CS-GY 6643 Project 1: Extended Otsu's method

By Vin Liu zl1477

Code:

You could also find this code in the file `Project1.java`.

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

public class Project1 {
    private final static String[] FILENAMES = {"data13", "fruits2b",
"tiger1-24bits"};
    private final static int[][] DIMENSIONS = {{320, 240}, {484, 363},
{690, 461}};

    public static void main(String[] args) {
        BufferedImage image = null;
        for (int i = 0; i < FILENAMES.length; i++) {
            // Read image file
            image = readImage(FILENAMES[i], DIMENSIONS[i][0],
DIMENSIONS[i][1]);
            // Convert to grayscale
            image = convertToGrayScale(image);
            // Generate magnitude distribution of all pixels
            double[] distribution = generateDistribution(image);
            // Compute thresholds
            int[] thresholds = computeThresholds(distribution);
            // Write image
            writeImage(image, FILENAMES[i], thresholds);
            System.out.println("-------------");
        }
    }

    private static BufferedImage readImage(String fileName, int width, int
height) {
        BufferedImage image = null;
        try {
            // Read in the image
            File inputFile = new File(fileName + ".bmp");
            image = new BufferedImage(width, height,
                    BufferedImage.TYPE_INT_RGB);
            image = ImageIO.read(inputFile);
            System.out.println("Reading in " + fileName + ".bmp");
            return image;
        } catch (IOException e) {
```

```
            System.out.println("Error: " + e);
            return null;
        }
    }

    private static BufferedImage convertToGrayScale(BufferedImage image) {
        // Get width & height of the image
        int width = image.getWidth();
        int height = image.getHeight();

        // Convert to grayscale image
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int p = image.getRGB(x, y);
                int r = (p >> 16) & 0xff;
                int g = (p >> 8) & 0xff;
                int b = (p >> 0) & 0xff;
                int gray = (int) Math.round(0.299 * r + 0.587 * g + 0.114
* b);
                p = (gray << 16) | (gray << 8) | (gray << 0);
                image.setRGB(x, y, p);
            }
        }
        return image;
    }

    private static void writeImage(BufferedImage image, String fileName,
int[] thresholds) {
        // Modify image according to the threshold
        int width = image.getWidth();
        int height = image.getHeight();
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int p = image.getRGB(x, y);
                int v = p & 0xff;
                if (v <= thresholds[0]) {
                    image.setRGB(x, y, (0 << 16) | (0 << 8) | (0 << 0));
                } else if (v <= thresholds[1]) {
                    image.setRGB(x, y, (85 << 16) | (85 << 8) | (85 <<
0));
                } else if (v <= thresholds[2]) {
                    image.setRGB(x, y, (170 << 16) | (170 << 8) | (170 <<
0));
                } else {
                    image.setRGB(x, y, (255 << 16) | (255 << 8) | 255);
                }
            }
        }

        try {
            // Write the image into file
            File file = new File(fileName + "_out.bmp");
            ImageIO.write(image, "bmp", file);
            System.out.println(fileName + "_out.bmp is written");
```

```
        } catch (IOException e) {
            System.out.println(e);
        }
    }

    private static double[] generateDistribution(BufferedImage image) {
        double[] distribution = new double[256];
        int width = image.getWidth();
        int height = image.getHeight();

        // Read every pixel and plot a histogram of counts
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int p = image.getRGB(x, y);
                int v = p & 0xff;
                distribution[v]++;
            }
        }

        // Normalize the histogram to probabilities
        for (int i = 0; i < 256; i++) {
            distribution[i] /= (width * height);
        }

        return distribution;
    }

    private static double computeWeight(double[] distribution, int
boundaryLow, int boundaryHigh) {
        double weight = 0;
        // Add up the probabilities to get weight
        for (int i = boundaryLow; i <= boundaryHigh; i++) {
            weight += distribution[i];
        }

        return weight;
    }

    private static double computeVariance(double[] distribution, int
boundaryLow, int boundaryHigh, double weight) {
        // Calculate the mean
        double mean = 0;
        for (int i = boundaryLow; i <= boundaryHigh; i++) {
            mean += distribution[i] * i / weight;
        }

        // Calculate the variance
        double variance = 0;
        for (int i = boundaryLow; i <= boundaryHigh; i++) {
            variance += (i - mean) * (i - mean) * distribution[i] /
weight;
        }

        return variance;
```

```
    }

    private static int[] computeThresholds(double[] distribution) {
        // Calculate the thresholds
        System.out.println("Computing thresholds");
        int[] thresholds = new int[3];
        double varianceOverall = Double.MAX_VALUE;
        for (int i = 0; i < 254; i++) {
            for (int j = i + 1; j < 255; j++) {
                for (int k = j + 1; k < 256; k++) {
                    double[] weights = {computeWeight(distribution, 0, i),
computeWeight(distribution, i, j), computeWeight(distribution, j, k),
computeWeight(distribution, k, 255)};
                    double varianceTemp = weights[0] *
computeVariance(distribution, 0, i, weights[0]) + weights[1] *
computeVariance(distribution, i, j, weights[1]) + weights[2] *
computeVariance(distribution, j, k, weights[2]) + weights[3] *
computeVariance(distribution, k, 255, weights[3]);
                    // When new minimum is found
                    if (varianceTemp < varianceOverall) {
                        thresholds = new int[]{(int) i, (int) j, (int) k};
                        varianceOverall = varianceTemp;
                    }
                }
            }
        }
        System.out.println("t1: " + thresholds[0] + "\nt2: " +
thresholds[1] + "\nt3: " + thresholds[2]);

        return thresholds;
    }
}
```

How to run

1. Please make sure `Project1.java` is in the same folder with `fruits2b.bmp`, `data13.bmp`, and `tiger1-24bits.bmp`.
2. Run the following commands in your terminal:

