University of Washington—Bothell

Jpeg Analyzer

Rajneet Dhanju

Inclusions: JpegAnalyzer source files

Table of Contents

Introduction

The overall goal of the research is to use a series of images that will be compared with one another based on their RGB (red, green, blue) values for a factor we would like to call ‘closeness’. Then based on the closeness factor between two images we can make a suggestion about the extent to which the two images are similar.

Algorithm Description

Initial Approach

The initial approach of comparing the images using their RGB values was to use the munsell color system. After considerable research, we quickly realized that this would not be an easy or reliable approach. There is no direct way to convert RGB to munsell color system—idea was to convert and compare the RGB values for closeness. In order to utilize the munsell color system 2 steps have to be taken: 1-convert RGB to XYZ representation, 2-then convert from XYZ to munsell representation. However, according to my research of this conversion system I realized that I would have to either a 3rd party software to do this or implement the conversion system myself—which seemed to take away from the core goals of the research. Doing so would take up majority of my time to write the conversion system—which also seemed to be unreliable. The munsell color system is considered to be the closest representation to real-world colors. Taking RGB values and finding their respective munsell color representation might create one munsell representation for multiple different RGB values.

There are some basic conversion tables available for RGB to munsell however they are incomplete—meaning they don’t have a munsell representation for every possible RGB combination.

Assumptions

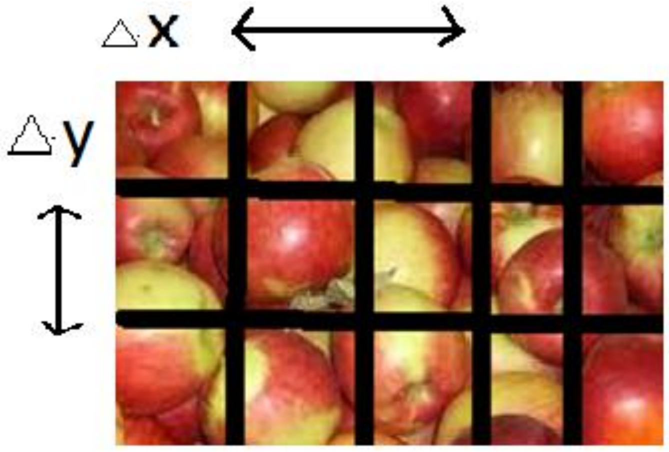
The following (next sub-heading) implementation approach makes these assumptions:

- images are not distorted in any manner (discoloration)
- two images that are the same (except one is in color and the other is black-white) will be considered as two distinct images as we compare based on RGB values

**Note: Algorithm takes into account images that are not the same size (different image sizes should not affect the outcome/results)

