Color-based Image Retrieval

A Machine Project in CSC741M

Richard Louie T. Orilla

College of Computing Education

University of the Mindanao

Davao City, Philippines

*Abstract*— Image retrieval is the concept of retrieving similar images on a database base on a single image which is known as the query image. In this paper, it highlights an implementation of color-based image retrieval in C# with no libraries used aside from .NET’s build in library. Various color-based methods are implemented which include the color histogram method, perceptual similarity, histogram refinement with color coherence, and color histogram with centering refinement. The results of image retrieval using these methods are compared and their effectiveness over the other approaches is discussed.

Keywords—color based image retrieval, color histogram, perceptual similarity, histogram refinement, centering refinement

# Introduction

Retrieving similar images in a large pool of images is a very challenging task. This is because images are considered as large 2D arrays of colors and the process of analyzing individual colors can be overwhelming given that the algorithms used is not efficient for the task. An image retrieval system focuses on browsing, searching and retrieving images from a large data of images.

In this project, a color-based image retrieval is implemented using C#. The project covers the histogram method, perceptual similarity, histogram refinement with color coherence, and color histogram with centering refinement. Images will be retrieved based on the query image. The will then be sorted to its relativeness base on the algorithm used. The results of the different implementations will be compared and their effectiveness over the other approaches is discussed. The evaluation metrics include the precision, recall, and F-measures of the implemented color-based retrieval methods.

# Design and Implementation

Each color-based image retrieval method uses their own unique histograms. It done by traversing each array of pixels in the image and obtaining their raw RGBColor values and then converting it to its respective LUVColor. Dictionary file is then used to keep each unique LUVColor and track the number of colors that belongs to that LUVColor. This process happens during the creation of the ImageInfo class. During the creation of the ImageInfo class, it will try to compute the image base on different presumptions such as regular histogram, color coherence histogram and center refined histogram. This has significant in the start-up time of the program but the program creates a cache (e.g filename.jpg.cache) file on the same directory of the image during the first time it encounters the file so that during the next iteration, if the .cache file exists, it will load the cache file instead reducing the amount of time it has to start-up the program given that it has to rerun the program.

The cause of the slow load-time without the cache is because of the preprocessing done for certain histograms. For example, in the color coherent histograms it is necessary to blur the image before it is computed. The bitmap class can handle such cases but the one created and hand drawn is not entirely optimized since the entire process is done CPU bound thus taking an extra half second to process a 100x100 resized bitmap. The existence of the cache file severely reduces this because of the removal of processes that normally takes time to finish. It should be noted that resized bitmaps are converted to 8bit colors in order to quantized the colors into specific numbers.

After all the preprocessing and histogram computation. The ImageInfo class contains all the details regarding the image such as histograms of different cases which is used for image retrieval purposes. The ColorAnalyzer class is created and designed to manipulate these histograms that the ImageInfo can throw in. The design and implementation for the different color-based image retrieval methods are discussed below.

## Color Histogram

Color histograms is the count of colors present on the image. The counting of colors depends on the type of algorithm applied thus different approaches lead to different histograms regardless if two algorithms is used in the same file.

In the first approach, the histogram is simply obtained by grouping the colors and obtaining the count of colors present in the group. Then histograms from one image to the other are compared using the Histogram Intersection formula as seen below.

In the implementation, in order to appeal the SortedList of the .NET framework, a modification of the formula is used instead since Sorted List data structure are designed to sort items in ascending order. Thus it is necessary for the similar images to be sorted in the fashion from 0 to 1 instead of the opposite.

## Perceptual Similarity

The perceptual similarity is done by using the normal histogram present in case A and then is computed by finding the differences between all pairs of LUVColors in order to find the as seen here.

The next step then is to find the by multiplying with p, which is defined in the program as 0.5. Similarity matrix is then generated by following the rules.

Then the matrix is then feeded to the Mahalanobis function using the Singular value decomposition and then finding its distance using x and y where x represents the count of a particular color in image one and y represents the count of a particular color in image two. It will then be returned to be displayed in the program.

## Histogram Refinement with Color Coherence

Color coherence is done by first applying a box blur in the image which is done using the following formula as seen in the implementation in ColorAnalyzer.Blur

Then the coherent pixels are obtained by creating a bucket of unique colors present in the image and then creating a dictionary of neighboring pixels known as neighborhood and then per unvisited pixel, check if in the neighboring pixels exists a coordinate that is close to the unvisited pixel. If the pixel found is close to a neighboring pixel, it will then be added to the neighborhood, otherwise it will create a new coordinate for that particular index and put it as a new neighborhood. New neighborhoods have an extra space defined which if an unvisited pixel is contained on these extra space, a search function for checking neighboring pixels is used otherwise, it will stop searching that particular neighborhood and move to the next neighborhood.

Each neighborhood that contains more 3 members are considered as coherent pixels (based on the 4 connected-ness) otherwise it is considered as incoherent pixels. Then histograms of the query image and the indexed images are compared using the L1 distance as seen below.



## Color Histogram with Centering Refinement

In the Center Refinement, it uses a different approach in counting colors. Instead of applying a max, since the program indexes and resizes images by 100x100 pixels, it checks during scan if whether or not the particular pixel is between the range of x {25, 75} and y {25, 75} and if this is the case, it is considered inside the center and otherwise it is considered outside of the center.

During the comparison, it is compared from center of the two histograms and outside of the two histograms using the histogram intersection method and then add the two values and obtain the average of the two.

# Conclusion

During implementation it is discovered that the histograms that there are certain histograms that require extensive preprocessing and naïve implementations will take time. For example, in Perceptual Similarity, the Mahalanobis function takes time before it can produce results specially if the matrix is really big matrixes. It is considered during the next implementation to either reduce the image for the perceptual similarity in order to reduce the matrix or feed the function to GPU as CPU is too slow for the task.

The comparison for each histogram given that there is a histogram provided shows that there is no problem with the comparison as long as preprocessing is introduced.

Evaluation Results.

Upon checking the results, it has been observed that the serial implementation takes up less time compared to the parallel implementation. There were instances, especially in sorting 50000 items and above, where the parallel implementation had the advantage. Based on our observation, the larger the number of items that need to be sorted, the better the performance of the parallel implementation over the serial one. However, when sorting just a few items, the serial implementation had the speed advantage. The graphs are shown in Figures 7 and 8.

Other factors that may have contributed to the results could be the number of processes that are currently running in the machine while the run times were being recorded. The machine was already running a good number of processes and was able to output the results in a timely manner, except when sorting more than 50000 items wherein it took some time before displaying the result.

No significant changes were observed regarding memory usage for both the serial and parallel implementations.