```
javac Project1.java

java Project1
```

Results and outputs

For `fruits2b.bmp`:

t1 = 88

t2 = 141

t3 = 173

For `data13.bmp`:

t1 = 77

t2 = 155

t3 = 222

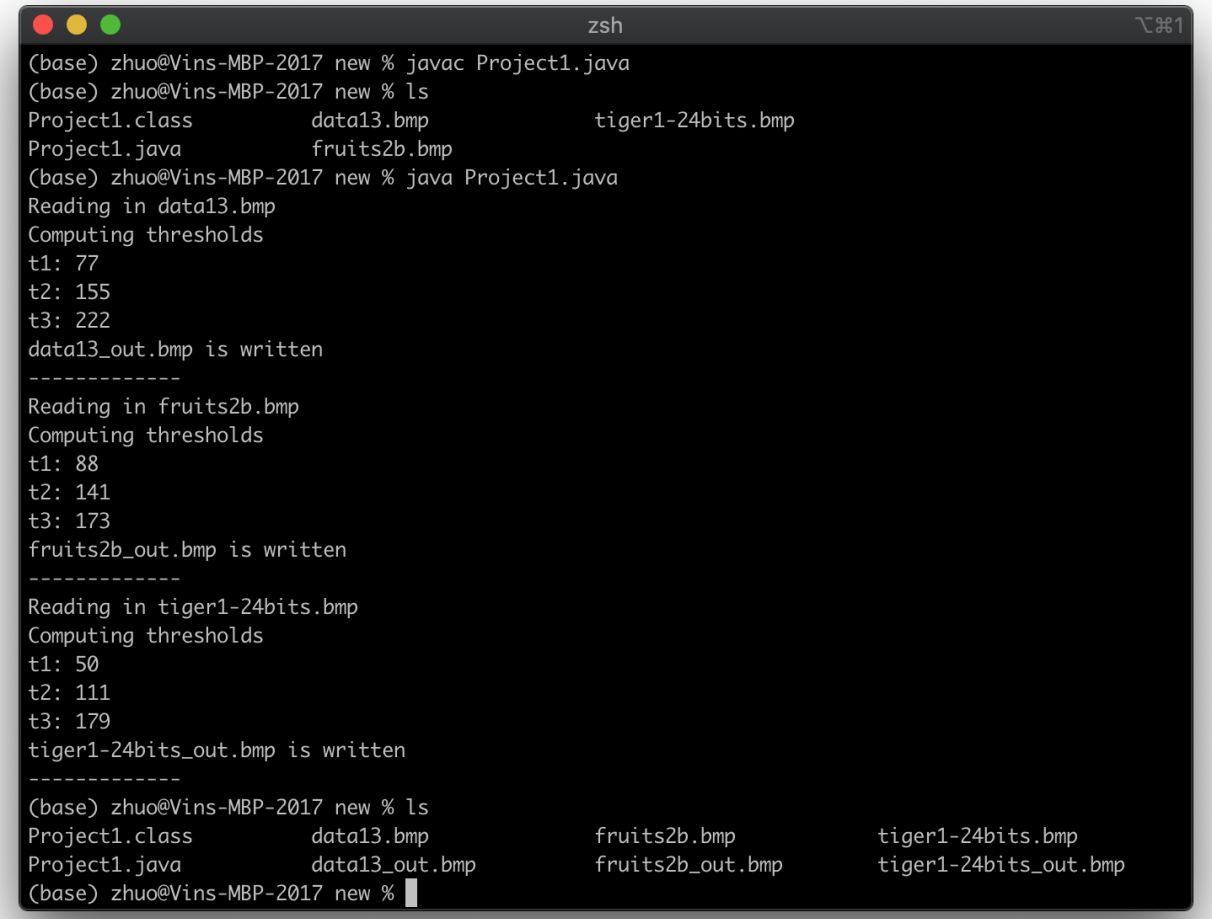
For `tiger1-24bits.bmp`:

t1 = 50

t2 = 111

t3 = 179

Screenshot of the overall results:


zsh
⌥⌘1
(base) zhuo@Vins-MBP-2017 new % javac Project1.java
(base) zhuo@Vins-MBP-2017 new % ls
Project1.class data13.bmp tiger1-24bits.bmp
Project1.java fruits2b.bmp
(base) zhuo@Vins-MBP-2017 new % java Project1.java
Reading in data13.bmp
Computing thresholds
t1: 77
t2: 155
t3: 222
data13_out.bmp is written
Reading in fruits2b.bmp
Computing thresholds
t1: 88
t2: 141
t3: 173
fruits2b_out.bmp is written
Reading in tiger1-24bits.bmp
Computing thresholds
t1: 50
t2: 111
t3: 179
tiger1-24bits_out.bmp is written
(base) zhuo@Vins-MBP-2017 new % ls
Project1.class data13.bmp fruits2b.bmp tiger1-24bits.bmp
Project1.java data13_out.bmp fruits2b_out.bmp tiger1-24bits_out.bmp
(base) zhuo@Vins-MBP-2017 new %