can also apply the HE algorithm separately to each of the three color channels. To implement this, we pass the each color channel to the function `map_image` and combine the mapped channels to form the complete mapped image. Our implementation is as follows:



```python
    # SPLIT RGB INTO SEPARATE CHANNELS
    image_list = image.tolist()
    image_r = []
    image_g = []
    image_b = []
    for row in image_list:
        for rgb in row:
            image_r.append(rgb[0])
            image_g.append(rgb[1])
            image_b.append(rgb[2])

    mapped_image_r = map_image(image_r)
    mapped_image_g = map_image(image_g)
    mapped_image_b = map_image(image_b)

    mapped_rgb = []
    mapped_image_rgb = []
    for r,g,b in zip(mapped_image_r, mapped_image_g, mapped_image_b):
        mapped_rgb = [r,g,b]
        mapped_image_rgb.append(mapped_rgb)

    # Convert to numpy array and reshape
    mapped_image_rgb = np.asarray(mapped_image_rgb).astype('uint8')
    mapped_image_rgb = np.reshape(mapped_image_rgb, image.shape)
```

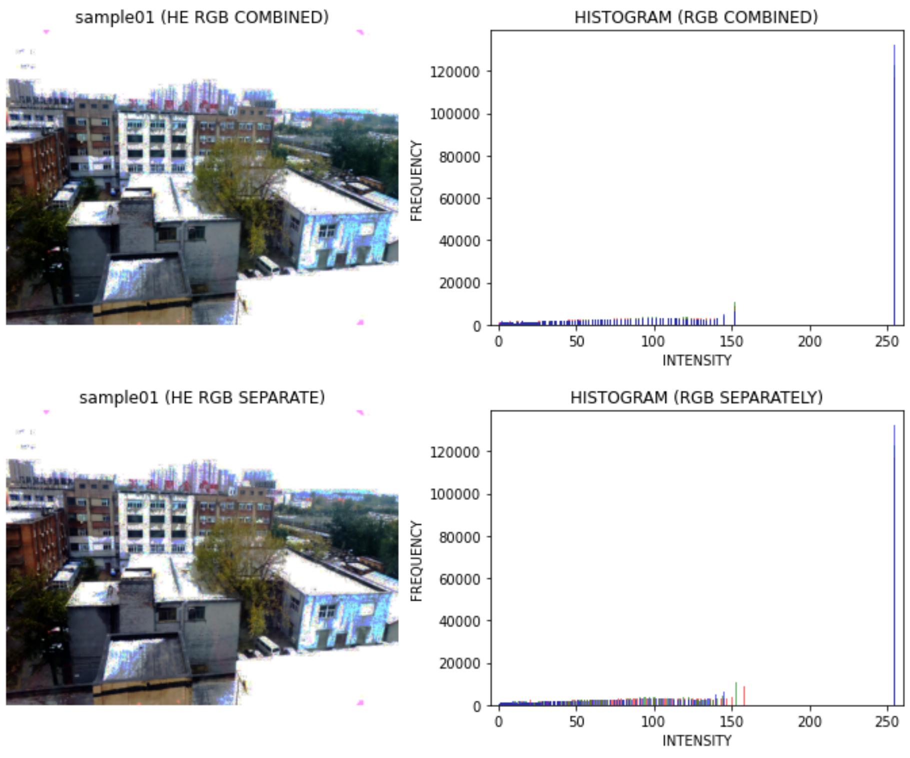


**Figure 1:** HE applied to RGB color space (sample01)

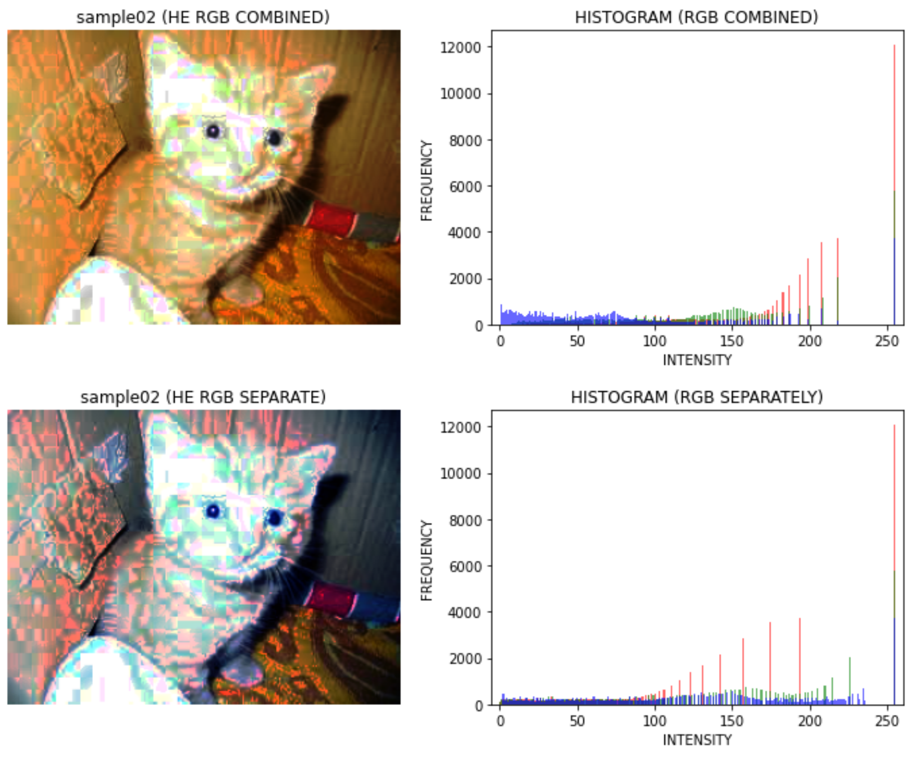


**Figure 2:** HE applied to RGB color space (sample02)

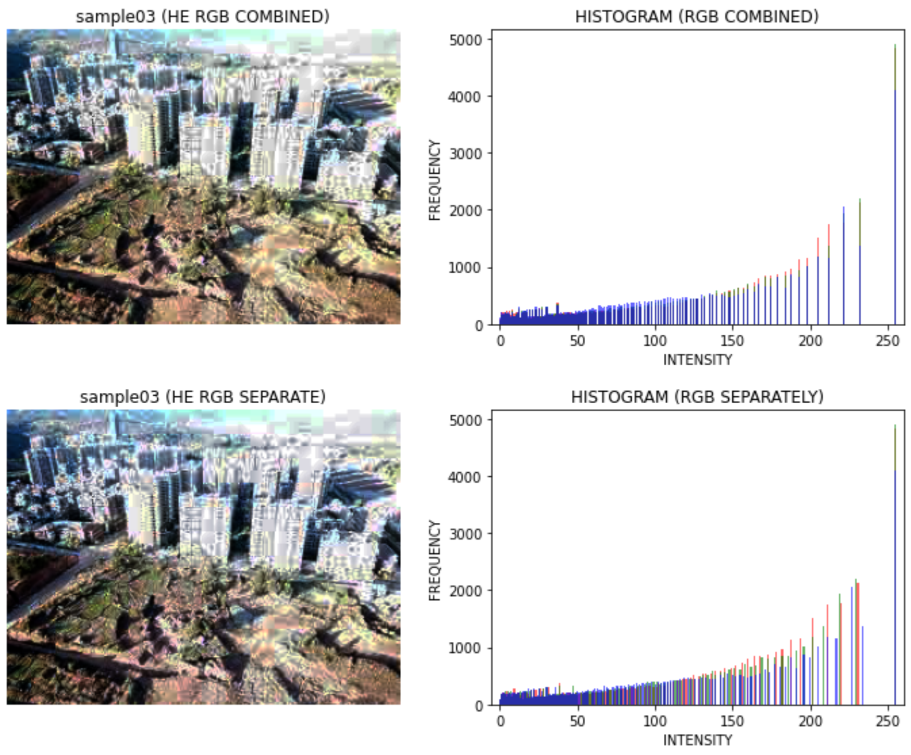


**Figure 3:** HE applied to RGB color space (sample03)

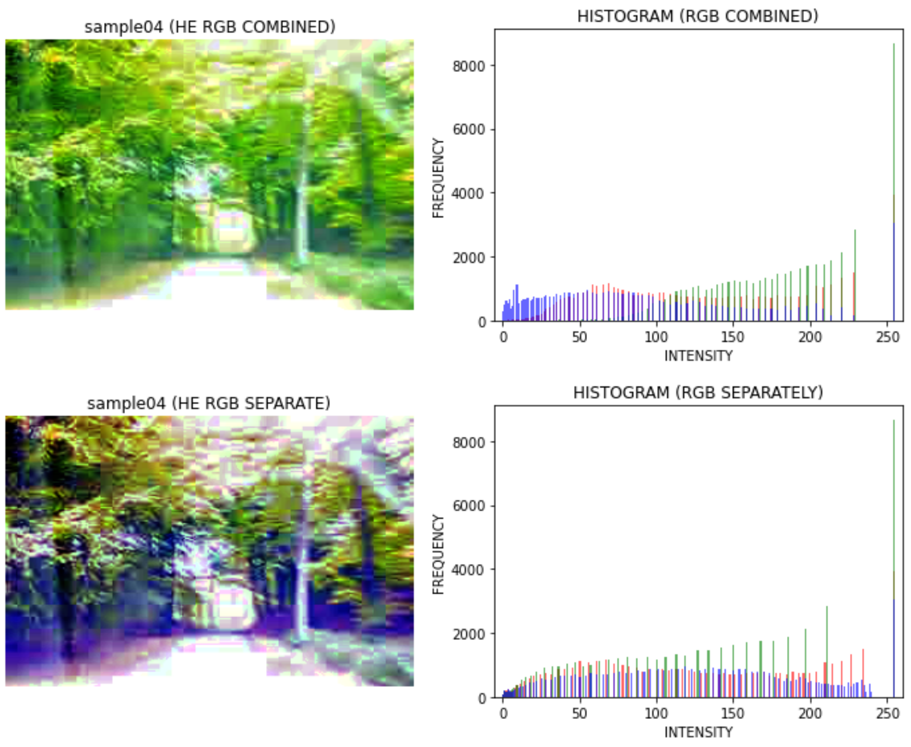


**Figure 4:** HE applied to RGB color space (sample04)

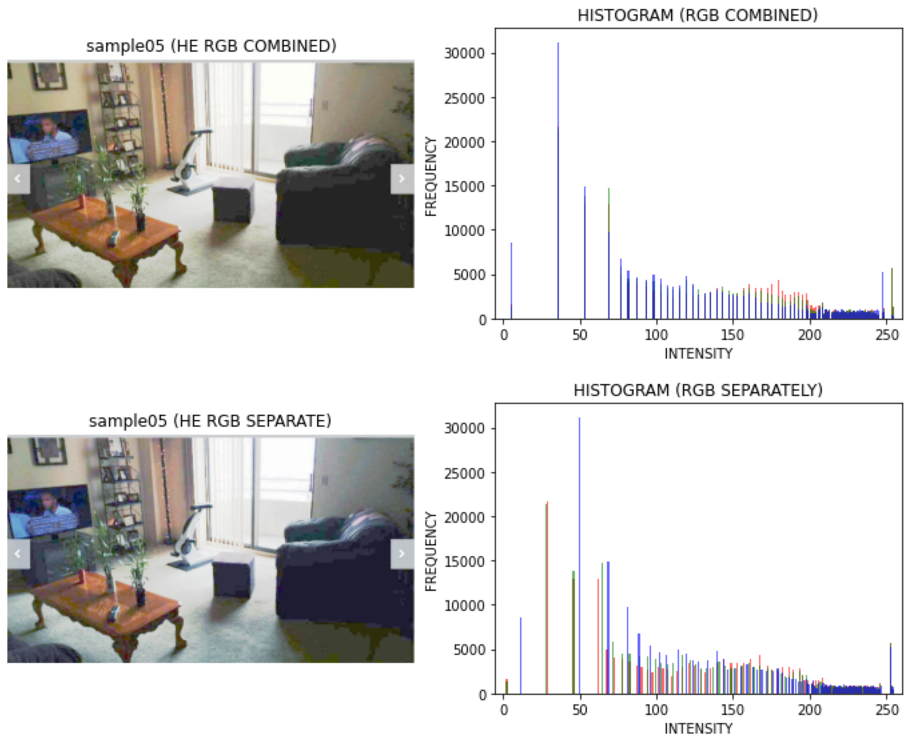


**Figure 5:** HE applied to RGB color space (sample05)

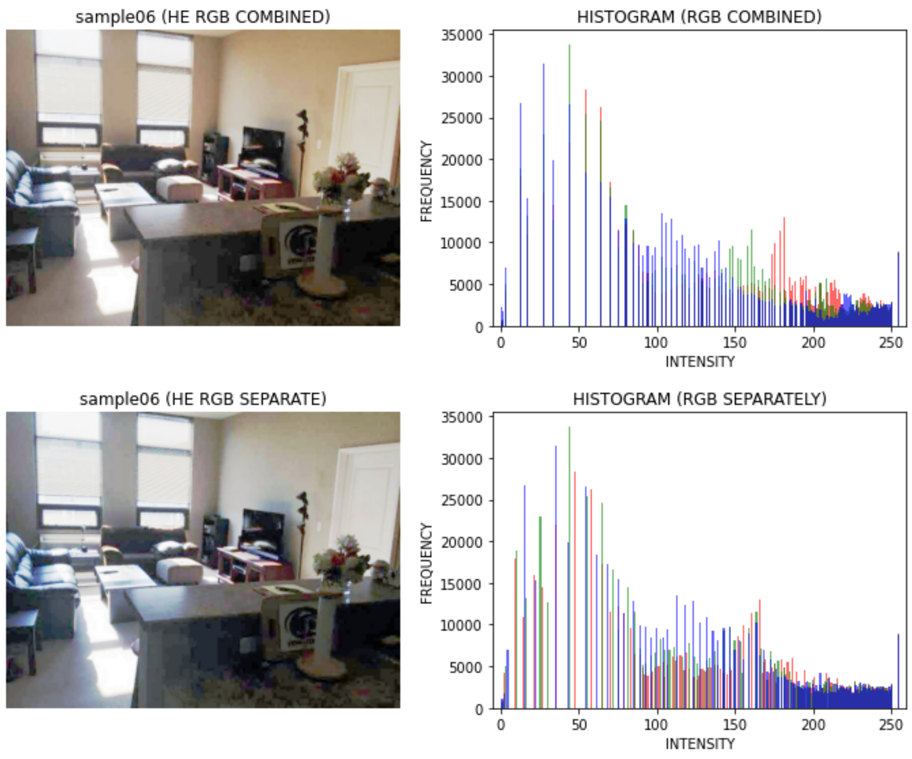


**Figure 6:** HE applied to RGB color space (sample06)

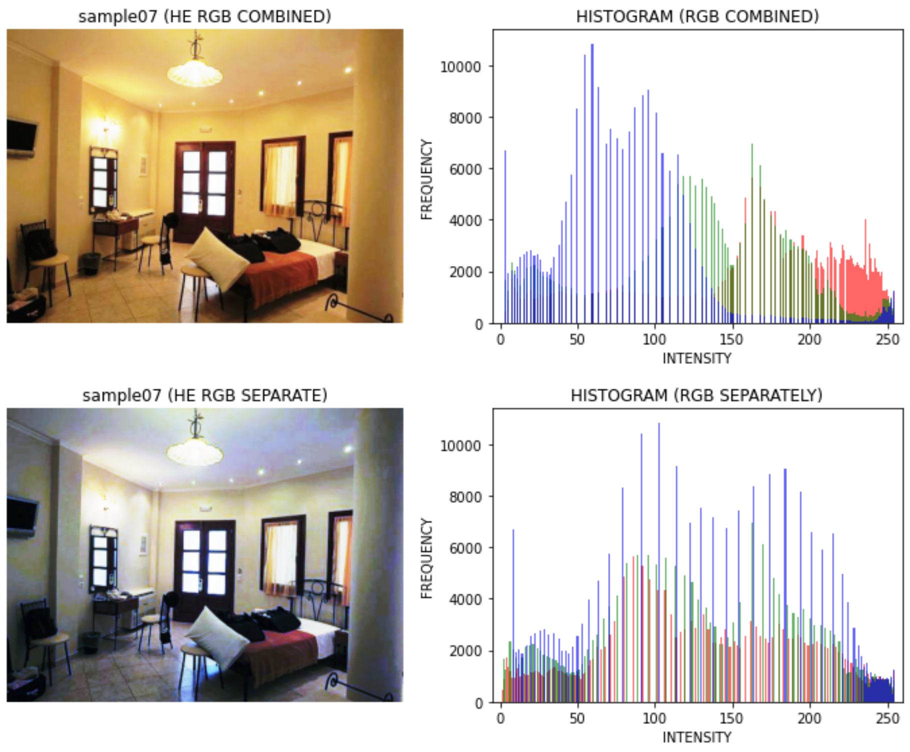


**Figure 7:** HE applied to RGB color space (sample07)

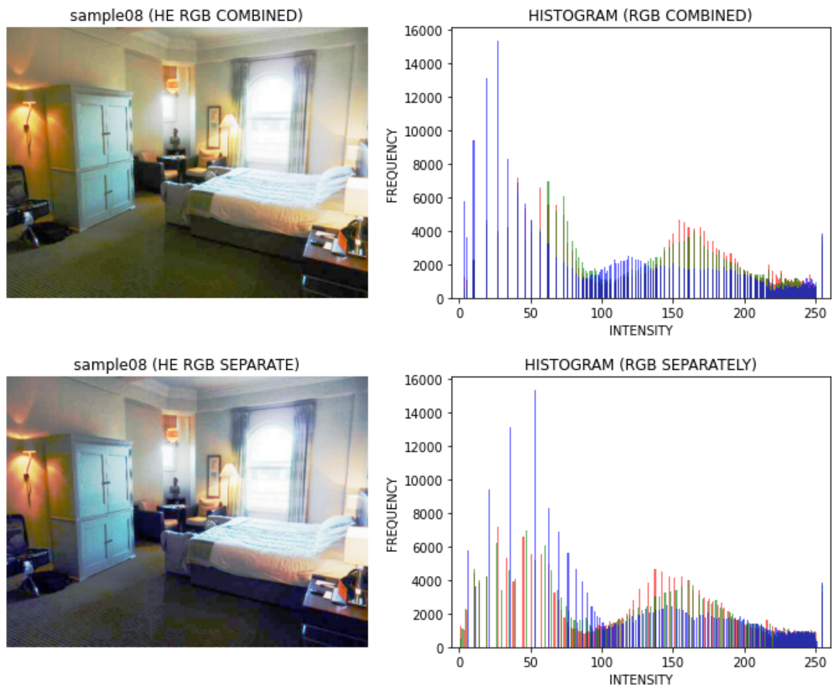


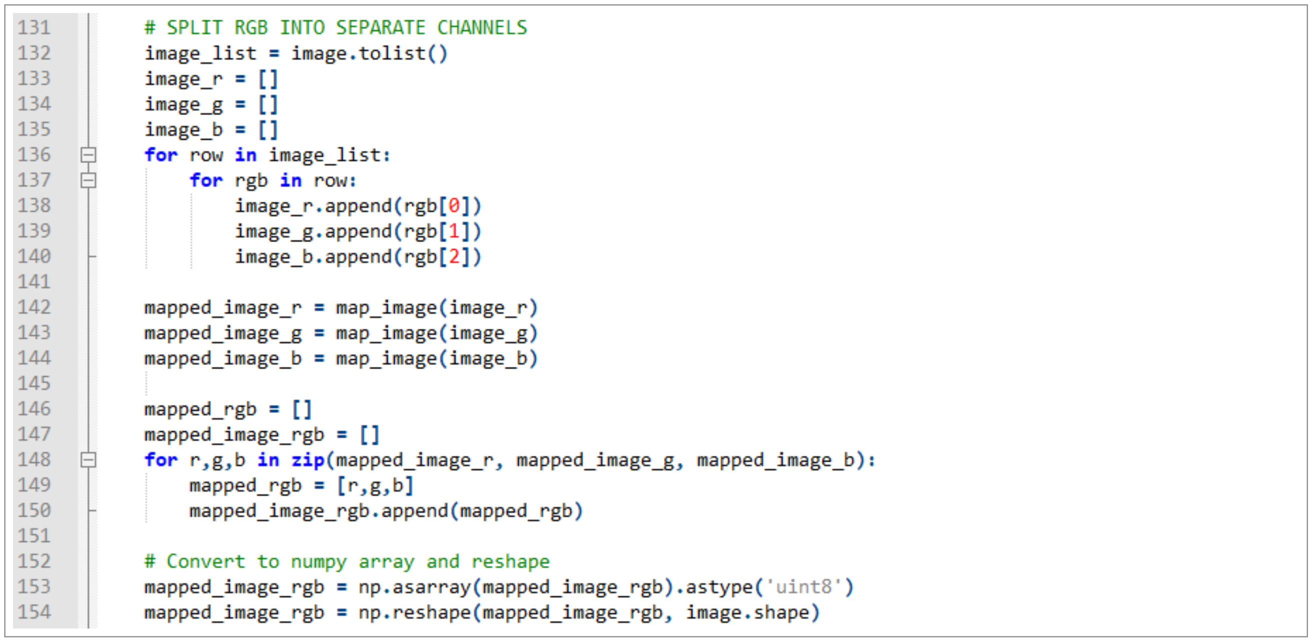
**Figure 8:** HE applied to RGB color space (sample08)

## 2.5 HE Applied to Value Channel (HSV Color Space)

Applying HE to RGB color channels is likely to produce mapped images that are very different from the original images. The color balance may be changed and the results may be overly dramatic.

