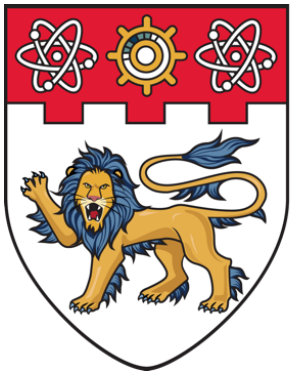

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

# AI6121 Computer Vision

## Assignment 1 Report


authored by

Ong Jia Hui
JONG119@e.ntu.edu.sg
G1903467L

**Abstract**

Histogram Equalization (HE) is an image processing technique that has been widely adopted to improve the contrast in images. It accomplishes this by spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. It usually increases the global contrast of images when the image pixels fall within a narrow range of intensity values. This allows for areas of lower local contrast to gain a higher contrast.

This assignment tasked the students to explore various ways of implementing histogram equalization on colored images. The report begins with an overview of the topic of HE. Chapter 2 will present the implementations of three different basic HE algorithm; namely 1D HE, RGB HE and Luma HE. The source codes and output of the algorithms will also be documented in the chapter. The superseding Chapter 3 will analyze the results of each HE method shown in the previous chapter and discuss their pros and cons. Lastly, the last Chapter 4 will document the motivation behind the three improvement ideas and verify their effectiveness in improving basic HE.

# Contents

# 1. Introduction

## 1.1 Background

Histogram is a graphical representation of the intensity distribution of an image. It captures the occurrence frequency of pixel intensity values with which multiple image statistics can be calculated. Many image processing methods make use of image histograms for different purposes.

Histogram Equalization (HE) is an image processing technique that has been widely adopted to improve the contrast in images. It accomplishes this by spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image. It usually increases the global contrast of images when the image pixels fall within a narrow range of intensity values. This allows for areas of lower local contrast to gain a higher contrast.

## 1.2 Motivation

This assignment consists of two main tasks focusing on the implementation and discussion of basic HE algorithm. There is also one bonus task on ideas to improve the HE algorithm.

Given a dataset of eight sample images, the first task requires the implementation and application of basic HE algorithm to generate enhanced images of them. Three techniques were experimented in this report. The first method termed as Basic HE will demonstrate the four main steps required for histogram equalization. Enhanced images are generated from a flatten one-dimensional array of the colored images. The following method, called RGB HE, applies the same methodology to the Red, Green and Blue components of the images. The final method, Luma HE repeated the same HE processing steps on only the luminescence channel of images after LAB/ HSV color space conversion.

The second task focuses on the discussion on the results obtained using basic HE algorithms. The pros and cons will be enumerated and supported by the outputs from the previous chapter. The possible causes to unsatisfactory contrast enhancement caused by using such point-based histogram equalization will also be summarized.

In the optional task, three improvement ideas were derived from the observations made in Chapter 3. This report presents the motivations and hypothesis of each idea. Finally, after implementation, the results were analyzed, and their evaluated effectiveness were documented.

# 2. Task 1: Implementation

## 2.1 Pre-requisites/ Setup

The source code of this assignment are implemented in Python with the use of Numpy library as well as Matplotlib for graphical representations. OpenCV2 library is used for conversion of image color channels, which will be explained in later sections.

## 2.2 Dataset Analysis

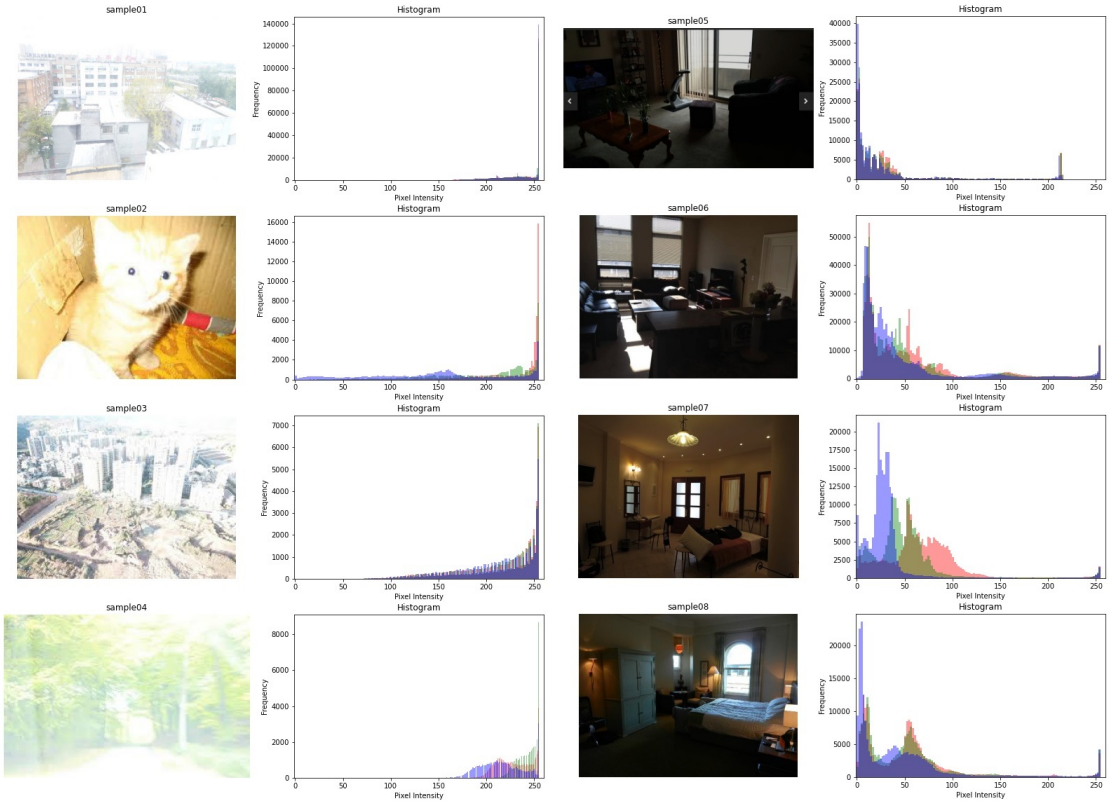


Figure 2.1: Sample images and their RGB histogram distribution.

The sample images are analyzed by plotting a histogram of their pixel intensities against frequency. As shown in Figure 2.1, the histograms are representative of the images as they reflect the distribution of the pixel values.

For the histogram distributions of samples 1 to 4, they are shown to be right skewed towards 255. From this observation, they reflect that these images are bright images with high-contrast. On the other hand, for samples 5 to 8, the left skewed distributions show that the images are dark and have low-contrast issues.

## 2.3 Basic HE Algorithm

This section breaks down the basic histogram equalization procedure, which is a image processing method for contrast adjustment using the image's histogram. In this implementation, the colored image channels will first be flattened into an one-dimensional array. The general steps for basic HE algorithm's implementation is summarized as shown below:

**Algorithm 1:** Basic HE Implementation

**Result:** HE transformed images
Load the dataset into dictionary $D$. See Appendix.
Write a helper function to display image & histogram side by side. See Appendix.
**for** *img in D* **do**
    Flatten the image;
    Create a histogram of pixel values count;
    Compute CDF of the histogram;
    Normalize CDF values;
    Map the original pixel values to the normalized CDF;
    Reshape the transformed values back to image size;
    Plot transformed image and its histogram;
**end**

### 2.3.1 Source Code

```
def create_histogram(image, bins=max_intensity):
    """
    Creates a histogram containing number of pixels with value x
    """
    hist = np.zeros(bins, dtype=int)
    # count the number of pixels add to bin
    for pixel in image:
        hist[pixel] += 1
    return hist
```

Listing 2.1: Function to create a histogram

```
def compute_cdf(histogram, bins=max_intensity):
    """
    Creates an array that represents the cumulative sum of the histogram
    """
    cdf = np.zeros(bins, dtype=int)
    cdf[0] = histogram[0]
    for i in range(1, histogram.size):
        cdf[i] = cdf[i-1] + histogram[i]
    return cdf
```

Listing 2.2: Function to compute CDF of the histogram

```
def normalize_cdf(cdf, L=max_intensity):
    """
    Creates the transformation mapping array,
    which normalizes CDF to range between 0 and 255
    """
    nj = (cdf - cdf.min()) * (L-1)
    MN = cdf.max() - cdf.min()
    return (nj / MN).astype('uint8') # round values
```

Listing 2.3: Function to normalize CDF to range between 0 to 255

```
def global_he(img_arr):
    """
    Returns new image array after histogram equalization steps
    """
    # step 1: Count pixels in bins
    hist = create_histogram(img_arr)
    # step 2: Compute cumsum of histogram
    cdf = compute_cdf(hist)
    # step 3: Create the transformation function s_k
    sk = normalize_cdf(cdf)
    # step 4: Apply transformation
    he = sk[img_arr]
    return he
```

Listing 2.4: Function that performs all HE steps

```
def histogram_equalization_1d(img):
    """
    Applies HE after flattening image array
    """
    img_1d = img.flatten() # 1D Histogram Equalization
    he_1d = global_he(img_1d)
    he_1d = np.reshape(he_1d, img.shape)
    return he_1d
```

