CENG 466 Fundamentals of Image Processing HW1

Andaç Berkay Seval
2235521

Asrın Doğrusöz
2380301

***Abstract*—This report is prepared for the first homework of CENG 466.**

I. Introduction

This report includes methodology and rationale behind the algorithms that are implemented in the homework, difficulties and errors encountered in the design, implementation, experimentation stages and their solutions, analysis and comments on the results and the requirements of the code.

II. Requirements

This homework is done with the help of some libraires. Therefore, there are 2 requirements for the code:

- scikit-image (skimage)
- matplotlib

The details and the purposes of these libraries will be explained in the next section.

III. Implementation

A. Affine Transformation

Firstly, the rotation function was implemented without any library. However, since we are allowed to use external libraries, we switched to use "skimage.transform.rotate" function to achieve a more compact and readable form. That rotate function takes input image as ndarray, angle as float as rotation angle in degrees in counter-clockwise direction, order as int in range [0-5] as the interpolation type, 1 is bilinear interpolation and 3 in bicubic interpolation and then, it rotates the image by given angle around its center with applying desired interpolation.

The design of the algorithm was not hard. When interpolation type is linear, we chose the order of 1 and when it is cubic, we chose 3. Also, we took - degree for rotation in order to apply that degree in clockwise direction. Read image and write image functions were also implemented with the help of skimage. Thus, "skimage.io.imread" and "skimage.io.imsave" functions were used. In the first implementation step, there was a warning for all the generated images "Lossy conversion from float64 to uint8. Range [0, 1].". This warning occurred since rotate function changes the pixel value range from uint8 to float64 between [0, 1]. Therefore, we had to map these values again to uint8 with the help of "skimage.img_as_ubyte" function which converts image to unsigned byte format with values [0, 255]. Furthermore, the warning disappeared.

"a1.png" image has a shape of (256, 256, 3). Thus, it is already a low resolution image. When we looked at the 45 degrees rotation outputs, there were some differences between bilinear and bicubic interpolated images. Bicubic interpolated image has some slight improvements over bilinear interpolated image. The difference between them is not huge for human eyes since "a1.png" already has a low resolution. However, when we compared the pixel values, there were differences between bilinear and bicubic interpolations. Then, when we compared the 90 degrees rotation outputs we noticed that there were not any pixel value differences between bilinear and bicubic interpolated types. All pixel values were the same. The reason behind that is probably in 45 degrees rotation, some source pixels mapped into the same destination pixels due to roundings. However, in 90 degrees rotations, source to destination maps are exact. The rotation matrix in counter-clockwise direction is $\begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

"a2.png" image has a shape of (4032, 3024, 3). Thus, it is already a high resolution image. When we looked at the 45 degrees rotation outputs, there were some differences between bilinear and bicubic interpolated images. When we compared the pixel values, there were differences. However, since "a2.png" already has a high resolution, the differences are hard to see with human eyes but bicubic interpolated image has some slight improvements over bilinear interpolated image.

B. Histogram Equalization

Histogram equalization function was implemented with the help of "skimage.exposure.equalize_hist" function. It takes the input image and returns histogram equalized version of the image. Saving histograms of the images was implemented with the help of "matplotlib" library. Thanks to that, histograms of the images can be easily plotted and saved.

The design of the algorithm was not hard. However, there was an error in the histograms of the images in the first implementation since "matplotlib" saves the previous histogram and cannot plot the next one onto it. We did not know that yet when we realized that, it was simply solved by "plt.clf" function that clears the figure. Therefore, all the

problems were solved.

When we compared the histograms of original image and enhanced image, we noticed that histogram equalization flattened the original histogram. Thus, equalized histogram was closer to uniform distribution. Then, we compared the original image "b1.png" and enhanced image and we noticed that histogram equalization increased the contrast of the original image. The most obvious difference was that the darker pixels get brighter so the details were more detectable.

