# CENG466-Take Home Exam 1

1st Seda Civelek
*METU Computer Engineering*
*No: 2237147*
seda.civelek@metu.edu.tr

2nd Kağan Erdoğan
*METU Computer Engineering*
*No: 2098986*
e209898@metu.edu.tr

***Abstract*—This document is a report for Ceng466-Fundamentals of Image Processing course's Take Home Exam 1. This homework is given to learn histogram processing, histogram matching, edge detection algorithms and image interpolation. The answers of each question can be found in subsections respectively. In order to implement algorithms, python is used. To run the code for part1 and part2 write following commands:**

- **python the1_part1.py inputimage referenceimage**
- **python the1_part2.py inputimage**

***Index Terms*—histogram processing, histogram matching, canny edge detection, sobel filters, image interpolation**

## I. Histogram Matching

In the first question, it is asked to do histogram matching on input image with reference images. Firstly, histogram processing with equalization is applied to both images, then histogram matching is applied. Steps are explained in detail in the subsections respectively.

### A. Histogram Processing

In histogram processing, firstly we count number of pixels having intensity color from 0 to 255. It can be see that, A1.jpg has more pixels that are close to intensity value of 255. Unlikely, A2.jpg has more pixels that are close to intensity value of 0.

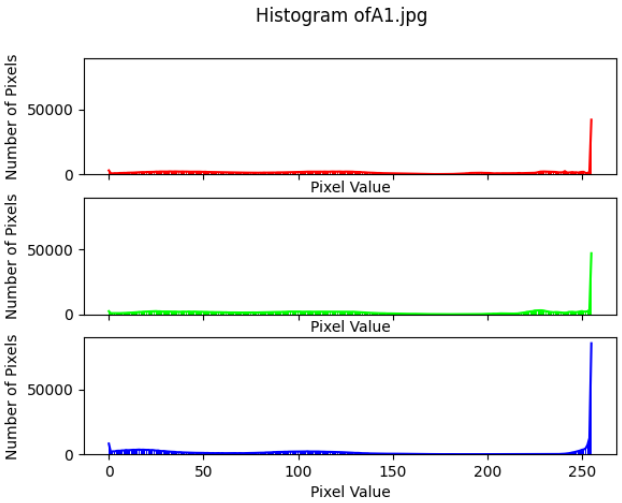


Fig. 1. Histogram of A1.jpg

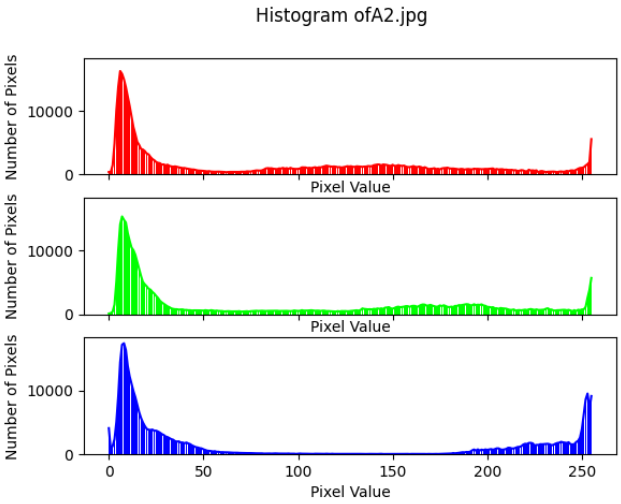


Fig. 2. Histogram of A2.jpg

After we computed histogram of both two images, we applied cumulative distribution and equalization. You can see cumulative equalized histograms of "A1.jpg" and "A2.jpg".