Listing 2.5: Function for 1D Basic HE

### 2.3.2 Output

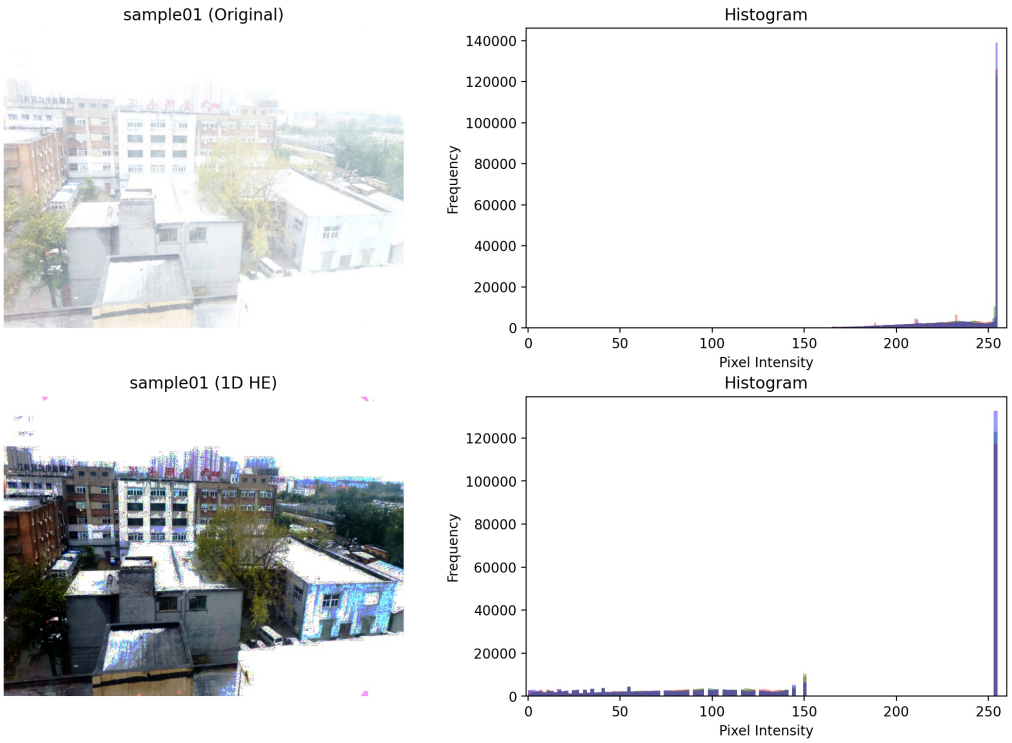


Figure 2.2: Sample01 - Before and After Basic HE

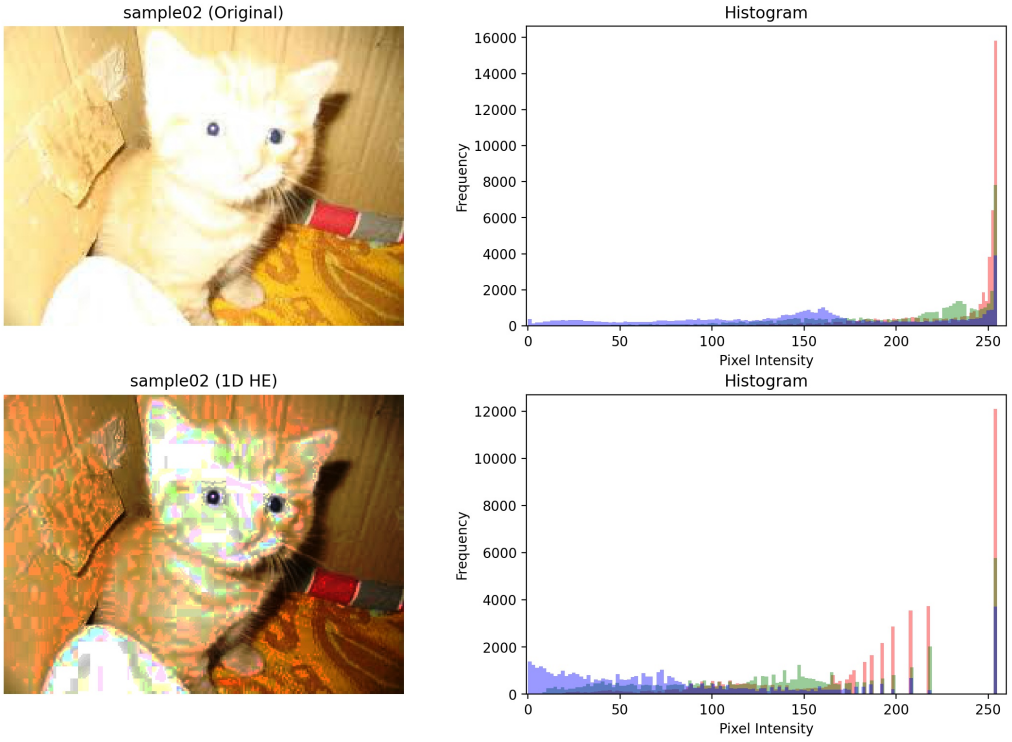


Figure 2.3: Sample02 - Before and After Basic HE

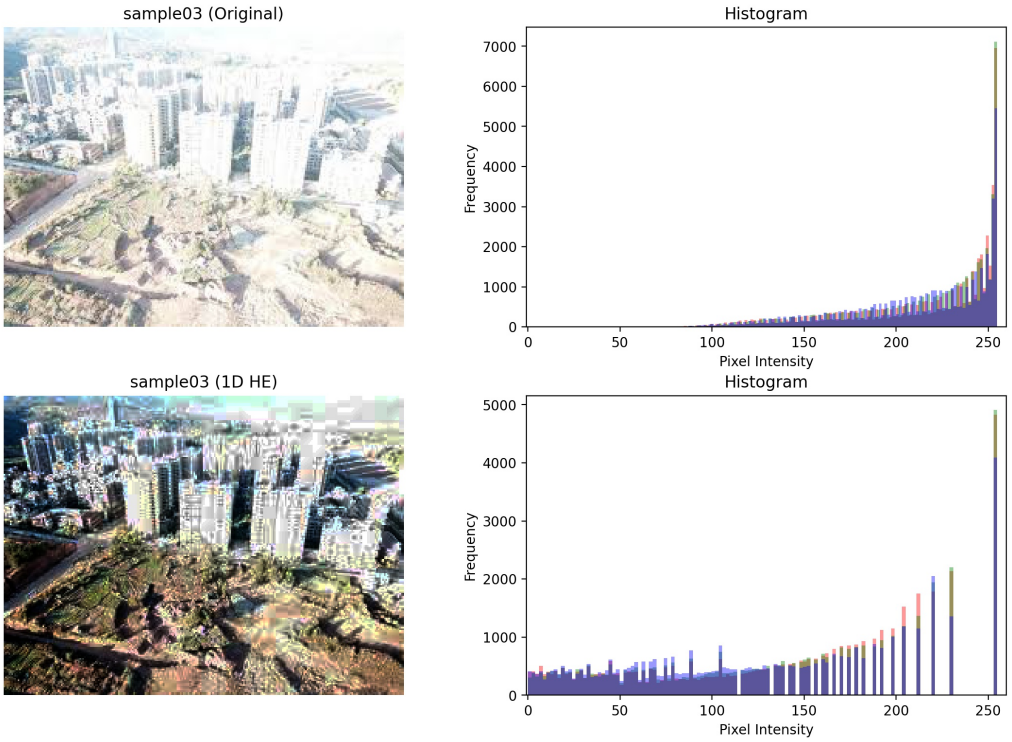


Figure 2.4: Sample03 - Before and After Basic HE

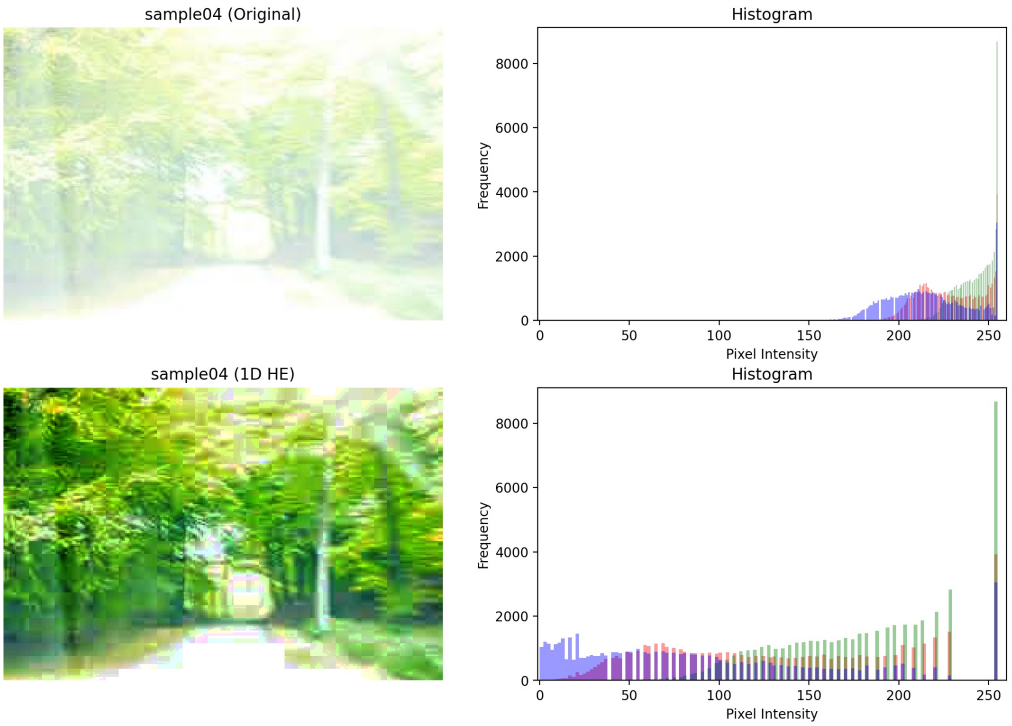


Figure 2.5: Sample04 - Before and After Basic HE

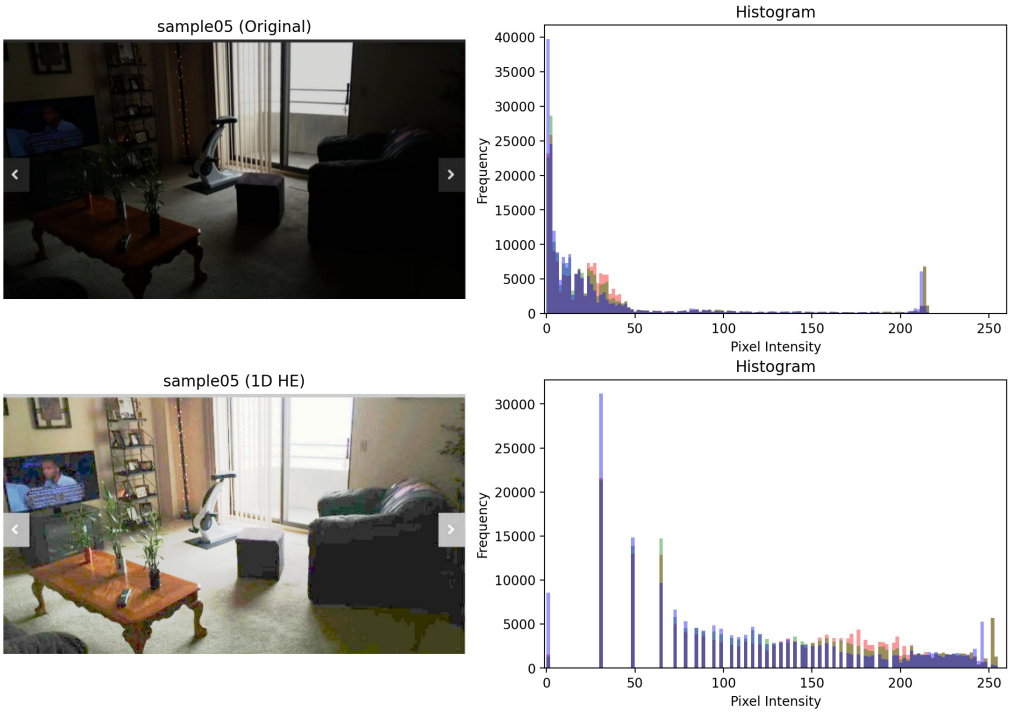


Figure 2.6: Sample05 - Before and After Basic HE

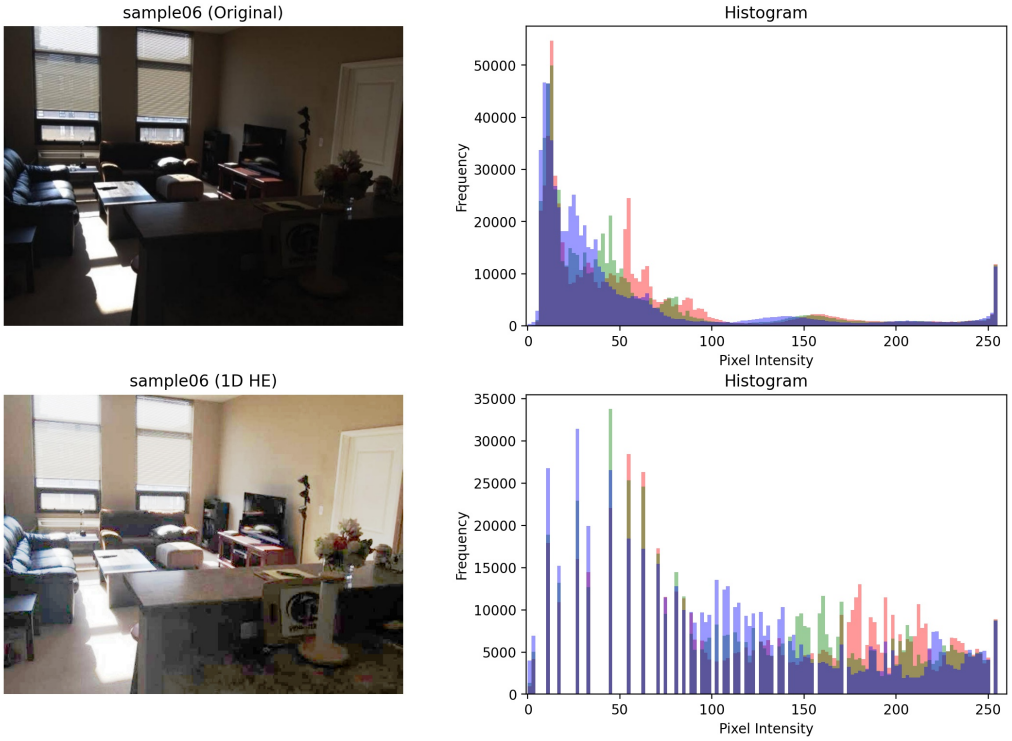


Figure 2.7: Sample06 - Before and After Basic HE

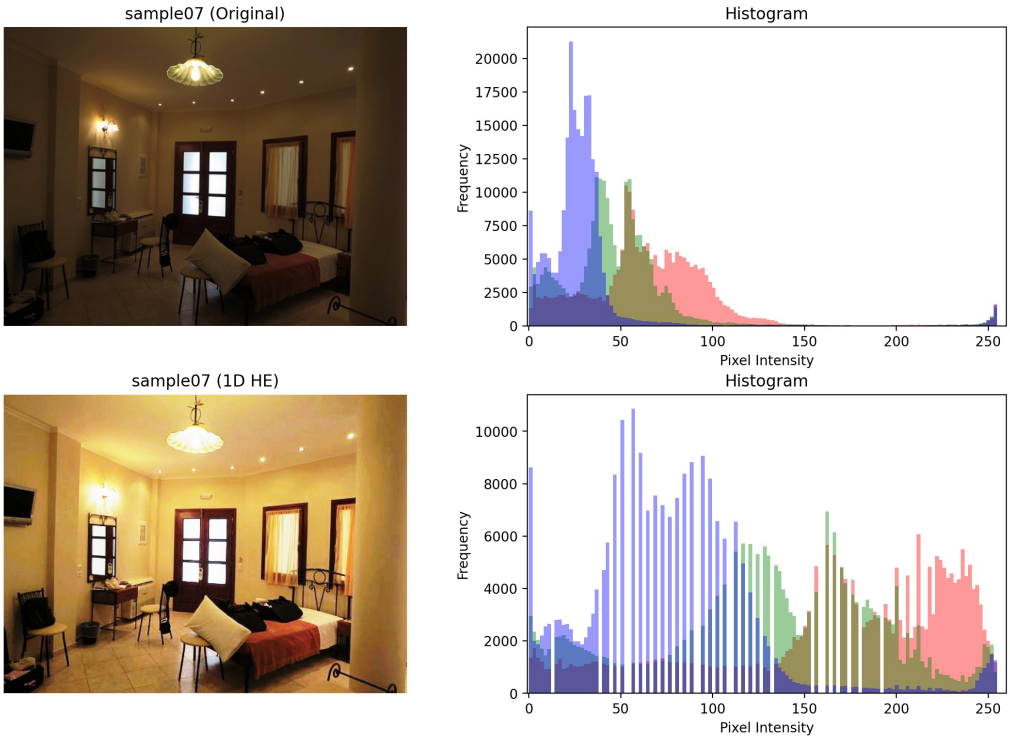


Figure 2.8: Sample07 - Before and After Basic HE

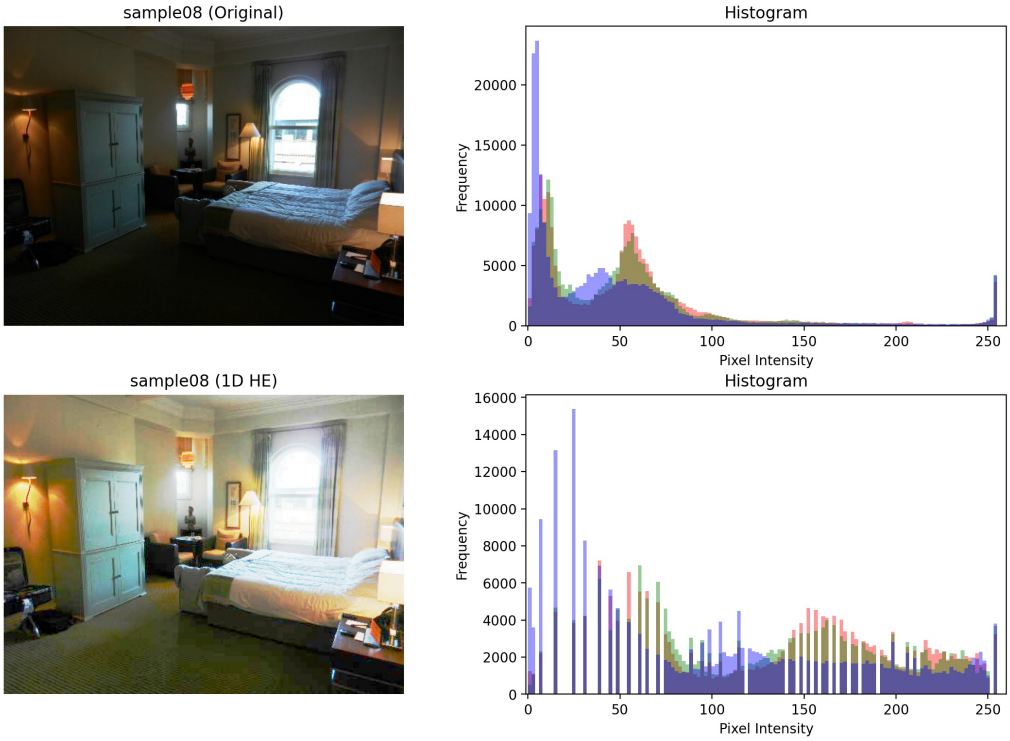


Figure 2.9: Sample08 - Before and After 1D HE

## 2.4 RGB HE Algorithm

Section 2.3 described the basic HE algorithm usually performed on grey scale images (1D array). As the sample images are all colored images, this section will describe the attempt made to equalize the histogram based on the Red, Green and Blue color channels. The implementation is as follows:

**Algorithm 2:** RGB HE Implementation

**Result:** HE RGB transformed images
Load the dataset into dictionary $D$. See Appendix.
Write a helper function to display image & histogram side by side. See Appendix.
**for** *img in D* **do**
    Split the image into Red, Green, Blue channels $C$;
    **for** *color in C* **do**
        Create a histogram of pixel values count;
        Compute CDF of the histogram;
        Normalize CDF values;
        Map the original pixel values to the normalized CDF;
    **end**
    Merged the processed channels back to one array;
    Reshape the transformed values back to image size;
    Plot transformed image and its histogram;
**end**

### 2.4.1 Source Code

```
def histogram_equalization_rgb(img):
    # split into 3 color channels R G B
    img_red = img[:, :, 0].ravel()
    img_green = img[:, :, 1].ravel()
    img_blue = img[:, :, 2].ravel()

    he_r = global_he(img_red)
    he_g = global_he(img_green)
    he_b = global_he(img_blue)

    he_rgb = np.dstack((he_r,he_g,he_b))
    he_rgb = np.reshape(he_rgb, img.shape)

    return he_rgb
```

Listing 2.6: Function for RGB HE

The implementation for `global_he` can be found in Listing 2.4.