Implemented Approach

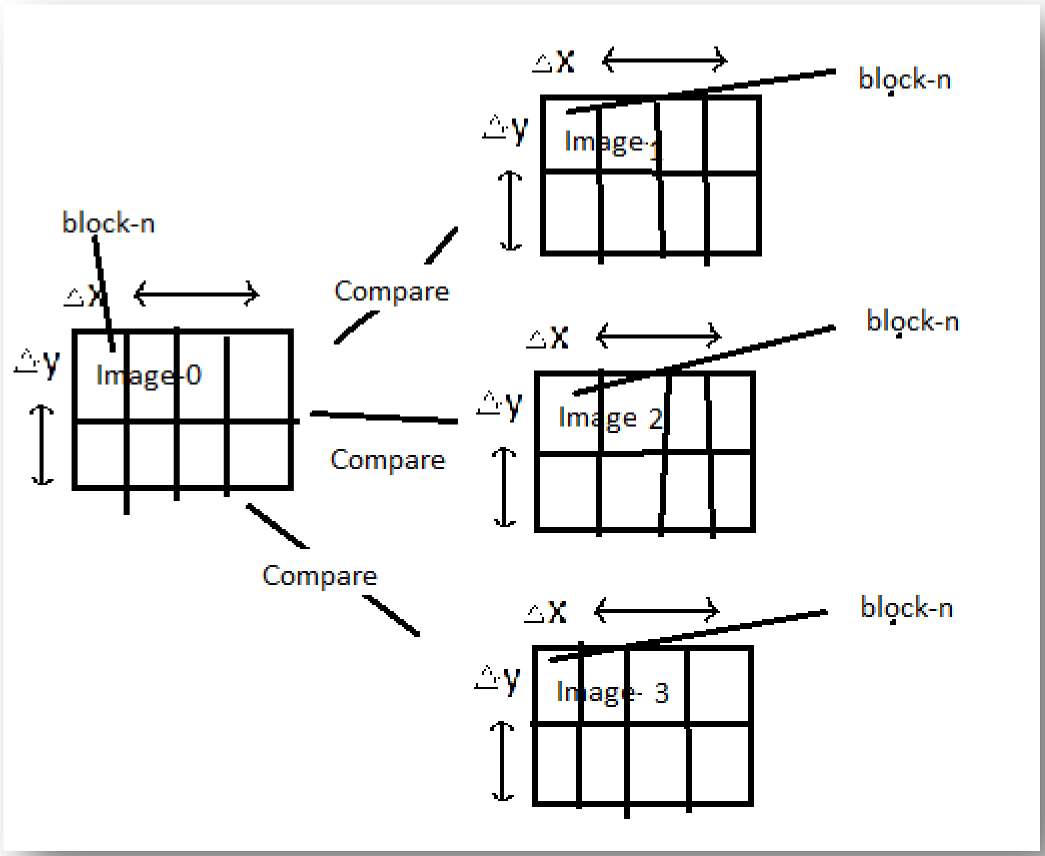
Figure 1: Algorithm Example



General Comparison Algorithm logic

1. Define a delta x and delta y for the purpose of referring to a subsection of the image
    - this will be used to divide the image into subsections **(called blocks)**
    - for example: in this algorithm I have use a delta x = 10 and delta y = 10
    - therefore each image will be divided into 100 blocks (10x10)
    - **Note:** the smaller the delta (x and y) the more precise the image comparison algorithm will be
2. Get each block's value for each image (ColorImageScale.java, setpallet(…))
    - For each block
    - Add up all the red, green, and blue pixel values
    - Calculate the average (from summation above of the RGB values)
3. Argument 0 will define the target image to which we want to compare all of our images with

Figure 2: Block Comparison



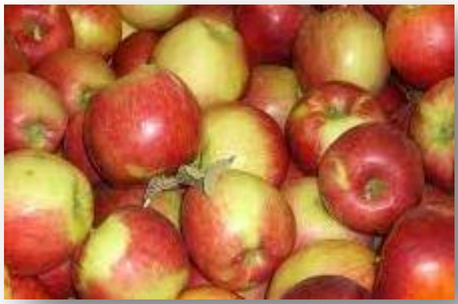
- For each image following image-0 (target image) compare each block-n with the corresponding block in image-0 (see figure 2 above)
- Calculate the relative percentage for each color (for each block)
  - Example
    - (maxRed – minRed) / maxRed
    - (maxBlue – minBlue) / maxBlue
    - (maxGreen – minGreen) / maxGreen
- Step-3 implemented in BlockCollection.java calculateClosenessFactor( )

4. For each image
   - sum up the RGB
   - take the average of each color
   - sum up the averages
   - implemented in BlockCollection.java summation( )
5. Now we have a single value that represents the closeness factor for each image with the target image (image-0)
6. Perform merge sort algorithm on each closeness factor to rate them and print them out in order of most-close (lower closeness factor) to least-close (larger closeness factor)

Input

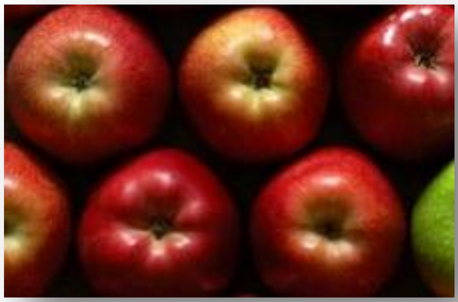
The majority of the implemented algorithm is tested on the following jpeg files (actual size). **NOTE:** During the course of using the JpegAnalyzer (for the actual raw image analysis) I have noticed that image larger than 600x600 pixels will cause the program to crass with an “Java out-of-memory error”.