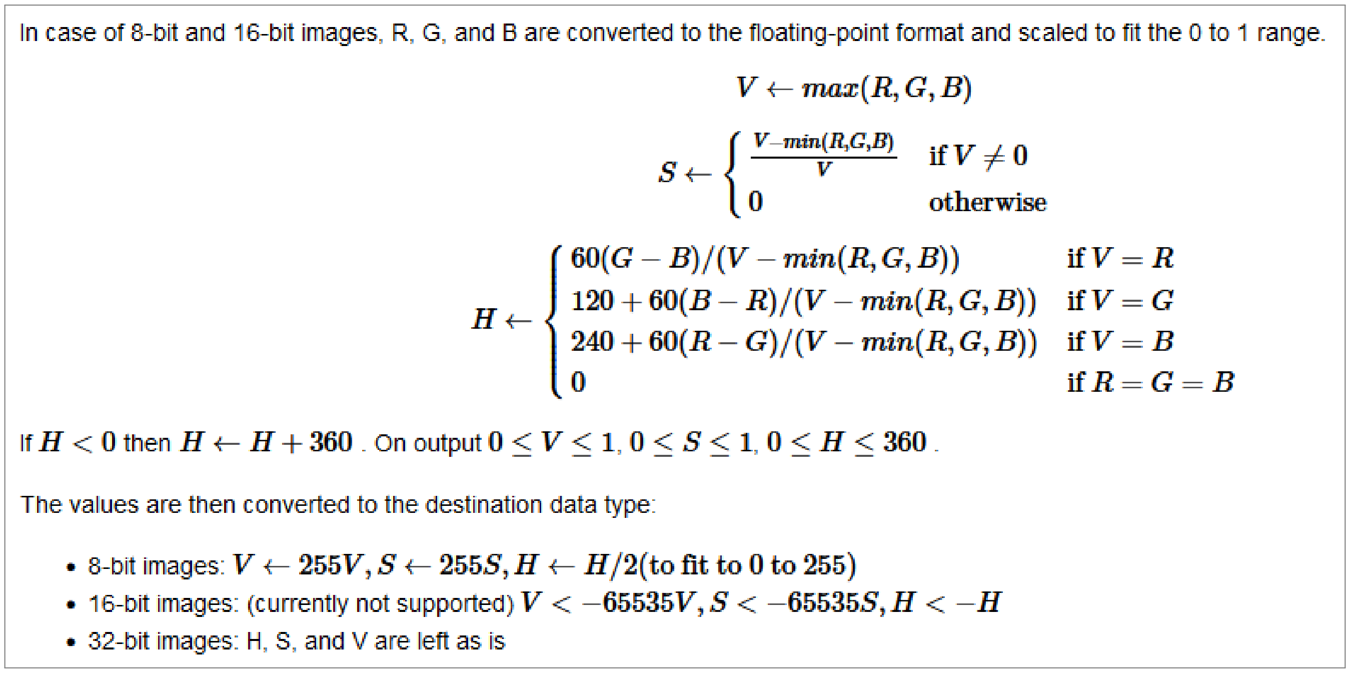
Instead of using the RGB color space, we can use the HSV (hue, saturation, value) color space and apply HE only to the *value* channel without any changes to *hue* and *saturation*. However, all red, green and blue channels may still be mapped to new values after converting back to RGB. The *value* channel in the HSV color space is equivalent to $max(R, G, B)$ in RGB color space.

In our implementation, we first convert the color space from RGB to HSV. In the HSV color space, we apply the HE algorithm to the value channel only. After that, we convert it back to the RGB color space. In the process, the RGB values may be mapped to new values. Our code is as follows:



```
    # SPLIT RGB INTO SEPARATE CHANNELS
    image_list = image.tolist()
    image_r = []
    image_g = []
    image_b = []
    for row in image_list:
        for rgb in row:
            image_r.append(rgb[0])
            image_g.append(rgb[1])
            image_b.append(rgb[2])

    mapped_image_r = map_image(image_r)
    mapped_image_g = map_image(image_g)
    mapped_image_b = map_image(image_b)

    mapped_rgb = []
    mapped_image_rgb = []
    for r,g,b in zip(mapped_image_r, mapped_image_g, mapped_image_b):
        mapped_rgb = [r,g,b]
        mapped_image_rgb.append(mapped_rgb)

    # Convert to numpy array and reshape
    mapped_image_rgb = np.asarray(mapped_image_rgb).astype('uint8')
    mapped_image_rgb = np.reshape(mapped_image_rgb, image.shape)
```

For conversion of color space, we use the OpenCV library. In OpenCV, RGB is converted to HSV as shown below in its documentation[1]:



In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V \leftarrow max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V - min(R,G,B)) & \text{if } V = R \\ 120 + 60(B-R)/(V - min(R,G,B)) & \text{if } V = G \\ 240 + 60(R-G)/(V - min(R,G,B)) & \text{if } V = B \\ 0 & \text{if } R = G = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$ . On output $0 \leq V \leq 1$, $0 \leq S \leq 1$, $0 \leq H \leq 360$ .

The values are then converted to the destination data type:

- 8-bit images: $V \leftarrow 255V, S \leftarrow 255S, H \leftarrow H/2 (\text{to fit to 0 to 255})$
- 16-bit images: (currently not supported) $V < -65535V, S < -65535S, H < -H$
- 32-bit images: H, S, and V are left as is