### 2.4.2 Output

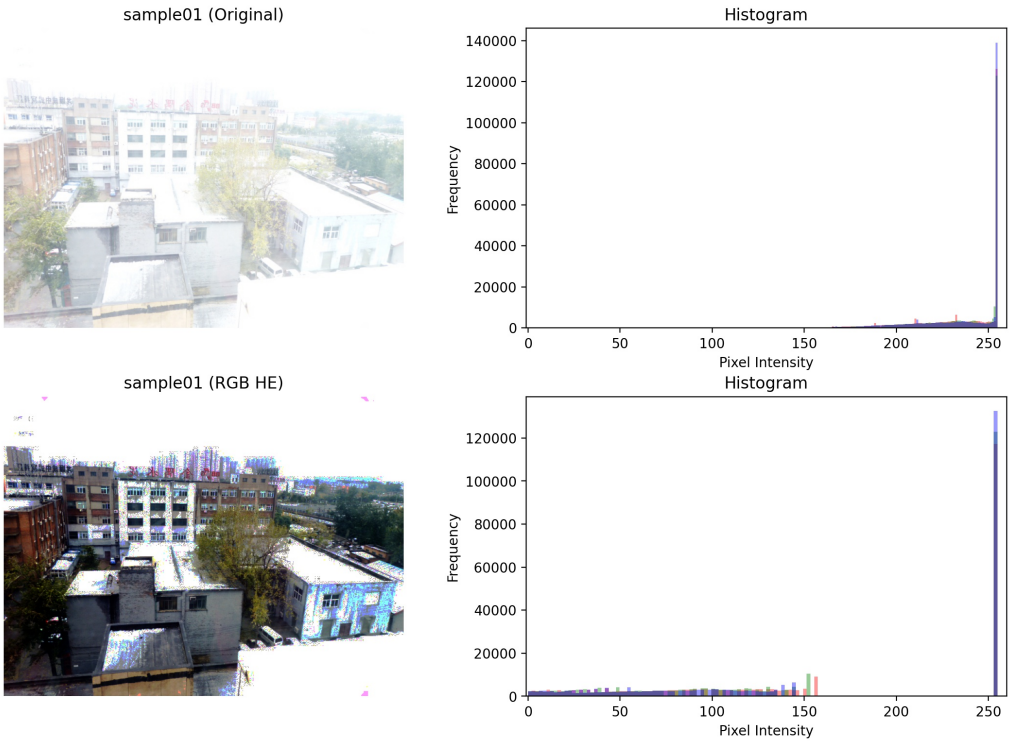


Figure 2.10: Sample01 - Before and After RGB HE

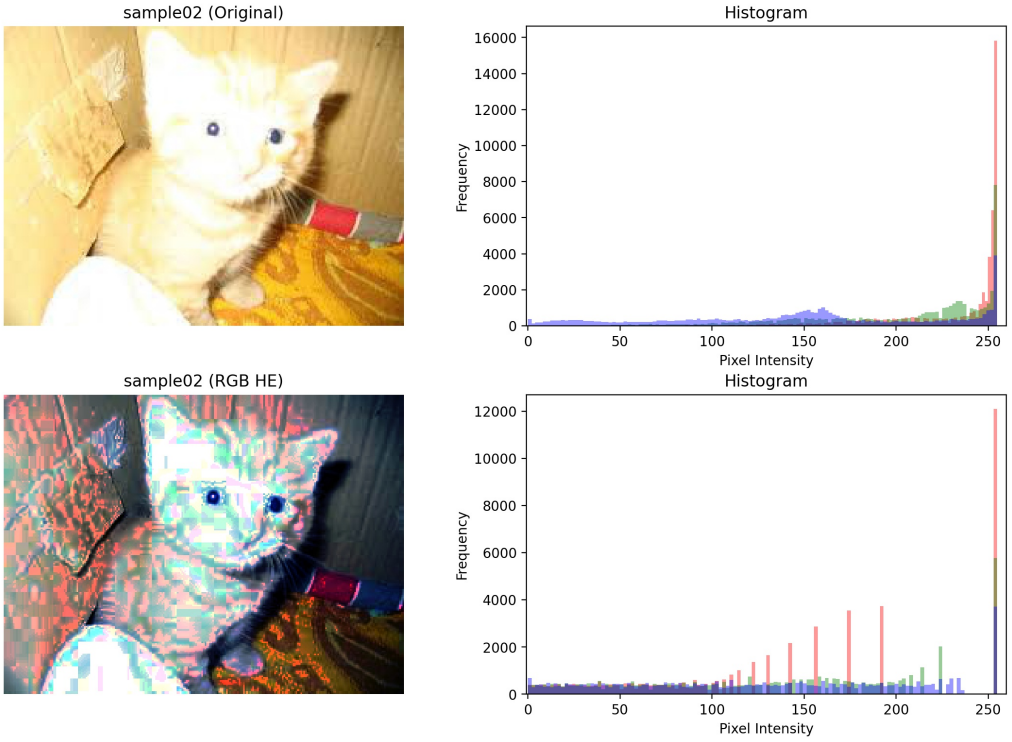


Figure 2.11: Sample02 - Before and After RGB HE

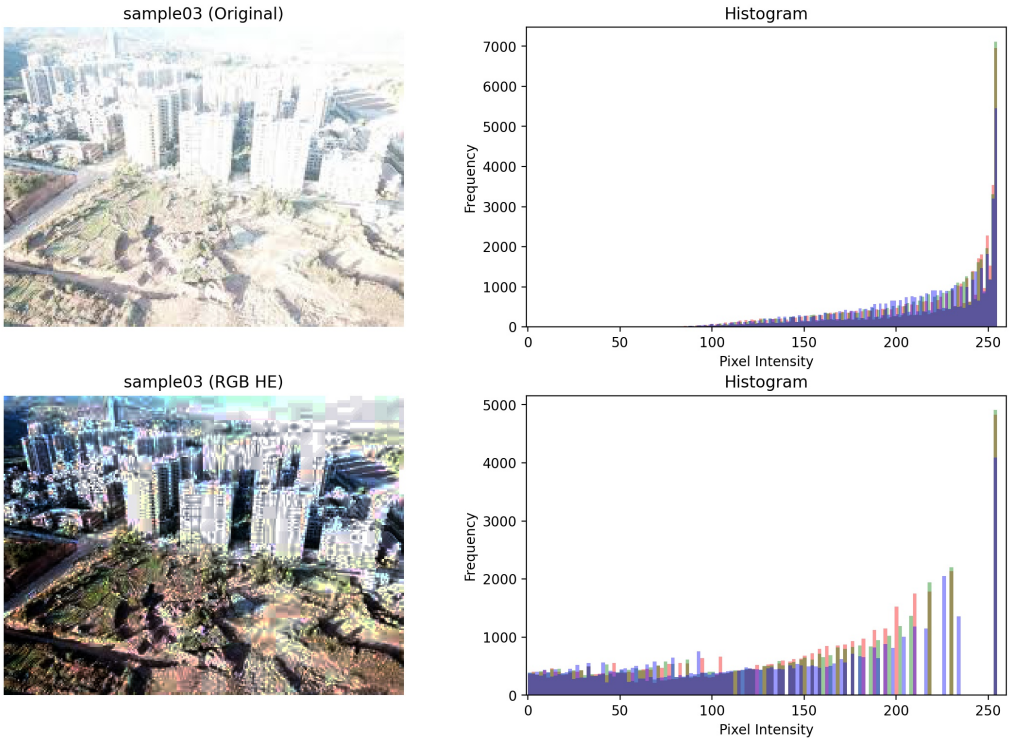


Figure 2.12: Sample03 - Before and After RGB HE

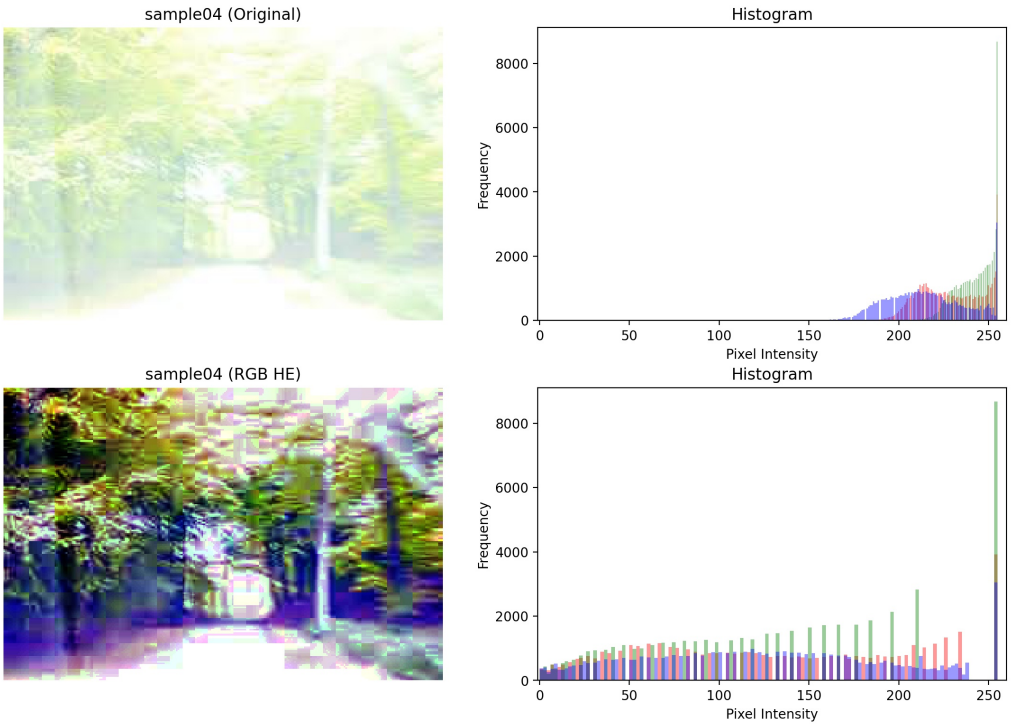


Figure 2.13: Sample04 - Before and After RGB HE

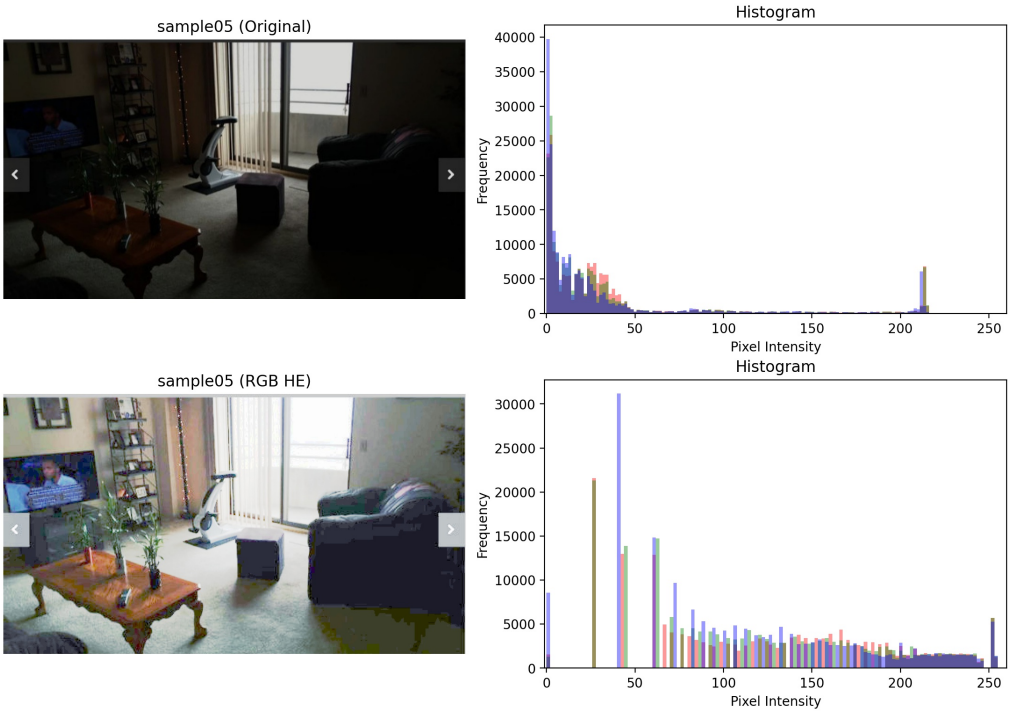


Figure 2.14: Sample05 - Before and After RGB HE

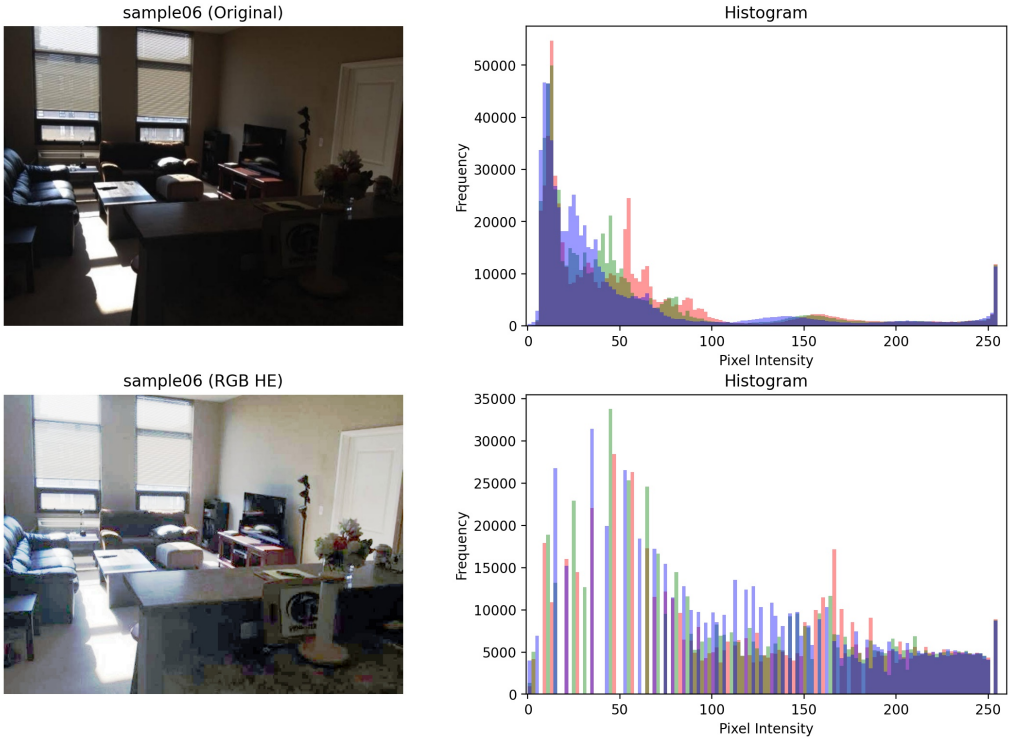


Figure 2.15: Sample06 - Before and After RGB HE

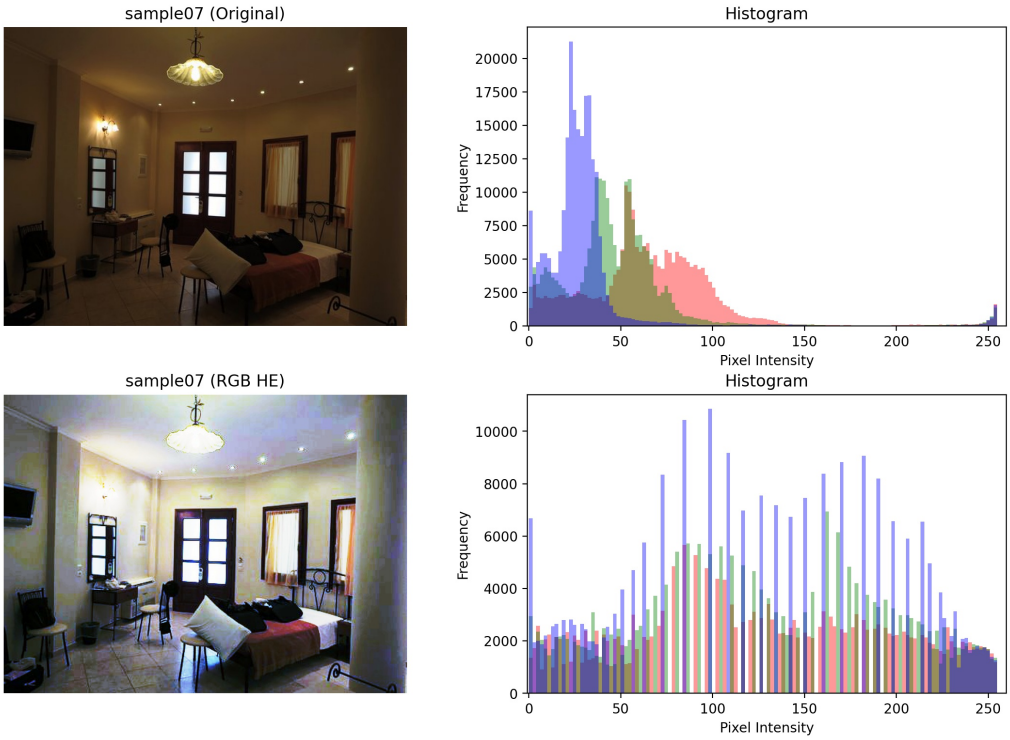


Figure 2.16: Sample07 - Before and After RGB HE

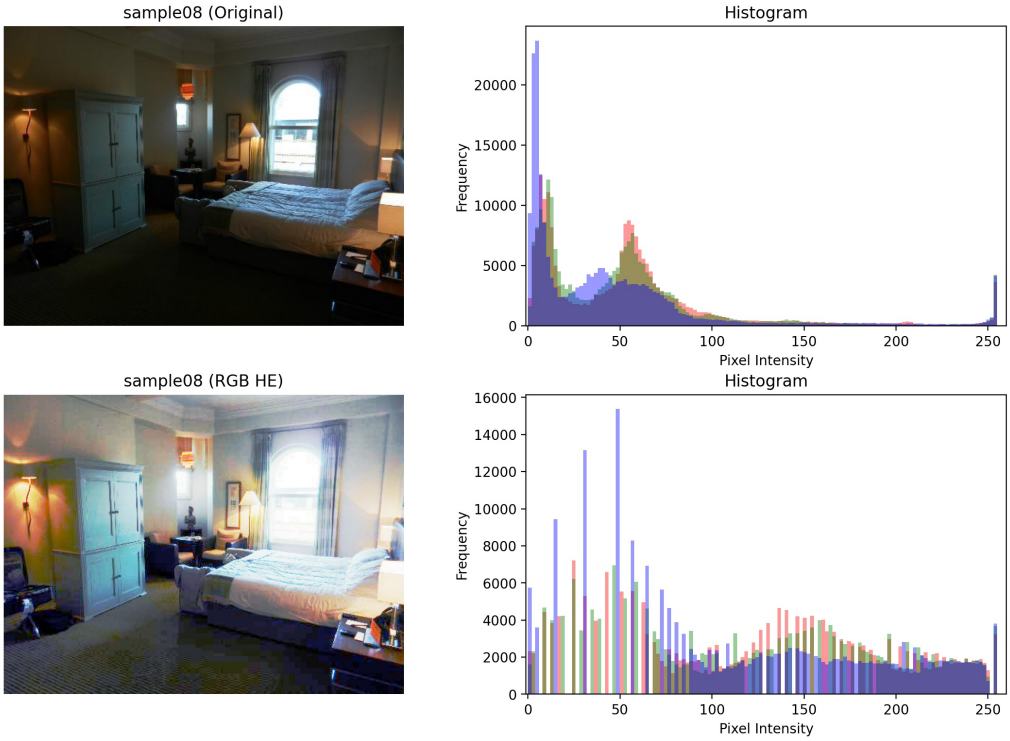


Figure 2.17: Sample08 - Before and After RGB HE

## 2.5 Luma (LAB/HSV) HE Algorithm

This section will describe the implementation of histogram equalization on the intensity component of the luminescence color spaces. For instance, in the HSV color space:

- H stands for Hue, which refers to the spectrum of colors.
- S stands for Saturation, which refers to the intensity of the primary colors.
- V stands for Value, which refers to the lightness or darkness of a color.

On the other hand, in the LAB color space:

- L stands for Lightness, which scales from 0 (black) to 100 (white) like a similar to grayscale image.
- A stands for Green-red color spectrum, which scales from green (neg) to red (pos).
- B stands for Blue-yellow color spectrum, which scales from blue (neg) to yellow (pos).

The idea behind this implementation is to only adjust the luminescence/intensity value of the image (L channel in LAB or V channel in HSV). The pseudo code of the implementation is shown below:

**Algorithm 3:** LAB HE Implementation

**Result:** HE LAB transformed images
Load the dataset into dictionary $D$. See Appendix.
Write a helper function to display image & histogram side by side. See Appendix.
**for** *img in D* **do**
    Convert the RGB image to LAB color space;
    Split the image into LAB color channels $C$;
    Create a histogram of pixel values count on the $V$ channel;
    Compute CDF of the histogram;
    Normalize CDF values;
    Map the original pixel values to the normalized CDF;
    Merged the LAB channels back to one array;
    Convert the LAB image back to RGB color space;
    Plot transformed image and its histogram;
**end**

The same method is also applied in the HSV color space, whereby the image is first converted to HSV color space and HE is applied only on the V channel. This will be termed as HSV HE and its results will be shown in Chapter 3.

### 2.5.1 Source Code

```
def histogram_equalization_LAB(img):
    img_H, img_W, _ = img.shape
    img_lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
    ch_lab = cv2.split(img_lab) # split color channels

    ch_lab[0] = global_he(ch_lab[0].flatten()) # apply HE on L channel
    ch_lab[0] = ch_lab[0].reshape(img_H, img_W)

    he_lab = cv2.merge(ch_lab) # merge back the channels
    he_lab_rgb = cv2.cvtColor(he_lab, cv2.COLOR_LAB2RGB) # convert back to RGB
    return he_lab_rgb
```

Listing 2.7: Function for LAB HE

The implementation for `global_he` can be found in Listing 2.4.