Picture 1



File Size = 16 KB
Dimensions = 225 x 147

Picture 2



File Size = 19.6 KB
Dimensions = 225 x 146

Picture 3

File Size = 9.22 KB
Dimensions = 183 x 275

Picture 4

File Size = 4.39 KB
Dimensions = 265 x 190

Picture 5

File Size = 10.4 KB
Dimensions = 350 x 275

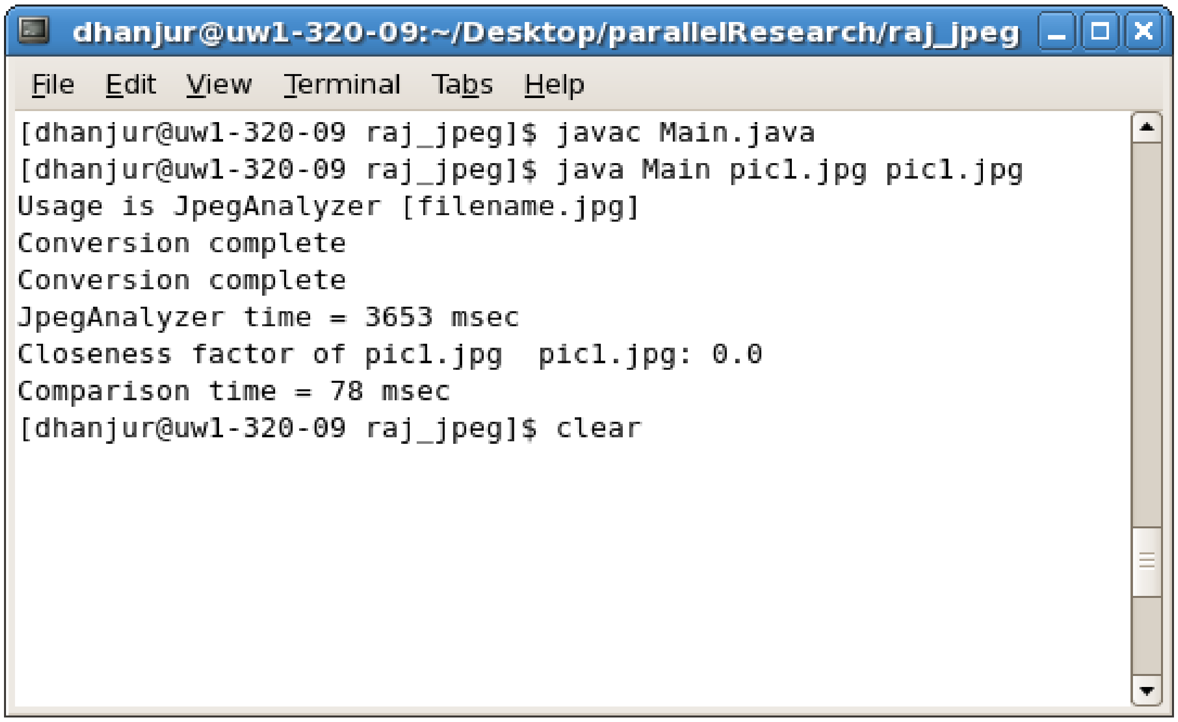
Picture 6

File Size = 4.71 KB
Dimensions = 213 x 237

Output (execution)

