Assigment 1: Basic Image Manipulations

Matias Cinera
Computer Science Department, *University of South Florida*
Tampa, Florida, 33612, USA
cinera@usf.edu

I. Introduction

This document is the report of my Assignment 1 from my Fall 2023 class CAP-5400 Digital Image Processing, as a graduate student. The individuals who are overviewing this assignment are our professor Dr. Dmitry Goldgof and Mr. Anthony McCofie (Teaching Assistant). The purpose of this assignment is to implement basic image transformation (binarization, brightness, scale, rotation) for grey-scale and color images.

Figure 1: Sample Images used

II. Image Transofrmations

The following are my grey-scale image transformations, although they also work for color images. Both images used to test them were 512x512 pixels, although, for the equations I will refer to an image as an **n·m** (rows = m, columns = n) matrix.

A. Add Grey

The add grey function simply adds a **value (**specified **by** the user**)** to each *I(i,j)* pixel value. The effect of this function increases or decreases the overall brightness of the image.

$$\hat{I}(i,j) = \begin{cases} I(i,j) + v, & \text{if } I(i,j) + v < MaxRGB \\ MaxRGB, & \text{otherwise} \end{cases}$$

Figure 2: Add grey equation

The same opposite equation will work for "remove grey", just inverse the equation and use a MinRGB value instead.

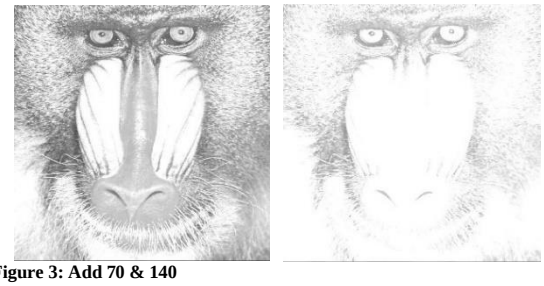

Figure 3: Add 70 & 140

Once the absolute value of ±**v** goes over ~220 the image will either be completely white or black. Thus, any values between *-220 < **v** < 220* will retain some information from the original image.

B. Binarize

Binarize is a function that aims to only have 2-pixel values for all pixels in the image. This function works by setting all *I(i,j)* values to either MinRGB or MaxRGB values based on a threshold input.

$$\hat{I}(i,j) = \begin{cases} MinRGB, & \text{if } I(i,j) < t \\ MaxRGB, & \text{otherwise} \end{cases}$$

Figure 2: Binarize equation

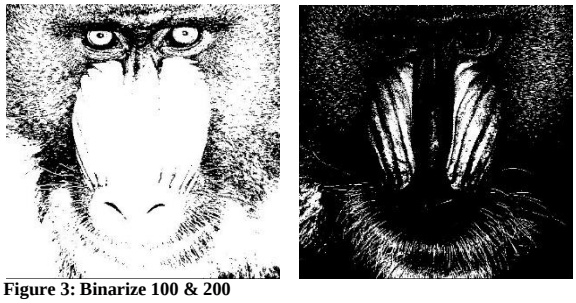

Figure 3: Binarize 100 & 200

Values between *30 < **t** < 230* will retain some information from the original image. Any values outside this range will result on a mostly white or black image with little or none information of the original image.

C. Scaling

My scaling equation is achieved by multiplying a scaling factor to get the new number of columns and rows for the output image. The scaling factor is also used to map the pixels, by getting pixels from the source image and rounding them down.

$$\hat{I}(i,j) = I(\frac{i}{ratio}, \frac{j}{ratio})$$

Figure 5: Scaling equation

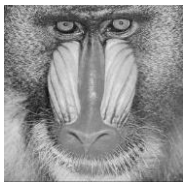

Figure 6: 0.5 Scale image

D. Rotation

In this assignment the rotation equations only needed to handle 3 cases, 90, 180 and 270 degree turns. Thus, I manually implemented the 3 cases through matrix manipulations.

$$\hat{I}(i,j) = I(m - i, n - j)$$

Figure 7: 180 degree rotation equation

Figure 8: 0.5 Scale image

However, a better implementation could be achieved by multiplying the original matrix with a scaling factor and a rotational matrix in a similar way to the affine transformation. In this case the scaling factor is not needed.

$$\hat{I}(i,j) = s \cdot I(i,j) \cdot R + t$$

III. Color Image Transofrmations

The following are transformations specifically made for color images, although they can also be used for grey images.

A. Multiplicative Color

The purpose of this function is to intensify each pixel of the image. To achieve this just multiply each *I(i,j)* pixel value with a multiplicative factor (C). This process is repeated for all RGB values, thus is done 3 times.

$$I(i,j) = I(i,j) \cdot C$$

Figure 9: Multiplicative color equation

Figure 10: Multiplicative factor (C) = 1.7

B. Brightness

The brightness function is essentially the same as the **Add Grey** equation. However, the **AC** value is bounded, and the operation is done through all the RGB channels.

$$-50 \leq AC \leq -50$$

$$\hat{I}(i,j) = \begin{cases} I(i,j) + AC & \text{if } I(i,j) + AC < MaxRGB \\ MaxRGB & otherwise \end{cases}$$

Figure 11: Brightness equation

Figure 12:Brightness (AC) = 45

IV. Implementation & Multiple Parameters

This assignment required to apply multiple parameters sequentially to the same image. The assignment also required for all parameters to allow a **Region of Interest (ROI)**, which is a subset of the image in which the image operation can be performed. To achieve this, all my image operations are independent, meaning all of them can only receive one input and produce one output. By implementing it this way, I can call as many operations as I want sequentially on the same image. I also implemented a **merge ROI** function, this function merges and ROI produced image with the original image.

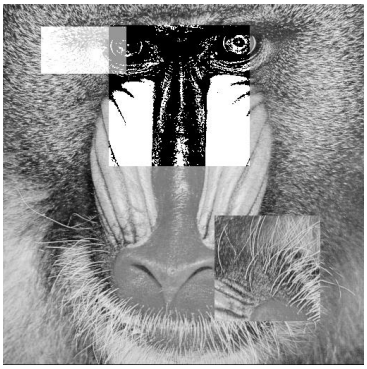

Figure 12: Grey-Scale Image with 3 image operations:

Rotate: degree = 270, **ROI**: i=300, j=300, i size = 150, j size = 150

Binarize: t=158, **ROI**: i=30, j=150, i size = 200, j size = 200

Add Grey: v=150, **ROI**: i= 32, j= 54, i size = 67, j size = 121

Figure 12: Color-Scale Image with 3 image operations:

Brightness: AC=-30, **ROI**: i=0, j=0, i size = 180, j size = 180

Rotate: degree = 90, **ROI**: i=150, j=150, i size = 200, j size = 200

Binarize: t=200, **ROI**: i= 300, j= 300, i size = 70, j size = 70

Images don't necessarily need to be merged back, the code is flexible to only output the ROI. All images generated with multiple operations were merged back for convenience/accesibility purpuses.

V. Conclusions

Most of the functions were already provided by the TA, and the textbook. Implementing the logic was simple. However, I don't like the code base of this project. Changing simple things like returning types of functions made the whole silently crash the whole program (not return any type of debugging code or let you know that the program crashed). Thus, debugging this code was a nightmare! Would prefer to implement this in python with a library to just display images