Report

AI6121 Assignment 01 Histogram Equalization

Group Members:

Wang Yuhui (Matriculation Number: G2202262J)

Kishore Rajasekar (Matriculation Number: G2101949G)

Sean Goh Ann Ray (Matriculation Number: G2202190G)

Team Contribution:

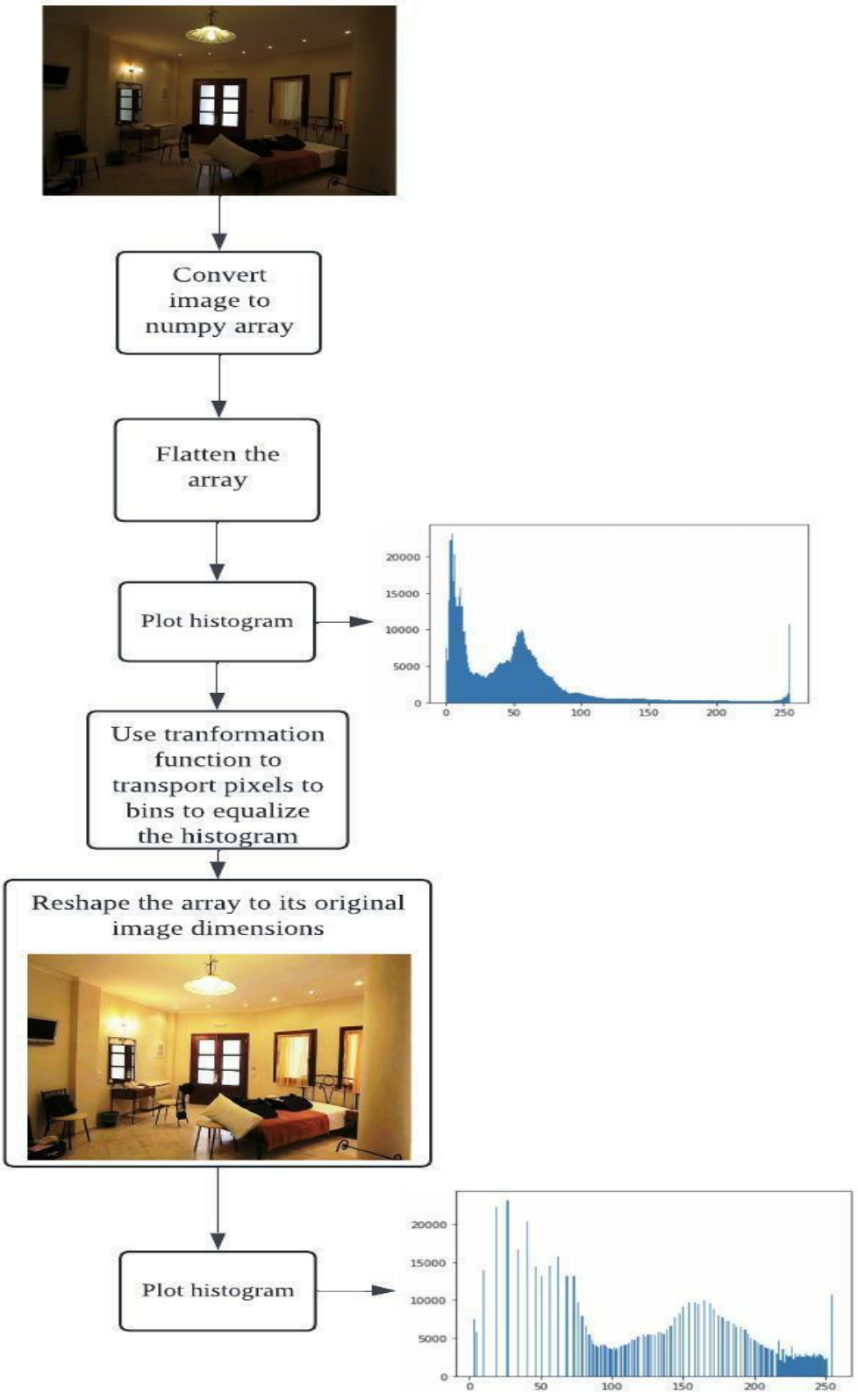
1. Wang Yuhui
    - Information Research, Report Draft & Refinement, Discussion
2. Kishore Rajasekar
    - Code - HE Algorithm Implementation, Report - Flow Chart, Code, Results, Discussion
3. Sean Goh Ann Ray
    - Code - HE Algorithm Implementation, Improvements (Code), Report - HE Algorithm Implementation, Results, Discussion, Improvements

1. Introduction

In this report, we will first give a detailed explanation of the histogram equalization algorithm we implemented, which includes the pre-processing process, introduction of the main algorithm, results display and the code to implement the algorithm. Then we analyze the advantages and disadvantages of this algorithm. Finally, we will try to give possible improvements to this algorithm.