### 2.5.2 Output

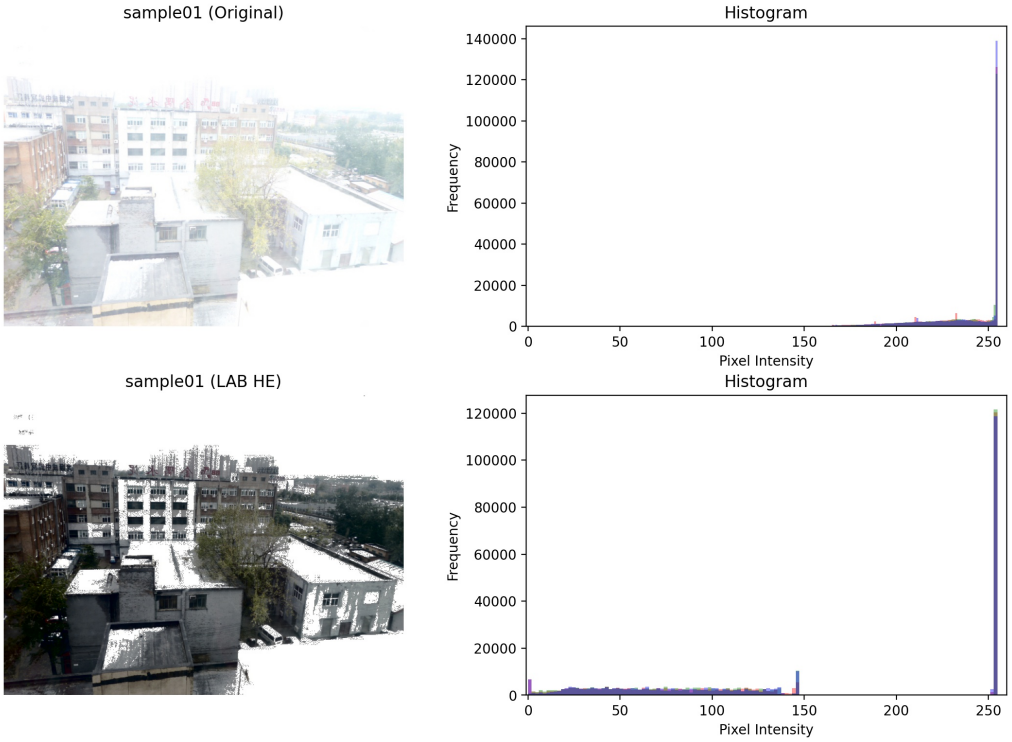


Figure 2.18: Sample01 - Before and After LAB HE

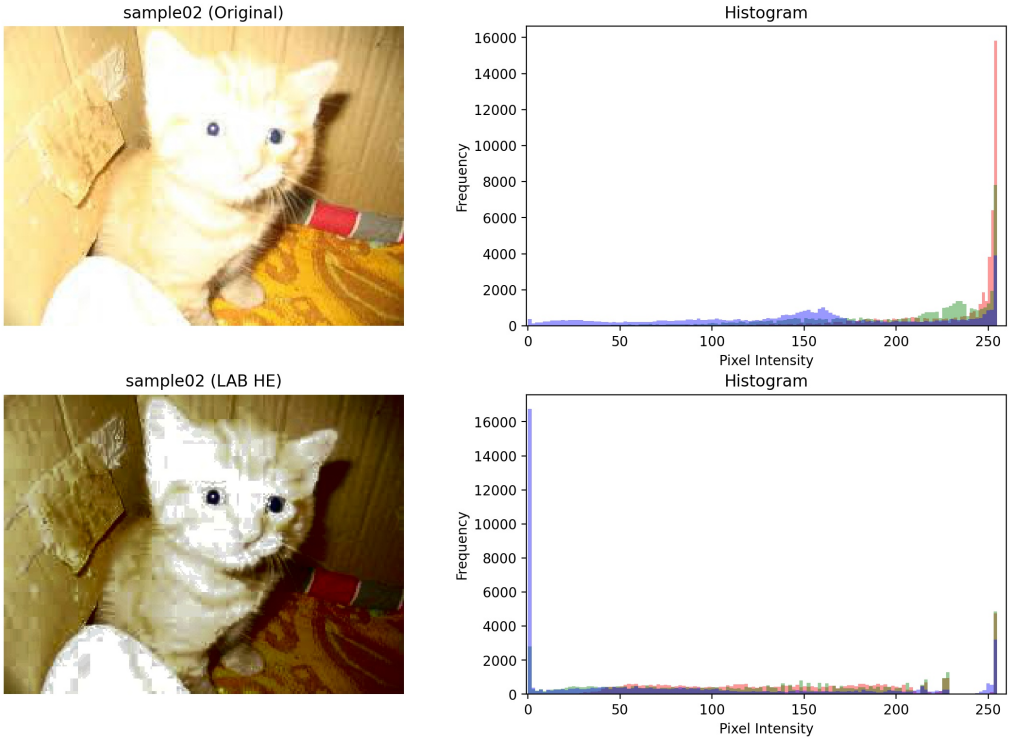


Figure 2.19: Sample02 - Before and After LAB HE

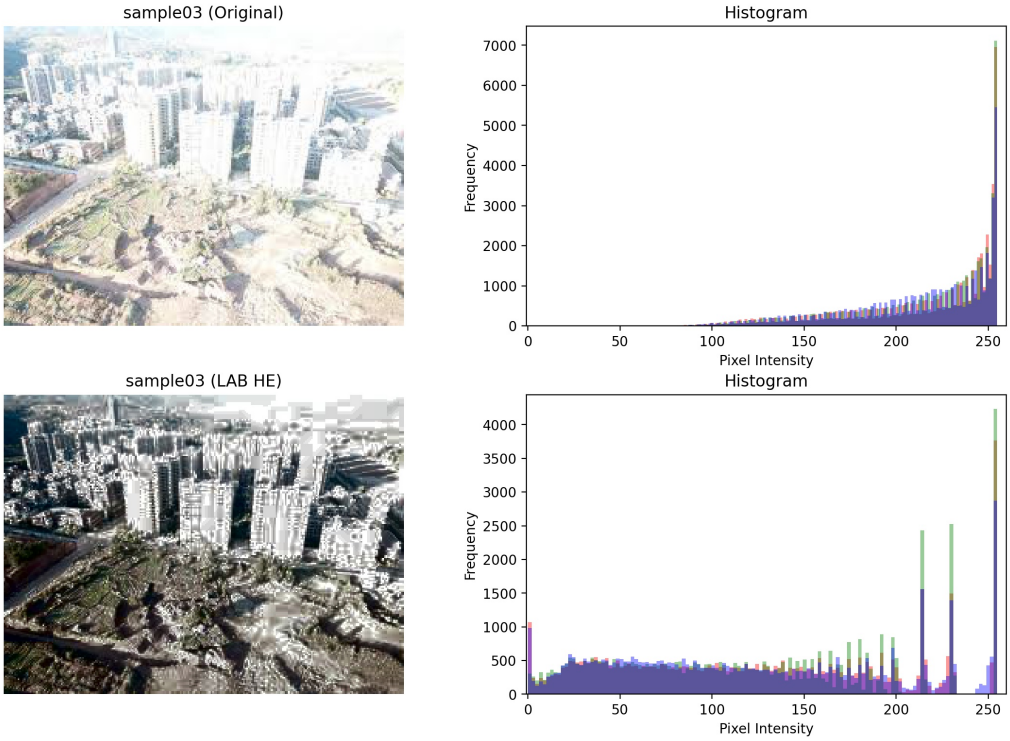


Figure 2.20: Sample03 - Before and After LAB HE

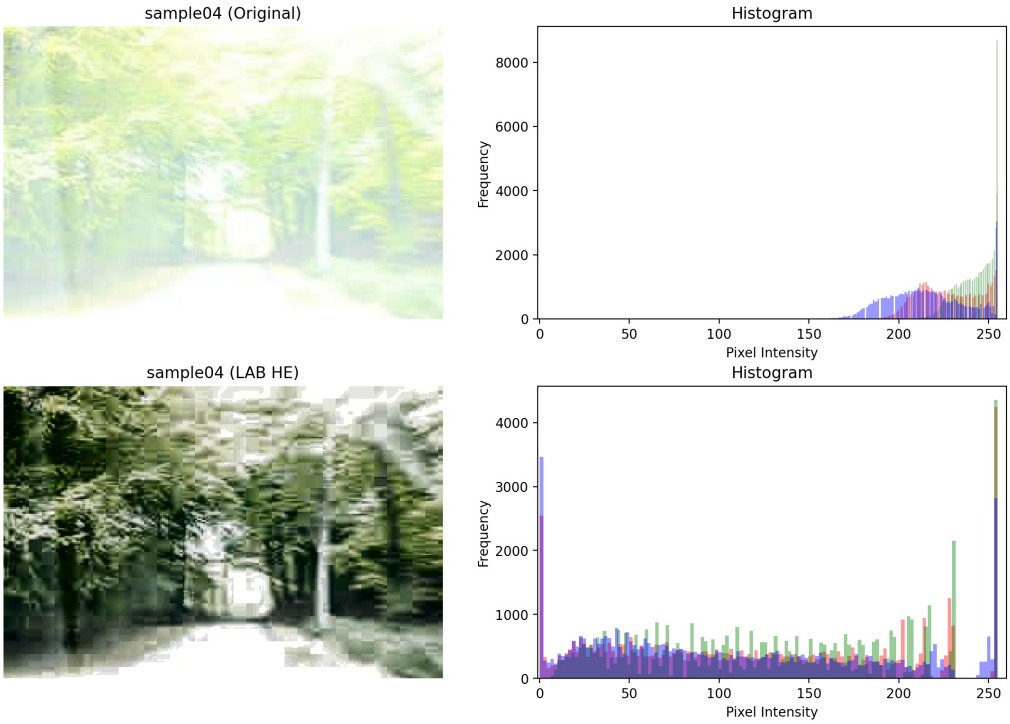


Figure 2.21: Sample04 - Before and After LAB HE

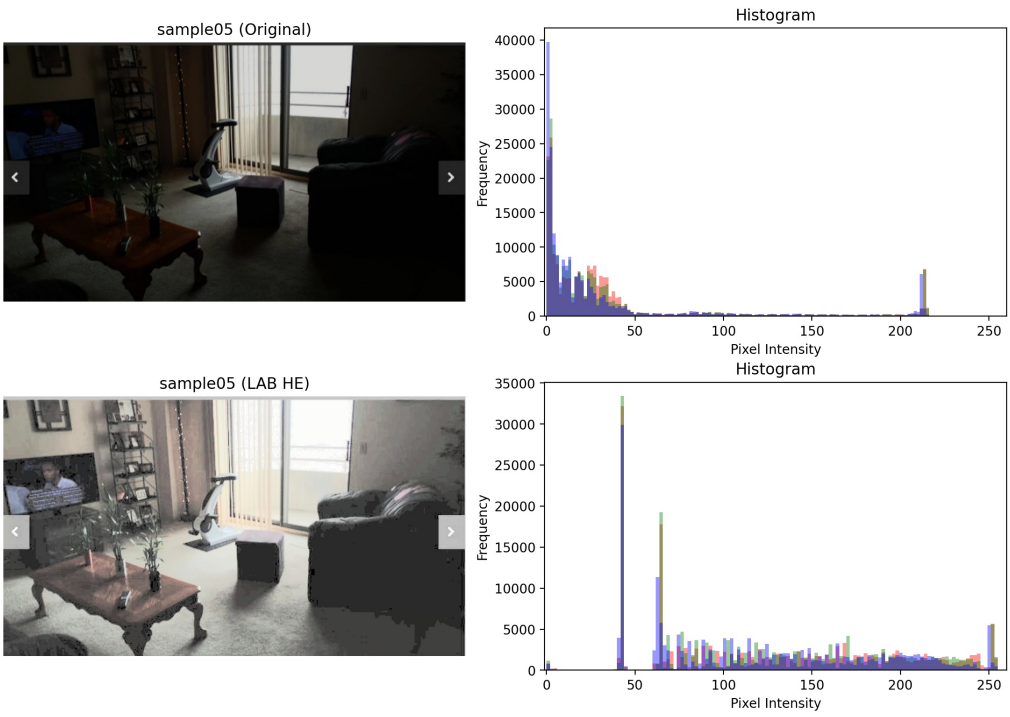


Figure 2.22: Sample05 - Before and After LAB HE

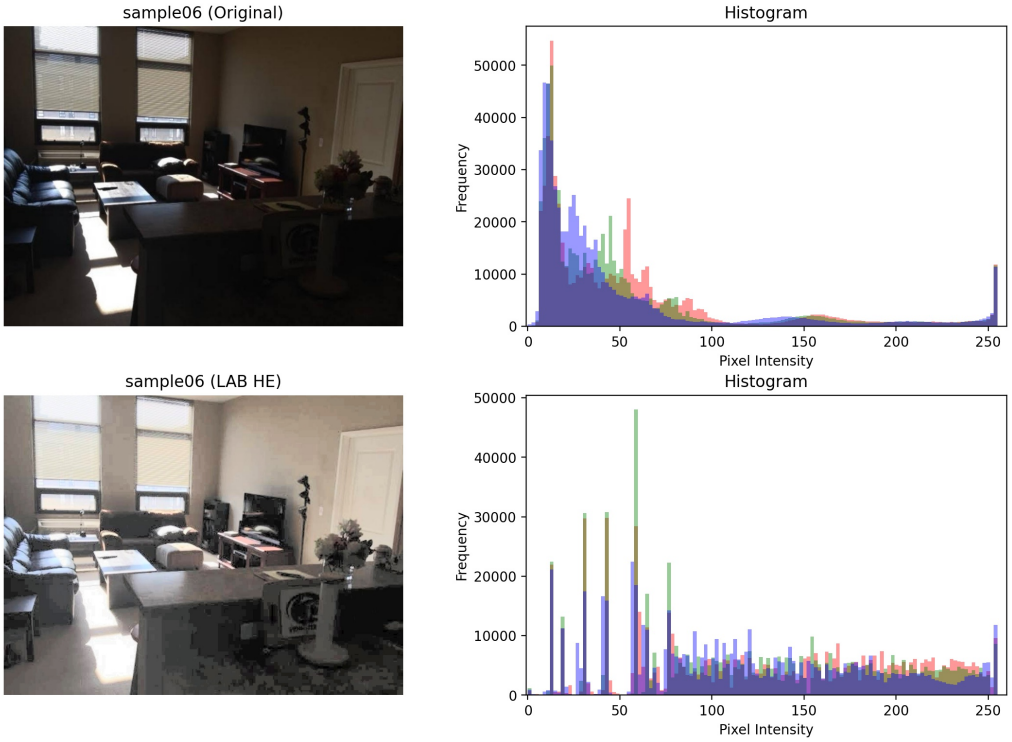


Figure 2.23: Sample06 - Before and After LAB HE

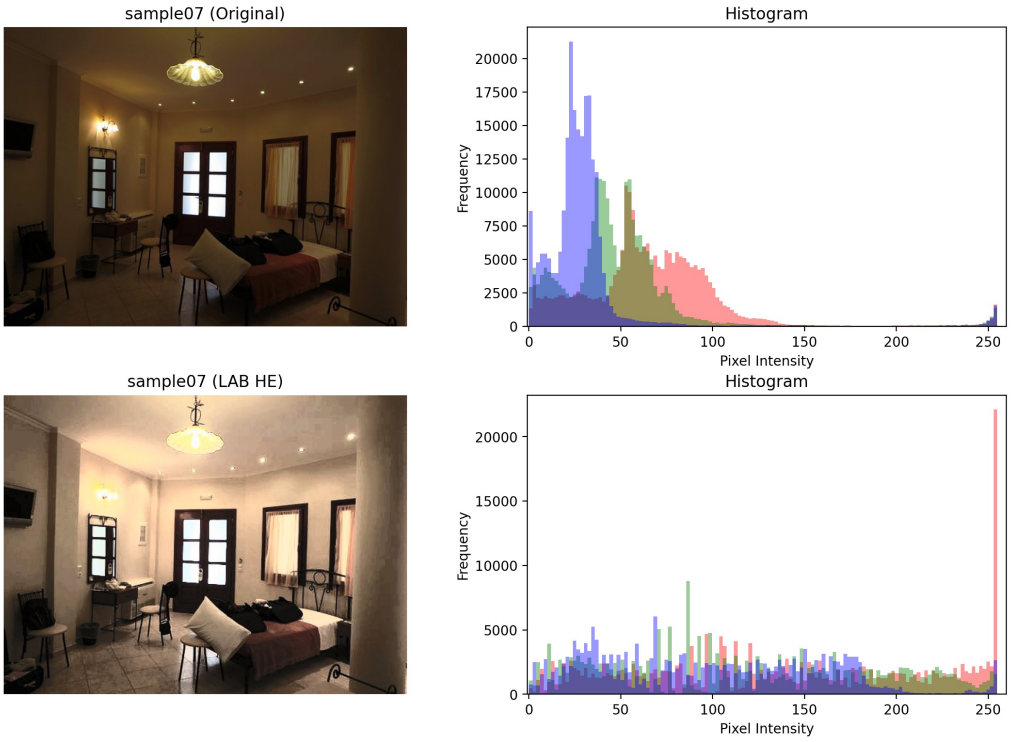


Figure 2.24: Sample07 - Before and After LAB HE

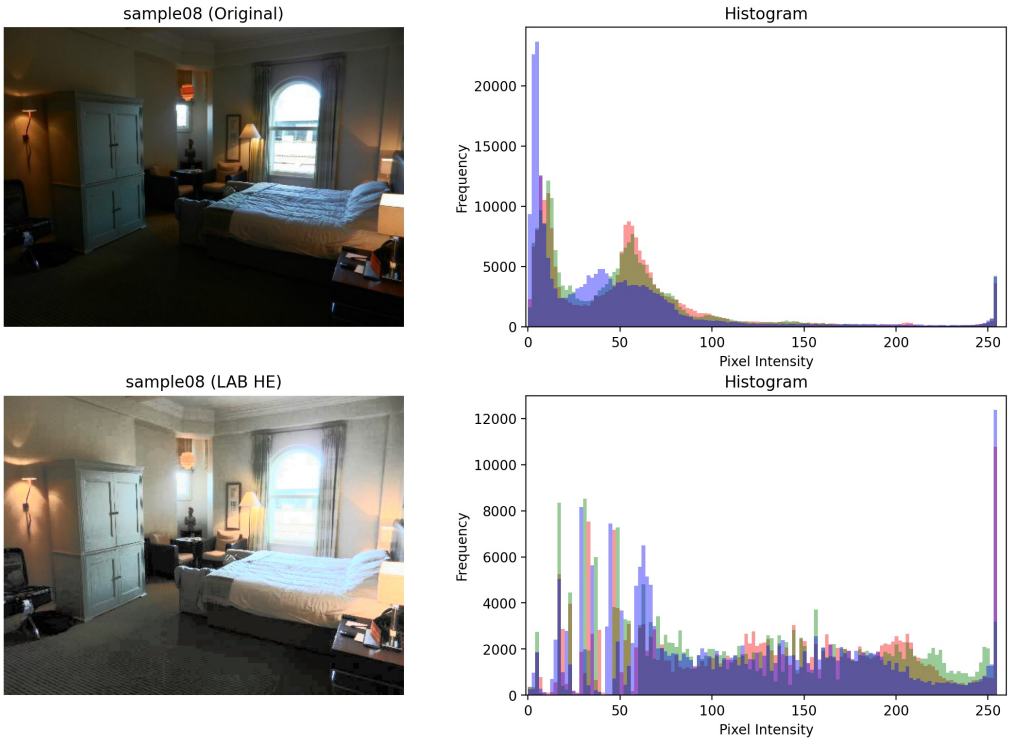


Figure 2.25: Sample08 - Before and After LAB HE

# 3. Task 2: Results Discussion

## 3.1 Results Evaluation

In this section, the output of each sample will be analyzed and the pros and cons of each methodology will be assessed.

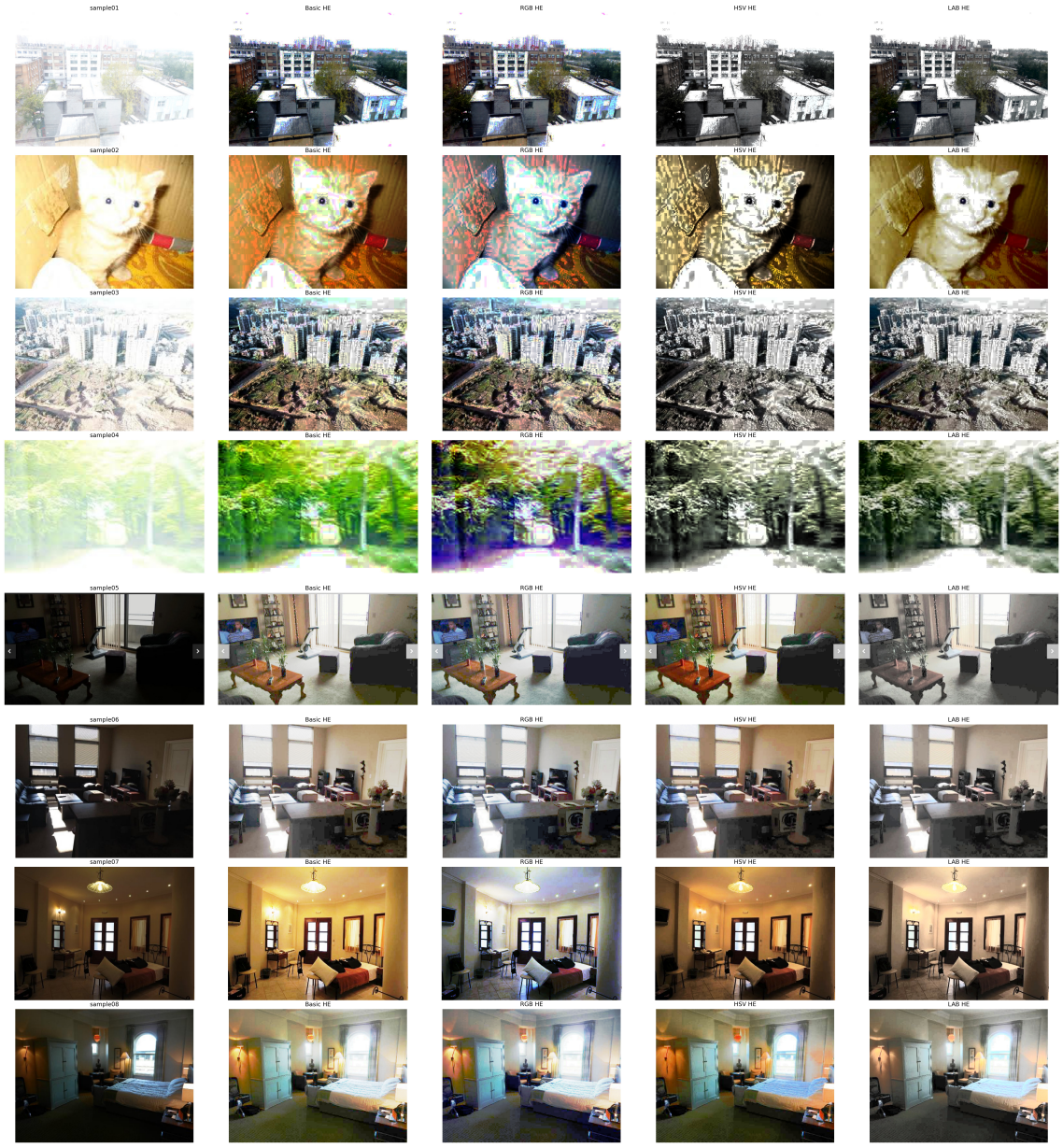


Figure 3.1: Processed images of all samples starting from original image (1st column), Basic HE (2nd), RGB HE (3rd), HSV HE (4th) and LAB HE (5th)

### 3.1.1 Basic HE Results Evaluation

In general, histogram equalization (HE) is a simple-to-implement and fast method for contrast enhancement. As seen from the output images, images that are originally bright (samples 1, 2, 3 and 4) are "darken". Likewise, images that are initially dark (samples 5, 6, 7 and 8) are "brighten". With HE contrast enhancement, more details of the images can be seen. For example in sample 5, the person in the TV is now visible.

However, the downsides of HE are also apparent from the output images. HE tends to introduce unwanted artifacts and unnatural enhancements as seen in Sample 1 and 2 (Figure 2.2 and 2.3). Basic HE also tends to over enhance saturation, which results in outputs like Sample 4 (Figure 2.5), where the leaves of the trees turned lime green. Furthermore, the illumination of the images will also be significantly altered, leading to unnatural augmentations as well. This can be seen in Sample 8 (Figure 2.9).

