IMAGE COMPARISON

Nowadays, people usually need to compare the similarity of images, to judge how close the different pictures are. For example, suppose we are running a database storing various kinds of images, and when we want to add a new one inside the database, we prefer to compare the target picture with old ones, it's no need for us to add a duplicate picture if there is already a same picture existing in the database. Another instance, police officer usually need to compare a picture of suspect with the photos in the crime database, to determine whether this suspect had committed crimes previously. Thus the application for image comparison is broad and common, and many researchers have proposed numerous mature algorithms to solve this puzzle. In this article, I will introduce three types of algorithms which is popular and efficient.

I. KEYPOINT MATCHING

Instead of comparing each pixel in every image, we could choose important points which contains more information than others (particularly at edges and corners), these special points are named as keypoints. Finding identical keypoints and tracking them across a series of images allows structure from motion software to retrieve 3D information from the images. Automated computer vision approaches thousands of keypoints even from low resolution. Each keypoint may not be tracked with enough accuracy, but be compensated by huge quantity of them.

There are lots of algorithms implemented this theory, and among them, ***SIFT(Scale-invariant feature transform) keypoints*** are definitely the most popular, since it could compare images with high accuracy regardless to the rotation, re-scale, cropping and lighting of the images. This algorithm was published by David Lowe in 1999.

SIFT mainly has four basic steps:

1. Scale-invariant feature detection: search all around the pictures, and located the points with ***difference of Gaussians*** function which are more stable for matching and recognition;
2. Feature matching and indexing: locate the candidate point according to identify its neighborhood. Lowe used a modification of ***k-d tree*** algorithm called the ***Best-bin-first search*** method that can identify the nearest neighbors with high probability using limited computation;
3. Orientation assignment: each keypoint is assigned one or more orientations based on local image gradient directions. This is the key step in achieving invariance to rotations as the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation;
4. Keypoint descriptor: compute a descriptor vector for each keypoint such that the descriptor can be used distinctively to identify this keypoint and partially invariant to the remaining variations such as illumination and 3D viewpoint, etc.

We can use SIFT features to recognize objects by the following procedures:

1. Compute the SIFT features of the input image with the above algorithm;
2. These features are matched to the SIFT feature database obtained from training images. This feature matching is done by measuring the ***Euclidean-distance*** based on nearest neighbor approach. To increase robust, some matches will be discarded if the ratio of the nearest neighbor to the second nearest neighbor is larger than a specific threshold;
3. Using ***Hough transform*** to cluster those features that belong to the same object and reject the matches that are left out in the clustering process. When clusters of features are found to vote for the same pose of an object, the probability of the interpretation being correct is much higher than for any single feature;
4. For each candidate cluster, a least-squares solution for the best estimated affine projection parameters relating the training image to the input image is obtained.

SIFT features are essentially applied to any task that requires identification between images, such as special object recognition in 2D images, 3D motion tracking and image panorama stitching and so on.

II. HISTOGRAM CALCULATION

Histogram calculation is another method of image comparison, it's less robust but much faster. Briefly, we could build feature histograms for each image, and choose the image with the histogram closest to the input image's histogram. We can generally perform this algorithm with the following procedure:

1. Construct the color histogram: it can be built from images in various color spaces, whether RGB, rg chromaticity and any other color space of any dimension. A histogram of an image is produced first by discretization of the colors in the image into a number of bins, and counting the number of image pixels in each bin. For example, a Red–Blue chromaticity histogram can be formed by first normalizing color pixel values by dividing RGB values by R+G+B, then quantizing the normalized R and B coordinates into N bins each. A two-dimensional histogram of Red-Blue chromaticity divided into four bins (N=4) might yield a histogram that looks like this table:

		Red			
		0~63	64~127	128~191	192~255
Blue	0~63	43	78	18	0
	64~127	45	67	33	2
	128~191	127	58	25	8
	192~255	140	47	47	13

2. Compare the histogram in the database with the target histogram of input image: to compare two histograms (H_1 and H_2), firstly we have to choose a metric $\left(d\left(H_1, H_2\right)\right)$ to express how well both histograms match. We could use 4 different metrics to compute the matching:

a) Correlation:

$$d(H_1,H_2)=\frac{\sum_I\left(H_1(I)-\overline{H_1}\right)\left(H_2(I)-\overline{H_2}\right)}{\sqrt{\sum_I\left(H_1(I)-\overline{H_1}\right)^2\sum_I\left(H_2(I)-\overline{H_2}\right)^2}}$$

where

$$\overline{H_k}=\frac{1}{N}\sum_J H_k(J)$$

And *N* is the total number of histogram bins.

b) Chi-Square:

$$d(H_1,H_2)=\sum_I\frac{\left(H_1(I)-H_2(I)\right)^2}{H_1(I)}$$

c) Intersection:

$$d(H_1,H_2)=\sum_I\min\left(H_1(I),H_2(I)\right)$$

d) Bhattacharyya distance:

$$d(H_1,H_2)=\sqrt{1-\frac{1}{\sqrt{\overline{H_1}\overline{H_2}N^2}}\sum_I\sqrt{H_1(I)\cdot H_2(I)}}$$

For the *Correlation* and *Intersection* methods, the higher the metric, the more accurate the match. For the other two metrics, the less the result, the better the match.

