Project 1 - Color, Texture, and Shape

Max Ehrlich, Paul Ton, Xiaoran Liu, Cheng Liu, Miao Li

Introduction

The goal of this project was to extract low-level features for Color, Texture, and Shape from a set of test images. These features provide a uniform mathematical description of the properties of an image. We worked with the Color Histogram and Color Correlogram features for quantifying color, the Local Binary Pattern Histogram for texture (patterns of intensity), and the Histogram of Oriented Gradients for shape information. We also used the L1, L2, KL and Chi-Square methods for measuring distance, and the Cosine Similarity and Histogram Intersection methods for comparing histograms.

Design

Each of the five features used in this project depend on a histogram. This is reflected in the library's class design. The library is written in C++, adhering to the C++11 standards for best practices as much as possible and can be compiled for both dynamic and static linking. It is able to build cross platform. Figure 1 below shows a UML class diagram of the libraries components. Histograms are represented as simple double arrays (using std::vector).

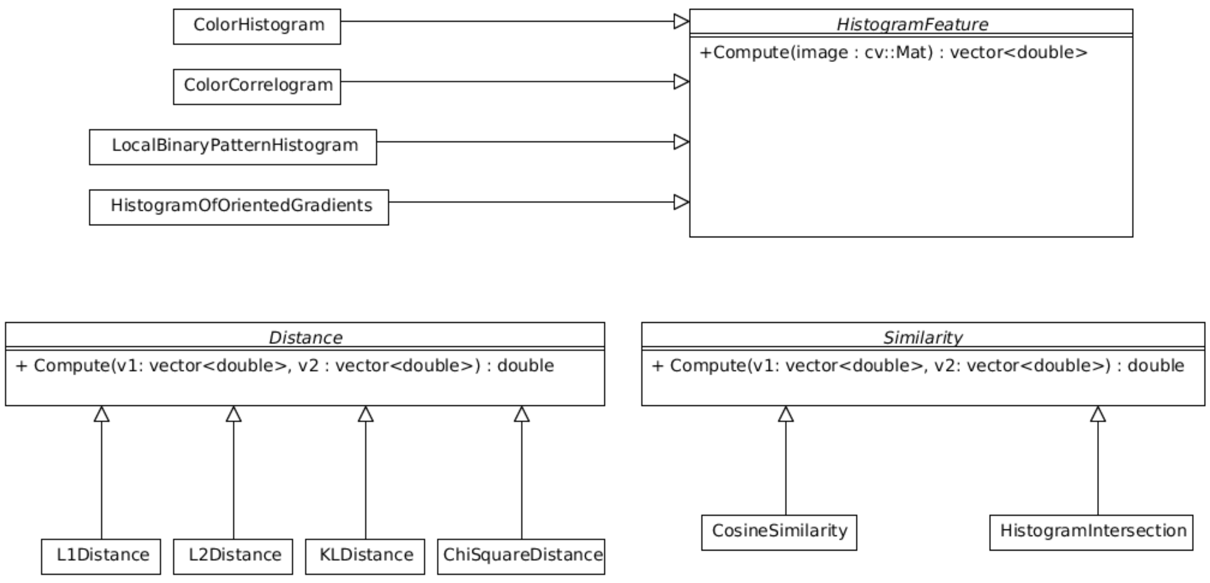

ColorHistogram
ColorCorrelogram
LocalBinaryPatternHistogram
HistogramOfOrientedGradients
HistogramFeature
+Compute(image : cv::Mat) : vector<double>
Distance
+ Compute(v1: vector<double>, v2 : vector<double>) : double
Similarity
+ Compute(v1: vector<double>, v2: vector<double>) : double
L1Distance
L2Distance
KLDistance
ChiSquareDistance
CosineSimilarity
HistogramIntersection

Figure 1
UML Class Diagram

Features

Color Histogram

The Color Histogram feature quantizes the distribution of color information in an image. The image is broken up into bands of unique colors, for our implementation, this quantization is done in a uniform manner, but more complex schemes based on K-Means, for example, are possible. Pixels in the image are then binned based on which color band they fall into, with no special weighting (e.g. each pixel contributes one unit to the histogram bin). The process is described in detail below. One consequence of storing the color information like this as a simple count is that spatial distribution of colors is lost. This can be rectified somewhat by switching to a more complex representation, such as the Color Correlogram described in the next section.