[1] https://docs.opencv.org/master/de/d25/imgproc_color_conversions.html

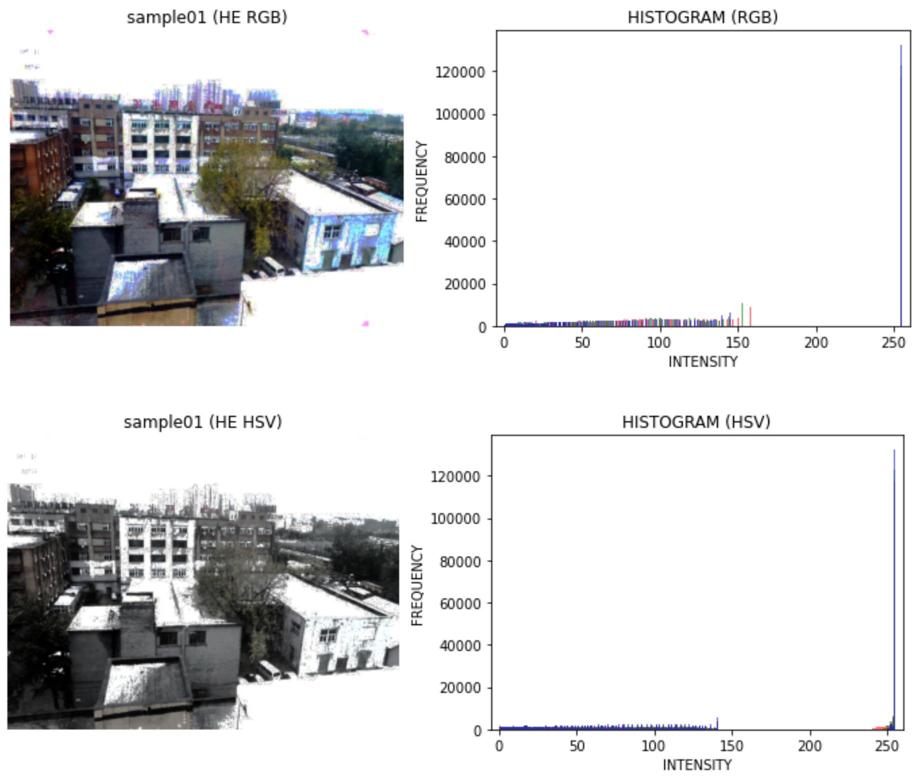


**Figure 9:** HE applied to RGB and HSV color spaces (sample01)

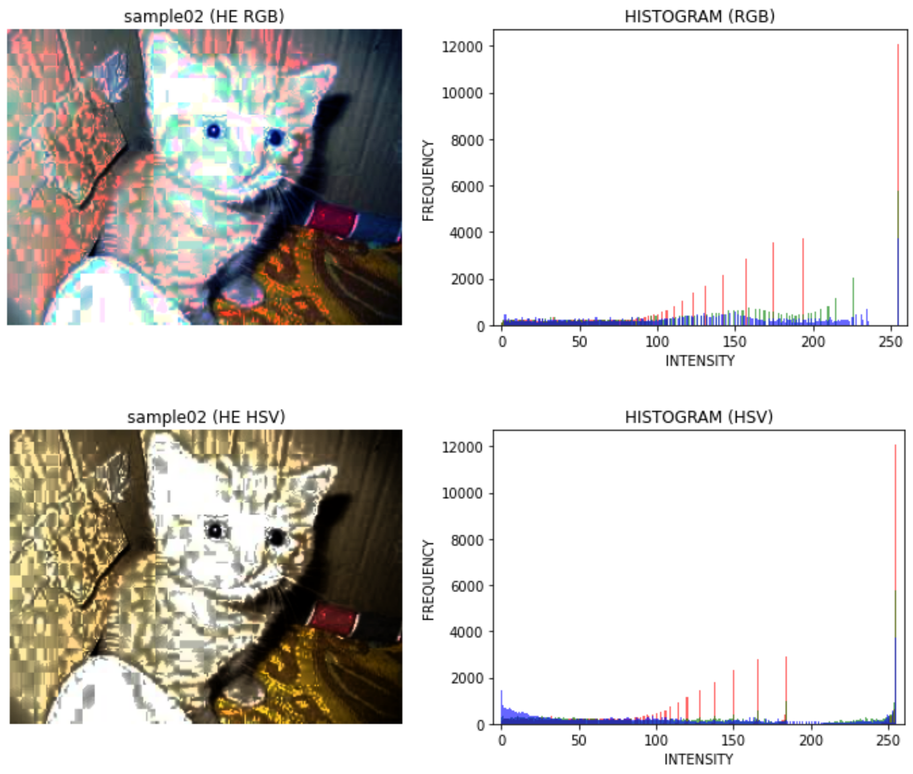


**Figure 10:** HE applied to RGB and HSV color spaces (sample02)

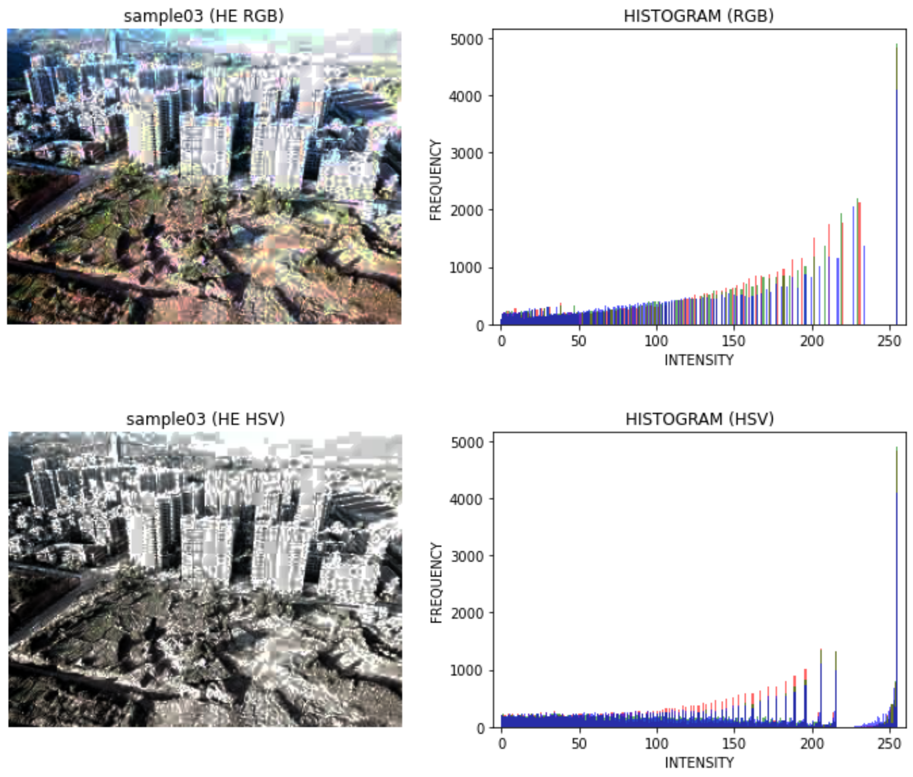


**Figure 11:** HE applied to RGB and HSV color spaces (sample03)

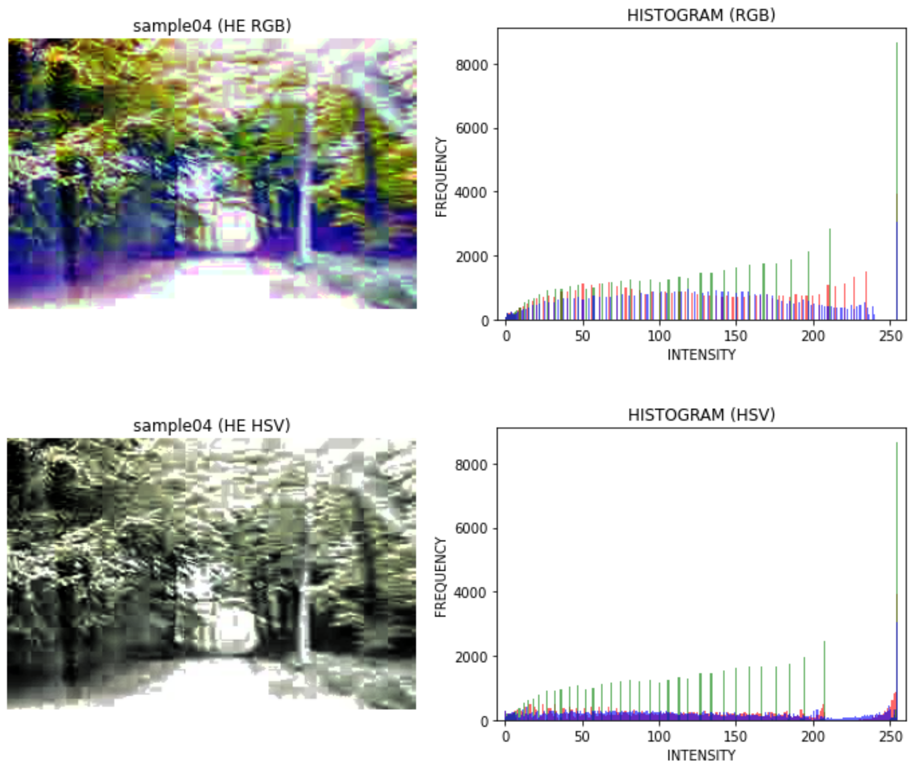


**Figure 12:** HE applied to RGB and HSV color spaces (sample04)

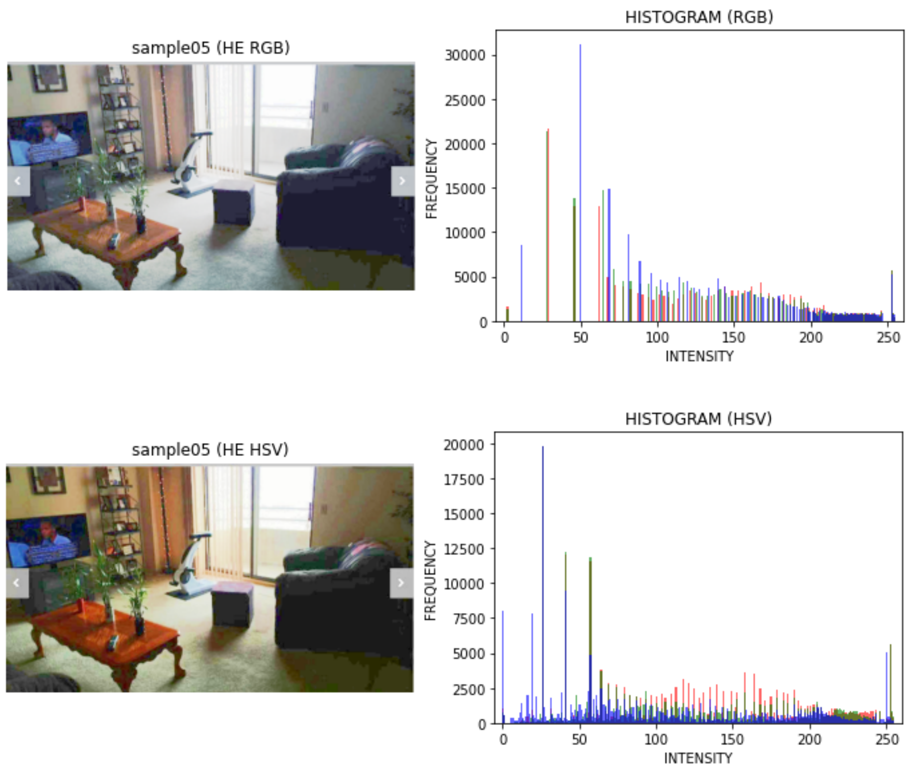


**Figure 13:** HE applied to RGB and HSV color spaces (sample05)

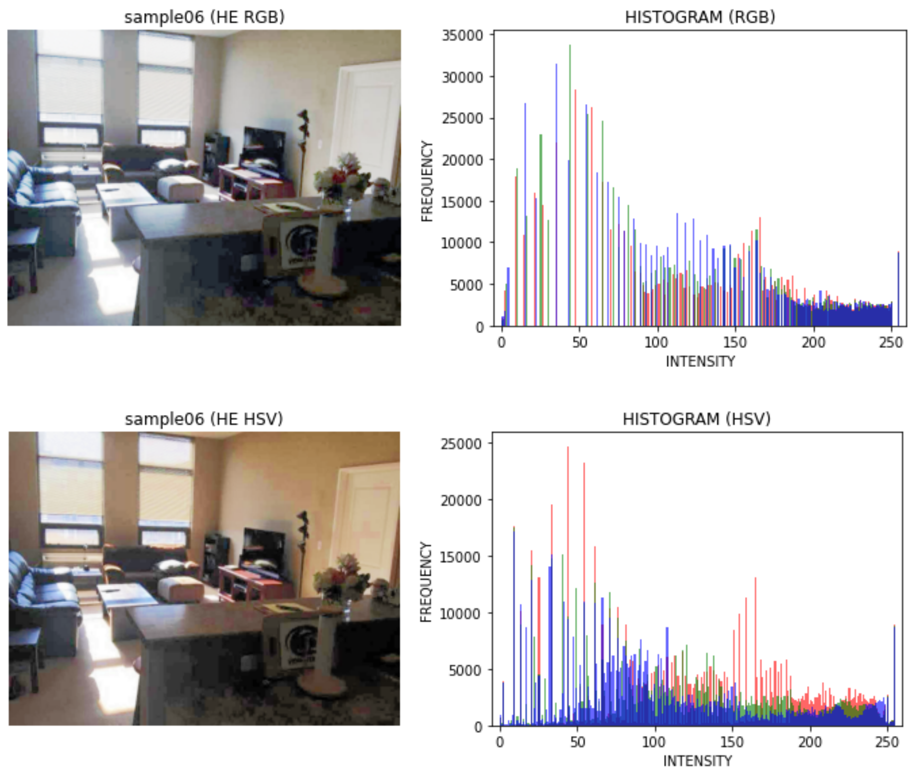


**Figure 14:** HE applied to RGB and HSV color spaces (sample06)

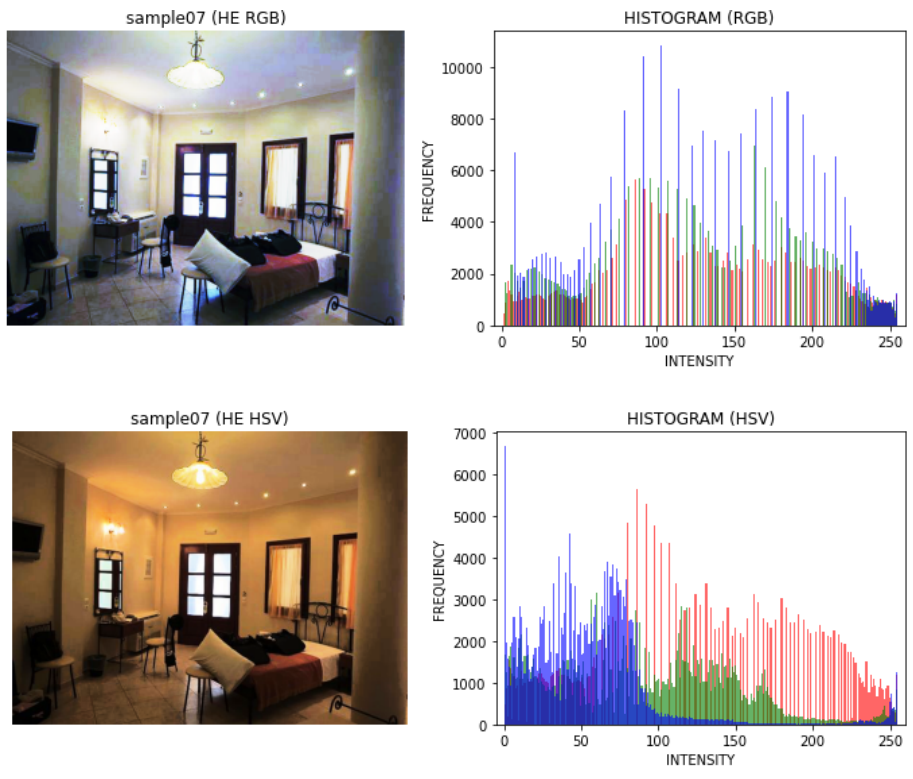


**Figure 15:** HE applied to RGB and HSV color spaces (sample07)

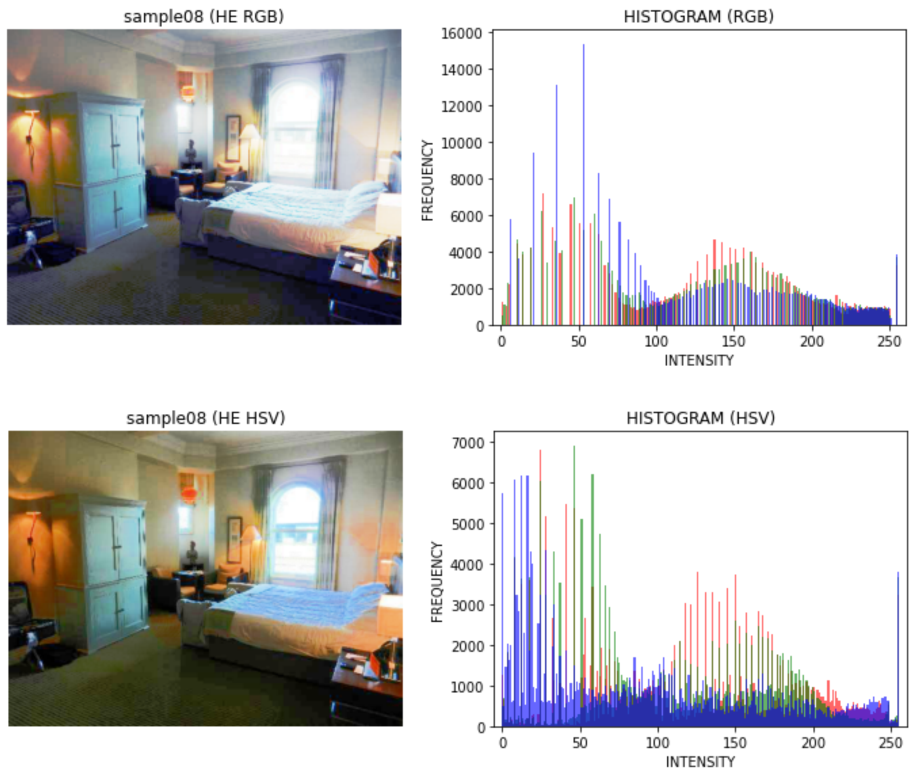


**Figure 16:** HE applied to RGB and HSV color spaces (sample08)

## 3 Comparison and Discussion of Results (Task 2)

### 3.1 RGB Color Space

The results for the eight sample images with HE applied to RGB color space (across all channels and separately to each channel) are shown in Figures 1, 2, 3, 4, 5, 6, 7 and 8.

From the resulting images, we see that the results are quite similar when the original images contain dull colors (samples 01, 03, 05, 06, 08). Samples 02, 04 and 06 have more vibrant colors (yellow, green and orange respectively) and we see a significant difference between the two methods. Both methods appear produce much better results for darker images (samples 05, 06, 07, 08) compared to the first four sample images. For samples 02 and 04, applying HE globally appears to produce better results. For these two sample images, the original colors are significantly changed when HE is applied separately to each color channel.

Overall, applying HE to RGB channels changes the histogram distributions significantly, resulting in changes in colors and color balance. This method is useful if we want to darken light images or lighten dark images significantly.

### 3.2 RGB and HSV Color Spaces

We now compare the mapped images and histograms for HE applied to RGB (separately to each channel) and HSV. The results are shown in Figures 9, 10, 11, 12, 13, 14, 15 and 16.

For the lighter images 01, 03 and 04, HE applied to HSV produced images with more greyscale colors. The colors are less vibrant. However, for darker images (samples 05, 06, 07, 08), the opposite is true. HE applied to HSV produced images with significantly more vibrant colors compared to HE applied to RGB. As before, the results for darker sample images are much better than for the lighter images.

The distribution of the histograms are spread more evenly across the entire range. The result of this is a significant change in the colors, especially of darker images. This method is useful if we want more vibrant colors in darker images.

## 4 Possible Improvements (Task 3)

### 4.1 Modifying the Mapping Function

To improve the quality of mapped images, we can use a mapping function of the following form [1]:

$$f(k) = \alpha c(k) + (1 - \alpha)k, \text{ where } \alpha \text{ is a constant.}$$

