William Beninger 001159516
Krysti Bouttell-Bonnar 001148083
November 25, 2013

FINAL REPORT
CPSC 4660
Image Comparison Algorithms Implemented

Summary of Components Being Implemented:

How can we teach a computer if two images are similar? In the code provided, we have implemented three different algorithms that are commonly used to determine exactly this. Typically the most common methods involve analyzing the colors that are present in the picture. These are the least computationally expensive but can be swayed by changes in lighting. Other methods do exist that attempt to pick out specific shapes in an image but are more computationally expensive. For our tests we elected to only use data relating to the specific colors.

The first algorithm is the simplest and the least computationally expensive, the Histogram.[4] It functions as a base line for our other two algorithms and produces results that are very hit or miss. It will only look at the colors that are present in a picture but not factor in any positioning or grouping of these colors. As such, it can frequently suggest best match images that have no similarities beyond the colors that are present.

The second algorithm is from our first research paper and involves the use of Color Coherence Vectors (CCV).[1] CCV's analyze if groupings of colors are considered "coherent" or, so to say, sufficiently large. It will count the number of coherent groupings and the number of incoherent groupings. The algorithm uses these as a metric to determine the difference between images. CCV's pitfall seems to be that although it will look for groupings of colors, it cannot discern between one large pixel grouping and a few smaller groupings that meet minimum coherence and sum up to the size of the larger grouping.

The final algorithm that was implemented involves the use of Color Correlograms.[2] It functions by creating a probability of pixels that are the same color within a certain distance of one another. In this way, it creates a method of spatially reasoning what colors are present in an image. The results it creates are typically more accurate than the two earlier algorithms and relatively insensitive to shifts in perspective of an image as well as some light distortion.

Implementation Details:

The entire program is written in Java since it was friendlier to working on images than C++. It is comprised of a main driving class titled "FrontEnd.java" and an assortment of classes used to create objects necessary for smoothly using the three algorithms.

Images were taken from our personal collection and consist mainly of environment scenes including geography, buildings, landmarks, animals, and plants. Images were

specifically selected that had a wide range of colors while others were selected for being very bland. This was done so that we could introduce quite a bit of noise into the algorithm and make it harder to locate the matching pictures. In total, there are 697 images that are included alongside the code. Due to the size of the complete project, no images will be submitted along with the code. All images have been scaled down to 220 x 165 pixels (36,300 total pixels). This scaling is necessary for a reasonable running time of the database and it is very important that all images are the same size so as not to throw off their proper matches. For all algorithms, the image is eventually discretized into specific color bins. So to say, colors of each pixel in the image are converted to a closest matching color. Researchers in both papers found that 64 color bins was an appropriate trade-off between algorithm speed and accuracy so it has been implemented across the board in our implementation. The color bins are placed so as to have an even distance between each bin. During discretization, all RGB colors are gamma corrected so as to provide a color to the computer that is closer to what the human eye might perceive.

The histogram is implemented through an array of integers that represent the number of pixels of a certain color. The HistogramVector.java class takes in a picture in its constructor, discretizes the image into the specified colors (in this case 64 color bins), and counts the number of pixels that belong in each bin. All of this is stored in a database class that has a few more functions to make the class more transparent. It's running time is O(n*m) where n is the height of the image and m is the width of the image. As an improvement, we ran the blurring and the counting concurrently allowing for it to run in O(n*m) instead of O(2(n*m)). For comparisons, it attempts to find the least number of differences in a matching picture. Comparing two pictures runs in O(C) time where C is the number of color bins.

The CCV is implemented with an array of points each storing two integer values; the first value is the number of Coherence pixels and the second value is the number of non-Coherence pixels. Coherence is defined as the number of pixels in a single area that are all connected to another pixel of the same color in any of the 8 directions. For this algorithm to run smoothly the data must be cleaned before the algorithm can be run. It is first smoothed by blurring the picture. The picture is blurred by selecting each pixel and averaging the colors of the 3x3 area (including the pixel), this allows for more coherent regions to form. Blurring runs in the same time as the binning operation O(n*m). After blurring, the data is binned into the 64 color bins using the same strategy as above. Finally, the algorithm can be run on the data. The first stage of the algorithm requires that all pixels be labelled based on their connected components.[5] Connected-Component Labeling is a two pass process each with a complexity of O(n*m). The first pass involves iteration through the image with labeling done in said order while keeping track of labels that can be thought to be equivalent. The second pass puts equivalent labels all to the lowest equivalent label. After all pixels have been labeled, a third pass is required to associate all labels with a specific color and tally the number of pixels in a label. An iteration is done through the label data object to check whether the specific label meets the minimum threshold which in this case is set to be at least 1% of the image. So a pixel region must consist of 1% of the image to be considered Coherent. Comparison is run extremely quickly in O(C) time (C being number of colors). It involves taking the sum of the absolute value of the difference between coherent values and the absolute value of the difference between non-coherent values.

