Assignment 1- Histogram Equalizer

Samuel Samsudin Ng sa0002ng@e.ntu.edu.sg
G1903488D

For this assignment, a histogram equalizer was implemented in Python with NumPy [1] computational package. As a multi-purpose equalizer, it was designed to handle multiple channels of two dimensional input array. Each input channel is equalized independently. This implementation uses Pillow [2] and scikit-image [3] packages for digital image interface and formatting.

Histogram equalization can be applied to enhance image details and contrast. In this assignment, eight images were provided as input to the equalizer. Three equalization methods were applied: (1) **RGB_Eq**: equalization of each RGB channel independently, (2) **HSV_Eq**: equalization of V channel in HSV color space, and (3) **LAB_Eq**: equalization of L channel in LAB color space.

This report is organized as follows. Section 1 provide an introduction and overview of digital image and histogram equalization. Section 2 provides the implementation details and equalization methods performed. Section 3 presents and discusses the histogram equalization results on the test images. Finally, Section 4 discusses possible improvement strategies to address the limitations of global histogram equalization.

1. Introduction

Digital images are composed of discrete elements referred to as *pixels*. Each pixel is represented by its intensity value(s), and has a unique position in the two dimensional image plane. These values are discrete, quantized values which may represent different information depending on the image format and color models.

1.1 Color Models

Color model is a method to specify color for storage, transmission and reproduction [4]. Three color model were selected for this assignment, based on the chrominance (color) and luminance (brightness) properties of the models which will be elaborated in the following paragraphs.

RGB is the most commonly used format in computer, television and video systems. RGB images are composed of three channels. Each pixel is composed of a mixture of three color elements: (R)ed, (G)reen and (B)lue, each having intensity between 0 - 255 for a common 8-bit unsigned integer representation. It does not represent the chrominance and luminance components separately. Hence this model is not very suitable for direct histogram equalization. However, this model is included for comparison with the other two models where

the chrominance and luminance are separated.

HSV represents colors by three components: (H)ue, (S)aturation and (V)alue or intensity. In this model, color is perfectly separated into its chrominance (H, S) and luminance (V) components. This allows independent adjustment of these components, which is advantageous for image enhancement and processing.

LAB, similar to HSV, represents color by its chrominance (A, B) and luminance (L) components. It was designed to approximate human perception of color.

1.2 Histogram Equalization

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.

Figure 1i shows an example of a grayscale image. The corresponding histogram is shown in Figure 1ii. From the histogram, it can be seen that the pixel intensity values are concentrated in the range of 120 and 200. This results in image that appears washed out with low-contrast and details, as all the information in the image are compressed into a small, high intensity range. Histogram equalization attempts to spread the pixels intensity distribution such that it is as uniform as possible across the full intensity range. The histogram of uniformly distributed pixel values is flat, containing the same number of pixels for all intensity values. Consequently, the cumulative histogram for uniformly distributed pixel values is linearly increasing across the range of intensity values.

Figure 1iii shows the cumulative density function of the same grayscale image. Cumulative density function (cdf) is obtained by normalizing the cumulative histogram by the total number of pixels in the image. It can be seen that the cdf of this image is non-linear across the range. Hence, the task of histogram equalization is to transform the cdf into linearly increasing function by re-distributing the values of each pixel.

2. Histogram Equalization Implementation

2.1 Algorithm

Given a discrete grayscale image $\mathbf{x}$, or in fact any two dimensional array of discrete values, the normalized histogram can be computed as

$$p_x(i) = p(x = i) = \frac{n_i}{n}, 0 \leq i < L \tag{1}$$

where i is the pixel intensity value and L is the total number of intensity level (typically L = 256 for 8-bit representation). The corresponding cumulative distribution function can be computed as

$$cdf_x(i) = \sum_{j=0}^{i} p_x(x = j). \tag{2}$$

A mapping function $\mathbf{y}$=T($\mathbf{x}$) is derived such that $\mathbf{x}$ is transformed into $\mathbf{y}$ which has a flat histogram. The resulting image will have a linear cumulative density function