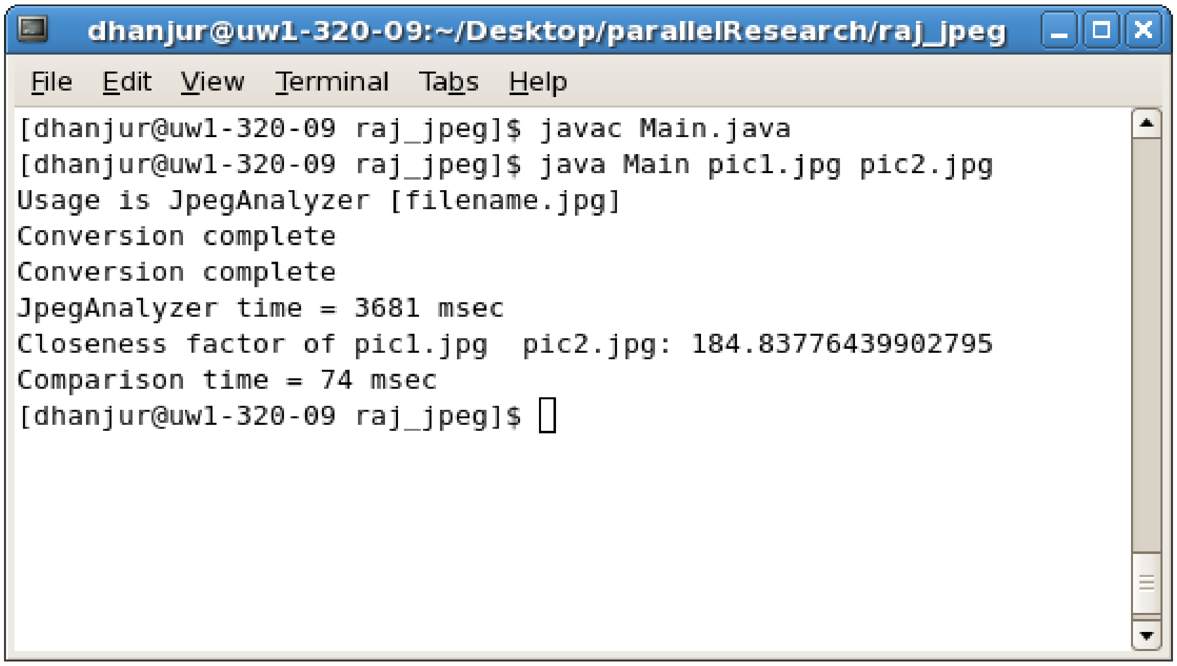
Without MASS

Figure 3: Same Images (2)


dhanjur@uwl-320-09:~/Desktop/parallelResearch/raj_jpeg
File Edit View Terminal Tabs Help

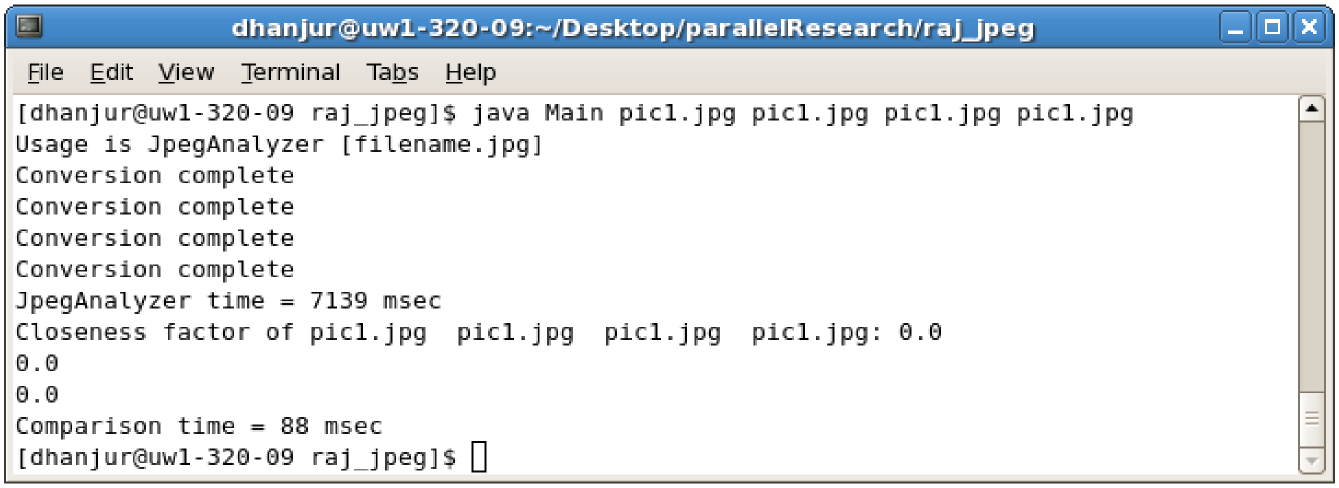
```
[dhanjur@uwl-320-09 raj_jpeg]$ javac Main.java
[dhanjur@uwl-320-09 raj_jpeg]$ java Main pic1.jpg pic1.jpg
Usage is JpegAnalyzer [filename.jpg]
Conversion complete
Conversion complete
JpegAnalyzer time = 3653 msec
Closeness factor of pic1.jpg  pic1.jpg: 0.0
Comparison time = 78 msec
[dhanjur@uwl-320-09 raj_jpeg]$ clear
```

Figure 4: Different Images (2)


dhanjur@uwl-320-09:~/Desktop/parallelResearch/raj_jpeg
File Edit View Terminal Tabs Help

