Fundamentals of Image Processing THE1 Report

1st Yasemin Hale KARAKAŞ
Computer Engineering
Middle East Technical University
Ankara, TURKEY
e223755@metu.edu.tr

2nd Nilüfer TAK
Computer Engineering
Middle East Technical University
Ankara , TURKEY
e231050@metu.edu.tr

***Abstract*—Affine transformations ,image interpolation, linear interpolation, cubic interpolation, histogram extraction, histogram equalization are some of the important concepts of image processing. In this document, these concepts are presented and discussed while referencing code implementations written in Python3.**

***Index Terms*—affine transformation, linear interpolation, cubic interpolation, histogram extraction, histogram equalization, rotation**

I. Introduction

In section Affine Transformations, linear and cubic interpolations are discussed and their implementations are explained. In section Histogram Equalization, extraction method of histograms and equalized histograms of gray-scale images and their implementations are discussed.

II. Affine Transformation

In this part, it is asked to read the images and perform rotation and interpolation on these by given degree values. As a result of rotating $45°$ some images end up having blank areas on the corners. To solve this issue zero padding, edge padding and mirror padding methods were tried and it was decided mirror padding gave the best result given the patterns on the input images.

A. Methodology & Algorithm Design

The methodology was basically finding the correct functions to rotate the image and then applying either linear or cubic interpolation on them based on the input parameters. Since preexisting library functions are used for all operations, there was not an algorithm design.

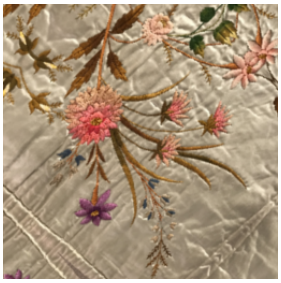
B. Implementation

The implementation written in Python3 included linear interpolation on various images. Here is the algorithm implemented step by step :

Step 1: Read the given image as a numpy array.
Step 2: Rotate the image either $45°$ or $90°$.
Step 3: Do either linear or cubic interpolation on the image.
Step 4: Save the resulting image.

C. Analysis & Comments

Firstly, the image a1.png is rotated $45°$. As it can be seen on the figure below, when rotating, the information on the image from corners are lost. However, when the image is rotated by $90°$, it can be seen that the image remains intact which means there is no need for padding after interpolation.



(a) $45°$ rotation

(b) $90°$ rotation

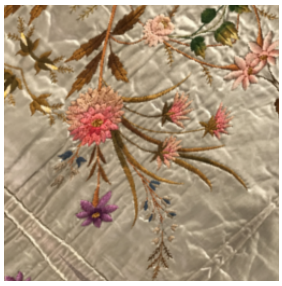
Secondly, linear and cubic interpolation of the same image resulted in slightly different images as can be seen in the below figure. Even though the difference can hardly be seen between these two images, the difference is caused by the fact that linear interpolation employs 4 or 8 nearest neighbors to estimate the intensity of the center pixel whereas cubic interpolation uses 16 nearest neighbors. Hence the image with the cubic interpolation implementation is expected to have a smoother look.

(c) Linear Interpolation

(d) Cubic Interpolation

Below are $45°$ rotation of a1.png and a2.png which are different sized and resolutioned versions of the same image. a2.png has way higher resolution than a1.png and that can also

be observed on the results of the rotation. Also the fact that the images have different shapes make the resulting rotated images look different due to the choice of padding mode.



(e) 45° rotation of a1.png

(f) 45° rotation of a2.png

D. Requirements

For the implementation of this first part of this project several functions from openCV, matplotlib, SciPy, imageIO and scikit-image libraries were used. Therefore the only specific requirement is having preferably an up-to-date version of these libraries.

Since most of the functions are common between all parts, the library requirements are the same. Hence requirements section is removed from part two and bonus part.

III. Histogram Equalization

In this part, it is asked to extract two histograms, namely original and equalized histograms, out of the inputted image b1.png, which can be found right below. As a design choice, the colored version of the image was read as grayscale before extracting the histograms.

Fig. 1: Input image b1.png

A. Methodology & Algorithm Design

There was not much of an algorithm design involved in this part as it was allowed to use libraries such as opencv, matplotlib and scikit-image. All that was needed to be done was finding the proper functions to perform necessary operations on the images.

Following the instructions given, image was first read then the histogram of the image was extracted and saved. In the next step the previously extracted histogram was equalized and saved under a different name. Finally, the enhanced image that was constructed from the equalized histogram was saved. The details about implementation will be discussed in the following part.

B. Implementation

In this section, only extract_save_histogram and histogram_equalization functions will be explained as the others are already explained in Affine Transformation. The explanations will be fairly brief as both are only two lines long functions.