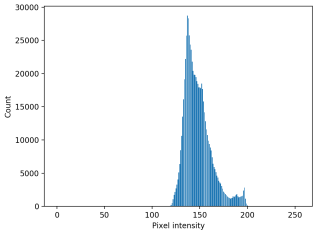
$$cdf_y(i) = iK \tag{3}$$

for some constant K. The transformation function can be calculated from cdf according to

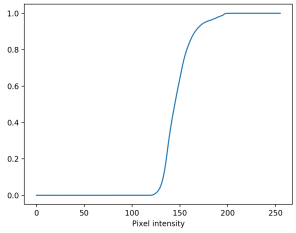
$$k = T(i) = (L-1) * cdf_x(i), 0 \leq k < L \tag{4}$$

where k is the transformed pixel value that equalizes the histogram.

(i) grayscale image

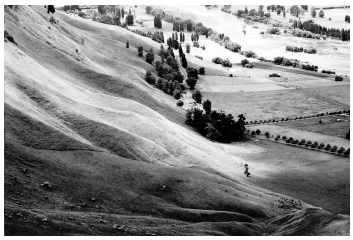


(ii) histogram

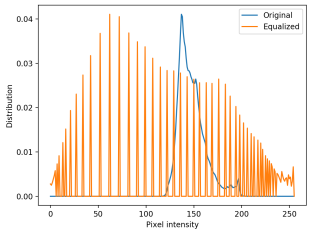


(iii) cumulative distribution

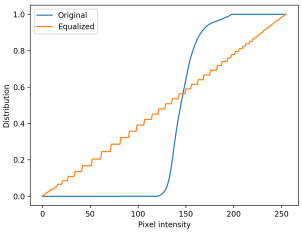
Figure 1: Original low-contrast image



(i) grayscale image



(ii) pixel value distribution



(iii) cumulative distribution

Figure 2: Equalized image

2.2 Python Implementation

The algorithm was implemented in Python using *NumPy* for numerical computation, and *Pillow* and *skimage* for image interface and color conversion. The implementation was first verified by equalizing the grayscale image presented in Figure 1i.

Figure 2i shows the output image after such transformation has been applied. Compared to the unequalized image, more contrast and details can be observed. Histogram and cdf of the equalized image are shown in Figure 2ii and 2iii. It can be seen that the pixel values are now spread over the entire range of the intensity, and the cdf is a linearly increasing function.

For this assignment, three equalization methods were tested:

1. **RGB_Eq**: histogram equalization is performed on each color channel of the RGB image independently.
2. **HSV_Eq**: RGB image is first converted into HSV image, and subsequently the histogram equalization is performed on the luminance (V) channel.
3. **LAB_Eq**: RGB image is first converted into LAB image, and the histogram equalization is performed on the luminance (L) channel.

2.3 Python Code and Usage

The Python codes are attached with this submission. The codes can also be pulled from its github repository at `https://github.com/samsudinng/cv_histogram_equalization`. The histogram equalizer is implemented in *histogram_equalizer.py*. The script to process the test images is *CV_Assignment1.py*. The test images should be placed in *jpg/* folder. An output folder *jpg/equalized/* must be created, where the equalized images will be written to.

To use the equalizer, the following command can be coded:

```
from histogram_equalizer import histogram equalizer

#to use the histogram equalizer
eq_img_array, transform_map, cdf, pdf = histogram_equalizer(img_array)
```

Input:

- *img_array*: 2D NumPy array of values to be equalized, in the range of 0 to 255 (uint8). If the array is multi-channel (eg. RGB image array), each channel will be equalized independently.