2. HE Algorithm Implementation

2.1 Flow chart



2.2 Pre-processing

The essence of histogram equalization is to make the distribution of pixel intensities uniformly distributed, which means that we need to do some calculations on the original pixels of the image. So, the first thing is to convert the opened image into a numpy array and then flatten the array. The flattening of the array is necessary as the numpy array from the image is a 3D array, consisting of a 2D pixel map and the red, green and blue (RGB) values of each pixel. Furthermore, 1D arrays require less computational time and memory as compared to multi-dimensional arrays. .

2.3 Main Algorithm

Having the 1-D pixel intensity array, the next step is to loop through this array to count the number of occurrences per intensity (ranging from 0 to 255) in order to generate the histogram. Then we can calculate the probabilities for each bin (intensity value) by simply dividing the number of occurrences in every bin by the total number of occurrences.

After that is the most important step: we should use the transformation function. Here we refer the function from lecture: $s_k = (L - 1) \sum_{j=0}^{k} p_j$ where L is the total number of bins and p_j is the probabilities for each bin which we have calculated in the previous step. Then we round every s_k because we want an integer rather than a float for the new intensity values.

The final step to get the new image is to simply get the value from the cumulative rounded sum of bins for every index in the flattened image array then we get the 1-D array for the image we want to get. Lastly, we reshape this array back into the original 3D array shape before converting the 3D array back into an image. .

2.4 Results Observation

After implementing our HE algorithm to the 8 sample pictures provided, we can see that the contrast in all of the images improved. Although some pictures are still unsatisfactory, there is no doubt that all of them looked better than the original.

Below are all the 8 pictures after implementing the HE algorithm:

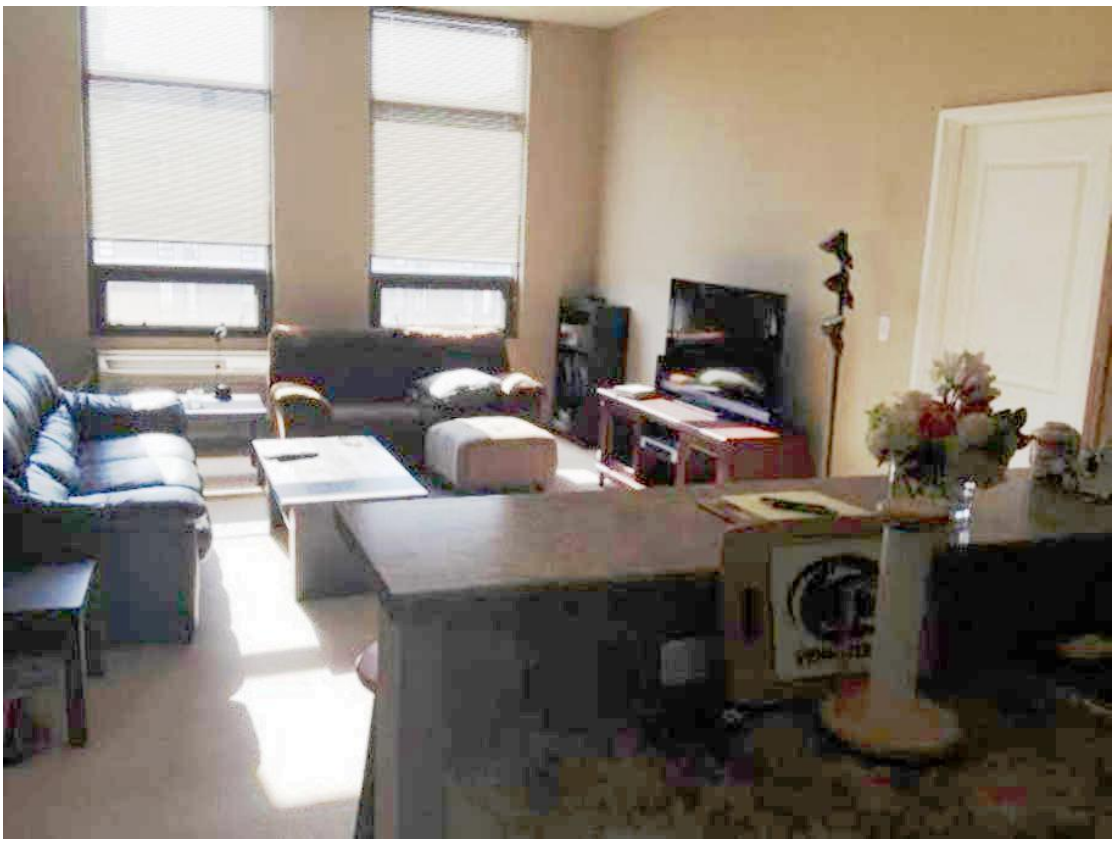
HE_sample01

HE_sample02

HE_sample03

HE_sample04

HE_sample05



HE_sample06

HE_sample07

HE_sample08

2.4 Code

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

img = Image.open('/content/gdrive/MyDrive/AI6121/sample08.jpg')

# display the image
plt.imshow(img)

img = Image.open('/content/gdrive/MyDrive/AI6121/sample08.jpg')

# display the image
plt.imshow(img)

# create our own histogram function
def get_histogram(image, bins):
    # array with size of bins, set to zeros
    histogram = np.zeros(bins)

    # loop through pixels and sum up counts of pixels
    for pixel in image:
```

```
        histogram[pixel] += 1

    # return our final result
    return histogram

# execute our histogram function
hist = get_histogram(flat, 256)
plt.plot(hist)

#Calculating the probabilites
p_hist = hist/hist.sum()

#finding sk
bins=256
mod_hist = np.zeros(bins)

for i in range(bins):
  mod_hist[i] = (bins-1)*sum(p_hist[:i+1])
  # print(255*sum(p_hist[:i+1]))

#rounding the bin numbers
# round_mod_hist = [round(num) for num in mod_hist]
round_mod_hist = mod_hist.round().astype('uint8')

# get the value from cumulative rounded sum of bins for every index in flat, and set that as img_new
img_new = round_mod_hist[flat]
# put array back into original shape since we flattened it
img_new = np.reshape(img_new, img1.shape)

# set up side-by-side image display
fig = plt.figure()
fig.set_figheight(15)
fig.set_figwidth(15)

fig.add_subplot(1,2,1)
plt.imshow(img)

# display the new image
fig.add_subplot(1,2,2)
plt.imshow(img_new)

plt.show(block=True)
ims = Image.fromarray(img_new)
ims.save("HE_sample08.jpg")

img_n = np.asarray(img_new)

# put pixels in a 1D array by flattening out img array
flatten = img_n.flatten()
```

```
# show the histogram
plt.hist(flatten, bins=256)
plt.show()
```

3. Discussion

3.1 Advantage

The advantage of the HE algorithm is that it improves the image quality by increasing the global contrast of the image. It can make a bright image darker (result 01-04) and a dark image brighter (result 05-08), all of the sample images are obviously clearer after implementing the HE. Details that were not visible due to the low contrast of the original images could also now be observed with the increased contrast of the processed images.

3.2 Disadvantage and possible causes

The biggest disadvantage is color distortion. Especially for those samples that have high brightness (Sample 01-04). One possible cause is in our algorithm, we only care about how to make the pixel intensity uniformly distributed. However, simply changing the pixel intensity may cause other unexpected problems. Because in fact, a colored image has three channels Red, Green and Blue. HE might bring a pixel's R component higher but lower its G and B components, causing that pixel to change its color completely when the RGB components are put back together. Of course, another simple guess may also be that the extent of color distortion depends on the quality of the original image. By comparison, we can find the quality of samples 05-08 is better than samples 01-04. Correspondingly, there is less color distortion in the results. Another disadvantage is that the algorithm does not differentiate between foreground and background in an image. It can't identify important aspects in an image and work towards highlighting that in particular

4. Improvements

4.1 HE Algorithm on HSV Color Space

Since HE is meant to improve contrast by equalizing the pixel values, one improvement that can be done is to perform the HE algorithm in the HSV color space. The Hue, Saturation, and Value (HSV) color space mimics how the human eyes perceive color more closely as compared to the RGB color space. It is different from the RGB color space in that hue determines the color on the RGB circle, saturation determines the amount of color used, and value determines the brightness of the color, which is the amount of black present. The HE algorithm can be performed on only the value to avoid color distortion. The value array ranges from 0 to 255, where 0 would mean pure black and 255 would mean pure color (no black), similar to how a HE algorithm would work on a grayscale image.

4.1.1 Code