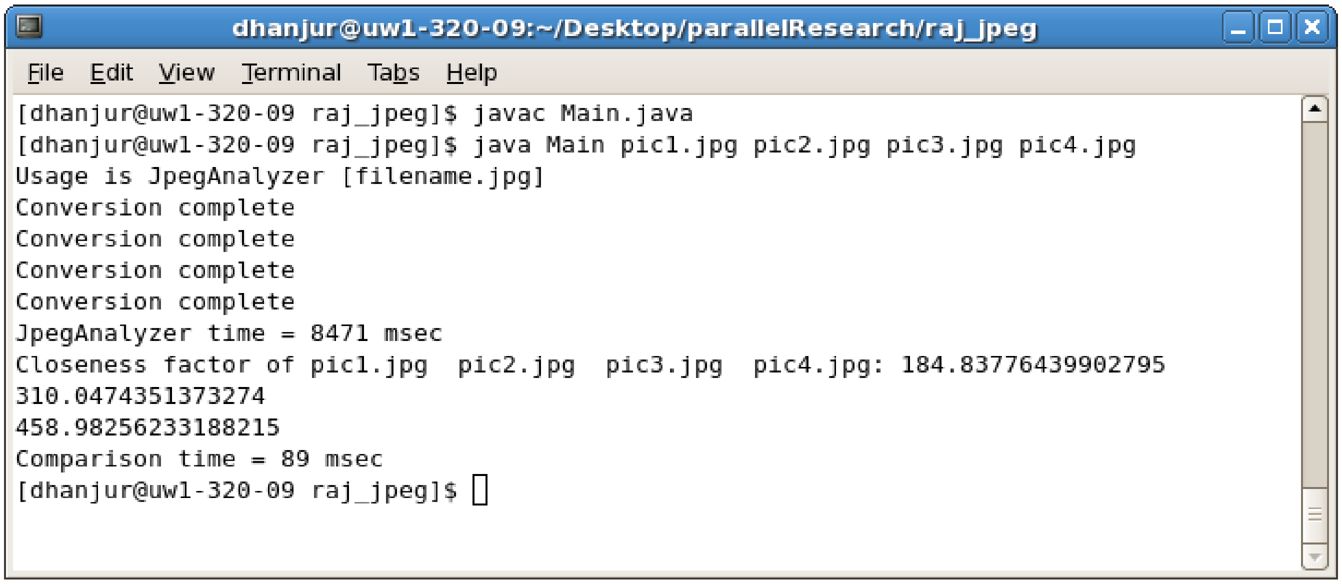
```
[dhanjur@uwl-320-09 raj_jpeg]$ javac Main.java
[dhanjur@uwl-320-09 raj_jpeg]$ java Main pic1.jpg pic2.jpg
Usage is JpegAnalyzer [filename.jpg]
Conversion complete
Conversion complete
JpegAnalyzer time = 3681 msec
Closeness factor of pic1.jpg  pic2.jpg: 184.83776439902795
Comparison time = 74 msec
[dhanjur@uwl-320-09 raj_jpeg]$ 
```

Figure 5: Same Images (4)


dhanjur@uw1-320-09:~/Desktop/parallelResearch/raj_jpeg
File Edit View Terminal Tabs Help

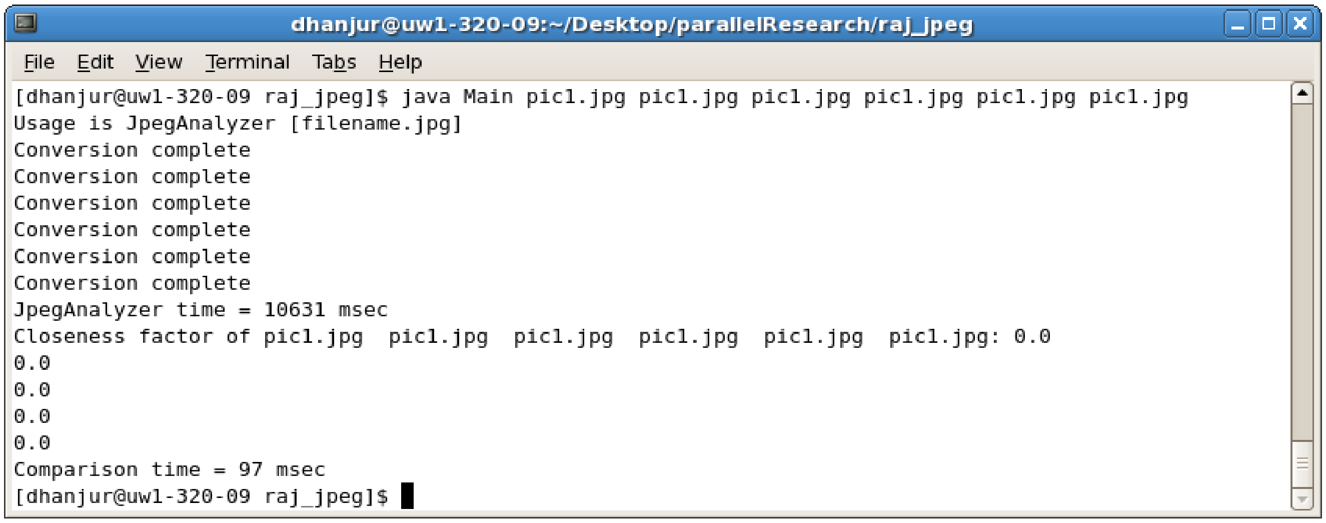
```
[dhanjur@uw1-320-09 raj_jpeg]$ java Main pic1.jpg pic1.jpg pic1.jpg pic1.jpg
Usage is JpegAnalyzer [filename.jpg]
Conversion complete
Conversion complete
Conversion complete
Conversion complete
JpegAnalyzer time = 7139 msec
Closeness factor of pic1.jpg  pic1.jpg  pic1.jpg  pic1.jpg: 0.0
0.0
0.0
Comparison time = 88 msec
[dhanjur@uw1-320-09 raj_jpeg]$ 
```

