Image Classification and Processing Report

Fares Wael

May 10, 2025

Abstract

This report provides a detailed analysis of a Java-based image classification and processing program. The program handles image loading, organization, processing (including RGB and YUV color space conversions), similarity calculations, and dataset statistics generation. The report includes visual examples of RGB and YUV processed images, a structural overview of the code, and sample terminal output.

Contents

1 Introduction **2**

2 Code Structure **2**

3 Image Processing: RGB and YUV **2**
3.1 RGB Processing 3
3.2 YUV Processing 3

4 Terminal Output **4**

5 Code Example **5**

6 Conclusion **6**

7 References **6**

1 Introduction

The Image Classification and Processing project is a Java application designed to process and analyze images from a dataset organized into categories (`Animals`, `Faces`, `Nature`) and directories (`train`, `test`). Key functionalities include:

- Loading and organizing images by category.
- Processing images (resizing, grayscale conversion, YUV conversion).
- Calculating image similarity in RGB and YUV color spaces.
- Generating dataset statistics and compression ratio reports.
- Compressing processed outputs into a ZIP file.

This report explains the code structure, demonstrates RGB and YUV processing with sample images, and provides terminal output examples.

2 Code Structure

The program is implemented in a single Java class, `ImageClassification`, with modular methods for different functionalities. Below is a visual representation of the code structure. The main components include:

- **Main Method**: Orchestrates the workflow, calling functions for dataset processing, image processing, similarity calculations, and compression.
- **Dataset Processing**: Methods like `processDataset` and `printStatistics` handle dataset analysis.
- **Image Processing**: Methods such as `processImage`, `convertToYUV`, and `saveYUVImage` manage image transformations.
- **Similarity Analysis**: `calculateImageSimilarity` and `calculateYUVSimilarity` compute image similarities.
- **Reporting**: `generateCompressionRatioReport` and `batchProcessToYUV` produce detailed reports.
- **File Management**: `organizeImages` and `compressOutputToZip` handle output organization and compression.

3 Image Processing: RGB and YUV

The program processes images in both RGB and YUV color spaces, each with distinct characteristics.

```
Starting Image Classification and Processing...
Found 10 images in train/Animals
Found 10 images in train/Faces
Found 9 images in train/Nature
Found 5 images in test/Animals
Found 5 images in test/Faces
Found 5 images in test/Nature

--- Dataset Statistics ---
Animals: 15 images
Faces: 15 images
Nature: 14 images
Total images: 44

--- RGB vs YUV Comparison ---
Original image size: 9909 bytes
RGB Processing:
  - Processing time: 8 ms
  - Output size: 12045 bytes
  - Compression ratio: 0.82
YUV Processing:
  - Processing time: 14 ms
  - Output size: 7580 bytes
  - Compression ratio: 1.31
RGB Quality (PSNR): 39.70 dB
YUV Quality (PSNR): 12.64 dB
Conclusion: YUV provides better compression
RGB Similarity: 0.8401283081538626
YUV Similarity: 0.9465209233663107
Compression ratio report generated: output\reports\compression_ratio_report.txt
Images organized successfully in: output\organized
YUV processing report generated: output\yuv_processing_report.txt
Successfully batch processed all images to YUV
Successfully compressed output to zip file
Image Classification and Processing completed successfully!
```

Figure 1: Structure of the ImageClassification Java Code

3.1 RGB Processing

RGB (Red, Green, Blue) is the standard color model for digital images. The program resizes images and optionally converts them to grayscale. Below is a sample RGB-processed image.

3.2 YUV Processing

YUV separates luminance (Y) from chrominance (U, V), often used in compression due to human vision's sensitivity to luminance. The `convertToYUV` method converts RGB pixels to YUV using the following equations:

$$Y = 0.299R + 0.587G + 0.114B \tag{1}$$

$$U = 0.492(B - Y) \tag{2}$$

$$V = 0.877(R - Y) \tag{3}$$

The YUV image is visualized by mapping Y, U, and V to RGB channels. Below is a sample YUV-processed image.

Figure 2: RGB Processed Image (Animal Category)

Figure 3: YUV Processed Image (Animal Category)

4 Terminal Output

The program generates detailed console output, including dataset statistics, similarity scores, and processing results. Below is a sample terminal output. Example output includes:

- Dataset statistics (e.g., number of images per category).
- RGB and YUV similarity scores (e.g., `RGB Similarity: 0.85`).
- Compression ratio analysis (e.g., `RGB Compression Ratio: 1.45`).
- Confirmation of file organization and ZIP compression.

```
Starting Image Classification and Processing...
Found 10 images in train/Animals
Found 10 images in train/Faces
Found 9 images in train/Nature
Found 5 images in test/Animals
Found 5 images in test/Faces
Found 5 images in test/Nature

--- Dataset Statistics ---
Animals: 15 images
Faces: 15 images
Nature: 14 images
Total images: 44

--- RGB vs YUV Comparison ---
Original image size: 9909 bytes
RGB Processing:
  - Processing time: 8 ms
  - Output size: 12045 bytes
  - Compression ratio: 0.82
YUV Processing:
  - Processing time: 14 ms
  - Output size: 7580 bytes
  - Compression ratio: 1.31
RGB Quality (PSNR): 39.70 dB
YUV Quality (PSNR): 12.64 dB
Conclusion: YUV provides better compression
RGB Similarity: 0.8401283081538626
YUV Similarity: 0.9465209233663107
Compression ratio report generated: output\reports\compression_ratio_report.txt
Images organized successfully in: output\organized
YUV processing report generated: output\yuv_processing_report.txt
Successfully batch processed all images to YUV
Successfully compressed output to zip file
Image Classification and Processing completed successfully!
```

Figure 4: Sample Terminal Output

5 Code Example

Below is a key method, `convertToYUV`, which demonstrates the YUV conversion process.

```
1  private static BufferedImage convertToYUV(File input) throws
       IOException {
2      BufferedImage originalImage = ImageIO.read(input);
3      int width = originalImage.getWidth();
4      int height = originalImage.getHeight();
5
6      BufferedImage yuvImage = new BufferedImage(width, height,
           BufferedImage.TYPE_INT_RGB);
7
8      for (int y = 0; y < height; y++) {
9          for (int x = 0; x < width; x++) {
10             Color pixel = new Color(originalImage.getRGB(x, y));
11
```

```
            double r = pixel.getRed();
            double g = pixel.getGreen();
            double b = pixel.getBlue();

            double Y = 0.299 * r + 0.587 * g + 0.114 * b;
            double U = 0.492 * (b - Y);
            double V = 0.877 * (r - Y);

            int yVal = (int) Math.min(255, Math.max(0, Y));
            int uVal = (int) Math.min(255, Math.max(0, U + 128));
            int vVal = (int) Math.min(255, Math.max(0, V + 128));

            Color yuvPixel = new Color(yVal, uVal, vVal);
            yuvImage.setRGB(x, y, yuvPixel.getRGB());
        }
    }

    return yuvImage;
}
```

6 Conclusion

The Image Classification and Processing program effectively handles image processing tasks, with robust support for both RGB and YUV color spaces. The YUV conversion leverages human visual perception for better compression, as evidenced by the compression ratio reports. The modular code structure ensures maintainability, and the generated reports provide valuable insights into dataset characteristics and processing outcomes.

7 References

- Java AWT Documentation: https://docs.oracle.com/javase/8/docs/api/java/awt/package-summary.html
- YUV Color Space: https://en.wikipedia.org/wiki/YUV