This function is a weighted linear combination of the cumulative distribution function and a linear function, resulting in *partial histogram equalization* which will maintain more of its original distribution. We will get less dramatic changes in color observed in our implementations.

### 4.2 Adaptive Histogram Equalization

For images in which there is a wide range of luminance values (i.e., having extremely light and dark regions), we can obtain better results by dividing the image into $M \times M$ rectangular blocks and performing HE separately in each block. These blocks may or may not overlap one another. Non-overlapping blocks are more efficient but the resulting image may have artifacts at the block boundaries.

In Adaptive Histogram Equalization (AHE), we eliminate such artifacts by interpolating with mapping functions from neighboring blocks. To illustrate, in Figure 17[2], the image is partitioned into nine blocks. We form a mapping function by combining the four mapping functions $f_1$, $f_2$, $f_3$, and $f_4$ for four adjacent blocks, each weighted by distances $s$ and $t$ from the pixel to the center of each block [1]:

$$f_{s,t}(k) = (1-s)(1-t)f_1(k) + s(1-t)f_2(k) + (1-s)tf_3(k) + stf_4(k)$$

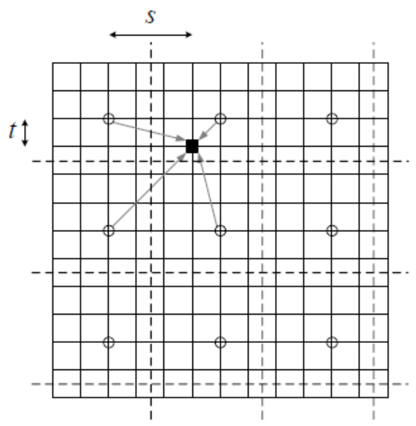
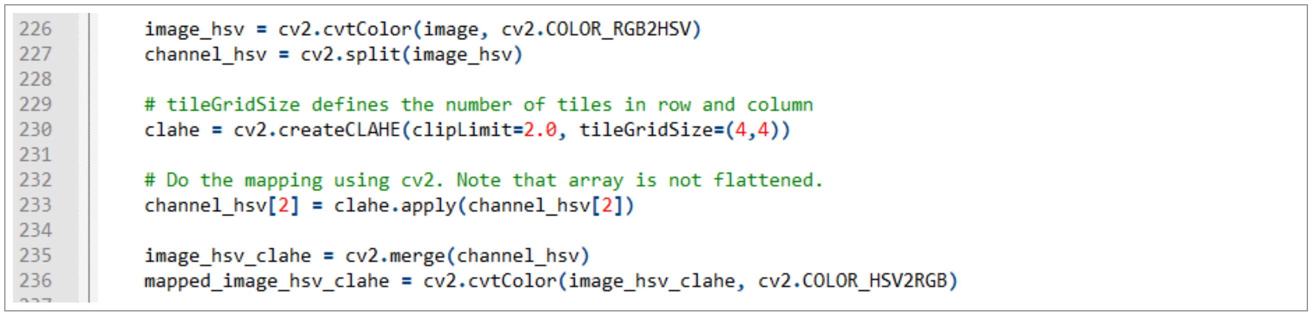


**Figure 17:** Adaptive Histogram Equalization: Interpolation

[2]Richard Szeliski Richard. Computer Vision: Algorithms and Applications, 2010. https://szeliski.org/Book/

## 4.3 CLAHE

A version of AHE is known as CLAHE (Contrast Limited Adaptive Histogram Equalization), so named because noise amplification is limited (i.e., reduced) in regions of near-constant intensity. We applied CLAHE (from the OpenCV library) with HSV to the eight sample images using 16 blocks (called *tiles* in OpenCV) in each image. As before, CLAHE is applied only to the *value* channel. Our code is as follows:



```
    image_hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    channel_hsv = cv2.split(image_hsv)

    # tileGridSize defines the number of tiles in row and column
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(4,4))

    # Do the mapping using cv2. Note that array is not flattened.
    channel_hsv[2] = clahe.apply(channel_hsv[2])

    image_hsv_clahe = cv2.merge(channel_hsv)
    mapped_image_hsv_clahe = cv2.cvtColor(image_hsv_clahe, cv2.COLOR_HSV2RGB)
```

## 4.4 Comparison with Earlier Results

We now present the results from the three methods along with the original images: RGB (each channel equalized individually), HSV, CLAHE with HSV. Figures 18, 19, 20, 21, 22, 23, 24 and 25 show the results.

For lighter sample images 01, 02, 03 and 04, results from CLAHE are clearly much better than our earlier results. The changes from original images are much less dramatic and the color balance is very close to that of the original images.

For darker images 05, 06, 07 and 08, the results are again less dramatic. For images 05 and 06, where there are regions with extreme lightness and darkness, the results are much better than earlier results. The colors are even and overall, the images are no longer unnaturally patchy.