Output:

- *eq_img_array*: NumPy array of the equalized values with the same shape as the input array
- *transform_map* : the mapping array to transform the intensity values
- *cdf* : cumulative density function of the input array
- *pdf* : normalized value distribution of the input array

Dependencies: The submitted code and scripts depends on the following Python packages:

- NumPy
- PIL
- skimage

- matplotlib
- pandas

On *conda* environment, the dependencies can be installed via the provided *requirements.txt* file with the following command:

```
pip install -r requirements.txt
```

3. Results and Discussion

3.1 Analysis of Test Images

The original test images can be seen on the 'Original' column in Appendix A. It can be seen that images *sample01* to *sample04* are over-exposed, resulting in washed out and overly bright images dominated by white hue. Images *sample05* to *sample08* are under-exposed, resulting in dark image with unclear details dominated by black hue. This is confirmed by looking at the skew of the density function (normalized histogram) towards the higher and lower intensity range as shown in Figure 3.

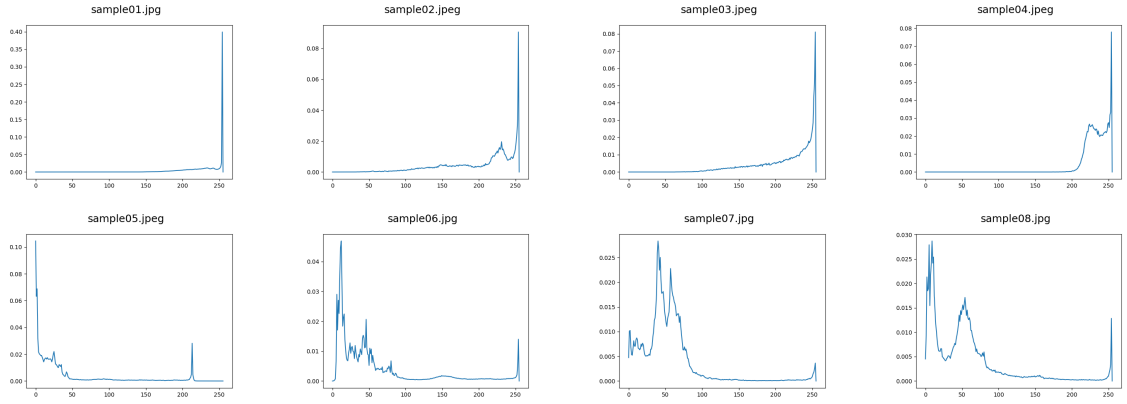


Figure 3: Distribution of pixel intensities in original test images

3.2 Equalization Results and Discussion

The results of the histogram equalization is presented in Appendix A. The comparison of the cdf before and after equalization is presented in Appendix B. In general, all three histogram equalization methods enhance the images, resulting in better contrast and details. For *RGB_Eq* method, independent equalization on RGB channels tends to create color imbalance, which is expected. This is evident especially for *sample01* to *sample04*, where the resulting color seems unnatural.

For *HSV_Eq* and LAB_Eq methods, the resulting colors are more natural as the equalization is performed on the luminance channel, which is separated from the chrominance

channels in these two color spaces. *HSV_Eq* method gives more vibrant output as observed in *sample05* and *sample07*, with the exception of *sample02* where *LAB_Eq* method produces more natural image.

As the equalization is applied based on the histogram of the whole image, it is usually referred to as *global histogram equalization*. The pro and cons of global histogram equalization are summarized as follows.

Pro: Global histogram equalization is simple and effective. It is relatively simple to implement, as elaborated in Section 2. The equalization process involves calculating the histogram, deriving the transformation function from the cumulative density function, and then mapping the values accordingly to reshape the histogram. It is effective, as shown by the results in Appendix A. Image contrast and details are enhanced.

Cons: Despite its simplicity, global histogram equalization does not take into account the local information in image. As the range of intensity are stretched, some information maybe lost or overly enhanced. Bright areas in the image might become too bright. For example, the equalized area around the window in *sample05* and *sample08*. False contours and blocking artifacts may appear, for example in *sample02* and the equalized dark areas in *sample06*.