Figure 6: Different Images (4)


dhanjur@uw1-320-09:~/Desktop/parallelResearch/raj_jpeg
File Edit View Terminal Tabs Help

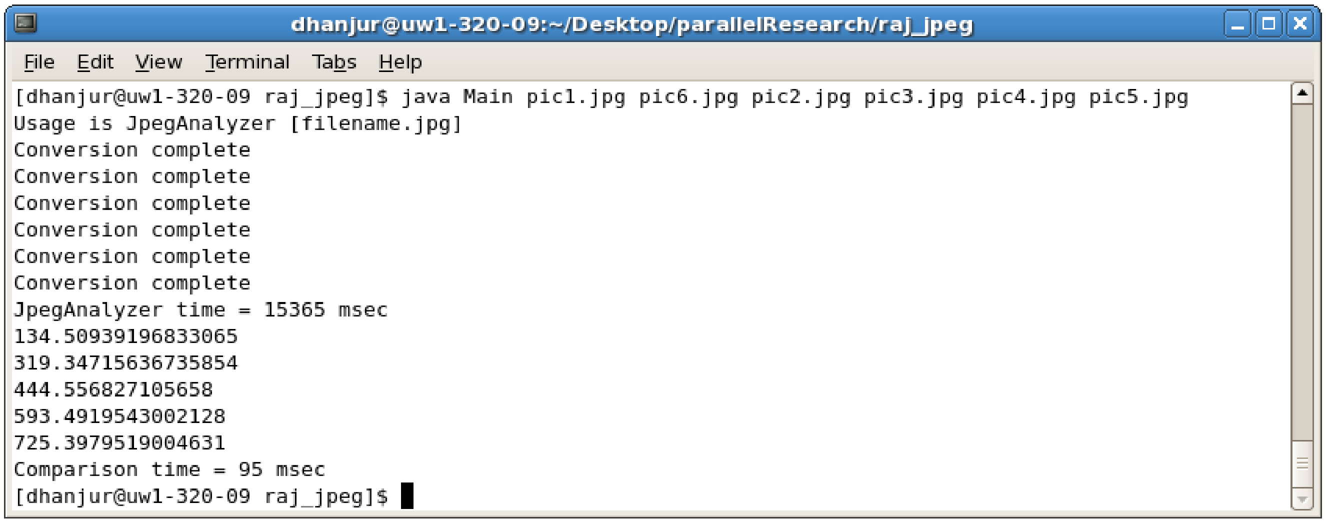
```
[dhanjur@uw1-320-09 raj_jpeg]$ javac Main.java
[dhanjur@uw1-320-09 raj_jpeg]$ java Main pic1.jpg pic2.jpg pic3.jpg pic4.jpg
Usage is JpegAnalyzer [filename.jpg]
Conversion complete
Conversion complete
Conversion complete
Conversion complete
JpegAnalyzer time = 8471 msec
Closeness factor of pic1.jpg  pic2.jpg  pic3.jpg  pic4.jpg: 184.83776439902795
310.0474351373274
458.98256233188215
Comparison time = 89 msec
[dhanjur@uw1-320-09 raj_jpeg]$ 
```