The histograms for CLAHE show partial equalization and are more similar to the original distributions.

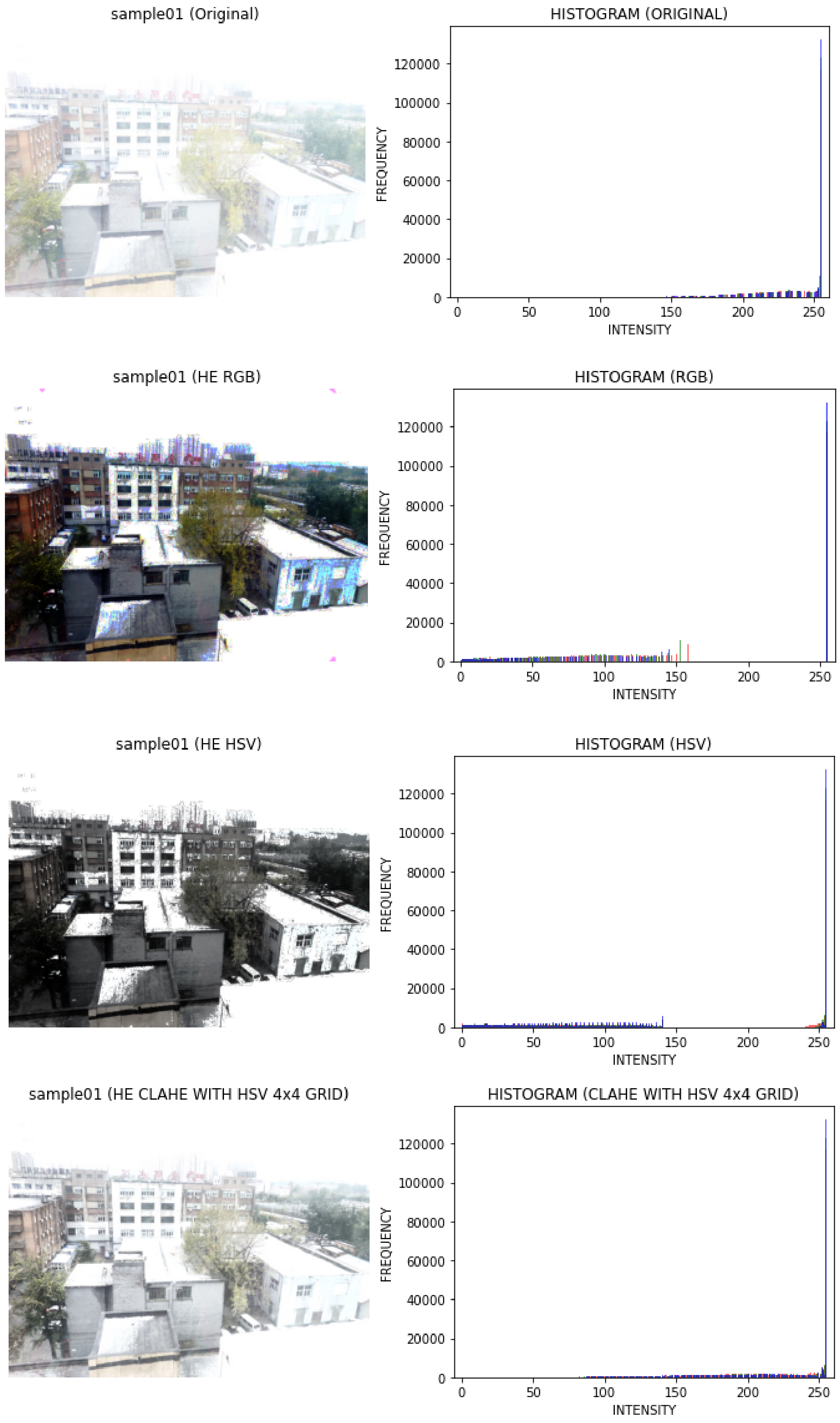


**Figure 18:** Sample01

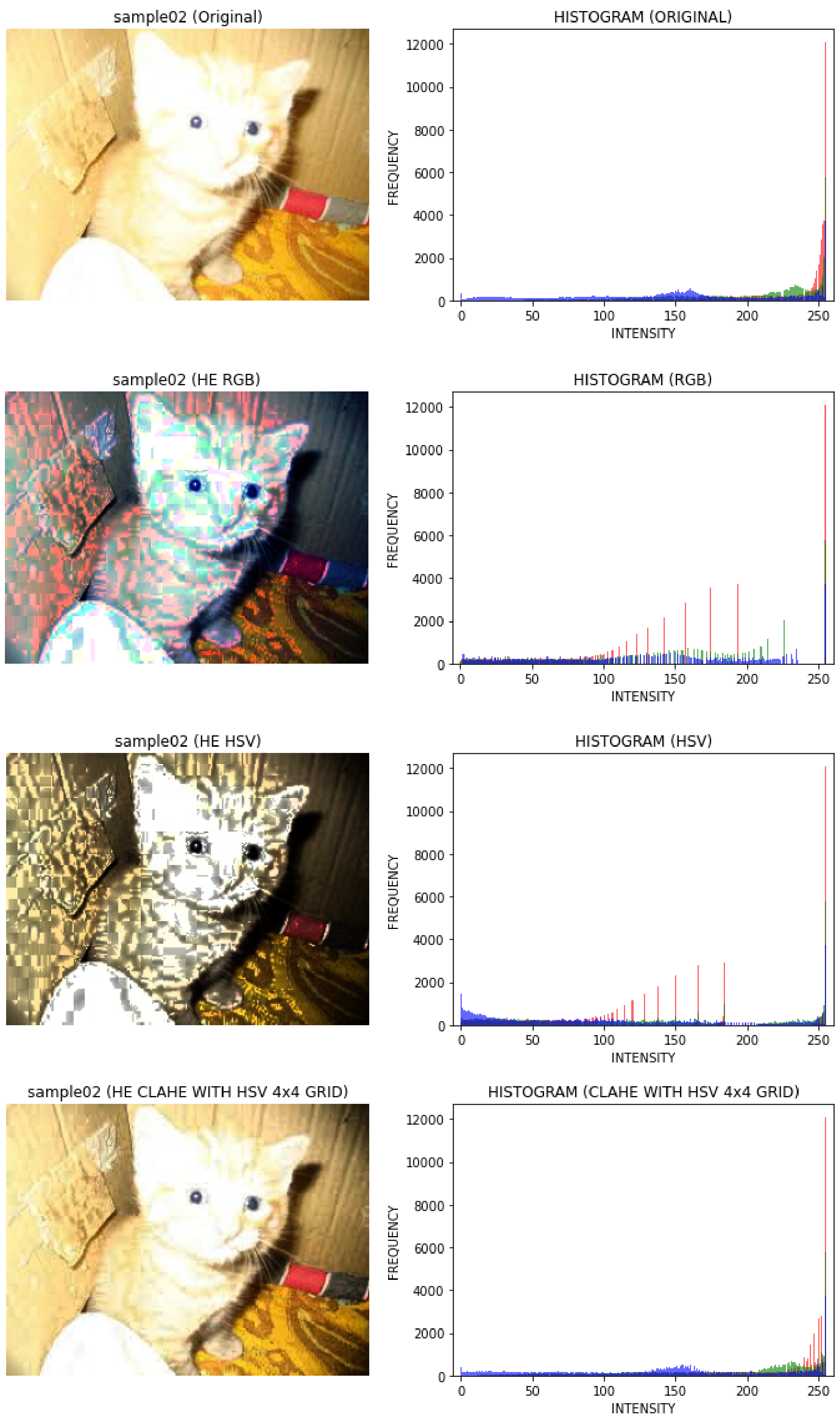


**Figure 19:** Sample02

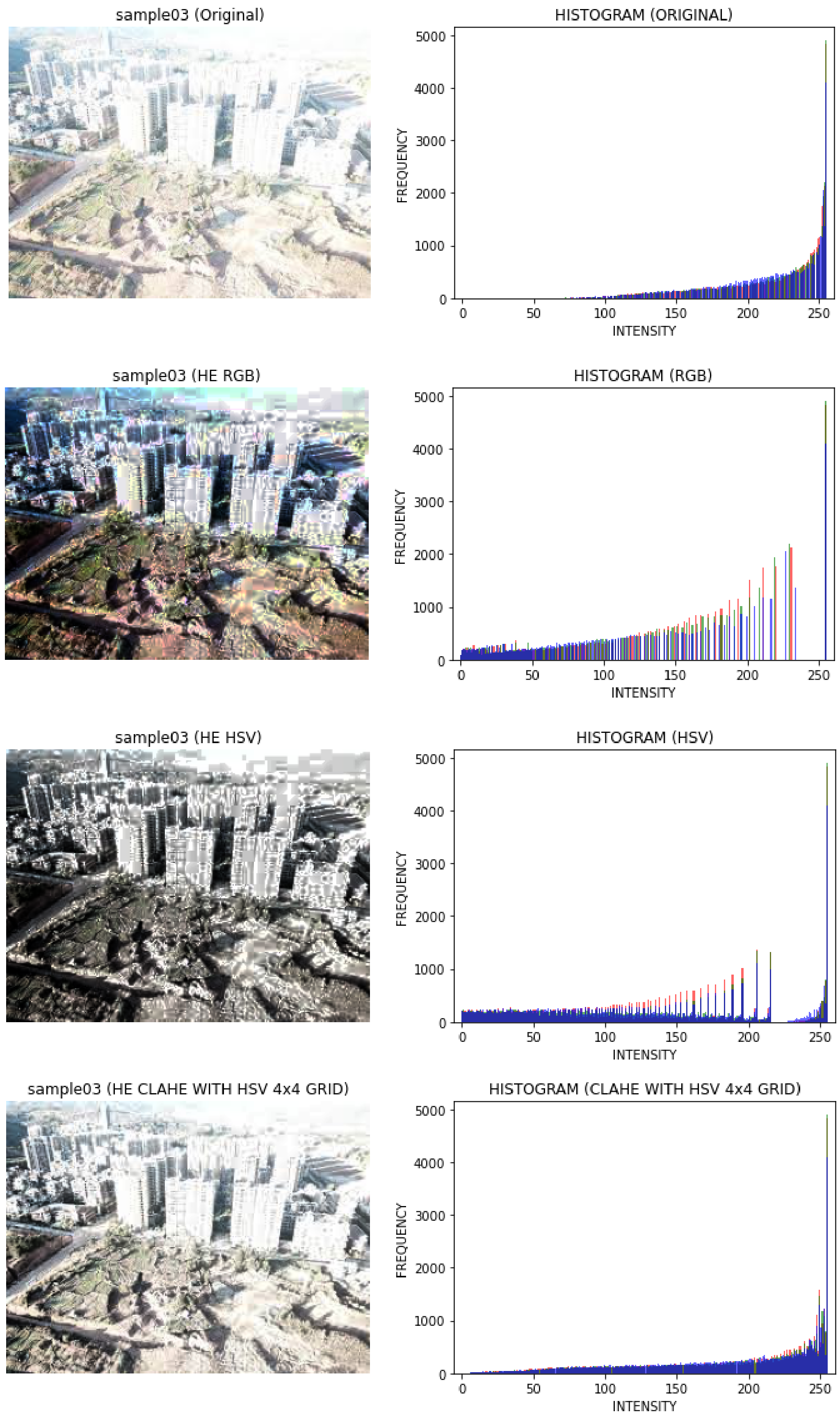


**Figure 20:** Sample03

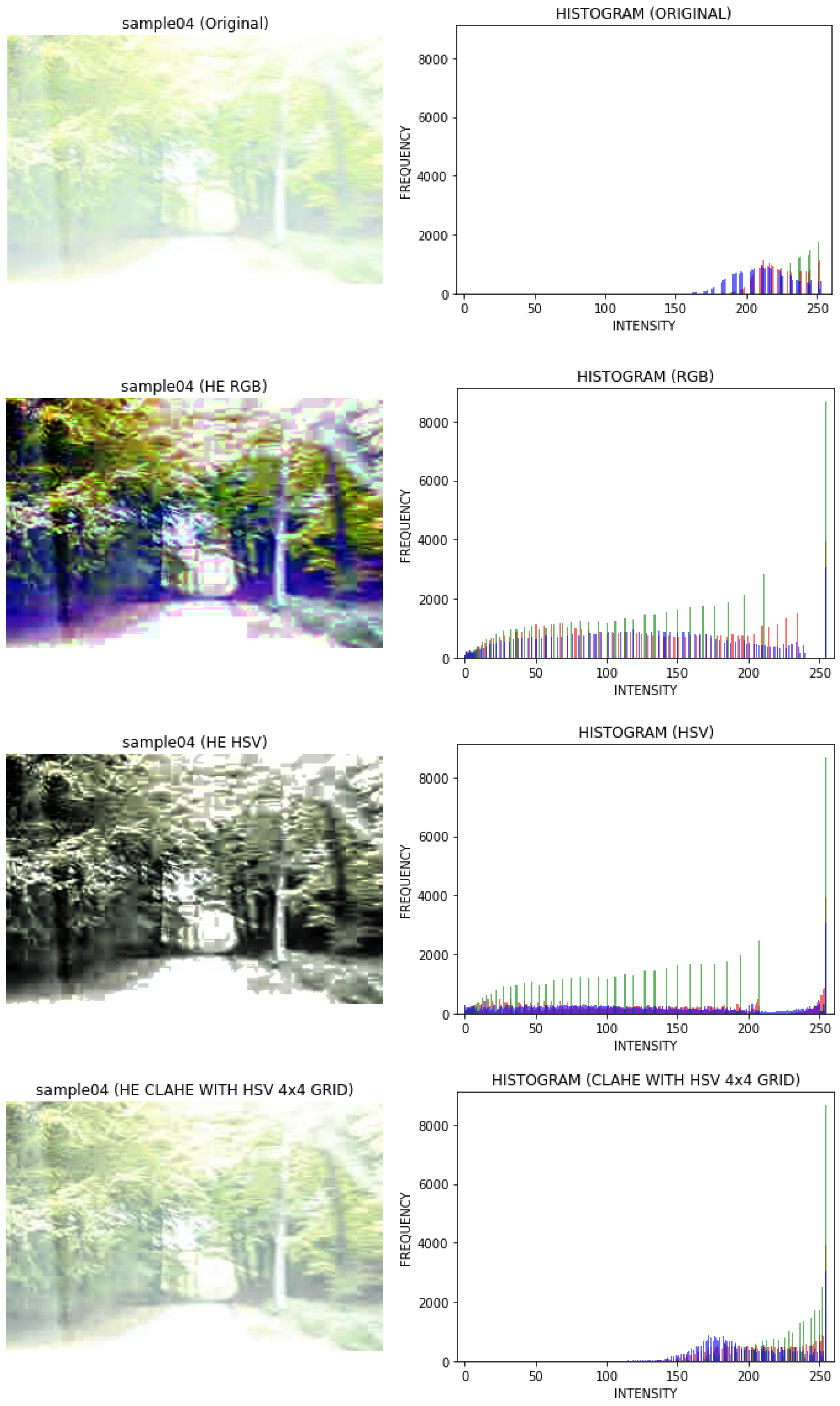


**Figure 21:** Sample04

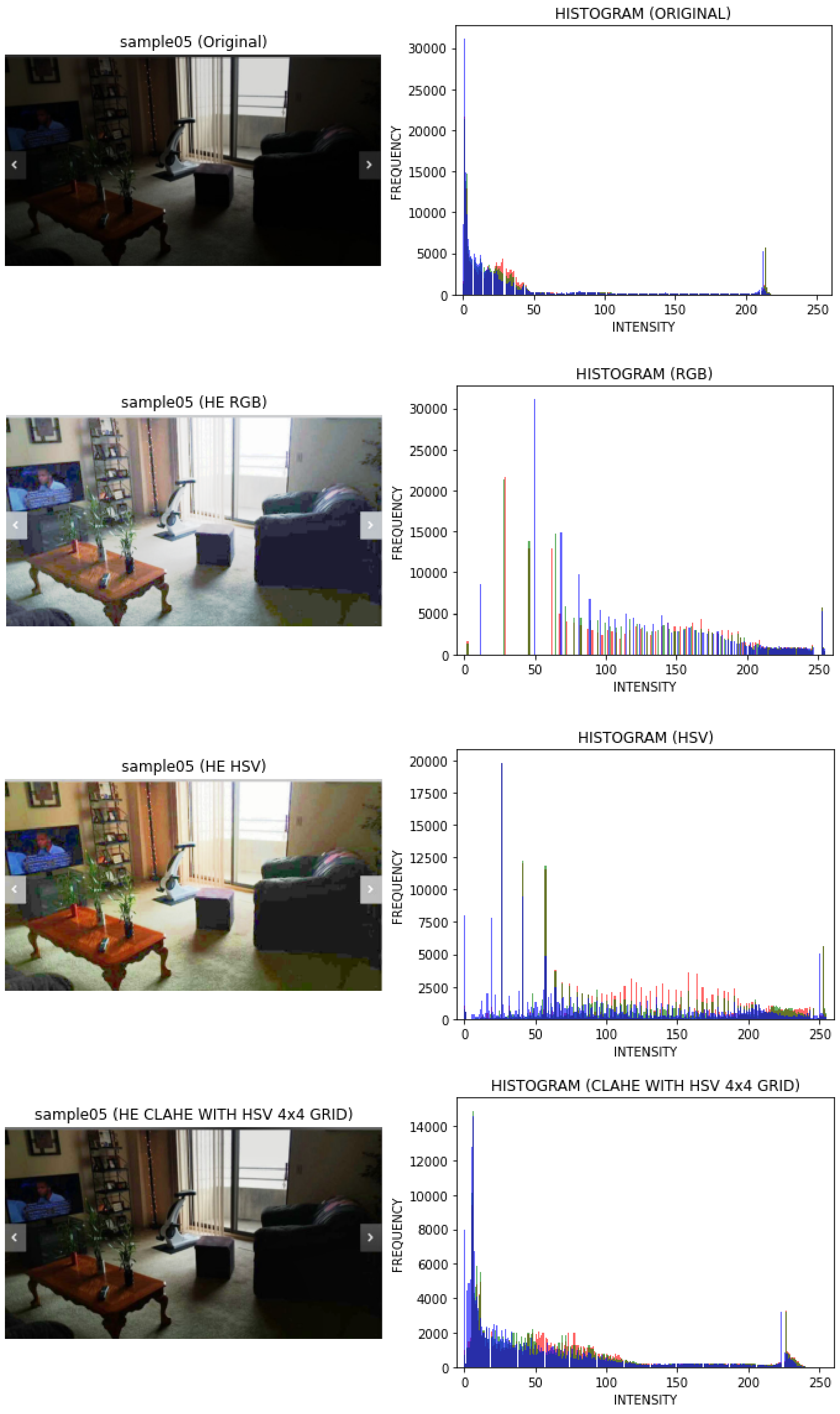


**Figure 22:** Sample05

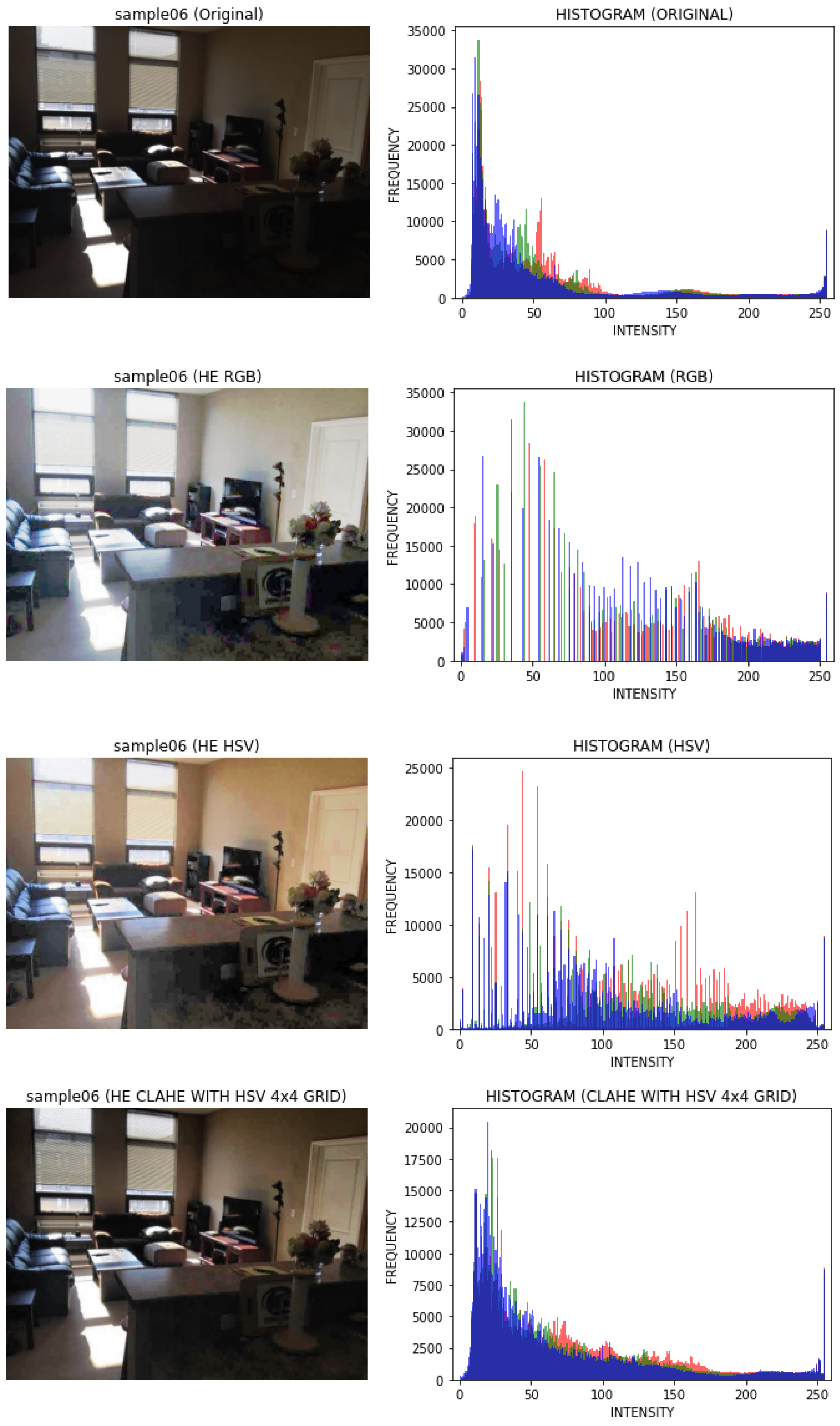


**Figure 23:** Sample06

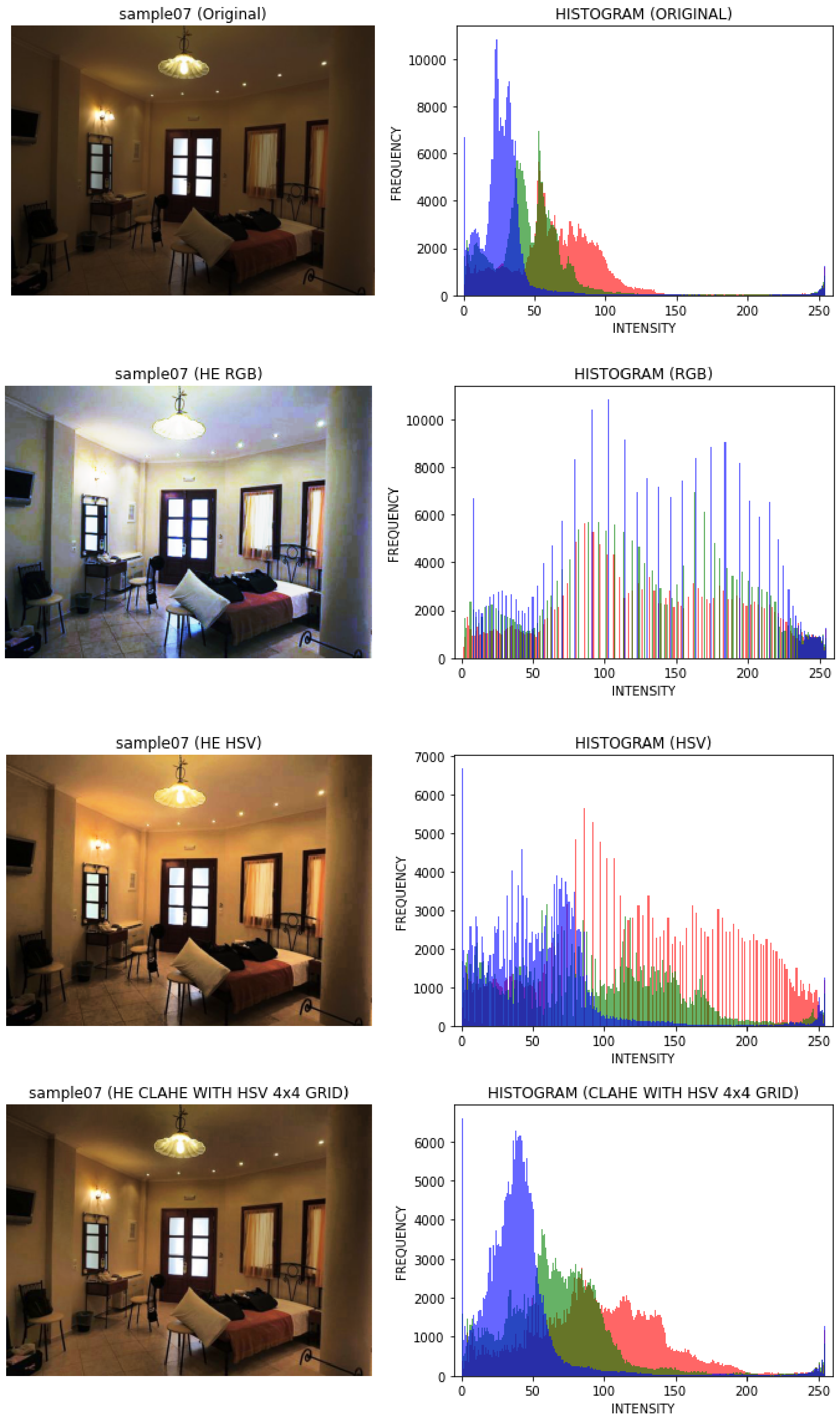


**Figure 24:** Sample07

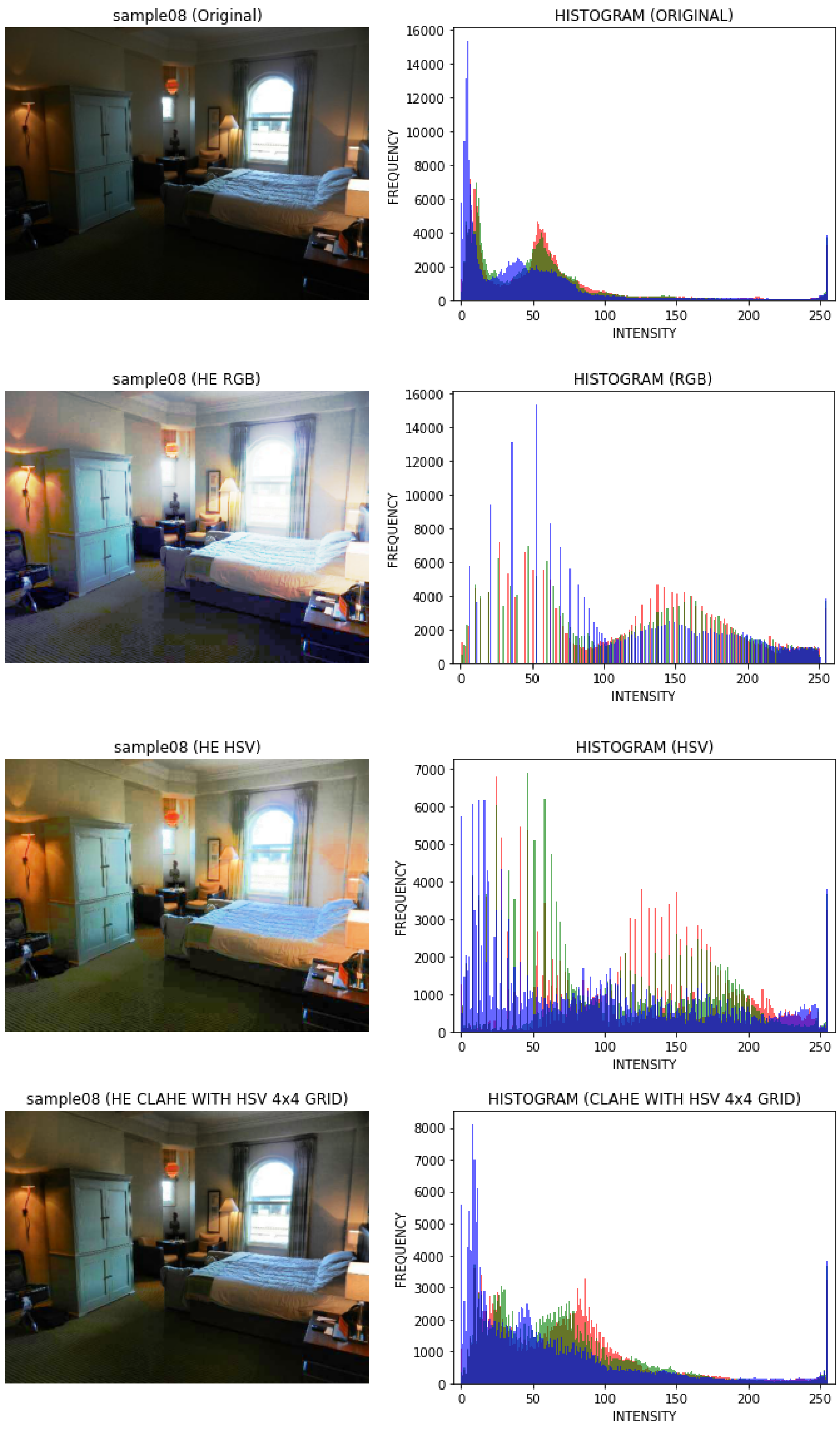


**Figure 25:** Sample08

## 5 Conclusions

We implemented the Histogram Equalization algorithm and applied the algorithm to channels in the RGB and HSV color spaces. We also used the OpenCV library to apply Contrast Limited Adaptive Histogram Equalization (CLAHE) to the eight sample images.

When applied to RGB color channels, the results are better for darker images than for lighter images. However, the color balance may be dramatically changed from the original color balance.

When applied to HSV, the resulting images contain more greyscale and less vibrant colors for lighter images. For darker images, HE applied to HSV produced images with significantly more vibrant colors compared to HE applied to RGB.

Applying HE to RGB channels is useful if we want to darken light images or lighten dark images. Applying HE to HSV is useful if we want more vibrant colors in darker images.

For lighter images, results from CLAHE are much superior than the results from our implementions. The changes from original images are much less dramatic and the color balance is close to the original color balance. For darker images, the colors are even and are no longer patchy. For CLAHE, the histograms show partial equalization and are more similar to the original distributions.

## References

[1] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010. 15