4. Improvements

Global histogram equalization is an effective image enhancement method, aside from the few drawbacks as discussed in the previous section. A possible improvement is to perform histogram equalization adaptively on different parts of the image. The image can be divided into smaller blocks or decomposed into different sub-images. Due to the time limitation, improvements were not implemented. However, I provide several improvement methods based on literature review.

In image sub-blocking techniques, methods such as adaptive histogram equalization (AHE) and contrast-limiting AHE (CLAHE) has been very popular [5][6]. The basic idea is to transform each pixel depending on a small region around the pixel, effectively localizing the equalization. This method might be computationally expensive, hence some variations has been proposed such as using non-overlapping or semi-overlapping blocks with interpolation.

In image decomposition techniques, the original image maybe decomposed into several sub-images based on its mean or median value [7][8]. Such techniques are known as bi-histogram equalization (BHE). The image might also be decomposed into sub-bands by means of low-pass filtering [9]. Such method is able to preserve the brightness, or mean and standard deviation of the intensity of the image.

References

[1] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[2] Alex Clark. Pillow (pil fork) documentation, 2015.

[3] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[4] M. H. Asmare, V. S. Asirvadam, and L. Iznita. Color space selection for color image enhancement applications. In *2009 International Conference on Signal Acquisition and Processing*, pages 208–212, 2009.

[5] Stephen M. Pizer, E. Philip Amburn, John D. Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B. Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer Vision, Graphics, and Image Processing*, 39(3):355 – 368, 1987.

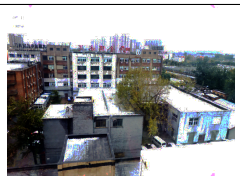
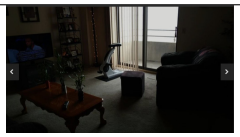
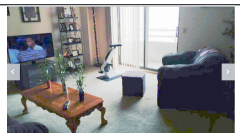
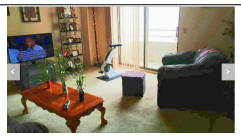
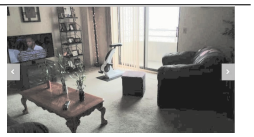
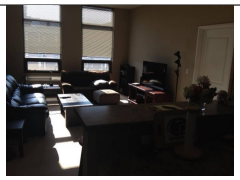
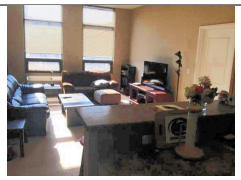
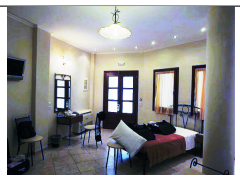
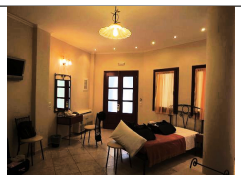
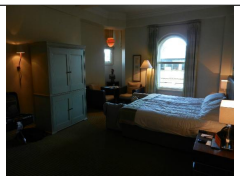
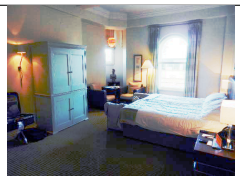
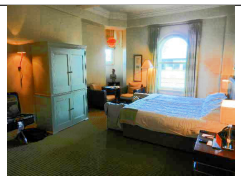
[6] S. M. Pizer, R. E. Johnston, J. P. Ericksen, B. C. Yankaskas, and K. E. Muller. Contrast-limited adaptive histogram equalization: speed and effectiveness. In *[1990] Proceedings of the First Conference on Visualization in Biomedical Computing*, pages 337–345, 1990.

[7] Yeong-Taeg Kim. Contrast enhancement using brightness preserving bi-histogram equalization. *IEEE Transactions on Consumer Electronics*, 43(1):1–8, 1997.