```
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt

# filename = "sample01.jpg"
# filename = "sample02.jpeg"
# filename = "sample03.jpeg"
# filename = "sample04.jpeg"
# filename = "sample05.jpeg"
# filename = "sample06.jpg"
filename = "sample07.jpg"
# filename = "sample08.jpg"
save_filename = "HSV_HE_" + filename

# import image and split into HSV arrays
img = Image.open(filename)
img= img.convert('HSV')
img = np.asarray(img)
h = img[:,:,0]
s = img[:,:,1]
v = img[:,:,2]

# Total no. of pixels
height = len(np.asarray(img))
width = len(np.asarray(img)[0])
N_pixels = height * width

# No. of bins
L = 256

# Histogram
histogram_v_array = np.bincount(v.flatten(), minlength=L)

# Probability
P_v_array = histogram_v_array / N_pixels

# Transformation function s_k
s_k_v_array = (L-1) * np.cumsum(P_v_array)

# Equalized histogram
eqhistogram_v_array = np.floor(s_k_v_array).astype(np.uint8)

# flatten image V array into 1D
im_v_flat = np.asarray(v).flatten()

# replace pixel values with values from equalized histogram to equalize
eq_img_v_flat = [eqhistogram_v_array[p] for p in im_v_flat]

# plot old vs new histogram for V array
plt.hist(v.flatten(),bins=L,label='Original V')
```

```
plt.hist(eq_img_v_flat,bins=L,label='New V')
plt.legend(loc='upper right')
plt.show()

# reshape equalized pixel values back into original V array shape
eq_img_v_array = np.reshape(np.asarray(eq_img_v_flat), np.asarray(v).shape)

# merge HSV arrays into one and save as a RGB image file
eq_img = np.zeros((height,width,3), 'uint8')
eq_img[:,:,0] = h
eq_img[:,:,1] = s
eq_img[:,:,2] = eq_img_v_array

