Image Processing

Homework 1

Fall 2017

Kynan Justis

10/1/2017

Overview

The main objective for this assignment was to write a program for grayscale, 8-bit image processing using histogram equalization. For example, grayscale images that are too bright, too dark, or otherwise imperfect can be processed in a way to improve contrast and brightness without compromising the main information of the image. However, although histogram equalization was the main method of the assignment, I will cover the results of several other methods of contrast/brightness recovery such as full-scale contrast stretching and linear transformation.

Full-scale contrast stretching

Full-scale contrast stretching (FSCS) works by considering where the majority of the data is laid out in a histogram. Based on the distribution, FSCS attempts to stretch the original distribution from an extreme range of values into a distribution of values in a more acceptable range. For example, an image with a lot of mid range values would come out with a distributions of many low-intensity values, many high-intensity values, and very few mid-intensity values. Conversely, an image with a lot of low or high-intensity values would come out with more mid-intensity values.

FSCS Results

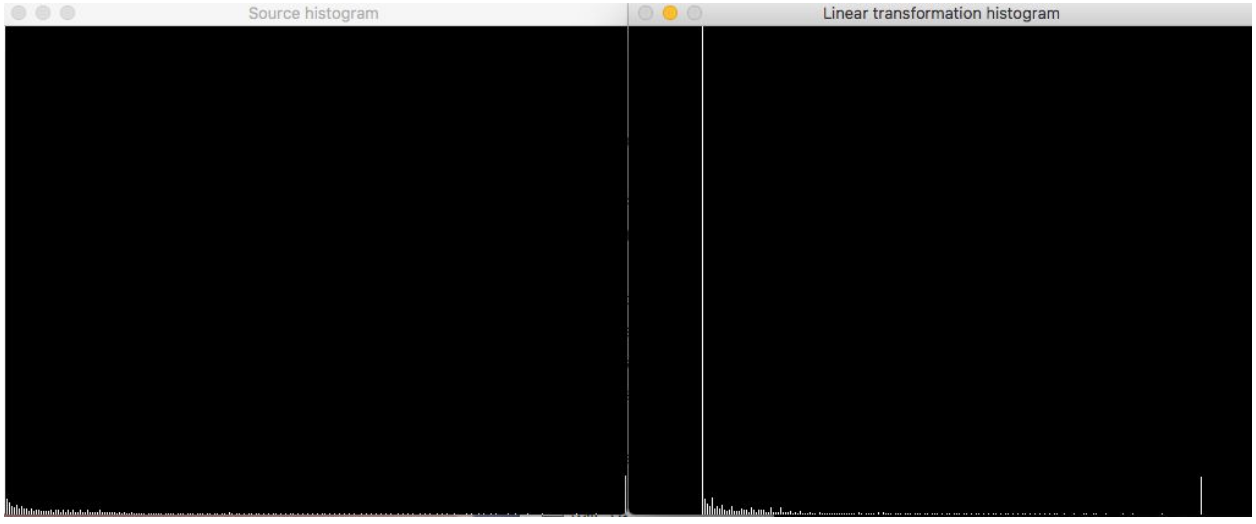
FSCS Summary

While FSCS does work somewhat well for recovering the contrast in this image, my implementation has some problems. When looking at the large bright area in the source image, it's clear that my version of FSCS puts too much of a weight on that section and pulls the intensity down much too sharply. This leads to a severely darkened area surrounded by a harsh, jagged outline. From what I can tell in my code, I believe this may have to do with problems passing data as different data types and casting between them. Since I used C++ with the OpenCV library, all of the images are read in as *cv::Mat()* objects. These are nice since they allow for pixel data to be retrieved as any data type with a single call. When I was writing the FSCS function, however, I was retrieving the pixel values as unsigned characters, which I then subtracted from by the '0' character literal in order to give me the actual intensity value. Although this was simple enough, it made getting the calculations for the output pixels a little more difficult. To compensate for subtracting the '0', I had to add it back in before the final return and it's reasonable for me to think that if I had just used double precision behind the scenes the whole time I would have gotten more fine-grain results.