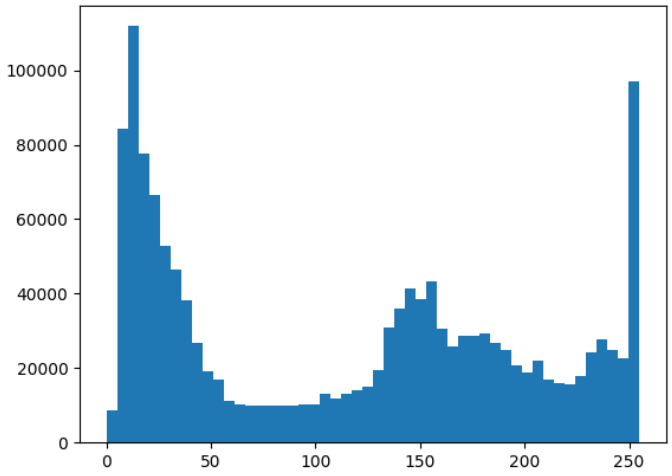

Fig. 1. Original Histogram

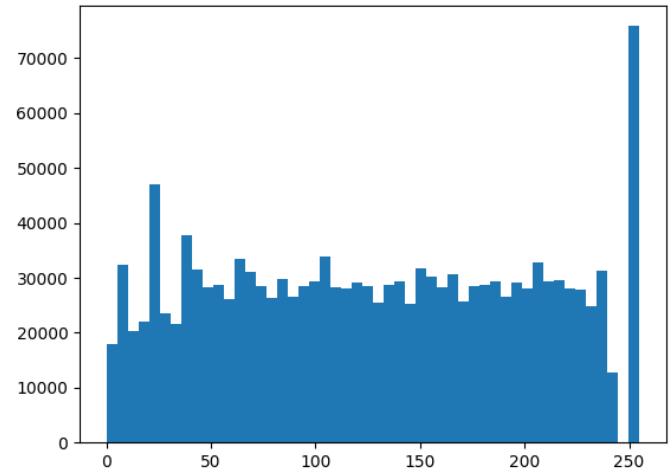

Fig. 2. Equalized Histogram

C. BONUS (Adaptive Histogram Equalization)

Adaptive histogram equalization function was implemented with the help of "skimage.exposure.equalize_adapthist" function. It takes the input image and returns adaptive histogram equalized version of the image by using histograms computed over different tile regions of the image.

The error due to the "matplotlib" library also occured in the histogram of adaptive enhanced image. However, it was solved easily by using "plt.clf" function.

When we compared the histograms of the original image, enhanced image and adaptive enhanced image, we noticed that adaptive histogram equalization changed the original histogram, but it did not majorly flatten as a whole like histogram equalization. Actually, it changed the local distribution of the pixel values of the image. Then, we compared the original image, enhanced image, adaptive enhanced image and we noticed that adaptive histogram equalization enhanced local contrasts of the original image. Thus, local details of the image enhanced in adaptive enhanced image even more than in enhanced image. Hence, it looks more natural than histogram equalized image.

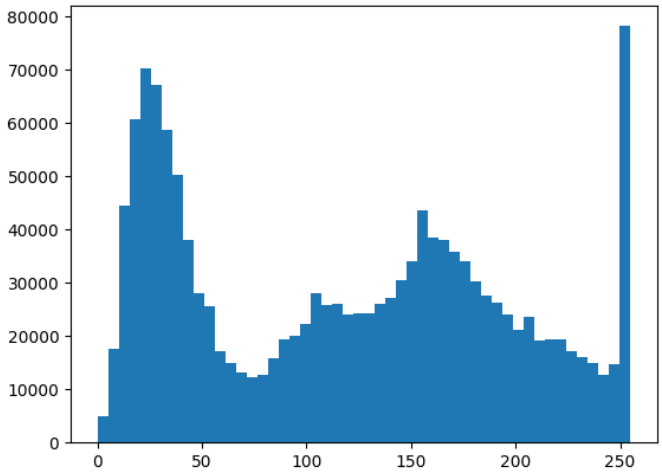

Fig. 3. Adaptive Equalized Histogram

REFERENCES

[1] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: Image processing in Python. PeerJ 2:e453 (2014) https://doi.org/10.7717/peerj.453.

[2] J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.