### 3.1.2 RGB HE Results Evaluation

As shown from the results, the RGB histogram equalization implementation not only suffers from the same issues of basic HE as discussed in Section 3.1.1, it has also led to even worse outcomes.

Figure 3.2: R, G, B color channels of sample 2

Comparing the histograms between RGB HE (Section 2.4.2) to basic HE (Section 2.3.2), the color channels will become more equally distributed among the pixel intensity bins. However, by processing each R, G, B color channels individually, the relevance among the colors become disregarded, which leads to severe color balance problem.

Taking sample 2 in Figure 2.11 as an example, as the frequency values of each color intensity is different, the Red (R) color is out of balance, leading to disproportional red artifacts on the image and also losing the brown color of the cardboard. The histogram also revealed that the red color pixels were shifted regardless of the original relationships between the other two channels. This caused the severe color distortion on the cat's face.

Similarly, in the results of sample 4 as shown in Figure 2.12, when the three color channels are adjusted independently, the red and blue pixels became prominent in the output image that was originally green dominated. Overall, applying global HE on R,G,B channels has the worst performance out of the three methods performed.

### 3.1.3 LAB/HSV HE Results Evaluation

Figure 3.3 and Figure 3.4 depict the color space of LAB and HSV respectively for sample image 8. The reason for performing histogram equalization only on the luminescence channel of the color space is to adjust the contrast without amending the saturation and hue of the color space of the image. In comparison with the outputs of Basic HE and RGB HE, the noise artifacts in the outputs of LAB HE were greatly reduced. This is highly evident from samples 2 (Figure 2.19) and 4 (Figure 2.21), where the noise were most apparent in the previous Basic HE and RGB HE output images.