Input: one image file with RGB colors

Process:

1. Take an image file and the quantization of Blue, Green and Red as four input parameters.
2. Store the image into a matrix.
3. Decide the number of bins based on the quantization of three colors (bin = rq*gq*bq).
4. Initialize the color histogram vector to zeros.
5. Based on the quantization of colors, decide the boundary of each bin.
6. Based on the RGB value of one pixel, plus 1 to the correct bin.
7. Return color histogram vector.

Output: a color histogram vector

Color Correlogram

The color correlogram is meant to address one of the main deficiencies of color histograms - the lack of any spatial information. The color correlogram addresses this issue by calculating the correlation between color pairs at a series of fixed distances. The core concept is that for every color pair (c1,c2) and distance (k), we find, given a pixel of color c1, the probability that a pixel of distance k away (L-infinity distance) is of color c2. In practice, this is done by solving a simple counting problem and then dividing by the histogram for the color to get the probability.

We use for our implementation, one of the dynamic programming algorithms described in footnote [1]. Our steps are as follows:

1. Reduce the image to the quantized color space (default 64 bins)
2. For each color, build a horizontal and vertical lambda table. *From [1], each lambda is*

counting, given color c, distance k, and iterating over all the pixels in the image, the number of pixels of color c that are within distance k. (in the horizontal or vertical direction).

3. Using the lambda tables for color c2, count all the pixels of color c2 a fixed distance k away from pixels of color c1
4. Divide by the histogram count for color c1 as well as a factor of 8k (an artifact of the counting strategy used), to produce the correlogram value for (c1, c2, k).

Assuming a square image of side length n and given distance cap d, the algorithm is O(n^2*d) for small d, and we have to run it for each color pair (or just for each single color if we are calculating the autocorrelogram). Although as d approaches n, this value becomes O(n^3). The naive counting method is O(n^2*d^2) [1]. In practice, we still found the correlogram calculation to be noticeably slower than our other features. Additionally, the space usage for the lambda tables is also O(n^2*d). Originally, we were building all the lambda tables first before using them, which we found to be a problem for very large images.

We ignore boundaries, meaning that we don't go outside the borders of the image to count any pixels, but we still use the imaginary pixels in the probability calculation. Effectively, we are doing the calculation on the image that was passed in but with an imaginary border of dead pixels of length k in each direction.

The usage for our ColorCorrelogram class follows the same pattern as for our other feature classes. First, the class is instantiated with the desired settings: the distance, a flag to denote whether to calculate autocorrelogram or full correlogram, and the quantized color spaced as represented by our ColorQuantization class. After the class is instantiated, the color correlogram for an image can be built by passing the image into the Compute function for the class. The result is a flattened 1D vector of the correlogram represented as a vector<double>.

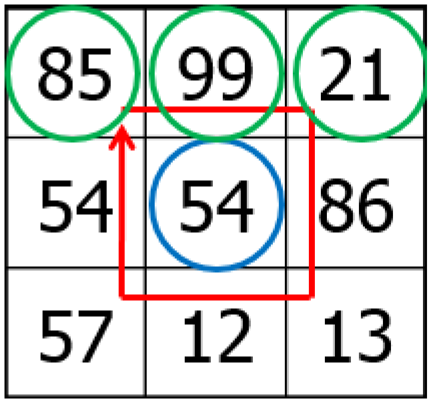
Local Binary Pattern Histogram

We used the very basic pattern of LBP algorithm that only calculate the center pixel with 8 pixels that around it. And we also create a structure named LBP including the bin id and the value.

The calculation is based on grayscale images, so that we don't need to do the calculation for rgb channels and this will increase the speed.

Process:

1. Load the image, convert the pixel's RGB to grayscale:
   <u>Grayscale = (R*299 + G*587 + B*114 + 500)/1000;</u>
   The reason why I didn't use float value is that float calculation may decrease the speed of the program.
2. Compare with pixels around it, generate the binary pattern code. If the grayscale value of adjacent pixel is no smaller than that in the middle, we set as '1', otherwise we set as '0'. The algorithm followed a sequence started from top left of the matrix. As image below:



3. Exclude the same LBP and convert binary to decimal.
   I do this by transfer every digit of the binary pattern until the last element is '1';
4. Calculate each LBP's number count.
   Scan the image for every pixel, if the outcome exist in our vector, increase the count value of that bin by 1, otherwise insert a new LBP value in the vector.
   As the max value of an eight-digit binary is 255, we set the vector size as 256 at the very beginning, if we get a new pattern, we insert it into the vector at the position that equals the bin id, so that we don't need to do the sorting after calculation.
5. Return a vector output.