[8] Jing Rui Tang and Nor Ashidi Mat Isa. Bi-histogram equalization using modified histogram bins. *Applied Soft Computing*, 55:31 – 43, 2017.

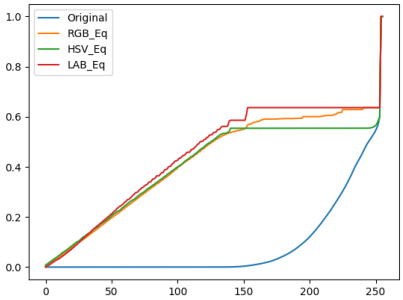
[9] Artyom Grigoryan and Sos Agaian. Gradient based histogram equalization in grayscale image enhancement. page 21, 05 2019.

Appendix A. Equalized Images

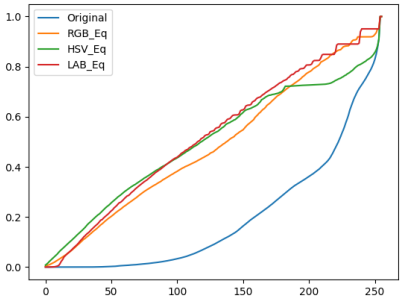
Image	Original	RGB_Eq	HSV_Eq	LAB_Eq
sample01.jpeg				
sample02.jpeg				
sample03.jpeg				
sample04.jpeg				
sample05.jpeg				
sample06.jpg				
sample07.jpg				
sample08.jpg				

Appendix B. Plots of cdf of Equalized Images

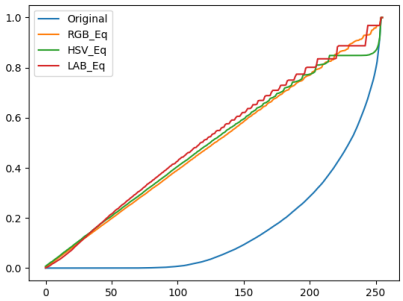
sample01.jpg



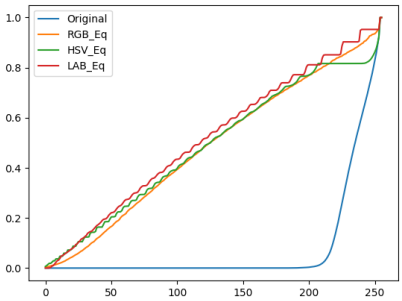
sample02.jpeg



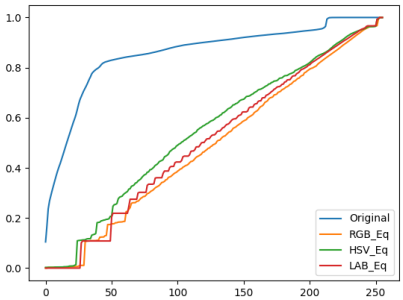
sample03.jpeg



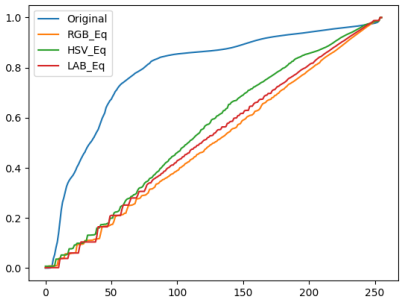
sample04.jpeg



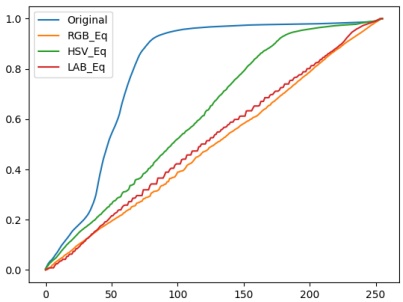
sample05.jpeg



sample06.jpg



sample07.jpg



sample08.jpg