Linear Transformation

While I was researching different methods for recovering contrast in grayscale images, I came across a paper that walked through a method using linear transformation (Source: http://www.ece.ubc.ca/~irenek/techpaps/introip/manual01.html). The function is simply applied to every pixel in the source image and is defined as: $X_{out} = a * X_{in} + b$ where 'a' is the contrast scale factor, 'X_{in}' is the pixel value from the source image, and 'b' is the brightness scale factor. The result is a more compressed intensity distribution range with larger intensity values overall.

Linear Transformation Results



Linear Transformation Summary

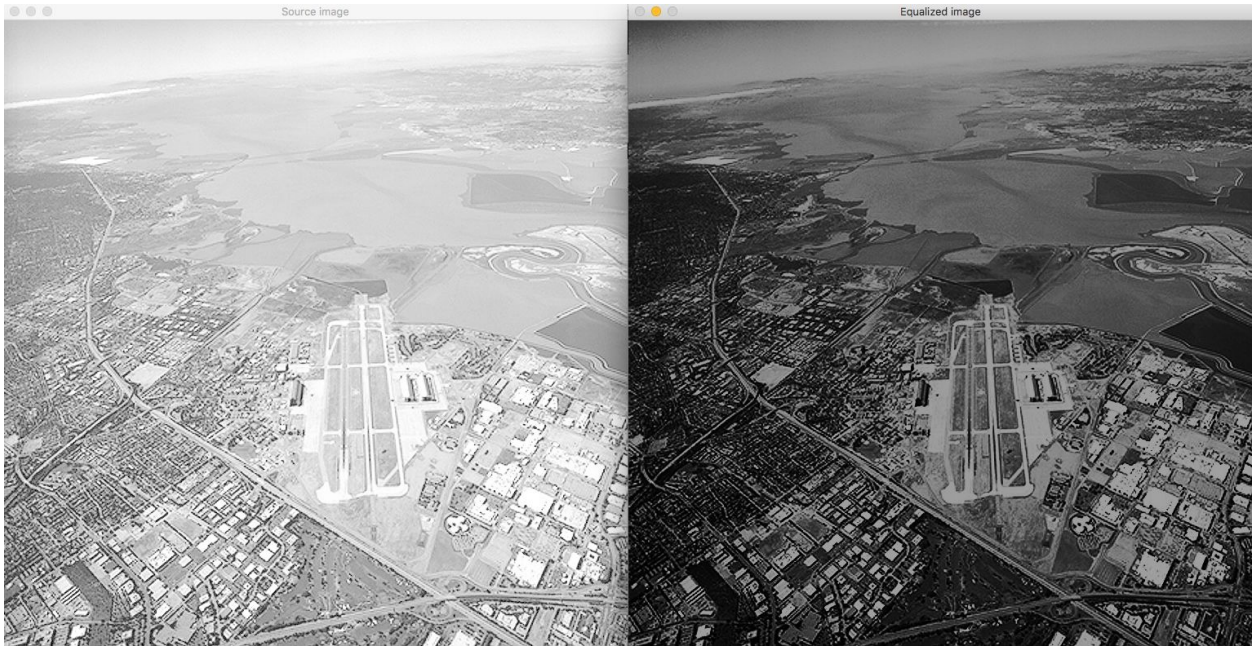
Overall, this method allows for fine-grain tuning of the input image. Given the proper contrast scale factor and subsequent brightness scale factor, all of the tested images showed improvements to a reasonable degree. The downside, however, is the trial-error process associated with it. A better implementation would put more effort into calculating those constants based on a given input image. While this may not be apparent information that can be derived through an imperative process, machine learning could help identify successful constants for different kinds of images (high brightness vs low contrast, high contrast vs high brightness, etc…). Alternatively, the problem could be given to the user to solve through a GUI that allows the user to change the contrast and brightness scale factors in real time.

