HW1 – IMAGE BINARIZATION AND PIXEL OPERATIONS IN C/C++

This assignment was used to teach us basic pixel operations using C++. Images had to be converted from RGB to Grayscale, images had to be binarized and compared to the Ground Truth image, and images had to have their brightness and/or contrast changed without using built-in functions. Having done this, images were able to be altered significantly using basic arithmetic commands.

SUNDAY, SEPTEMBER 12, 2021

RYAN ORF

CS4650: Digital Image Processing

For this assignment, students were challenged to alter images in three different ways without using built-in functions. The first was to take a colored image and convert it into grayscale. The second was to take a grayscale image, binarize it based off of a threshold value, and compare the results to the ground truth image. The third and final was to take a grayscale image and alter the brightness and/or contrast using either an intensity value or minimum and maximum values. With completion of these goals, students would be able to perform pixel operations using simple arithmetic to alter an image and better understand how image processing works at the pixel level.

These experiments did not produce any data such as graphs or tables to display here. However, each challenge can be explained in great detail. For the conversion to grayscale, students had to access the blue, green, and red channels at each pixel, obtain a value, and then plug those values into an equation that would be set equal to the corresponding pixel of a grayscale image. This process was performed through obtaining the row and column sizes of the input images, looping through the indices, and obtaining the RGB values at the given index, which would then be inserted into a formula that would initialize the corresponding pixel's index for the grayscale image. The image was then written to the disk with the given output name and displayed, showing the conversion to grayscale.

The image binarization was almost just as straightforward. An input image was opened in grayscale, and once again the row and column values were obtained to loop through all the pixels. A threshold value was also given, which would be vital in determining whether a pixel was considered in the foreground or background. While looping through the array, the grayscale value at a given index of the input image would be compared to the input threshold. If it were greater than or equal to, the pixel value for the binarized image would be set to 255, else it would

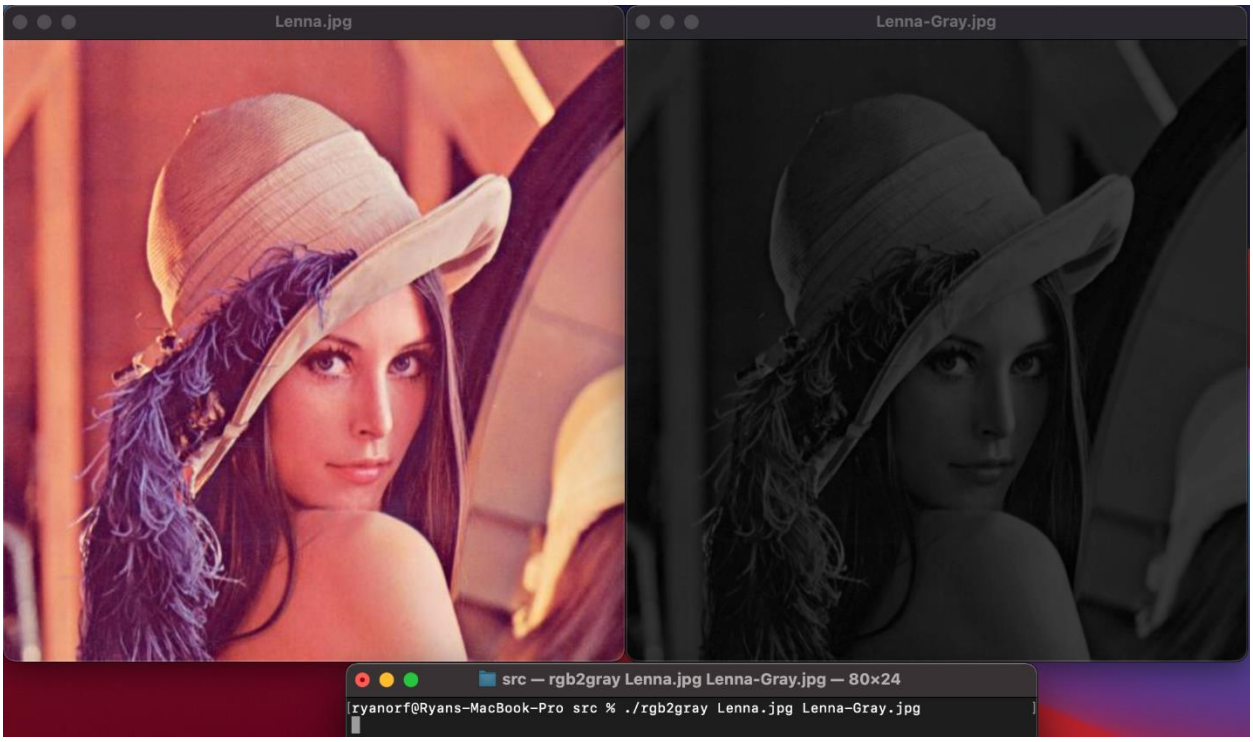
be set to zero. Simultaneously, the input ground truth image pixel value at that index would be compared to the index of the binarized image. If both were 255, a variable true positive was incremented. If only the binarized image was 255, a variable false positive was incremented. If both were zero, a variable true negative was incremented. Finally, if only the binarized image was zero, a variable false negative was incremented. The binarized and ground truth images would then be displayed to compare, and the statistics that were calculated aforementioned will be displayed to the user. The image was written to the disk with the provided output name.