Figure 7: Same Images (6)


dhanjur@uw1-320-09:~/Desktop/parallelResearch/raj_jpeg
File Edit View Terminal Tabs Help

```
[dhanjur@uw1-320-09 raj_jpeg]$ java Main pic1.jpg pic1.jpg pic1.jpg pic1.jpg pic1.jpg pic1.jpg
Usage is JpegAnalyzer [filename.jpg]
Conversion complete
Conversion complete
Conversion complete
Conversion complete
Conversion complete
Conversion complete
JpegAnalyzer time = 10631 msec
Closeness factor of pic1.jpg  pic1.jpg  pic1.jpg  pic1.jpg  pic1.jpg  pic1.jpg: 0.0
0.0
0.0
0.0
0.0
Comparison time = 97 msec
[dhanjur@uw1-320-09 raj_jpeg]$
```

Figure 8: Different Images (6)


dhanjur@uw1-320-09:~/Desktop/parallelResearch/raj_jpeg
File Edit View Terminal Tabs Help

```
[dhanjur@uw1-320-09 raj_jpeg]$ java Main pic1.jpg pic6.jpg pic2.jpg pic3.jpg pic4.jpg pic5.jpg
Usage is JpegAnalyzer [filename.jpg]
Conversion complete
Conversion complete
Conversion complete
Conversion complete
Conversion complete
Conversion complete
JpegAnalyzer time = 15365 msec
134.50939196833065
319.34715636735854
444.556827105658
593.4919543002128
725.3979519004631
Comparison time = 95 msec
[dhanjur@uw1-320-09 raj_jpeg]$
```

Performance

Without MASS Library

Table 1: Performance without MASS

Number of images	Same images? (Y/N)	Execution Time JpegAnalyzer (msec)	Execution Time Comparison Algorithm (msec)
2	Y	3653	78
2	N	3681	74
4	Y	7139	88
4	N	8471	89
6	Y	10631	97
6	N	15365	95

With MASS Library

Table 2: Performance with MASS

Number of images	Same images? (Y/N)	Execution Time JpegAnalyzer (msec)	Execution Time Comparison Algorithm (msec)
2	Y		
2	N		
4	Y		
4	N		
6	Y		
6	N		

Concluding Thoughts

Challenges and Difficulties

Some of the difficulties that I faced during the project include

- Understanding how the JpegAnalyzer works (inherited code) and how to withdraw the information I need from it (raw pixel values of each image)
- Some test code existed within the JpegAnalyzer that took me a while to figure out that it was not a main part of the program (system)
- Working with the MASS Library to parallelize the JpegAnalyzer part

  I was not able to accomplish this part of the program successfully as I had difficulties understanding how the library actually works. I had access to the manual and sources code—however, the manual did not fully explain how to use each of the function calls. There is one main sample code provided that uses the library (Wave2DMass.java); however, it does not showcase each function use with the various different overridden functions.

  For example, my parallelization requires me to use each argument passed into the command as arguments to each of the elements in 'Places'. So I want to be able to use the callAll function and pass in the arguments so that they can be used for the parallelization process. I followed the example and the manual to the best of my understanding and made the following changes in the JpegAnalyzer.java to accommodate the MASS library:

  - public JpegAnalyzer(Object fileName)
  - public callMethod(int funcId, Object fileName)
  - public init(Object fileName)

  The callMethod must be overridden in order for the callAll(..) function to work properly—which is what I implemented in my attempt to parallelize. The issues I ran into this were that the callAll function seems to work for the 1st image I pass in but not for any subsequent images. For any image after the first one I get null pointer errors.

  I made multiple attempts to explain and resolve this problem by consulting with Tim (the author of the library)—however I was unsuccessful. Tim was only available to work with me remotely. I was having problems that I could not explain clearly enough remotely and needed to show him in person—unfortunately we were not able to make any accommodations to meet in person. Therefore I was unable to answer my questions/problems and thus unable to implement the parallel portion of the program.

  I also tried to receive help from Tosa (graduate student working on the C++ version of the MASS library)—and he indicated to me that he faced similar problems while working with this particular version of the MASS library. He too was unable to write a simple application with MASS using certain function.

Overall, my experience with the MASS library has been really frustrating and at times I felt that I was spending too much time on something that should not require that much time. I think that the MASS library can be a great tool for parallelization of applications—however it is not yet user friendly. The manual provide a good overview (concepts) of how the MASS library works and communicates between processes and threads (work-load) however it does not fully explain to an outsider what steps have to be taken to actually use the library. The documentation of each of the function need to have specific examples of their use rather than a description of what they do—this was the most frustrating part as I had to look at the Wave2DMass.java sources code repeatedly to have a sense of how these functions were getting called and used (user defined override functions—ex: callMethod(..) that communicates by callAll(..) function).