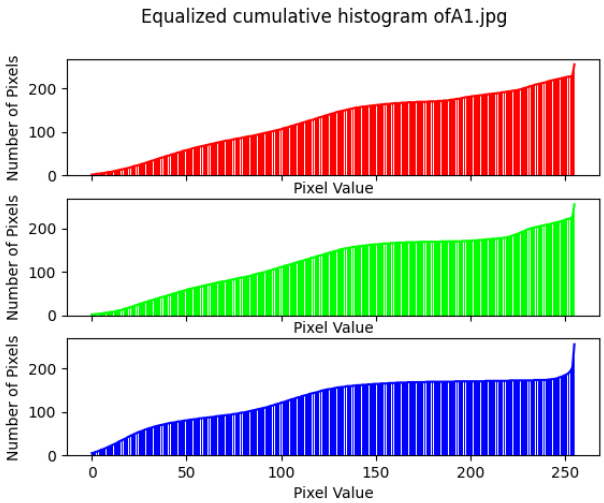


Fig. 3. Equalized Cumulative Histogram of A1.jpg

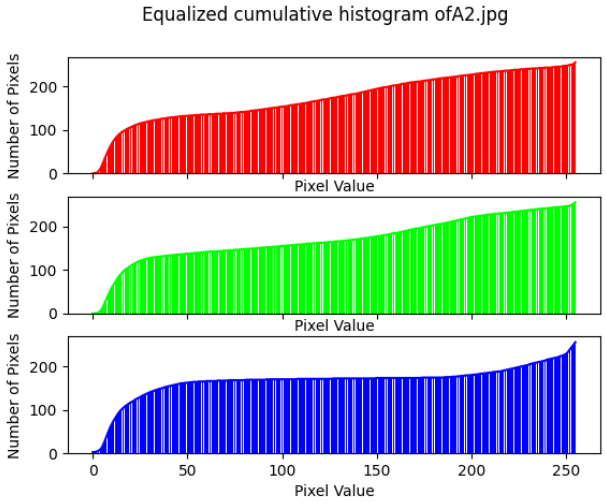


Fig. 4. Equalized Cumulative Histogram of A2.jpg

### B. *Histogram Matching*

In histogram matching, we modified the given input image based on the contrast of reference image. When you run the code you can simply give this commands(python the1_part1.py inputimage referenceimage) :

- python the1_part1.py A1.jpg A2.jpg
- python the1_part1.py A2.jpg A1.jpg

For matching, we simply mapped each pixel of input image based on the value of its equalized cumulative histogram to the value of reference image. If there is no mapping from input to reference, we mapped input to its closest value. You can see both matched equalized histogram for the cases that A1 is input and A2 is reference, and A2 is input and A1 is reference respectively.

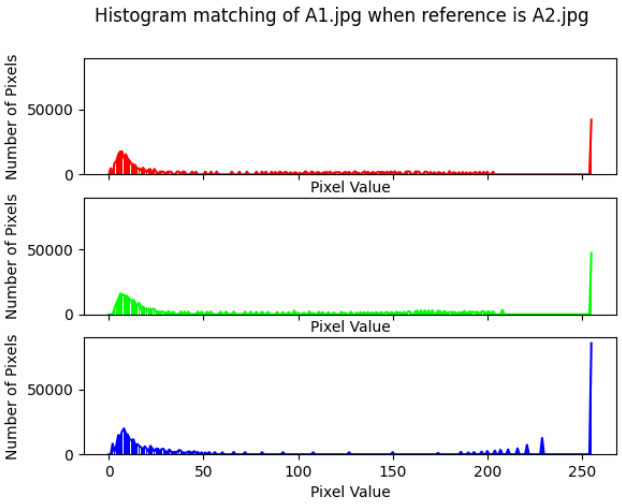


Fig. 5. Histogram matching of A1.jpg when reference is A2.jpg

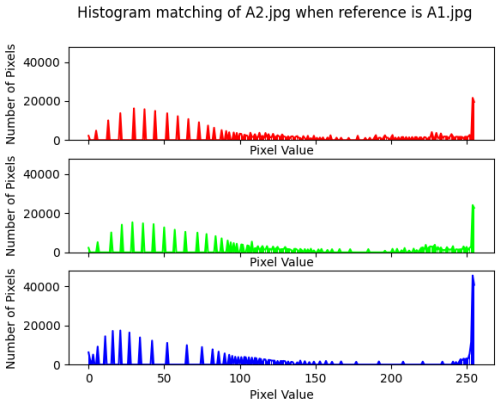


Fig. 6. Histogram matching of A2.jpg when reference is A1.jpg

If we look at the original histograms and matched histograms of A1.jpg and A2.jpg, we can conclude that A1.jpg has more pixels that are close to intensity value of 0 than before (Fig1.-Fig5.). Also, A2.jpg has less pixels that are close to intensity value of 0 than before (Fig2.-Fig6.). Finally, we obtained output images as follows:


Fig. 7. Transformed A1.jpg when reference is A2.jpg


Fig. 8. Transformed A2.jpg when reference is A1.jpg

## II. Edge Detection

In the second question, it is asked to implement a convolution function to do filtering. Then, it is asked to implement a canny edge detection algorithm, change the parameters and observe the affect of the changes.

