IPPR Lab 1: Basic Image Handling

Nischal Regmi
Associate Professor
Everest Engineering College

1 Introduction

The laboratory work for the course Image Processing and Pattern Recognition will be done in Java. Let me clarify why I selected Java for this course. If speed of execution is the main concern, C++ will be the better alternative. On the other hand, if we prefer ease of programming, then MATLAB or R could be good choices. Java falls somewhere between these extremes. It is fast, although slow in comparison to C++. Java compilers come along built-in class/functions for image handling. This makes easier to do image processing in Java, though using Java still somewhat harder compared to MATLAB/R. Java is a general purpose programming language, whereas MATLAB and R are not. Therefore Java would be a better alternative if you wish to develop software applications from your image processing research.

From my personal experience, there are other benefits of using Java as a student. There are numerous freely available Java libraries for scientific applications written by world class programmers. And installation of these packages is very easy – just download the appropriate 'jar' file and include the file in your linker setting. In contrast, installing third party libraries in C++ could be difficult. From my personal perspective, this is the strongest advantage of using Java. Finally, you need not worry about the operating system or underlying hardware while developing Java program. Your program will run on any computer that has the Java Virtual Machine. Of course, there are several weaknesses of Java too. But there are trade-offs in every choices we make.

So, install a Java IDE in your computer. I have been using the Eclipse IDE for Java, you can choose according to your preference. Regarding the Java programming language, the syntax of Java language is very similar to C++. As you all know C++, the transition would not be much difficult.

2 Reading, Saving, and Displaying Images

Java provides the 'BufferedImage' class that can be used to load and save image files, and access the image pixel by pixel. Below is a Java code to read a color image, convert it to gray-scale by averaging the RGB values, and save the resulting image. You can use this code as a template for your programs unless you get used-to with the image handling operations.

```
import java.io.File;
import java.io.IOException;
import java.awt.Color;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

public class ConvertToGray {

        public static void main(String[] args) {
                //create an object of buffered-image class
                BufferedImage image;
```

```
                //read an image from file
                try {
                        image = ImageIO.read(new File(``an_image_file.jpg''));
                        System.out.println(``Reading complete.'');
                }
                catch(IOException e) {
                        System.out.println(``Error: ''+e);
                        return;
                }

                //now loop over the pixel and average the RGB values
                for(int y=0;y<image.getHeight();y++)
                        for(int x=0;x<image.getWidth();x++) {
                                Color c = new Color(image.getRGB(x, y));
                                int red =  (c.getRed());
                                int green = (c.getGreen());
                                int blue = (c.getBlue());

                                int gray = (int) (red+green+blue)/3;
                                Color newColor = new Color(gray,gray,gray);
                                image.setRGB(x, y,newColor.getRGB());
                        }

                //save the image to file (if you would like)
                try {
                        ImageIO.write(image,``jpg'',new File``image_greyed.jpg''));
                        System.out.println(``Writing complete.'');
                }
                catch(IOException e) {
                        System.out.println(``Error: ''+e);
                        return;
                }
        }
}
```

Instead of viewed the output image in a separate viewer saved by our program, we would like to view the output using the Java program itself. For this, you need to insert the following piece of code appropriately in the program.

```
ImageIcon icon = new ImageIcon(image);
JFrame frame = new JFrame();
frame.setLayout(new FlowLayout());
frame.setSize(200,300);
JLabel lbl = new JLabel();
lbl.setIcon(icon);
frame.add(lbl);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

You should have a look into the details of Java Swing programming if you like to make more interactive displays. But making better GUI is not what we will do in this course.

3 Some Tips For Grayscale Image Processing

Most of the times we will be working with gray-scale images. In the previous example, we converted a color image to gray-scale. But what we saved is not actually in the gray-scale format. This is because in a gray-scale image, we only require one integer per pixel. Above program uses three (but save valued) integers for each pixel, which incurs unnecessary memory costs.

We could save the gray-scale image in a format that uses a single byte per pixel. BufferedClass provides allows us to save images in a special type 'TYPE_BYTE_GRAY'. This can be achieved by inserting a few lines of code in the program. I would suggest you to adopt the following strategy.

1. If the image you are processing is a color image
   - For each pixel, whenever necessary, get the gray-scale value by averaging RGB values as illustrated above.
   - While saving the image, set the same gray value to all three components as illustrated above.
2. If the image you are processing is already gray, whether that be in RGB or TYPE_BYTE_GRAY format, the getRGB() function will return equal values for all the three colors. So, you can use either of the three values and do your processing.

Theoretical discussions on image processing algorithms use matrix/array notations. So, instead of dealing with RGB components each time, it would be better if you extract red, green and blue components two three separate arrays and perform the required image processing tasks. The overall strategy for implementing image processing algorithms in Java could be as follows.

1. Extract grayscale values for the image and store it in a 2D integer array.
2. Perform the required operations on the array(s).
3. Convert the final array into an image.

This strategy is certainly not good in terms of memory overhead, but it makes programming much simple. To convert a 2D integer array into an image, you can use the following piece of code.

```
/*
The integer array A[][] needs to be converted to image.
So, first declare a new BufferedImage object with size equal to that of
the array */
BufferedImage image = new BufferedImage(A.length,A[0].length,
                                        BufferedImage.TYPE_BYTE_GRAY);
//now copy the values of A to the pixels of image
for(int x=0; x<image.getWidth(); x++)
        for(int y=0;y<image.getHeight();y++) {
                Color newColor = new Color(A[x][y],A[x][y],A[x][y]);
                image.setRGB(x, y,newColor.getRGB());
        }

//now the image is ready, you can save or display it, whatever required.
```

4 Problems

You can attempt these questions modifying and extending the example program given previously. For each of the questions, as suggested in section 3, first extract the grayscale values to a 2D integer array, apply the appropriate processing algorithms on the array, and convert the array into an image.

Before attempting the following problems, you need to convert the given example into a modular code. I will provide you the hint in the lab.

1. Create a binary image from a grayscale image by thresholding. That is, for example, you can convert to binary as following. Given that a pixel has intensity r, if $r > 128$, increase its intensity to 256, otherwise decrease it to 0.

2. In question 1, you created a binary image using an arbitrary threshold of 128. Instead, it could be better if we use the mean as the threshold. Modify your program for question 1 to convert a grayscale image to binary using mean intensity as the threshold.

3. Modify your program for question 2 using median intensity as the threshold, instead of the mean intensity.

4. Write short note on the following.

   (a) BufferedImage.TYPE_BYTE_GRAY

Note: You need to submit the source code of the functions for each questions in your report. Do not write the whole program code.