Figure 3.3: L,A,B color channels of sample 8

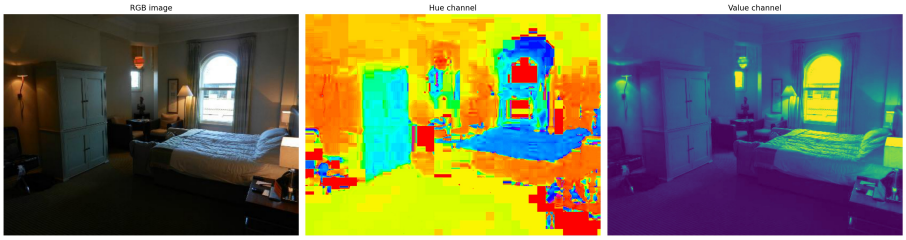


Figure 3.4: HS, V color channels of sample 8

The two different color spaces also rendered different results. Figure 3.1 shows that the outputs of HSV HE are more saturated than LAB HE (see sample 5). One anormaly occurred in the cat image of sample 2, where HSV HE produced a noisy output.

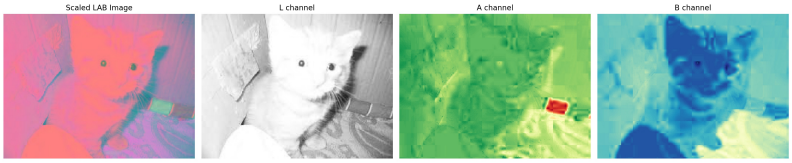


Figure 3.5: L,A,B color channels of sample 2

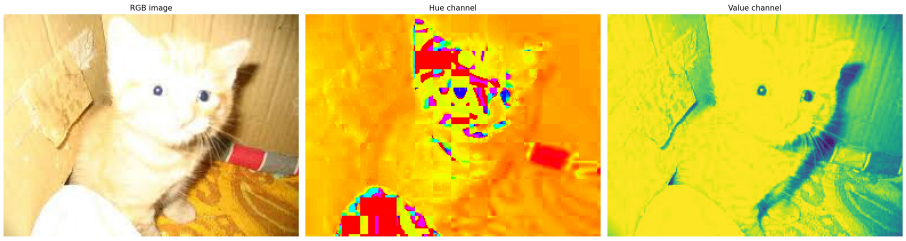


Figure 3.6: HS, V color channels of sample 2

By analyzing the HSV color space image of sample 2 (Figure 3.6), the distortion in HSV HE could be caused by the original Hue Channel after merging back with the adjusted Value channel, leading to the dark pixels on the cat's face. Unlike HSV color space, the A and B color spectrums in LAB color space (Figure 3.5) showed smoother gradient transitions.

In general, for Luma histogram equalization methods, either on the Value channel of HSV or Luminescence channel of LAB, render better outputs than Basic HE and RGB HE with reduced artifacts. However, it still suffers from effects of unnaturalness as seen in sample 1 and sample 3 with large amount of grey pixels.

## 3.2 Possible causes of unsatisfactory contrast enhancement

In Basic HE, as the method does not take into consideration of the mean brightness of a colored image, it leads to over saturation and noisy artifacts caused by the extreme pixel value bins of either white (256 in bright images) or black (0 in dark images). By only applying HE on the converted color space, these effects are reduced, but still persisted.

The possible causes of unsatisfactory contrast enhancement could be because Global HE methods are indiscriminate. It is a point-based image processing method such that the cumulative sum function does not consider relations with neighbouring pixels. This may increase the contrast of background noise while decreasing the usable signal.

# 4. Task 3: Improvement Ideas

In this chapter, three ideas on how to improve the overall contrast enhancement of the sample images will be discussed in this section.

## 4.1 Pre-process Image before Histogram Equalization

Due to the observation of overly bright and overly dark sample images, the idea is to first administer some image pre-processing before performing histogram equalization.

$$g1 = 255 \times \left(\frac{x}{255}\right)^2 \tag{4.1}$$

$$g2 = 255 \times \left(\frac{x}{255}\right)^{1/3} \tag{4.2}$$

Equation 4.1 is applied to the first 4 sample images, which have the issue of over brightness. By doing so, the intensity of pixels decreases, shifting the histogram values to the left. Similarly, equation 4.2 is applied to the last 4 sample images, which are overly dark. This increases the intensity of the pixels and make the images become lighter, shifting the histogram values to the right. After the non-linear adjustment, the image is then processed with LAB HE (Section 2.5). The idea for this is to disperse the histogram values from the extreme ends before histogram equalization.

### 4.1.1 Source Code

```
def non_linear_decrease_contrast(img):
    """
    Decrease intensity such that dark pixels become much darker,
    bright pixels become slightly dark
    """
    newImage1 = (255)*(img/(255))**2
    return np.asarray(newImage1).astype('uint8')

def non_linear_increase_contrast(img):
    """
    Increase intensity such that dark pixels become much brighter,
    right pixels become slightly bright
    """
    newImage1 = (255)*(img/(255))**(1/3)
    return np.asarray(newImage1).astype('uint8')
```

Listing 4.1: Functions to preprocess images

### 4.1.2 Output

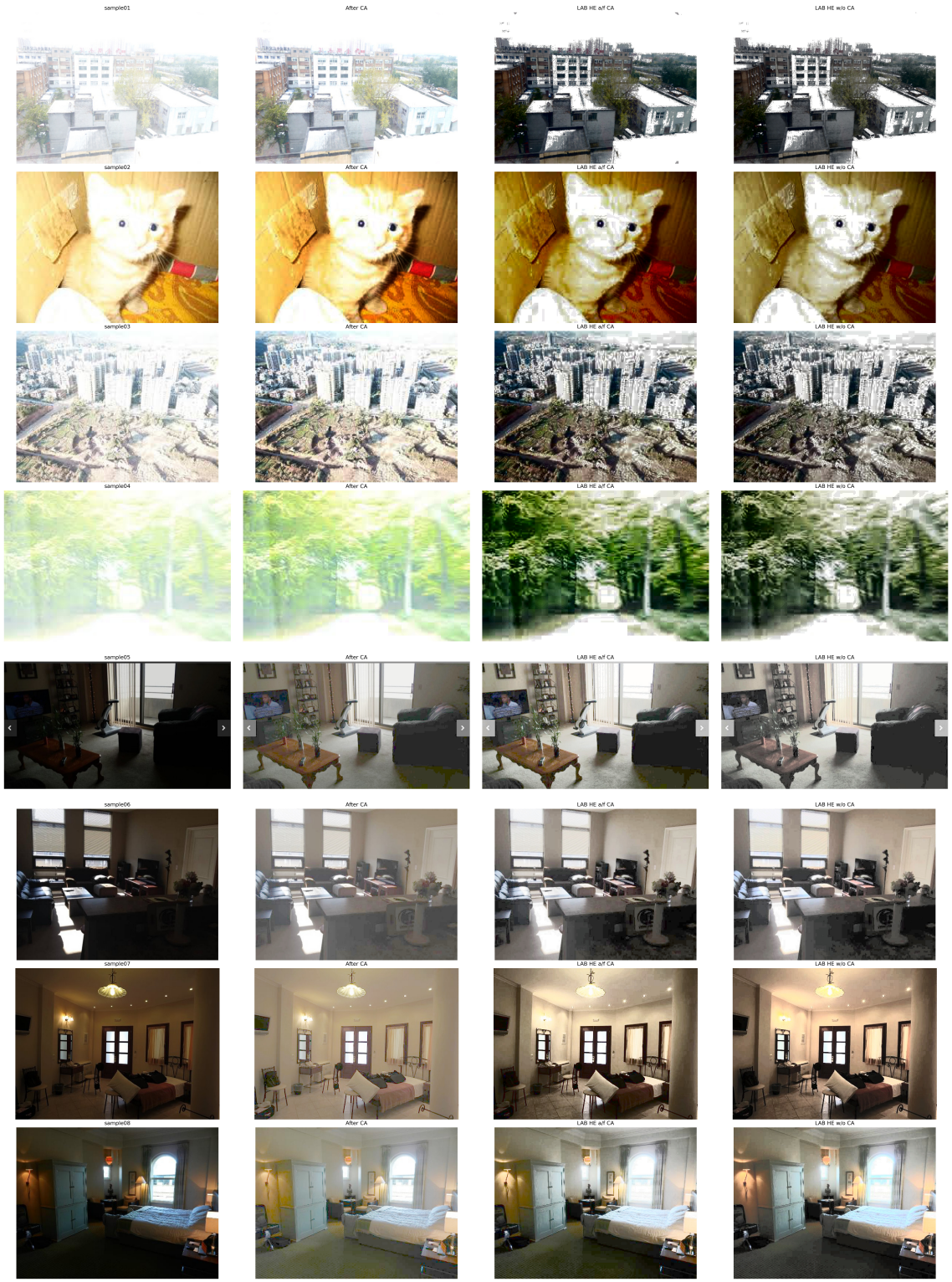


Figure 4.1: Processed images of all samples starting from original image (1st), After Contrast Adjustment (2nd), **CA+LAB HE** (3rd) and LAB HE w/o CA (4th)

### 4.1.3 Result Analysis

As shown in Figure 4.1, when comparing between CA+LAB HE (column 3) and LAB HE w/o CA (column 4), the method of applying LAB HE on non-linear contrast adjusted images improve the color definition of all the sample images. For instance, in sample 3, the green fields at the foreground are better presented. Similarly in sample 4, the green colored leaves on the trees looked more natural.

The benefits of applying LAB HE after non-linear contrast adjustment can also be seen from Figure 4.1. When only non-linear contrast adjustment (column 2) is applied, some images become unrealistically bright. This effect can be seen in Sample 6, 7 and 8. Applying LAB HE tones down the brightness and improve the overall contrast levels.

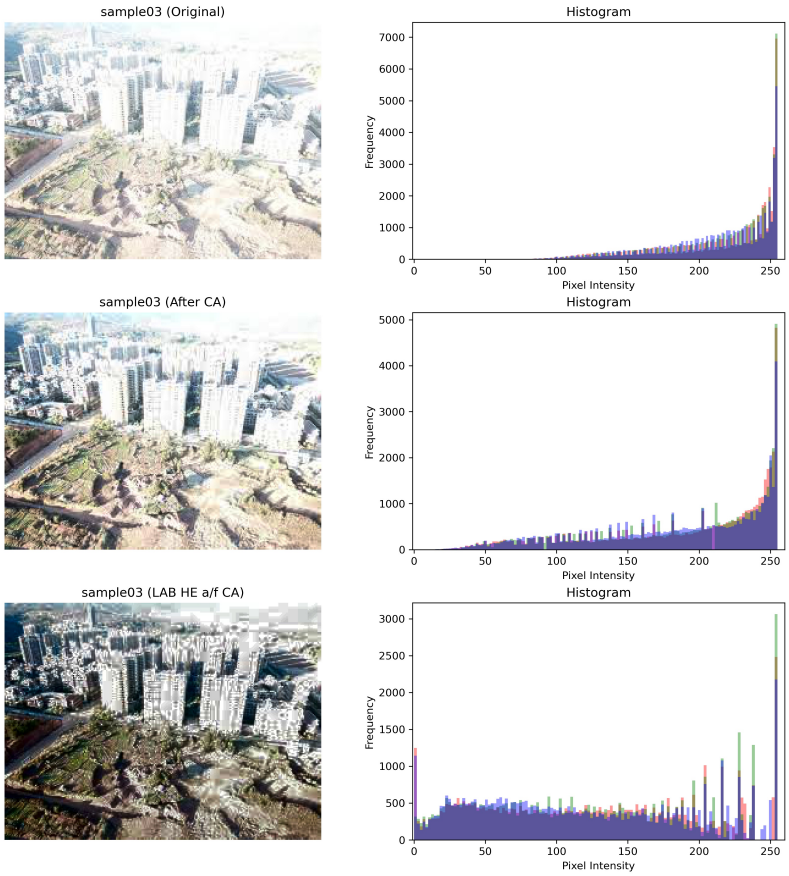


Figure 4.2: Sample03 - Histograms of Original (1st), After CA (2nd) and CA+Lab HE(3rd)

Figure 4.2 shows stages of applying contrast adjustment before LAB HE on sample image 3. The histogram plots supported the motivation behind the idea implemtation. As hypothesized, the non-linear contrast adjustment helps to spread the histogram values from the right (256 values) towards the left (0 values). Subsequent application of LAB HE (Section 2.5) flattens the histogram.

The same can be seen from dark images like sample 8 as shown in Figure 4.3, where the reverse contrast adjustment was made. It is evident from the histogram shifts right after CA and gets flattened after LAB HE.

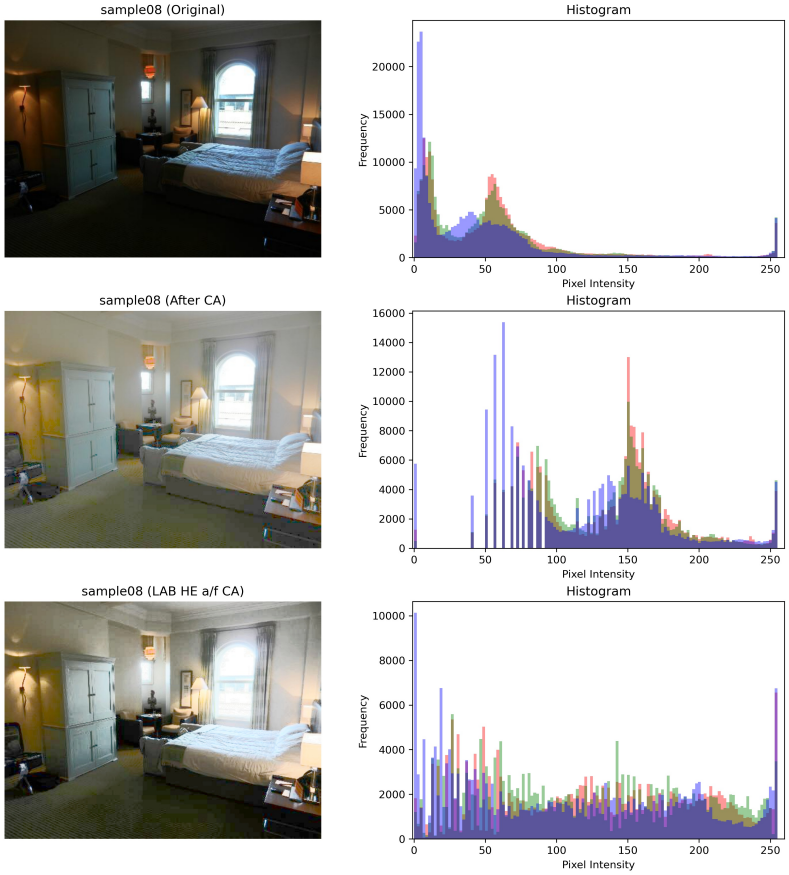


Figure 4.3: Sample08 - Histograms of Original (1st), After CA (2nd) and CA+Lab HE(3rd)

## 4.2 Context-aware Histogram Equalization

Instead of performing point-wise contrast enhancement, the idea behind this implementation is to perform histogram equalization on image pixel rows. It is hypothesized that the luminescence levels per image rows do not vary extensively in overly bright or dark images. By grouping the entire row pixels for histogram generation, the output of the LAB equalization will have an effect of localized context-aware contrast enhancement.