### A. Convolution

Firstly, we implemented our convolution function. We used zero padding for the pixels out of boundaries.We firstly tried to implement it for fixed sized filters (3x3), but in the canny edge detection phase, we need to implement 5x5 sized filter. So, we decided to change our convolution implementation and implement a function for various sized filters.

### B. Canny Edge Detection

Firstly, we applied Gaussian smoothing to remove the noise in image. Thus, we could apply edge tracking more correctly. Gaussian smoothing function has two parameters as kernel size and sigma. In the lecture notes, Gaussian function is given as following:

$$G(x,y) = \frac{1}{\sqrt{2\pi}\sigma} exp \frac{-1}{2\sigma^2}(x^2+y^2) \quad (1)$$

However,when we applied this function, we get unexpected results such as mixing of colors.So, we applied Gaussian function as follows:

$$G(x,y) = \frac{1}{2\pi\sigma^2} exp \frac{-1}{2\sigma^2}(x^2+y^2) \quad (2)$$

If we apply 3x3 kernel size with sigma value 1, we get the following image:

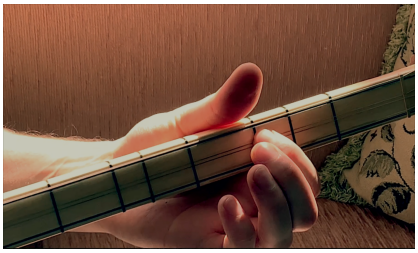

Fig. 9. Gaussian smoothing with 3x3 kernel size

If we apply 5x5 kernel size with sigma value 1, we get the following image:

Fig. 10. Gaussian smoothing with 5x5 kernel size

We can conclude that,if we applied bigger kernel, we obtain less noisy image. Also, if we change sigma value, we get different results too. If we apply 5x5 kernel size with sigma value 3, we get the following image: Secondly, we applied

Fig. 11. Gaussian smoothing with sigma value 3

Sobel filtering to find magnitude and direction gradient. When we applied vertical and horizontal filters:

$$w_v = \begin{bmatrix} 1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad w_h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

We obtained $g_v$ and $g_h$ which are filtered images with $w_v$ and $w_h$ respectively. In the lecture notes, direction of an edge

Fig. 12. Image filtered with $w_v$

Fig. 13. Image filtered with $w_h$

pixel is computed as following:

$$\Theta(x,y) = \arctan \frac{g_v(x,y)}{g_h(x,y)} \tag{3}$$

However, when we applied it, it didn't work. So, we searched and see that the correct equation for direction is the following:

$$\Theta(x,y) = \arctan \frac{g_h(x,y)}{g_v(x,y)} \tag{4}$$

We implemented second theta function and obtained better result.

Thirdly, we tried to implement Non-maximum Suppression. Our algorithms remove weak edges by comparing the pixel with the pixels in positive and negative gradient directions. Then, we implemented double thresholding with predefined low and high values. We assign pixel as strong if pixel value is bigger than high value. We assign pixel as weak if pixel value is smaller than low value. Finally, we kept track of edge by hysteresis. In the steps that are Non-maximum Suppression, double thresholding and hysteresis, since we didn't cover them in class, and there is no information in lecture notes, we got help from the article that our professor recommended [1]. You can see the output images below.