eq_img = Image.fromarray(eq_img, mode='HSV')
eq_img = eq_img.convert('RGB')
eq_img.save(save_filename)
```

4.1.2 Result Observation

It is observed that the HE algorithm can work in the HSV color space, however, it does not perform well where the original image has low contrast due to bright pixels, as seen in samples 01-04, where the bright pixels still remain bright.

However, it performs fairly well where the original image has low contrast due to dark pixels, as seen in samples 05-08, where there is little to no color distortion.

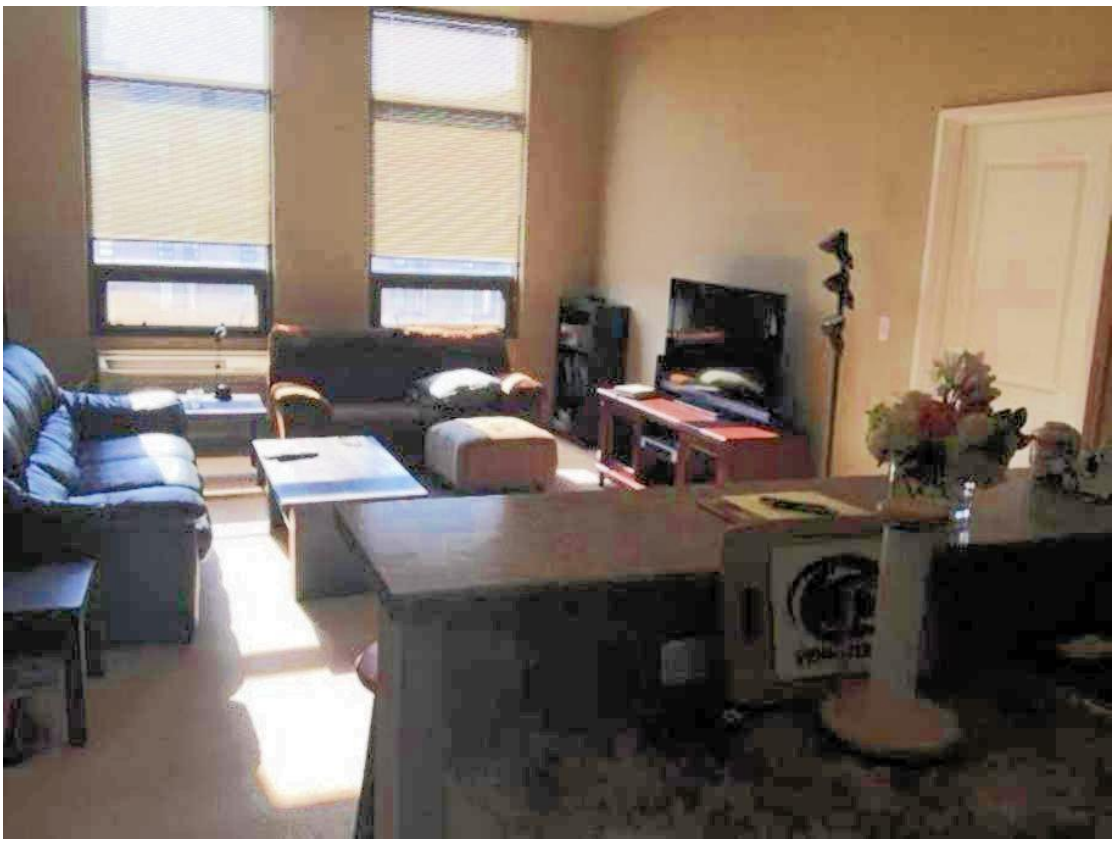
HSV_HE_sample01

HSV_HE_sample02

HSV_HE_sample03

HSV_HE_sample04

HSV_HE_sample05

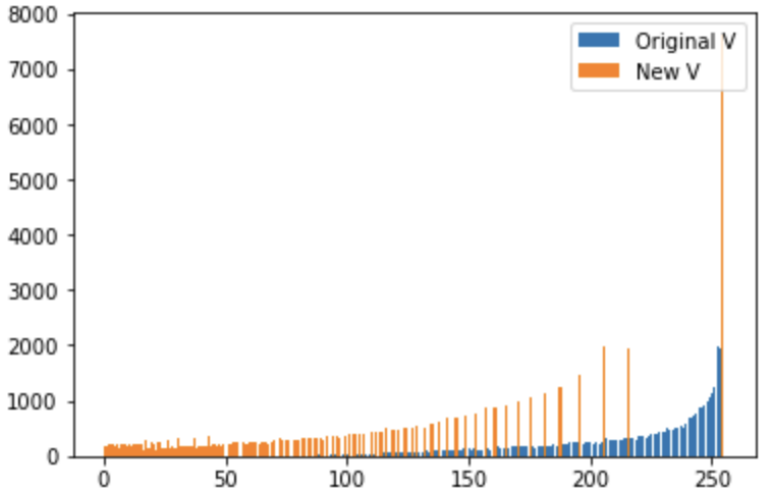


HSV_HE_sample06

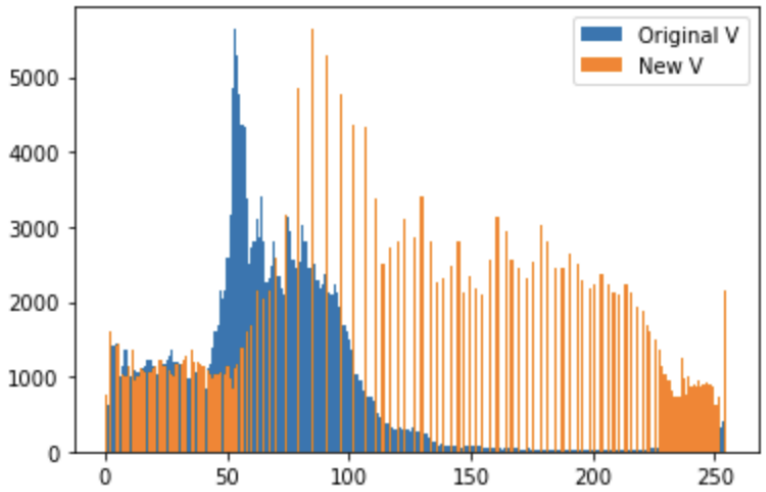
HSV_HE_sample07

HSV_HE_sample08

Below are the histogram for sample03 and sample07 respectively, showing how the algorithm worked in the HSV space for bright and dark images.



HSV_Histogram_sample03



HSV_Histogram_sample07

As seen in the histogram for image sample 03, the HE algorithm did spread the pixel values, however, the brightest pixels (value of 255) remained the same, showing that it does not work too well with images with a large number of bright pixels.

On the other hand, the HE algorithm worked very well in spreading the pixels value as seen in the histogram for image sample 07.