Binary Image Retrieval Using Local Binary Patterns

Will Bowditch,Ryan Chan, David Schmetterling

Department of Computer Science, Boston College

Abstract

The purpose of this study was to develop an algorithm that given a query image finds the closest entries to it on a database of images. All images files are of the same format, size and black and white, representing meaningful shapes. Throughout the development process, two versions were implemented and will be referred to as Version 1 and Version 2 throughout this paper. Version 1 processed each image by detecting distinct object within the matrix and caching their relevant attributes. The attributes of each database image were then compared to the attributes of each query image, and the k most similar database images were returned for each respective query. While Version 1 had an accuracy of 70% on small datasets, it failed to scale in runtime on datasets of size 1000 query images and 1000 database images. A new approach was implemented, Version 2, which compared the difference between local binary patterns, and up to 84% accuracy in 30 seconds on the larger provided dataset.

Version 1

For Version 1, each image was processed in an Image class that extracted specific features in the hopes of identifying patterns between similar binary images. The first step of extraction was to identify multiple objects within a matrix by using a depth first search to document adjacent object cells. This step also eliminated noise, such as stray points throughout the image, and was computed in linear time. After gathering a list of tuples representing the coordinates of an object, each object was placed in an Shape class for further feature extraction.

The center coordinate of a shape was calculated by finding the average coordinate point. The height of a shape was calculated by finding the difference between maximum and minimum row of the shape, with the same approach for the width. These three parameters were then used to center and scale the shape to appropriate size of the original matrix, along with caching these feature for later comparison among images.

The rotation of shapes were normalized by rotating shapes by their second moment of area. The second moment of area is a geometrical property of a shape which reflect how its points are distributed with regard to its axis. The second moment of area is a radian value which is cached as a feature and utilized as a parameter in a numpy.rotate method, which returns the matrix with the object rotated along its longest axis.

The corners of a shape were extracted by finding a shapes points that had more than five 0s as neighbors, and then reducing this set to the four closest points to the four matrix corners through minimum euclidean distances for the respective corners. The distances between corners were cached, along with the 5x5 neighborhood surrounding each corner.

After processing each database and query image through the Image class, a scoring function was called by each respective query. The function would construct a hash map for the query image, where each value was a database image with an initial key score of 0. The function would iteratively compare each database image with the query image, incrementing a database image's score corresponding to the difference between features collected. Since each matrix was normalized in terms of scale, rotation, and centering, calculating the hamming distance between pixels was found to be an extremely important feature for image comparisons. Furthermore, the four corner neighborhoods of each image were compared respectively by computing the difference of the singular variance decomposition, which is the generalization of the eigendecomposition for a normal matrix. The k highest scored database images were then return by the decision function for each query image.

Numbers										
	0	1	2	3	4	5	6	7	8	9
Θ	-.150	-.188	-.130	-.103	-.100	-.177	-.054	-.054	-.168	-.168
Surface Area	131.1	63.0	118.8	114.5	91.3	95.4	108.3	87.2	110.2	91.2
$\frac{\text{Height}}{\text{Width}}$	1.41	4.78	1.29	1.53	1.54	1.46	1.68	1.52	1.78	1.83
$\frac{\text{Area}}{\text{Size}}$	.474	.696	.401	.422	.347	.391	.454	.350	.469	.425
Corner Count	27.6	14.6	27.5	28.9	26.6	27.2	25.8	22.7	30.7	26.3
$\frac{\text{Area}}{\text{Matrix}}$	.174	.080	.152	.146	.116	.122	.138	.111	.141	.116

Figure 1: Data table demonstrating the differences in features extracted between categories.

Version 2

The shift in approach for this study occurred after recontextualizing the objective of the algorithm. Rather than trying to identify the images most similar to query, we sought to distinguish the images that were the least different. While these statements seem equivalent, the latter implies a minimization procedure essential to machine learning problems. Building off of this minimization approach, we needed an algorithm that could calculate the distance between two images while still generalizing for rotated shapes, and without sacrificing memory scalability.

Various approaches for image retrieval through distance minimization have been taken previously. A blog by Silviu Tanto [1] discussing the detection of duplicate images introduced the idea of building an accurate perceptual hash function by comparing adjacent pixels. This approach could be extended for our study by finding the minimum hamming distance on these hash values between the query image and database images. A paper by Yoosi Rubner, Carlo Tomasi, and Leonidas J. Uibas [2] introduced the idea of probability distribution comparisons by generating the earth movers distance between a query image and a database image. The earths mover distance is a measure of distance between two probability distributions, represented in computer science as normalized histograms. While we did not pursue an earth mover distance approach, analyzing the probability distributions of images was of great interest to this study, and it was found that a chi squared test of $\chi^2 = \frac{1}{2}\frac{(histA-histB)^2}{(histA+histB)}$ could be used to determine whether there is a significant difference between two probability distributions.