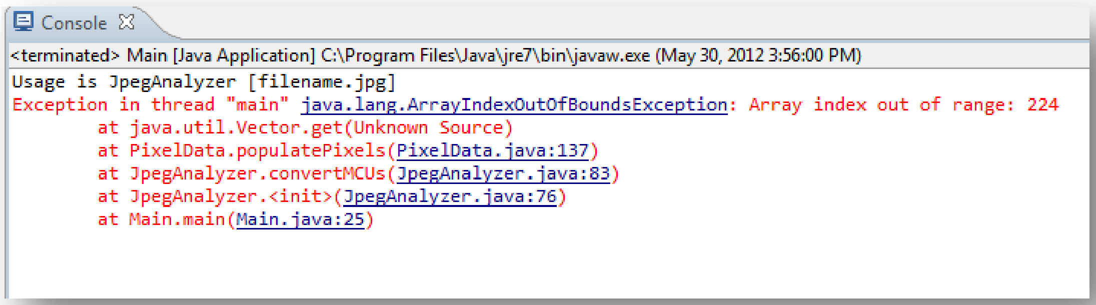
Results and Accomplishments

I have a working version of my comparison algorithm as explained above. It is able to calculate the compare 2 images and show their relative closeness as seen in the execution output section above.
However, there are some points to be noted about this algorithm as I have not been able to detect if it is the comparison that is causing these issues or the JpegAnalyzer:

- when comparing alike images: example pic1.jpg and pic1.jpg pic1.jpg
    - the closeness factor = 0 (correct)
- when comparing images in order of: pic1.jpg pic2.jpg pic1.jpg
    - the results will be pic2.jpg closeness factor = xxx
    - and pic1.jpg (3rd argument) closeness factor = xxx as well when in this case it should be 0 because it is referring to pic1.jpg which is the same image
    - this reveals that order of image input matter when it shouldn't
    - I have not been able to resolve this issue even with countless step-through and debugging

Problems with Jpeg Analyzer

Does not like images where x and y values (size) are the same (example 225 x 225) gives the following error—off by one error mistake in JpegAnalyzer—was unable to detect the exact location that was resulting in this outcome.



```
<terminated> Main [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (May 30, 2012 3:56:00 PM)
Usage is JpegAnalyzer [filename.jpg]
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Array index out of range: 224
	at java.util.Vector.get(Unknown Source)
	at PixelData.populatePixels(PixelData.java:137)
	at JpegAnalyzer.convertMCUs(JpegAnalyzer.java:83)
	at JpegAnalyzer.<init>(JpegAnalyzer.java:76)
	at Main.main(Main.java:25)
```

Dictionary

Term	Definition
Block	Refers to each subsection of an image each block size is defined by delta x and delta y (x time y = number of pixels included in that block) Refer to Figure 1.
Delta x and/or y	The value that refers to the number of pixels that will be looked at—for any given calculation (see Figure 1 for visual representation)
Relative value (percentage)	Equation used to represent each block's red, green, blue pixel in one value: i.e. (maxValue – minValue) / (maxValue) (% red_max - %red_min) / (red_max) Refer to Figure 2
Target Image	The image to which all the other images will be compared with.

How-to-Use Guide

1. Save each image to perform comparison on in the same directory as the source files
2. Run program as follows
   - java Main < target image name> <image 1> <image 2> … <image n>
   - For example: java Main pic1.jpg pic2.jpg pic3.jpg
3. For execution of the program see Output section above
   - Refer to 'Concluding Thoughts' section for known bugs with the program

Credits & References

Credits

- Professor Munehiro Fukuda (University of Washington—Bothell)
- Professor Clark Olson (University of Washington—Bothell)
- Tosa Ojiru (Master's Student, University of Washington—Bothell)
- Tim Chuang (Master's Student, University of Washington—Bothell)

References

MASS Library—found at dslab@hercules/SensorGrid/MASS/MASS-Merged/*

http://en.wikipedia.org/wiki/Merge_sort

http://www.brucelindbloom.com/index.html?ColorCalcHelp.html

http://www.brucelindbloom.com/index.html?Calc.html

http://munsell-to-rgb.blogspot.com/

http://casoilresource.lawr.ucdavis.edu/drupal/node/201

http://depts.washington.edu/dslab/SensorCloud/index.html

http://www.designersilverlight.com/wp-content/uploads/2009/09/ColorScaleImageRGB1.png