Correlogram is implemented with an array of points each storing four double values. Each value is the probability that a pixel p exists within distance d of a pixel q. The distance values implemented in the research paper were d={1, 3, 5, 7} and so were followed in our implementation. The image is discretized using the same method as above and runs in the same time. The next part of the algorithm involves iterating through all the pixels in the image and summing the probability based on each of the above distances. This runs in O(n*m*(k1+k3+k5+k7)) time where k1, k3, k5, k7 are the number of neighbors that exist for a pixel at each of the distances above. Finally the summed probability is divided by the total number of pixels of that color and each is stored in the points described above. Comparisons run in O(|d|*C) time where |d| is the number of distances implemented (in this case 4) and C is the number of colors (64).

To prevent the need to constantly recalculate the above values for each image comparison, the program will calculate them at startup and write their values to a flatfile. On future running of the program the flatfile can be loaded to quickly have all the same values and for quick lookup.

Timing metrics that were implemented are done in nanoseconds and carefully placed to only capture their intended meter. All display is output in seconds for easier readability.

We used two additional classes Picture.java and StdDraw.java. These two classes were created by Robert Sedgewick and Kevin Wayne from Princeton University.[3] They are open source and provided as free code classes for non-commercial use. The two classes allow for the reading in of images and manipulation of the pixels. It does not provide any of the implemented algorithms.

Evaluation Strategy:

Constants:

Number of Images: 697
Image Type: JPEG
Size of images: 220 x 165 Pixels
Total Image Pixels: 36,300 Pixels
Number of Colors: 64
Processor: i7 950 @ 3.07GHz
RAM: 12.0 GB
Java Version: Version 7 Update 25 (build 1.7.0_25-b17) x64 bit
Disk: 7200 RPM HDD
Integer Size: 32 bits (4 bytes)
Char Size: 16 bits (2 bytes)
Filename Length: 32 chars
Double Size: 64 bits (8 bytes)

Evaluation One:
The algorithms will be evaluated on the time necessary to process and index new images. Each algorithm will be timed as they index the entire image database collection (697 images) and they will be timed on the time necessary to index only one image to verify that the rates are linear.

Evaluation Two:
The algorithms will be evaluated on the time necessary to save their databases to flatfile and the time necessary to load from flatfile to database.

Evaluation Three:
The algorithms will be evaluated on the space necessary to store their image data on both an individual image level and for our sample database.

Evaluation Four:
The algorithms will be evaluated on the time needed to locate matching images in the database with a sample comparison image.

Evaluation Five:
The algorithms will be evaluated for their accuracy in determining matching images out of a test set of 30 images.

I will also be testing each picture against the r-measure and p1-measure that are described in the Correlogram paper. It involves recording the rank of the correct answer, for r-measure it is the sum of all ranks and for p1-measure it is 1/rank of the correct answer. Both can be averaged.

Results of Evaluation:

Evaluation One:

Time taken to process images

		Algorithm		
		Histogram	*CCV*	*Correlogram*
Number of Images	*1 Image*	3.08479 seconds	3.42455 seconds	3.5359 seconds
	697 Image	2249.64897 seconds	2390.51485 seconds	2557.26261 seconds

The time across the board is fairly consistent with a slight incline towards the more complex algorithms.

CCV takes roughly 11% longer than Histogram while Correlogram takes roughly 14.6% longer than Histogram.

Evaluation Two:

Time taken to save/load database to flatfile

		Algorithm		
		Histogram	*CCV*	*Correlogram*
Method	*Save*	0.04307 seconds	0.05408 seconds	0.21680 seconds
	Load	0.07282 seconds	0.12096 seconds	0.21323 seconds

The time across the board is fairly consistent with a slight incline towards the more complex algorithms.

CCV takes roughly 25.6% longer than Histogram while Correlogram takes roughly 503.4% longer than Histogram to save.

CCV takes roughly 166.1% longer than Histogram while Correlogram takes roughly 292.8% longer than Histogram to load.

Evaluation Three:

Space necessary to store image information

		Algorithm		
		Histogram	*CCV*	*Correlogram*
Item	*1 Image*	(char*filename) +(integer*colors) =(2*32)+(4*64) =320 bytes	(char*filename) +(2*integer*colors) =(2*32)+(2*4*64) =576 bytes	(char*filename) +(4*double*colors) =(2*32)+(4*64*64) =16,448 bytes
	DB	697*320 =223,040 bytes	697*576 =401,472 bytes	697*16,448 =11,464,256 bytes

CCV is 180% larger than Histogram while Correlogram is 5140% larger.

Evaluation Four:

Time taken to query entire database for matching pictures

		Algorithm		
		Histogram	*CCV*	*Correlogram*
Category	*Lookup*	0.00428 seconds	0.00876 seconds	0.02215 seconds

CCV takes 204.7% longer than Histogram while Correlogram takes 517.5% longer.