Output: LBP histogram vector.

Histogram of Oriented Gradients

The Histogram of Oriented Gradients (HoG) feature quantizes edge orientations of an image in blocks using a sliding window. First the image is converted from color to grayscale (as color information is not important for this computation), and then the intensity values are mapped to double precision floating point values in [0, 1]. Then, the vertical and horizontal gradients are taken. The default implementation convolves the image with [-1 0 1] and [-1 0 1]' in order to compute the gradient, but any kernel can be set by a programmer using the library. From here, orientation histograms are computed for each cell. The cell size can be set (in pixels) by a user, but the default is 6x6 pixels. In computing the cell histograms, unsigned angles are used (e.g. angles in the range [0, 180]). This is computed by taking the absolute value of the arctangent, using the previously computed horizontal and vertical gradients as inputs. Angle values are then binned into histograms proportional to the angular distance from the bin centers (split between the upper and lower angle bins) and weighted by the gradient magnitude. The number of bins can be adjusted by the programmer and defaults to 9. Then the cells are divided into overlapping, spatially connected blocks. Again, the programmer can choose the size of the blocks, which default to 3x3 cells. Circular blocks (CHoG) are not supported by our library, and the rectangular blocks have at most 50% overlap in each dimension depending on the parity (even or odd) of the dimension. If 50% overlap cannot be achieved then the computation takes the next lowest integer. The blocks are then normalized and concatenated into the final histogram. Any normalization scheme can be used, and we provide the L1, L1Sqrt, L2, and L2Hys methods.

Because the blocks overlap, it is important to note that all but the corner cells will contribute to the final descriptor more than once, although the repeated cells will have been normalized with respect to different spatial regions. All default values were taken from [2] which introduced the idea of the HoG feature for pedestrian tracking.

Distance Measures

L1

The L1 distance, we also call it absolute distance, city-block distance or Manhattan distance.
Input: two histograms of the same type: HI and HJ
Output:

$$D(I,J) = \sum_i |H_I(i) - H_J|$$

L2

The L2 distance, we also call it Euclidean distance.

Input: two histograms of the same type: HI and HJ

$$D(I,J) = \sqrt{\sum_i |H_I(i) - H_J(i)|^2}$$

Output:

KL

The KL distance, Motivated from Information Theory, KL is the cost of encoding one distribution as another, KL distances is the average of KL(I, J) and KL(J, I). We need to make the denominator is not 0 and check the logarithm of the number to the base e is not 0.
.
Input: two histograms of the same type: HI and HJ
Output:

$$KL(I,J) = \sum_i H_I(i) \log \frac{H_I(i)}{H_J(i)}$$

$$D(I,J) = \frac{KL(I,J) + KL(J,I)}{2}$$

Chi-Square

Motivated from nonparametric Chi-Square test statistics. We need to make sure the denominator is not 0.

Input: two histograms of the same type: HI and HJ
Output:

$$D(I,J) = \sum_i \frac{2(H_I(i) - H_J(i))^2}{H_I(i) + H_J(i)}$$

Similarity Measures

Cosine Similarity

Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them.

Input: two histograms of the same type: HI and HJ
Output:

$$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

Histogram Intersection

The Histogram Intersection, Measuring how much overlap two histograms have.

Input: two histograms of the same type: HI and HJ
Output:

$$S(I,J) = \sum_i min\,(H_I(i),\, H_J(i))$$

Testing and Results

General Testing:

correctness:

This was done on the feature histograms by computing the feature vectors for a small image and checking the values against a computation by hand. We were able to verify histogram, color correlogram, and histogram of oriented gradient in this way.

performance:

We computed the feature vectors over both large images of size roughly 1000x1000 pixels and a large-ish set of images - the 1000 image Wang database from reference [3]. From this we were able to get a sense of the comparative runtimes and space requirements. We discovered that color histograms are extremely fast to calculate, histograms of oriented gradient are still fairly fast, and color correlograms are comparatively time-intensive even for medium image sizes (the Wang-1000 database is composed of, roughly, 281x384 pixel images).