Function extract_save_histogram makes use of matplotlib library to plot the histogram of a flattened array of image and save the resulting figure to the given output path. Then the figure is cleared so the next time a figure is plotted it does not collide with the previous one.

Function histogram_equalization makes use of opencv library to equalize the histogram that was obtained in extract_save_histogram and returns. In the following line there is another call to extract_save_histogram with the equalized histogram as input to basically plot and save it as a figure.

C. Analysis & Comments

Comparing the original histogram and the equalized histogram of the image, it can be observed that the equalized histogram's y-axis values are distributed more evenly. This means that the image's contrast has enhanced.

This equalization technique typically improves many images' overall contrast, especially when the image is represented by a constrained range of intensity values. This adjustment allows for a better distribution of intensities on the histogram, employing the entire range of intensities. As a result, areas with low local contrast can have high contrast. Histogram equalization accomplishes this by effectively distributing a large number of input intensity values that are used to reduce image contrast.

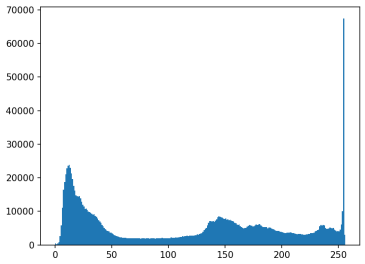


Fig. 2: The original histogram of image b1.png

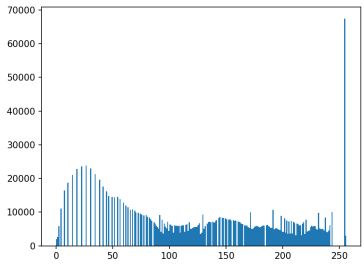


Fig. 3: The equalized histogram of the same image b1.png

The enhanced_image.png, which can be seen below, clearly shows that the contrast of the image is in fact enhanced which provides the viewer with a more comfortable experience.

Fig. 4: Enhanced version of image b1.png

IV. BONUS - Adaptive Histogram Equalization

Adaptive Histogram Equalization differs from ordinary histogram equalization in that it computes many histograms, each corresponding to a different area of the image, and uses them to disperse the image's brightness values. It is therefore suitable for improving the local contrast and enhancing the definitions of edges in each region of an image.[1].

The contrast limiting of Contrast Limited Adaptive Histogram Equalization (CLAHE) sets it apart from adaptive histogram equalization. For CLAHE, the contrast limiting process is used on each neighborhood that serves as the source of a transformation function. CLAHE is designed to prevent over-amplification of noise due to adaptive histogram equalization[2].

A. Methodology & Algorithm Design

By making use of opencv library obtaining the adaptive histogram was very easy straight from the image that was already read as grayscale.

B. Implementation

The only two lines in function adaptive_histogram_equalization are where we create Contrastive Limited Adaptive Histogram Equalization (CLAHE) function and then apply it to the image making use of opencv library functions.

clipLimit is the parameter for the threshold for contrast limiting and tileGridSize is the parameter for dividing the input image into M x N tiles and then applying histogram equalization to each local tile[3].

C. Analysis & Comments

As it was explained in previous chapters, adaptive equalized version of the image is calculated by several histograms, each histogram refers to a subsection of an image and, this approach achieves enhancement of local contrast.

When observing the adaptive equalized histogram of the image in below figure, it can be concluded that compared to equalized histogram, redistribution of the lightness values of the image is clearly results in more compact histogram. As a result, adaptive equalized version of the image b1.png, has more contrast and by having more contrast, the edges of the image can be seen more clearly.

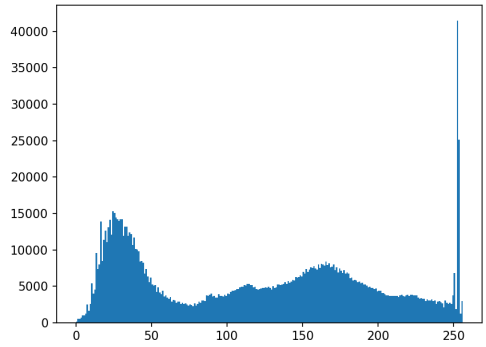


Fig. 5: Adaptive equalized histogram of image b1.png

Fig. 6: Adaptive equalized version of the image b1.png

References

[1] https://towardsdatascience.com/histogram-equalization-5d1013626e64
[2] https://towardsdatascience.com/histogram-equalization-5d1013626e64
[3] https://pyimagesearch.com/2021/02/01/opencv-histogram-equalization-and-adaptive-histogram-equalization-clahe/