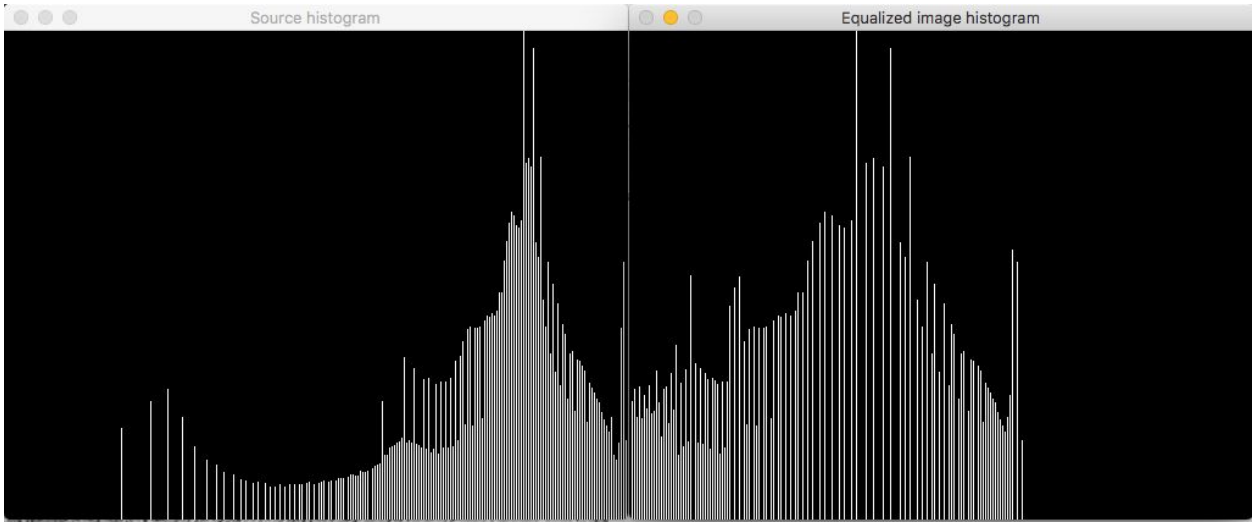
Histogram Equalization

As given by its name, this method uses histogram data calculated from an input image to adjust the brightness and contrast to be in an acceptable range. The process implemented in my program is based on both the OpenCV *cv::equalizeHist()* function and the notes provided in class. First, the input image and its histogram is passed to *GenerateEqualizeImage()*. Here, a new histogram is generated based on the normalized sum of the input histogram. By doing this, we essentially create a lookup table where the bin index represents a particular intensity value in the source image. This is then used as we run through the new matrix being generated where each pixel is filled in by the new value held at the intensity index of the pixel from the original source image. Referring to this new value as the "adjusted intensity", the adjusted intensity is calculated by taking the sum of that bin, multiplying it by some constant, and then dividing that value by

the source image dimensions. Afterwards, clamping is applied to ensure that no adjusted intensity is below 0 or above 255. By doing this for every pixel, the output image is generated. Following the process of generating the image, the histogram for the equalized image is then generated. This is done by passing the equalized image to *GenerateHistogramImage()*, which generates the histogram from the input pixel data, generates the actual image matrix for the histogram image, and normalizes the histogram so that the bins fall in the 0-255 range just as the histograms for all previous methods are calculated.

Histogram Equalization Results





Histogram Equalization Summary

In general, this method works very well for improving the contrast and normalizing the brightness in the input images. However, I have noticed some issues with my implementation. To begin, my calculation for the adjusted brightness seems to be incorrect. In my explanation I mentioned that some constant was used in the calculation and this constant is somewhat of an unknown to me. For some reason I used a value based on the image height, but for some images, using this value alone was not enough. To illustrate, the worst example was the Figure 3.8 image. The source image already starts with a large number of bright values, which is apparent from the source image histogram. In the output histogram, the majority of these bright values are still there although the values that approached a more mid-range intensity were shifted over somewhat. The final image that I produced was only generated after making changes to the code as described in the "notes.txt" file in the output folder of the project. One issue regarding this may be that I assumed that all images would be square in dimension, and so the histogram method worked much better on the more square images from the get go. To fix this, one would likely have to take the source width into consideration as well in the adjusted intensity calculation.

All of the results for each combination of method and source image can be found here in the output folder: https://github.com/kfjustis/ip1 If you want to build the project with CMake, assuming there is a valid install of OpenCV and CMake, just download the project, make a folder called "build" in the project, then change directory into it, run "cmake ..", and then "make". The binary will be in "build/src/".