CSCI 8810: Project #1

Due on Tuesday, February 10, 2015

Prof. Arabnia 08:00am

Sina Solaimanpour

Contents

I. Introduction to *"ColonD Image Processing"*

This project is aimed to get us more familiar with different Image Processing techniques. The overall work flow of this project is to read an image from an image source (here, the image source can be either a JPEG file or the computer's camera) and apply different image processing algorithms in the image separately and in combination.
I am using C++ programming language along with OpenCV library to implement the algorithms in this project. I chose C++ over other programming languages because of its superior performance and less overhead comparing to other high-level programming languages like Java or Python. Furthermore, I am using OpenCV just to read/write images from/to a file and also to maintain the image structure in the system. All other operations are implemented without any use of existing functions.
In this phase of the project, the following features have been implemented which will be discussed in more detail in the following sections:

- Read an image from the filesystem
- Capture an image from the computer's camera
- Create the image histogram and maintain it after each change
- Enhance the image contrast using image histogram
- Convert an image to a gray scale image
- Add artificially created noise to the image
- Smooth an image using "Average Filters"
- Smooth an image using "Median Filters"
- Convert an image to binary by simple thresholding
- Image slice thresholding an image
- Iterative thresholding an image
- Adaptive thresholding an image
- P-tile thresholding an image

These features are all implemented and tested with different images, individually and in combination of other operations. In the next sections, I will go over each one of these features in more detail and will present some results from the program.

II. Basic operations

In this section, I will go over some of the basic operations implemented in the project. These operations are:

1. Read/Write an image from/to the filesystem
2. Capture an image from the computer's camera

(a) Read/Write operations

For these two operations, I am using OpenCV's *"imread"* and *"imwrite"* functions. The imread function will read an image file and load it into a Mat data structure which is the image structure used in OpenCV. On the other hand, the imwrite function will get as parameter a Mat object and will serialize it to the filesystem. I faced some difficulties using these two functions. An example of these difficulties was the incompatibilities between color spaces. The imread function reads an image in BGR color space and I was processing the image in RGB space. This forced me to convert the read images to RGB, after reading it and, convert them back to BGR before writing to the filesystem.

II. Basic operations continued on next page...

(b) Capture image from camera

As an extra feature, I added the capture image functionality to the program. To do this, I get the system's default camera and open its video feedback in a different window. The user can hit the "Space" key to capture the image. After that, the captured image will be opened in the program automatically and it can be used as a normal image in the program.
Again, this image capturing process had the exact same problem with the color spaces as Read/Write operations. I have applied the color space conversion in this context too.

III. Histogram View

After opening or capturing an image in the program, a new window will appear which is consisted of the histogram of the image. For simplification, I have used the red channel to create and show the histogram and this will allow us to show almost the same histograms for both an color image and the gray scale version of it. This could be done by creating four different histograms, one for each RGB channels and one for the average of the three values.
To create the histogram chart, I have used GNU-Plot program installed on an UBUNTU machine. Each time the image changes, I will update its histogram and send it over to GNU-plot to create the chart. Then, I use the created histogram to update the Histogram view in the program.

Figure 1: Screen shot of a sample histogram created in the program.

IV. Gray Scale Conversion

As we are supposed to process only gray scale images, I needed to somehow convert the read image to gray scale. I tried three different methods to convert the color image to gray scale image. After comparing the results from different techniques on multiple images, I decided to use the weighted average as the main algorithm to convert the images to gray scale in my program. The following equation is used to get the resulting gray scale pixels in the image.

```
int gray_scale = 0.21 * red + 0.72 * green + 0.07 * blue;
```

Also, the following image is the result from converting the original version of the Lena image to gray scale using the ColonD program.

Figure 2: Convert lena's image to gray scale.

V. Add Noise

For learning purposes, we needed to add some artificially created noise to our images. In my project, using a slider control, one can add different amounts of noise to the opened image. The implementation of this feature was very simple but again, it could be done in different ways. What I am doing is, I generate a random number in $P\%$ of the pixels and replace the pixel value with the random value. Another way of doing this is to generate a random number and add the random value to the pixel values.
The user can use the slider to change the amount of added noise to the picture and by clicking on the "Apply" button the noise will be added to the image. If one forgets to click on the apply button or decide not to add noise to the image, the noise will not be applied and the original image will remain intact.

Figure 3: Added some noise to the lena's picture. The slider control and the Apply button can be seen in this snapshot of the program.

VI. Smoothing The Image

For smoothing an image, we can use two different techniques in this program. These two techniques will be explained in detail in the following sub-sections.

(a) Average Filter

This technique is nothing but using a simple filter over the image. The matrix which represents the filter have equal values as for each one its elements and this value is equal to $\frac{1}{Width^2}$.
The user can use the provided textfields to determine the width of the filter. The following figure will depict how the Average Filtering algorithm will work on an image.

Figure 4: Used an Average Filter with Width = 10 on lena's image.

This algorithm will leave out a few pixels from each side of the image and will not fill them with any other values hence the black frame around the image is formed.

(b) Median Filter

This filter will find the pixel values in corresponding to the filter elements, sort them and use the middle value (the median) to represent all other pixels. Then we replace the pixel corresponding to the filter's middle element.
I realized that this smoothing technique will work very well on noisy images I created using the Add Noise feature. The following figure will compare both the noisy and the smoothed image together. As you can see, the resulting smoothed image has a huge improved **visual quality** comparing to the noisy image.

Figure 5: Applying the Median filter to a noisy image of lena.

The width of the Median Filter can also be changed in the program. The interesting observation about this filter is that I used a Median Filter of width 3. If I use wider filters, the image will start to look like vector images and it starts to flatten out.

Figure 6: Applying a 10x10 Median filter to the original image of lena.

VII. Contrast Enhancement

As discussed in the class, the contrast of an image can be improved to a degree by stretching out the histogram to cover the whole space. To do this, first we need to find the dominant peak or peaks of the image. After finding the peak, we need to determine the valleys around the peak(s). The location of these valleys (we will have two valleys in this case, one on each side of the peak) can be used as stretching points of the histogram.
I have previously implemented a peak recognition system in a different context. In that method, first, I use a simple threshold (which can be determined using different statistical measures) to separate peak values from values located in valleys. After this step, I use Student T-test which is again another statistical test to figure

VII. Contrast Enhancement continued on next page...

out which peak is worth keeping and which peak is not. Also, using the resulting p-values from the T-test, I will decide whether I need to combine to consecutive peaks or treat them as separate peaks. This method has been implemented and is going to be published as an application note. But here, in our application, I just let the user to decide which two values he is going to use as the boundaries of the stretching process.

Figure 7: This figure shows an image before and after the contrast enhancement process. The image on the left is the original image. The right image is the enhanced version.

VIII. Binarization an image

As many other image processing techniques which don't have a unique way of doing, binarization of an image can also be done in many ways. Methods implemented in this project are,

1. Simple Thresholding
2. P-tile Thresholding
3. Iterative Thresholding
4. Adaptive Thresholding
5. Slicing an image and do Thresholding

In the following sub-sections I will present the results from running these techniques on different images and will briefly discuss each one of them.

(a) Simple Thresholding

A simple threshold will be used in this technique and simply turns all of the pixels below that threshold to black and the others to white. An example of this technique can be seen in the following figure,

Figure 8: Applying a simple threshold to an image.

The user can define the threshold he wants to use using a slide control which starts from 0 and ends at 255.

(b) P-Tile Thresholding

In this method, we get the P value and use it to find the *Pth* percentile of the pixels. This percentile value can be used as an threshold value to convert the image to binary. An example of this method can be used which is done with $P = 20$.

Figure 9: Applying a P-tile function with $P = 20$.

(c) Iterative Thresholding

This is just the implementation of the Iterative method discussed in the class. It doesn't need any input parameters and I am using 128 as the initial threshold value. The resulting image after applying this function on lena's image is depicted in the following figure.

Figure 10: Result of Iterative Thresholding on lena's image.

VIII. Binarization an image continued on next page...

(d) Adaptive Thresholding

Adaptive thresholding will cut the image into N * N smaller images and will apply an arbitrary thresholding techniques on each one these smaller images individually. In my program, I get the number of images in each dimension, cut the image into smaller ones and apply an Iterative thresholding technique on each one of them and stitch them back together at the end. The resulting image from the lena's picture can be seen in the following figure in which I will cut the image into 10 smaller images on each dimension which will give us 100 separate images.

Figure 11: Result of Adaptive Thresholding on lena's image with 10 images on each dimension.

(e) Image Slicing

This technique can also be done in many different ways. In this project, I decided to slice the histogram into **N** equal areas. For each one of these areas, the values located inside of the area will be shown white and the rest of the pixels will be shown black.
After calculating the results, a new window will appear, showing the resulting images as an animation which plays twice. Also, the program will ask for a location to save all the results in case you want to review the results individually.
The following figure shows a image sliced into five different binary images.

Figure 12: Lena's image sliced into 5 different binary images. The histogram is sliced from left to right in which the top-left image will corresponds to the left most slice of the histogram.

IX. Conclusion

As you may have figured out, Image Processing operations can be done in many different ways with many different parameters. Deciding on which technique to use and what parameters to use depends heavily on the application domain you are working in.