Altering the brightness and contrast was the final challenge, which was broken up into two files. The difference between these files was that the brightness file asked for an intensity value input, whereas the contrast file asked for a minimum value and a maximum value as input. For the brightness alone, each pixel was looped through the image in the same method previously mentioned, and the intensity value was added to the value of the pixel at that index. However, to ensure the picture did not become warped due to overflow, conditions were set in place to assure values over 255 would display as 255 and values under zero would display as zero. For the contrast file, instead of incrementing the pixel value by an intensity value, the minimum value was subtracted from the current pixel's value at that index, which was then divided by the difference between the maximum value and the minimum value, and finally this quotient was multiplied by 255 (since this was a grayscale image). This value was then set equal to the pixel's value at that index. In theory, this process changes the contrast of the image, as well as the brightness so long as the minimum value does not equal zero. Finally, the altered image is displayed and written to the disk with the given output name.

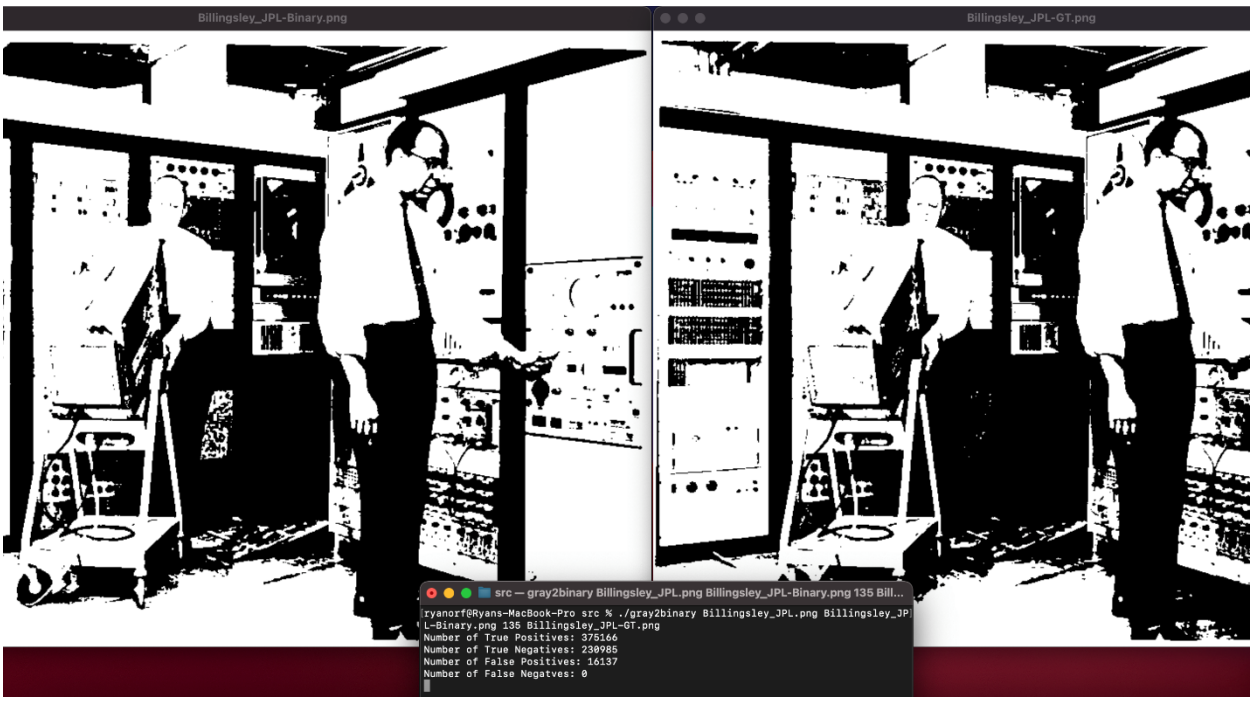
Outside of setting up OpenCV (the files required to run the image processes detailed), I did not find too much difficulty in these assignments. The grayscale and image binarizations

particularly came together rather naturally. I did encounter an issue when it came to the brightness and contrast, as my initial conditioning for the overflow did not work. However, this was simply due to a matter of order of operations when calculating the pixel's value. Other errors simply include syntax errors that were easily fixed once discovered. One interesting note I discovered while binarizing images was that, at least for the images I tested, there was not a possibility of having false positives and false negatives at a given threshold value. I also tested threshold values until I obtained one that only gave me true positive and true negative values. After experimenting with these arithmetic pixel alterations, I believe I have a much better understanding of how images are processed and changed at the pixel level. Please find below sample images of the programs running.

RGB2Gray:

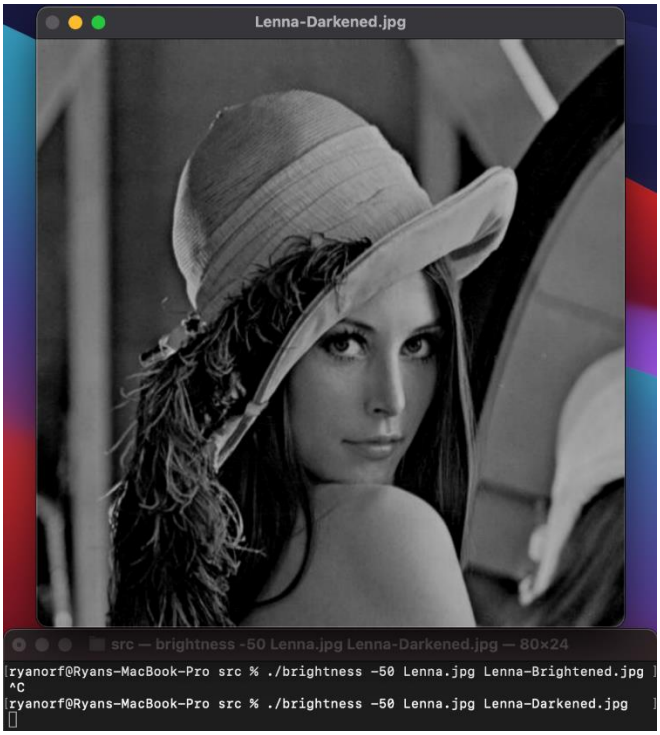


Gray2Binary:



Brightened Brightness:

Darkened Brightness:



Contrast: