CS512 Assignment 2 Report

Ronit Rudra
A20379221

October 9, 2017

Abstract

This report describes the implementation of a simple image processing framework. The desired outcomes of the framework, the problems faced, the implemented solutions would be discussed in the upcoming sections.

1 Problem Statement

The objective was to create a simple program which would apply various image processing techniques to an image.

The program should have the following:

- Would be run from the command prompt/terminal
- Arguments passed in the terminal would determine either of the two
  - Image filename is passed to the terminal
  - No arguments passed means image is to be captured from camera
- Functional menu system where the user is presented with choice of operations. The program then performs the operation corresponding to the key press and displays the result.

The next section describes how the program was structured and implemented

2 Proposed Solution

In order to implement the required problem, it was necessary to structure the program into the front-end and back-end.

The front-end would have the following:

Menu A system that presents the user with options to choose from.

File handling Handles reading of required files

Interface A way to interact with the backend which performs all the computations.

The back-end would have the following:

Processing Set of functions which process the image when called from the front-end

Data storage Stores all the data, final or intermediate

The program was implemented in *Python 3.6.1* with *OpenCV 3* and it made sense to implement Object-Oriented Programming to make the program as modular as possible.

The next section details the implementation of the program.

3 Implementation Details

The program is a set of three files:

main.py The main file which contains the script. This is the file that should be run when a user wants to run the program. The main file imports modules and classes from the below-mentioned files.

helper.py This file contains the class *Helper* which is responsible for the front-end. The class instance stores file locations, the current working directory (to facilitate relative directory paths) and has functions to read files and display the menu systems. The object is also responsible for taking user inputs and calling the back-end functions.

image_class.py This file stores the declaration of the class *Image*. This class stores the image itself and has methods which perform image processing. Since only one image is read at a time it was easier to wrap the data and functions into a class object.

Since there are around 15 operations the user can choose from, running an *if-else* or *switch* flow control would be tedious. An easier alternative was to store the function references in dictionaries with keys as the user input. This would ensure the operation to be called in $O(1)$ time instead of $O(n)$. The use of class methods removed the necessity of passing arguments to the function.

A few of the major problems and solutions is discussed below:

Webcam This was a major hurdle as my desktop does not have a webcam. Thus the webcam implementation was completely untested.

Manual Grayscale The manual grayscale conversion was done using the following equation:

$$I' = 0.229 \times I_r + 0.587 \times I_g + 0.114 \times I_b \tag{1}$$

The problem was, when displaying the image through the *OpenCV* function *imshow()* the image would appear as a white screen. It was noticed that using *matplotlib.pyplot.imshow(image,cmap="gray")* displayed the correct grayscale image. I have still displayed the image using the *OpenCV* function.

Gradient Vectors The plotting of the gradient vectors was done using *matplotlib.pyplot.quiver()* which plots directional arrows of the gradients. The problem encountered was in transferring the plot to *cv2.imshow()*. The ad-hoc workaround was to save the *matplotlib* plot as an image, read it into the program and then display it through *cv2.imshow()*.

Convolution The implementation of a manual convolution was done through a nested loop iterating over the rows and columns of the output. Since, the output size its taken to be the same as the input image size, zero padding was done to perform a full convolution. The padding was

calculated as follows:

$$H_{out} = \frac{H_{in} + 2 \times P_h - H_{kernel}}{S_h} \tag{2}$$

$$W_{out} = \frac{W_{in} + 2 \times P_w - W_{kernel}}{S_w} \tag{3}$$

where, H and W are the heights and widths of the input, output and kernel. P is the padding in the corresponding direction and S is the stride in the dimension. For our purposes, stride is always 1.

From this the padding can be calculated.

Slider There are four functions which use a slider and defining four different sliders was inefficient. Therefore, a single slider bar function was created which takes a flag as user data. The flag determines what operation is to be performed when the slider is moved, choosing from rotation, directional gradient spacing or blurring amount. The *_slider()* callback function in the image class performs the operations, not the functions which create the trackbar.

Downsample without Smoothing the function *cv2.pyrDown()* does downsampling with Gaussian Smoothing. But for simple downsampling by 2, the image array was subset by only taking alternate rows and columns.

Cycle Color Channels The problem with indefinitely cycling the color channel of the image was solved by using the function *cycle()* from the module *itertools*. The cycle function creates an infinite cycle of the list of items passed to it, thus it was trivial to write a loop to cycle through the image channels unless an exit key was pressed.

Rotation Performing a rotation transform leaves black dots or spaces in the output image. This is because some pixels in the input get mapped to the same output pixel. The transform is many-to-one. Using an inverse mapping ensures the transform is one-to-one. One more problem with rotation is that the image gets cropped. To deal with this, I adjusted the center of rotation such that the entire image is visible after rotation.

Help When the 'h' key is pressed, the program displays the contents of the *help.txt* file stored in the *doc* folder. This is easier than hard coding multiple print statements in the program.

4 Results and Discussion

The apart from a few bugs, the program works as intended. The only major concern is the untested camera capture which could potentially break the entire code.