III. PERCEPTUAL HASH

Perceptual hash is using algorithm to produce a snippet or fingerprint of various types of multimedia. For the image comparison, we try to generate a hash value for each image. However, the perceptual image hash is different from the cryptographic hash. In the latter one, the algorithm only generates with same digest when the input values are exactly the same. 1 bit change of the data will cause huge difference. Thus we could use cryptographic hash to determine whether two input values are the same, whereas we use image hash to compare the similarity of two images.

A perceptual image hashing system generally consists of four pipeline stages: the ***Transformation*** stage, the ***Feature extraction*** stage, the ***Quantization*** stage and the ***Compression and Encryption*** stage. In the *Transformation* stage, the input image undergoes spacial and/or frequency transformation to make all extracted features depend the the values of image pixels or the image frequency coefficients. In the *Feature Extraction* stage, the perceptual image hashing system extracts the image features from the input image to generate the continuous hash vector. Then, the continuous perceptual hash vector is quantized into the discrete hash vector in the *Quantization* stage. The third stage converts the discrete hash vector into the binary perceptual hash string. Finally, the binary perceptual hash string is compressed and encrypted into a short

and a final perceptual hash in the *Compression and Encryption* stage (Figure 1).

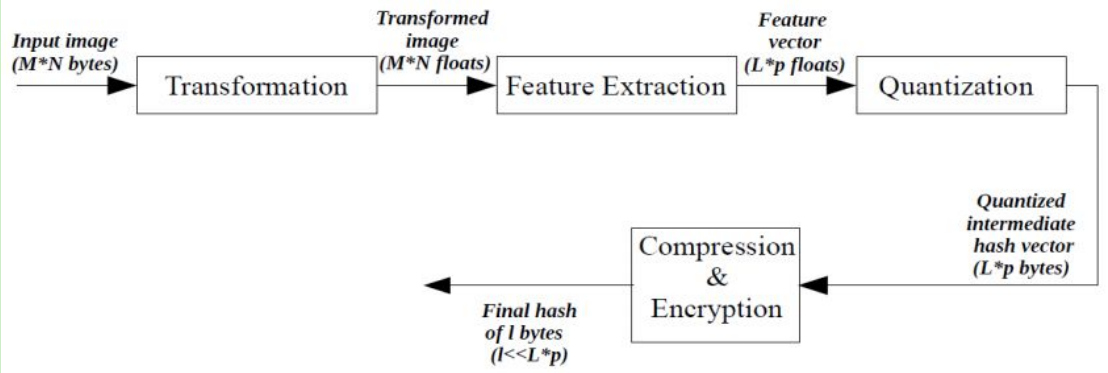

Fig. 1. Four pipeline stages of a perceptual image hashing system

There are several methods to implement the above procedure, one of them called pHash is a popular and widely used algorithm. We could generalize the algorithm to six vital steps:

1. Size reduction: diminish the size of picture to 8*8 (64 pixels), this step could remove the high frequency and keep all of different images to the same size;
2. Color simplification: transform the re-sized image to 64 gray level;
3. DCT transformation: use 32*32 DCT transformation to convert image and generate a 32 dimension matrix;
4. DCT matrix cropping: we pick up the 8*8 matrix only, which is on the top left corner of the original matrix obtained from the former step. This part represents the lowest frequency of the picture;
5. Calculate average: we calculate the mean of the 64 numbers in the matrix;
6. Hash calculation: we compare each number in the 8*8 matrix with the mean value. Set 1 when the number in the matrix is larger than average, set 0 otherwise. Then we transform this matrix into a long integer, which is 64 bit.

Through the above procedures, we finally get a 64-bit integer, which could be used as the hash value of a picture. If we want to compare different pictures, we actually compute the ***Hamming distance***, and we could regard the target image is similar with the source image if the hamming distance is larger than a specific threshold, otherwise is not.

REFERENCE:

1. Wikipedia SIFT: https://en.wikipedia.org/wiki/Scale-invariant_feature_transform
2. OpenCV Histogram Calculation: http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_calculation/histogram_calculation.html
3. Azhar Hadmi, William Puech, Brahim Ait Es Said and Abdellah Ait Ouahman, Perceptual Image Hashing, CNRS UMR 5506-LIRMM, 2012
4. pHash website: http://www.phash.org/