Evaluation Five:

r-measure & p-measure

		Algorithm					
		Histogram		*CCV*		*Correlogram*	
		r-measure	p-measure	r-measure	p-measure	r-measure	p-measure
Image #	*3*	9/696	1/9	13/696	1/13	55/696	1/55
	19	1/696	1/1	1/696	1/1	1/696	1/1
	39	93/696	1/93	83/696	1/83	11/696	1/11
	44	1/696	1/1	1/696	1/1	1/696	1/1
	50	3/696	1/3	2/696	1/2	9/696	1/9
	65	1/696	1/1	1/696	1/1	39/696	1/39
	70	1/696	1/1	2/696	1/2	2/696	1/2
	71	2/696	1/2	1/696	1/1	30/696	1/30
	93	43/696	1/43	26/696	1/26	9/696	1/9
	104	1/696	1/1	1/696	1/1	1/696	1/1
	138	1/696	1/1	2/696	1/2	6/696	1/6
	143	6/696	1/6	11/696	1/11	1/696	1/1
	175	1/696	1/1	1/696	1/1	1/696	1/1
	202	1/696	1/1	1/696	1/1	1/696	1/1
	237	49/696	1/49	19/696	1/19	2/696	1/2
	250	1/696	1/1	1/696	1/1	1/696	1/1
	252	2/696	1/2	1/696	1/1	1/696	1/1
	322	1/696	1/1	1/696	1/1	1/696	1/1
	333	1/696	1/1	1/696	1/1	1/696	1/1
	351	2/696	1/2	3/696	1/3	2/696	1/2
	407	6/696	1/6	8/696	1/8	2/696	1/2
	415	1/696	1/1	1/696	1/1	1/696	1/1
	426	2/696	1/2	4/696	1/4	5/696	1/5
	467	1/696	1/1	1/696	1/1	5/696	1/5
	558	5/696	1/5	9/696	1/9	1/696	1/1
	581	5/696	1/5	5/696	1/5	4/696	1/4
	599	1/696	1/1	1/696	1/1	1/696	1/1
	606	1/696	1/1	1/696	1/1	7/696	1/7
	608	2/696	1/2	1/696	1/1	5/696	1/5
	630	4/696	1/4	1/696	1/1	1/696	1/1
	Total	0.36	18.98	0.29	19.79	0.30	9.99
	Average	0.011877	0.63274	0.00977	0.659681	0.009914	0.333095

As defined in the Correlogram paper, a method is good if it has a low r-measure and a high p-measure. In the case of the pictures that we tested with, we found that the CCV outperformed both the Correlogram and the Histogram in the p-measure and just barely outperformed Correlogram in the r-measure.

Conclusions:

From the testing we have conducted in this project, we have found that on average, the CCV algorithm tends to predict the image more effectively than either the Histogram or the Correlogram measure. Although the Correlogram contains more measurements we found it to be less reliable. Where the Correlogram algorithm excelled was in finding similar pictures when the angle had changed dramatically or a section of the photo was zoomed in upon. In these scenarios, Histogram and CCV did not perform very well. Histogram out-performed CCV and Correlogram typically when the photo contained a large amount of an exotic color or the image lighting had changed. With space a factor, a database that needed to have good results and minimize the space required would do well to choose Histogram. In terms of efficiency and minimal space required, it out performs the other two more complicated algorithms.

In doing this project, we learned a significant amount about how computers perceive colors and how it differs from our own perception. Until we did this research and modified the algorithm, we found that there was a significant gap in the suggested results and what we were expecting. The run time of this program was a significant factor in forcing our hand at data storage. When the full run time of a program exceeds 2 hours, it becomes very important to find an alternate method of storing your data for quick retrieval. Flatfiles were an obvious and quick solution for quick storage and retrieval. They also allow the program to have limited functionality when the images are not present in the database.

References & Sources:

[1] **Comparing Images Using Color Coherence Vectors**
Greg Pass; Ramin Zabih; Justin Miller
ACM
MUTIMEDIA '96 Proceedings of the fourth ACM international, pp. 65-73 (1997)
Stable URL: http://doi.acm.org/10.1145/244130.244148

[2] **Image Indexing Using Color Correlograms**
Jing Huang; Kumar, S.R.; Mitra, M.; Wei-Jing Zhu; Zabih, R.,
IEEE
Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on , vol., no., pp.762-768, (17-19 Jun 1997)
Stable URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=609412&isnumber=13322

[3] **An Introduction to Programming in Java: An Interdisciplinary Approach**
Robert Sedgewick; Kevin Wayne
Addison-Wesley
(July 17, 2007)
ISBN-13: 978-0321498052

[4] **Color Histogram**
Wikipedia
Stable URL: http://en.wikipedia.org/wiki/Color_histogram
Accessed: Nov 25, 2013

[5] **Connected-Component Labeling**
Wikipedia
Stable URL: https://en.wikipedia.org/wiki/Connected_component_labeling
Accessed: Nov 25, 2013