<u>Time (s) for building the features for the entire Wang 1000 database</u>

Color Histogram: ~7.2 seconds

. 64 bin color quantization

Color Correlogram: ~ 875.7 seconds

. 64 bin quantization, depth = 5, and autocorrelogram = true

Histogram of Oriented Gradient: ~51.4 seconds

. cell size 6x6, block size 3x3, block step 2, 9 orientation bins

LBP: ---

We ran this for four trials for each of Color Histogram and HoG and once for Color Correlogram for the following system specs: Intel i5 processor, 2.80 Ghz, 4GB RAM

comparison:

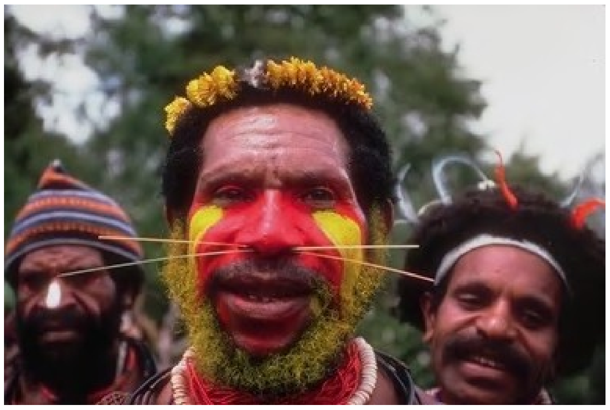
In the interest of time, memory, and processing power, we reduced the 1000-image Wang Database into 50 images, hand picking 5 images from each of the categories in the original set. For these 50 images, we performed the following procedure:

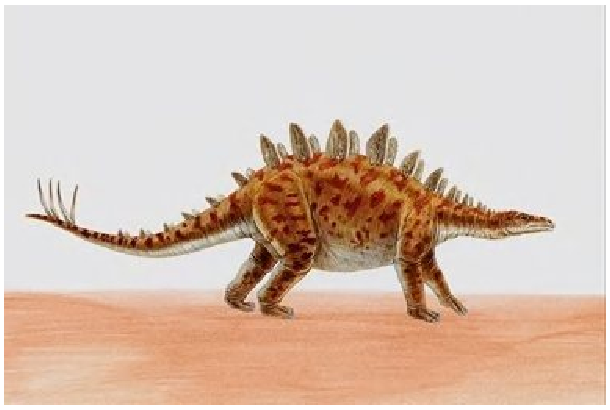
1. Computed the feature vector for each image and held it in memory
2. Calculated the pairwise L1 distance between each vector, treating one as the source and one as the destination. We use a weighted L1 mentioned in [1].
3. sorted the results for each source vector to get an ordered list.

Output files for our results are included as attachments.

interpretation:

We were able to do rudimentary recall and precision measures by hand (using our human judgment to decide correctness), although their value is dubious due to the small size of our image set. One interesting observations specific to the image-set is that HoG picked up these two images as the most similar:





potential future enhancements:

1. Trying a two pass strategy as mentioned in lecture: using the fast color histogram to pull back an initial set of results, and then running more expensive features on this initial set
2. Automated precision and recall calculation - by hand labeling or preclustering the data set before hand -- although this may be more complex than it looks.
3. Larger image database.
4. More thorough comparison/tuning of settings (quantization, depth, cell size, etc.)
5. Testing using different distance measures

Distribution of work

Max: github setup, workspace setup, cmake for cross-platform build, project design, histogram of oriented gradient, convolution, documentation, report
Paul: color correlogram, quantization, testing, report
Xiao: lbp, report
Cheng: color histogram, quantization, report
Miao: distance and similarity measures, report

References

1. Jing Huang, S. Ravi Kumar, Mandar Mitra, Wei-Jing Zhu, and Ramin Zabih. Image indexing using color correlograms. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 762--768, 1997.
2. N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: Computer Vision and Pattern Recognition (CVPR), June 2005.
3. Wang Image Database: http://wang.ist.psu.edu/docs/related/

4. Jia Li, James Z. Wang, ``Automatic linguistic indexing of pictures by a statistical modeling approach,'' IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 9, pp. 1075-1088, 2003.(download)
5. James Z. Wang, Jia Li, Gio Wiederhold, ``SIMPLIcity: Semantics-sensitive Integrated Matching for Picture LIbraries,'' IEEE Trans. on Pattern Analysis and Machine Intelligence, vol 23, no.9, pp. 947-963, 2001. (download)