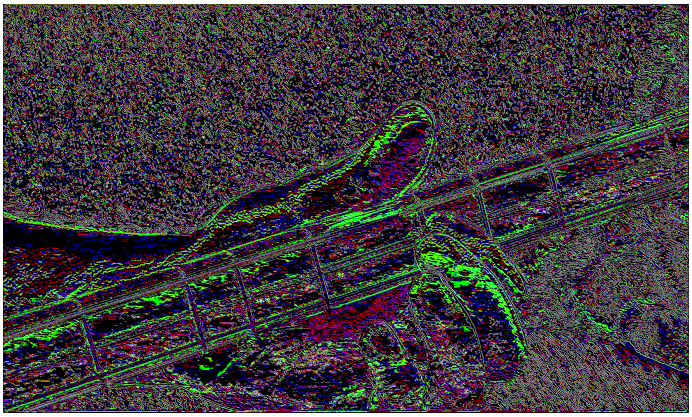

Fig. 14. Edge Detection of B1.jpg

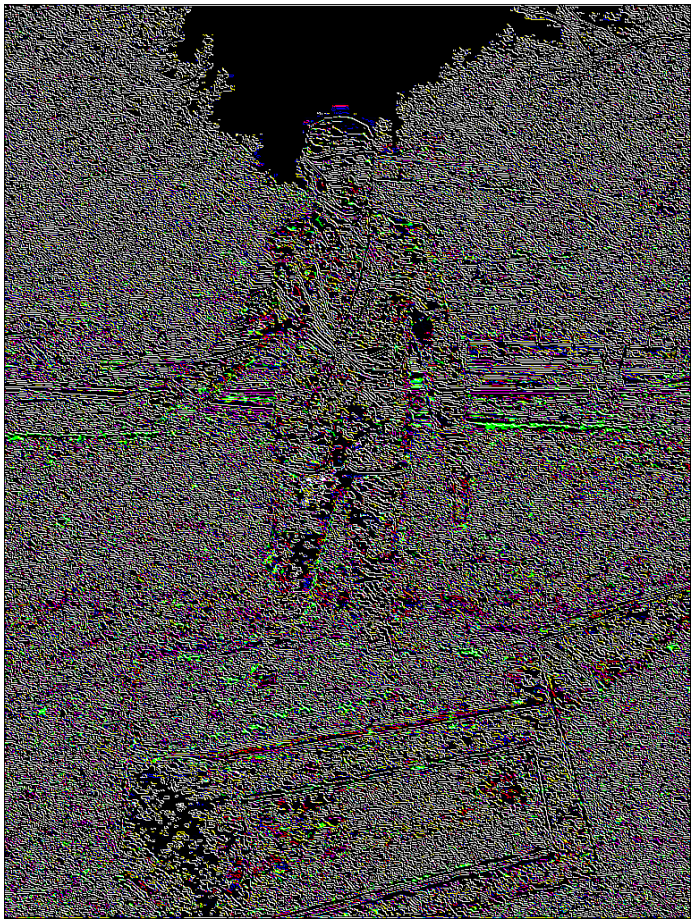

Fig. 15. Edge Detection of B2.jpg

## III. Interpolation

Firstly, to interpolate images we have determined the new coordinates of pixels for 8x8 images. Since we have 4x4 images we should multiply the coordinates by 2. For example, if our original coordinates are (0,1), then the new coordinates will be (0,2) in the new image. After identifying new co-ordinates, we have applied following rules to find missing brightness values of pixels according to nearest neighbor method :

- Assign top value
- If top value is empty, assign left value
- If left value is empty, assign left diagonal value

Here are the result of R,G,B image patch respectively (where black cells correspond to 0, and white cells correspond to 1) :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

The final images are very similar to original images as we expected. The only difference is number of pixels. In 8x8 images a pixel is represented with 2x2 square but pattern is the same. We think if we do not multiply with an integer value to obtain big image, the pattern will be different with respect to original image.

To find connected components in different neighborhood systems, we have examined the pixels that have brightness value 0 (black pixels). To understand how many unconnected components in an image we should understand adjacency concept in different neighborhood connectivity systems. If we can not find a path between components, then we can say that components are unconnected. We can draw a path between adjacent pixels. In 4-neighborhood system we can move only upward, downward, rightward and leftward. But in 8-neighborhood system we can move diagonal also. On the other hand, in mixed adjacency two pixels are adjacent if they have 4-adjacency or have 8-adjacency and they do not have mutual 4-adjacent pixel. In light of this information, here are the results for number of components in different neighborhood systems :

- In 8-neighborhood, R:1 G:1 B:1
- In 4-neighborhood, R:1 G:3 B:2
- In mixed neighborhood, R:1 G:1 B:1

## REFERENCES

[1] Sahir, S. (2019, January 27). Canny Edge Detection Step by Step in Python-Computer Vision. Retrieved November 29, 2020, from https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123