Local binary patterns is a type of visual descriptor that was proposed in 1990 by Dong-Chen He and Li Wang [3]. When this descriptor replaced our original algorithm, it reduced run time by up to 90% on provided data sets. The LBP feature computes the local representation of a texture by comparing each pixel with its surrounding neighborhood of pixels. The LBP feature vector for each image is created through the following steps:

1. Divide the matrix into cells
2. For each pixel in a cell, compare the pixels value with its eight neighbors.
3. Create an 8 bit binary number, representing each neighbor, where the digit at index i is equal to 1 if neighbor_i equals the center, and 0 otherwise.
4. Compute the histogram, over the cell, of the frequency of the 8 bit binary number occurring. Therefore, this histogram is a 256 dimensional vector.
5. Concatenate the normalized histograms of all cells, and return the concatenated histogram for the entire matrix.

This final histogram of a matrix is then compared to the final histogram of another matrix using the χ^2 test. The k database images with the lowest χ^2 test against a query image were then returned for each query image. The significance of the local binary pattern descriptor was that we were able to remove all other features that we had calculated and stored in memory, reducing the number of lines of code from 1400 to less than 200. However, a flaw of the LBP implementation is the inability to account for similar but scaled images. While solution to this problem was to re-implement our scaling and centering computations from Version 1, it was unwise to sacrifice the computational success that Version 2 had on the provided datasets when minimal accuracy was improved. However, a rotation invariant implementation of the LBP descriptor that also reduced memory scaling was built by considering the concept of local binary pattern uniformity.

A local binary pattern is considered uniform if it has most two transitions between 1 and 0. For example 11110000 and 00010000 have two transitions, while 10010001 and 00110010 both have 3 transitions. Uniform patterns are

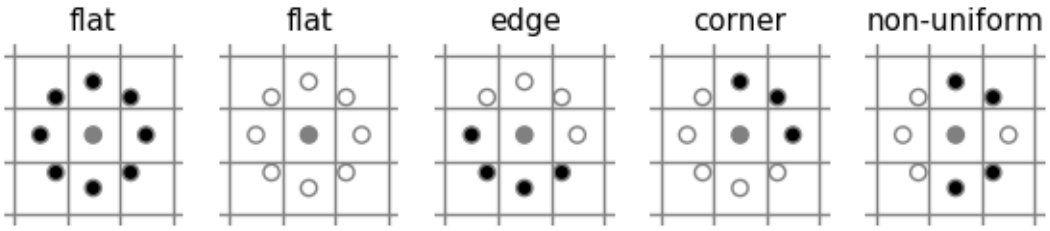

Figure 2: The figure above illustrates example patterns extracted from the image.

significant as they indicate distinct contrast around a pixel, similar to edges and corners. Uniform patterns are used by creating a separate bin in the histogram for each of the 58 uniform patterns, with the 59th bin accounting for all non-uniform patterns. This reduces the size of a histogram for each cell from 256 to 59, significantly reducing the memory required to cache each images total histogram. Ultimately, Version 2 was able to achieve 84% accuracy on Test 2 in 30 seconds by using 80% random sampling of the database images.

1. Conclusions

Version 2 of our algorithm succeeded by reducing the scale of memory and computation of Version 1 through local feature recognition. While it was far less code, the local binary patterns it extracted were much more relevant to image retrieval than the features of Version 1.

Some interesting features to our algorithm were developed but not implemented into the final product due to time constraints or from elimination in simplifying the algorithm This included edge detection and parallelization. The edge detection DFS algorithm was our first attempt at tracing the border of shapes to find significant edges that define their construction. This was later removed and replaced by our corner detection algorithm that grouped the furthest corners, which were used in other feature detection such as hamming distance. Parallelization would have helped our first algorithms fatal runtime flaw by taking advantage of the servers multiple processors. However due to limitations from python, this would have made more of a difference if it were coded in a language that is more geared towards parallelization, such as C.

A drawback of the histogram feature was its high dimensionality, which caused problems when attempting to implement nearest neighbor clustering rather than the $\mathrm{O}\left(n^2\right)$ histogram comparison. Further optimization of Version 2 would implement approximate nearest neighbor clustering through locality-sensitive hashing. Locality-sensitive hashing would reduce dimensionality for the histogram vector by placing similar nearby points into the same bins.

2. References

[1] Tanto, Silviu *Detecting duplicate images using Python.* IconFinder Blog http://blog.iconfinder.com/detecting-duplicate-images-using-python/

[2] Rubner, Yoshi *The Earth Mover's Distance as a Metric for Image Retrieval.* Science Department, Stanford University. https://users.cs.duke.edu/ tomasi/papers/rubner/rubnerTr98.pdf

[3] He, Dong-Chen *Texture Unit, Texture Spectrum, and Texture Analysis.* Transactions On Geoscience and Remote Sensing. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=572934