### 4.2.1 Source Code

```
def row_histogram_equalization_LAB(img):
    img_lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
    ch_lab = cv2.split(img_lab) # split color channels

    # row without flattening
    ch_lab[0] = global_he(ch_lab[0]) # apply HE on L channel

    he_lab = cv2.merge(ch_lab) # merge back the channels
    row_lab_rgb = cv2.cvtColor(he_lab, cv2.COLOR_LAB2RGB) # convert back to RGB
    return row_lab_rgb
```

Listing 4.2: Function to perform row-based context-aware LAB HE

### 4.2.2 Output

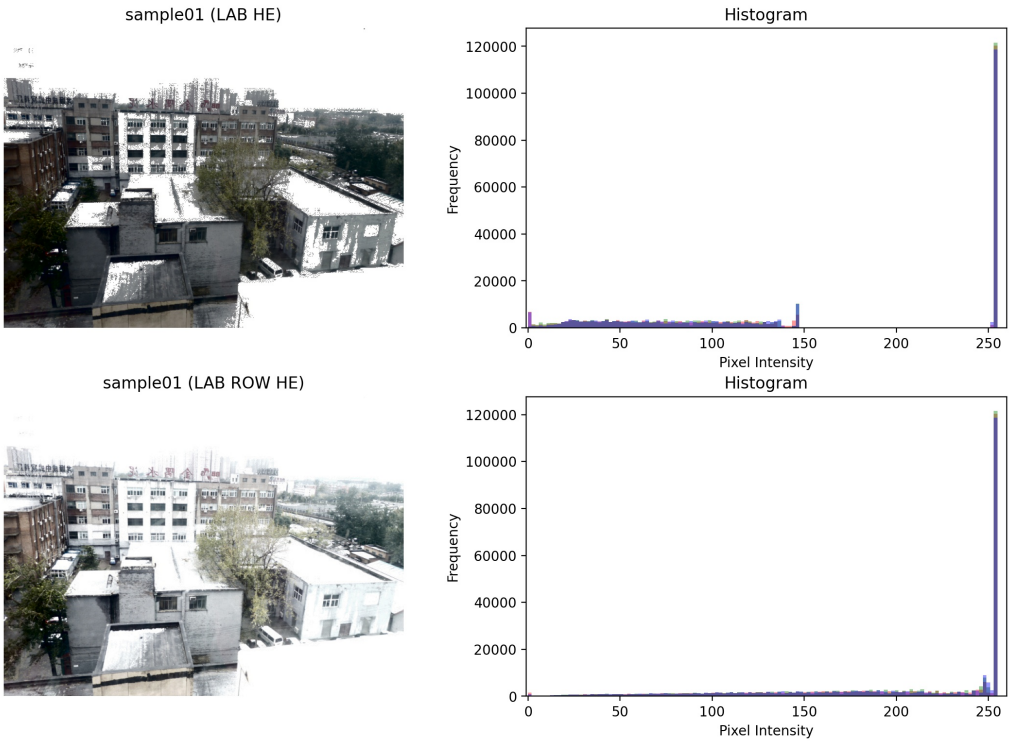


Figure 4.4: Sample01 - Point-wise and Row-based LAB HE

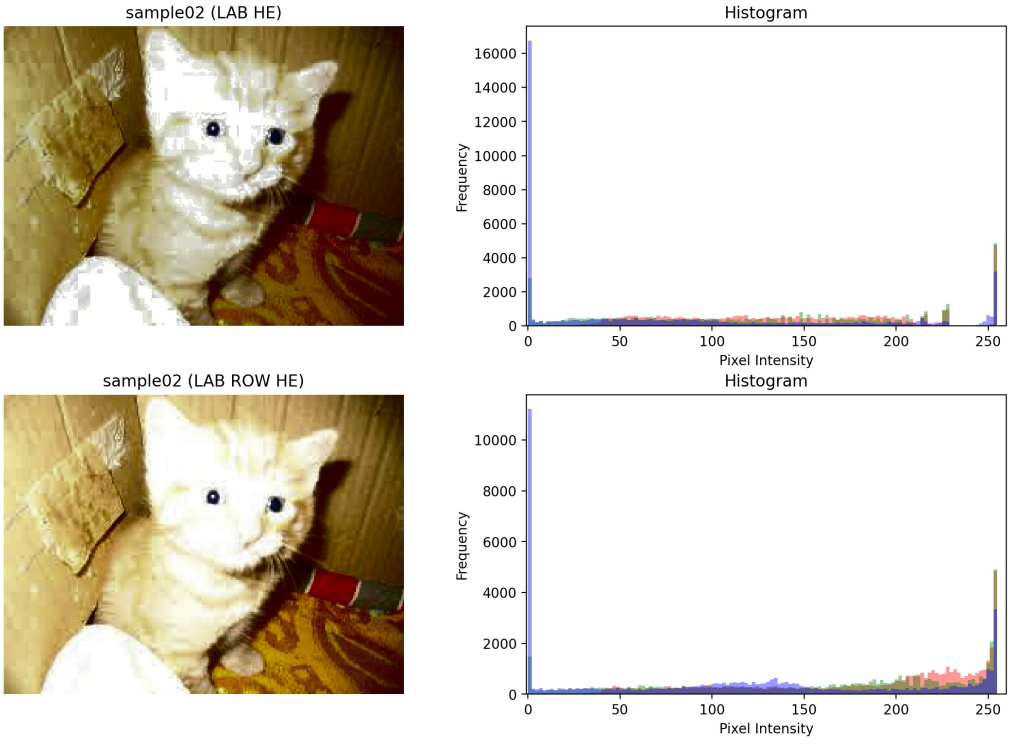


Figure 4.5: Sample02 - Point-wise and Row-based LAB HE

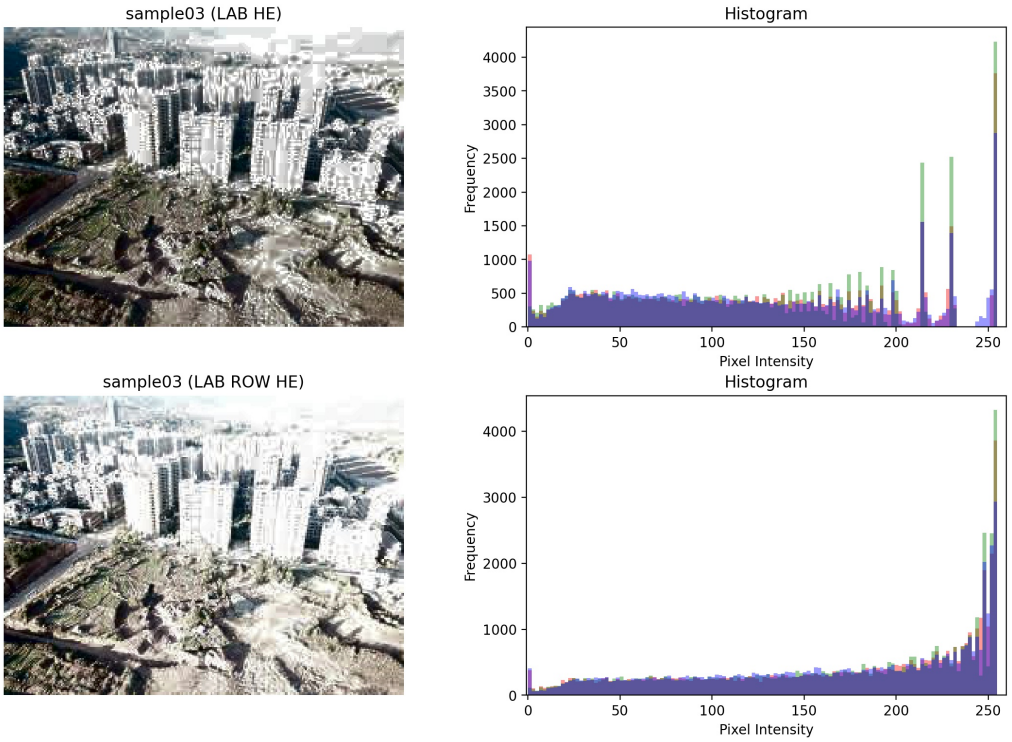


Figure 4.6: Sample03 - Point-wise and Row-based LAB HE

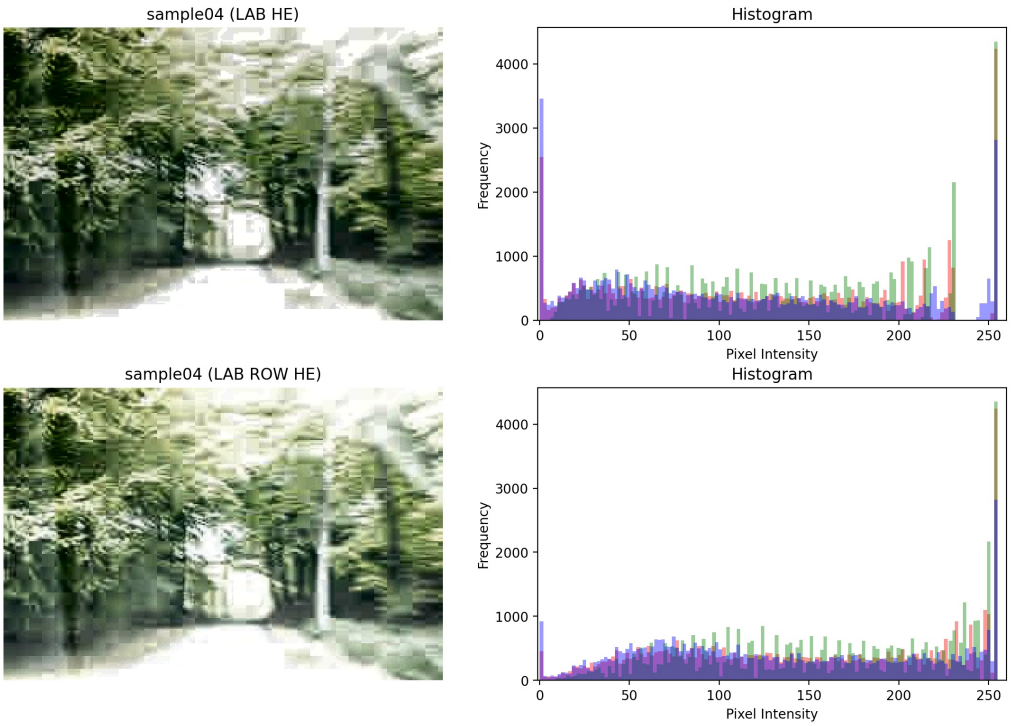


Figure 4.7: Sample04 - Point-wise and Row-based LAB HE

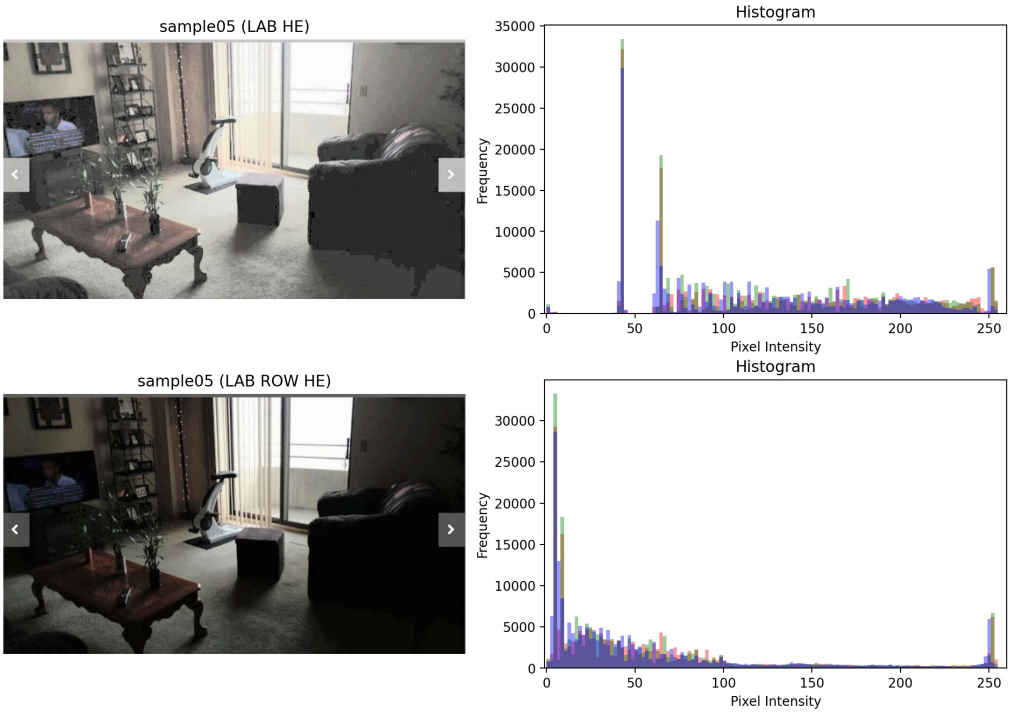


Figure 4.8: Sample05 - Point-wise and Row-based LAB HE

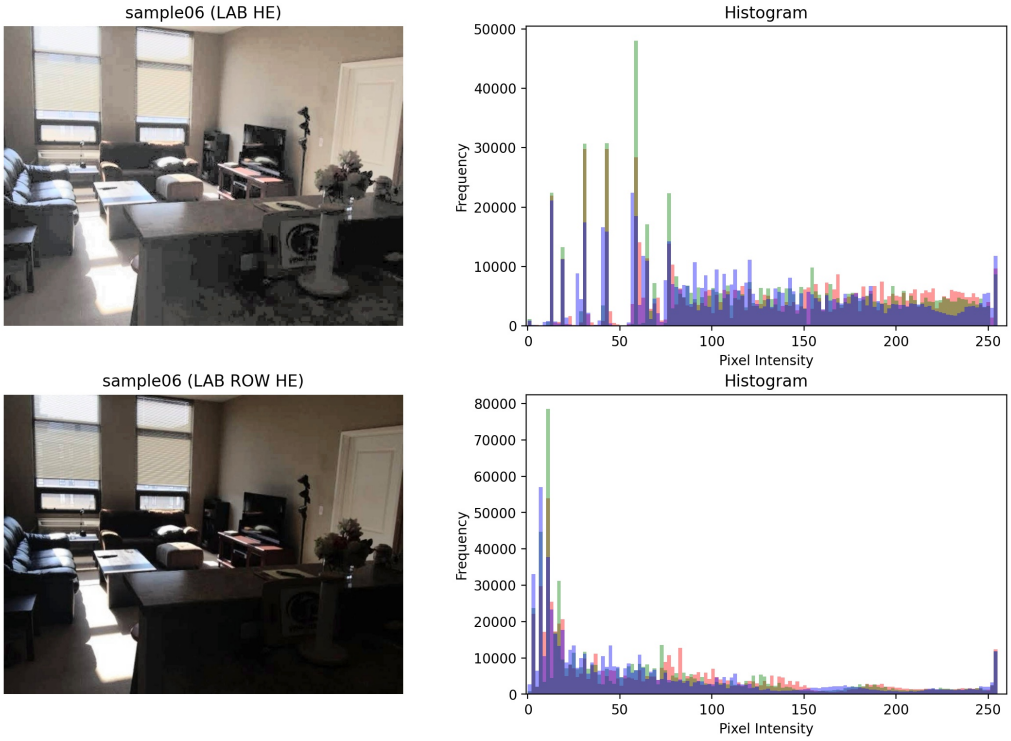


Figure 4.9: Sample06 - Point-wise and Row-based LAB HE

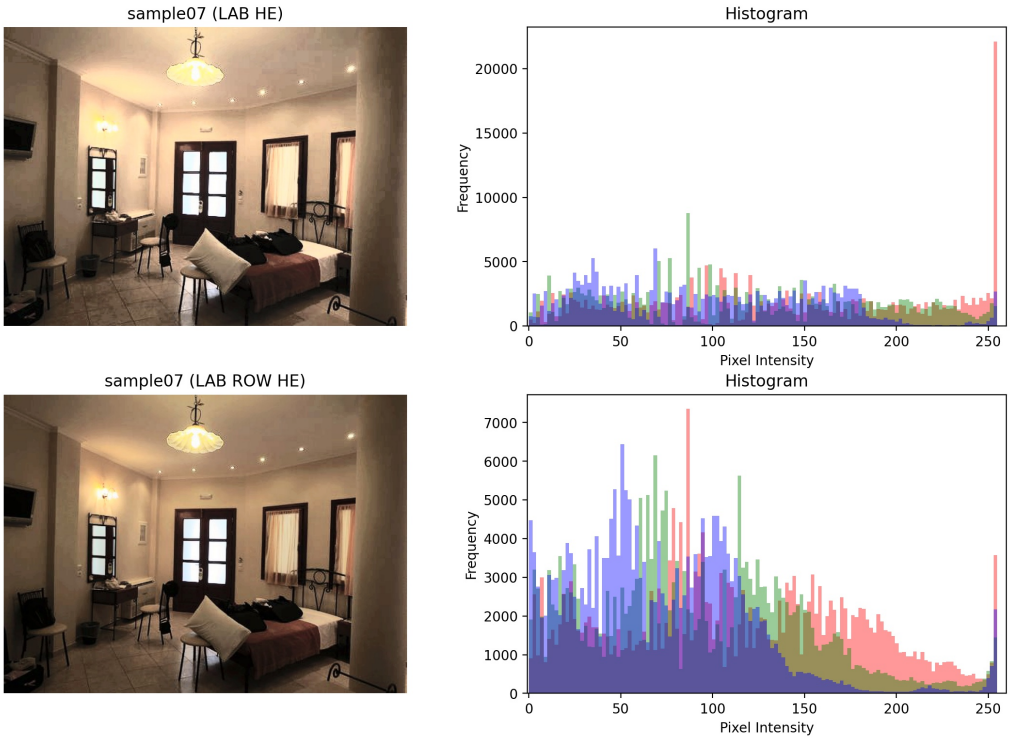


Figure 4.10: Sample07 - Point-wise and Row-based LAB HE

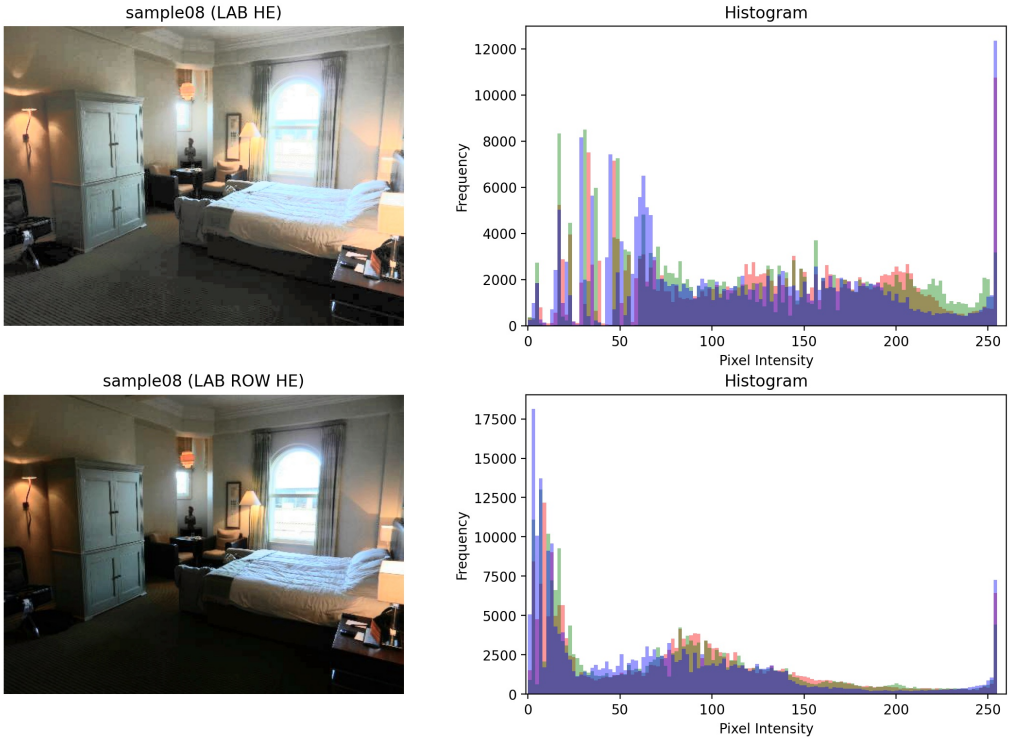


Figure 4.11: Sample08 - Point-wise and Row-based LAB HE

### 4.2.3 Result Analysis

The positive results of this implementation proved that the hypothesis cannot be rejected. The most evident improvement can be seen in sample 1 (Figure 4.4), where the empty gap of between 150 and 256 pixels in the histogram in the point-wise LAB HE is now removed and populated in the row-based LAB HE. The undesirable grey pixels can no longer be found in the new implementation. The resulting image has a very smooth color gradient. This helped to solve the original Luma HE problem as listed in Section 3.2.

The same can be seen in sample 2 (Figure 4.5) whereby the cat's face became less pixelated in the row-based HE, creating a natural look. The benefit of this context-aware HE can also be visualized in the results of darker images like sample 6 (Figure 4.9), where the noise artifacts in the foreground are removed.

## 4.3 Contrastive Limited Adaptive Equalization

Instead of performing global histogram equalization, it is possible to operate the contrast enhancement on small kernels, rather than the entire image. This is known as as adaptive histogram equalization. CLAHE (Contrast Limiting Adaptive Histogram Equalization) is one of the methods to implement such recursive region enhancement.

### 4.3.1 Source Code

```
def adaptive_he(img):
    img_lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB) # convert to LAB channels
    ch_lab = cv2.split(img_lab)

    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    ch_lab[0] = clahe.apply(ch_lab[0])

    clahe_lab = cv2.merge(ch_lab)
    clahe_lab_rgb = cv2.cvtColor(clahe_lab, cv2.COLOR_LAB2RGB)
    return clahe_lab_rgb
```

Listing 4.3: Perform CLAHE on L channel of images

### 4.3.2 Output

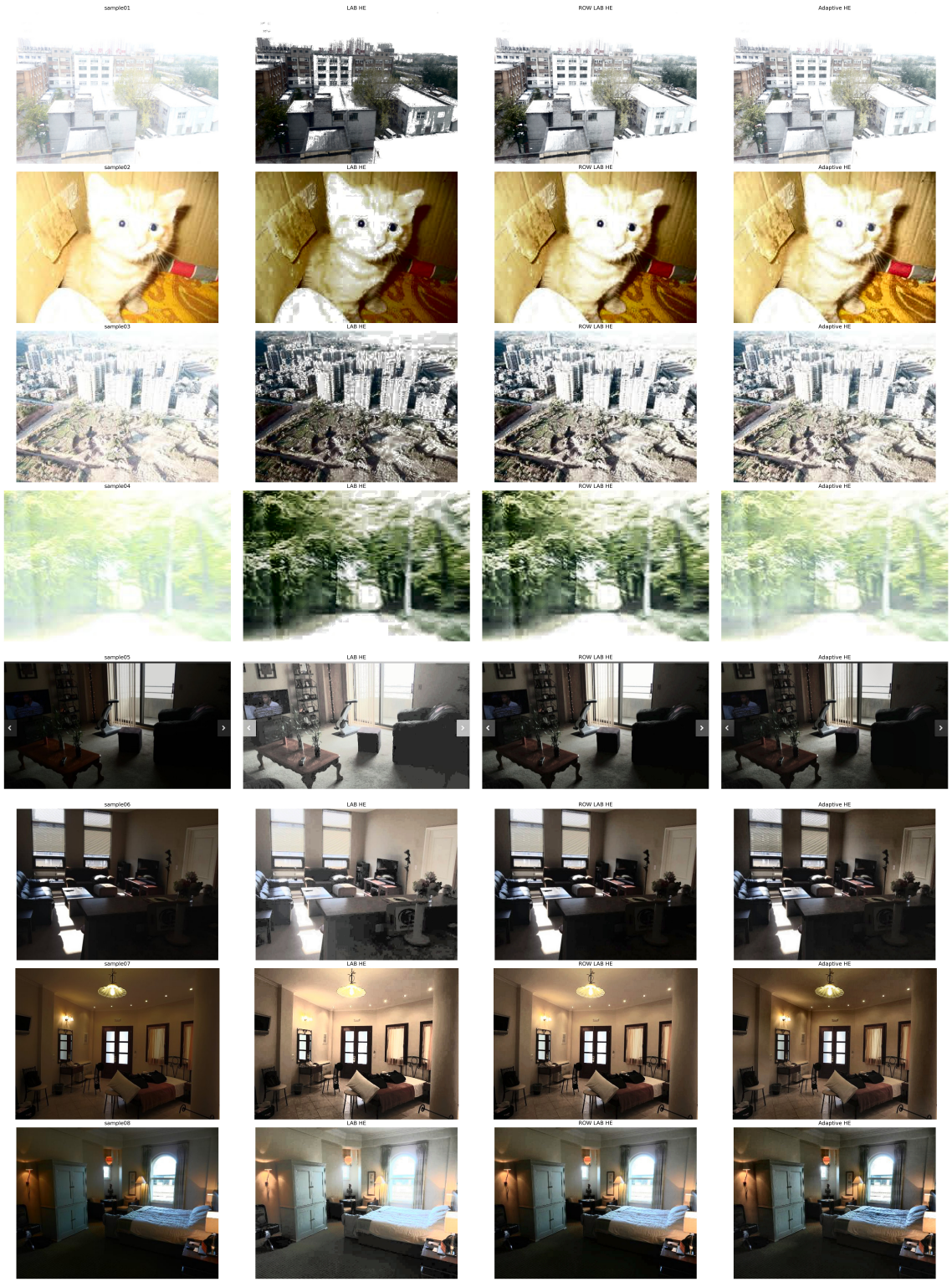


Figure 4.12: Processed images of all samples starting from original image (1st), LAB HE (2nd), Row-based LAB HE (3rd) and **AHE** (4th)

### 4.3.3 Result Analysis

The CLAHE algorithm creates more natural and less noisy outputs than the point-based LAB HE. Similar to row-based LAB HE (Section 4.2), it improved the local contrast and enhanced the definitions of edges in each region of an image.

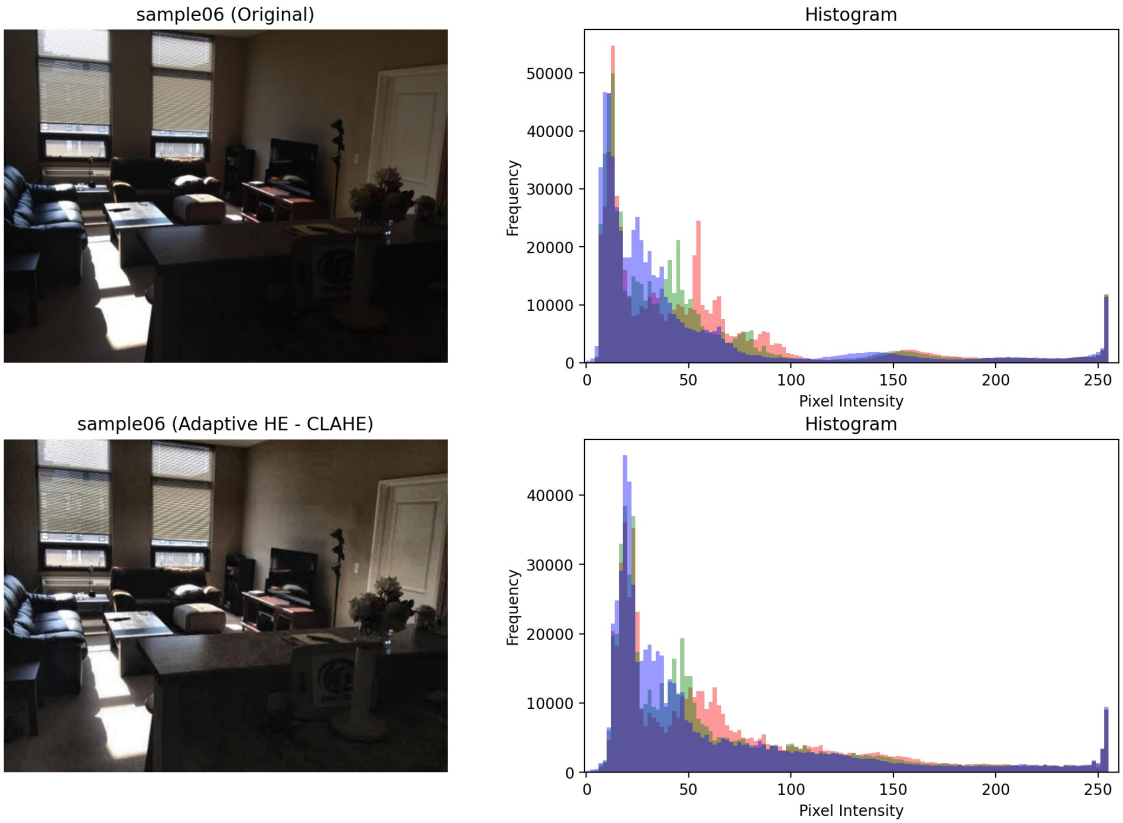


Figure 4.13: Sample06 - Original and CLAHE

The advantage of using AHE over basic global HE is most obvious in sample 6. By taking into consideration the neighbouring pixels, fine details like the lines of the roof outside the second window can be seen distinctively. Such details are lost in most other basic HE implementations.

However, the downside of using adaptive HE is that it is usually more computationally expensive and time-consuming since the recursions are performed sequentially using kernels.

# 5. Conclusion

In this report, various HE implementation methods were showcased to enhance the contrast of the given eight sample images. The three methodologies attempted were termed as basic HE, RGB HE and Luma (LAB) HE. All of their resulting processed images were thoroughly analyzed and the pros and cons of each methods were evaluated. LAB HE generally performs better than the other two basic HE by reducing noise artifacts and retaining saturation. However, there are still observations of unnaturalness and coarse gradient caused by the use of point-based HE.

Three other improvement methods to improve the overall contrast enhancement was discussed in the subsequent chapter. One of which is to pre-process the image by non-linearly adjusting the contrast before performing histogram equalization on the luminescence channel. The output of such joint method produced a flattened histogram stretched more equally amongst all pixel value bins. The second implementation was performed by applying histogram equalization on grouped localized regions such as rows. The improvement in results can be clearly seen using such row-based context-aware LAB HE. Lastly, an adaptive histogram equalization method was also trialed, and the resulting images showed finer details that were removed in other global HE methods.

# Appendices

## A Loading of images into dictionary

Listing 5.1 shows the source codes to load the eight sample images into a dictionary for easy access.

```
import glob
from pathlib import Path

def read_image_files(diry, exts = ["jpg", "jpeg"]):
    """
    Function to read image files from directory and return a sorted
    dictionary of images with filenames as indices
    """
    images = {}
    for files in [glob.glob(diry +'/*.%s' % ext) for ext in exts]:
        for file in files:
            filename = Path(file).stem
            images[filename] = mpimg.imread(file)
    return dict(sorted(images.items()))
```

Listing 5.1: Function for loading images into dictionary

# B Helper function to display image along with its histogram

Listing 5.2 shows the source codes to display image along with its histogram.

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def plot_img_and_hist(image, axes=None, bins=max_intensity, title=None,
↪  show_lines=False):
    """Plot an image along with its histogram and cumulative histogram.
    """
    if axes is None:
        fig, axes = plt.subplots(1, 2)
        fig.set_size_inches((12, 4))
    ax_img, ax_hist = axes

    # Display image
    ax_img.imshow(image)
    ax_img.set_axis_off()

    if title is not None:
        ax_img.set_title(title)

    # tuple to select colors of each channel line
    colors = ("r", "g", "b")
    channel_ids = (0, 1, 2)

    for channel_id, c in zip(channel_ids, colors):
        img_color = image[:, :, channel_id]
        # create bar histogram plot
        ax_hist.hist(img_color.ravel(), bins=bins, color=c, alpha=0.4)
        if show_lines:
            # create line histogram plot
            histogram, bin_edges = np.histogram(img_color, bins=bins)
            ax_hist.plot(bin_edges[0:-1], histogram, color=c)

    ax_hist.set_title('Histogram')
    ax_hist.set_xlabel('Pixel Intensity')
    ax_hist.set_ylabel('Frequency')
    ax_hist.set_xlim([-1, 260]) # adds some white space

    return ax_img, ax_hist
```

Listing 5.2: Function for displaying image along side its histogram