Image comparison via edge maps using Normalized Compression Distance

By:
David Cohen
<table>
<thead>
<tr>
<th>Ideas and goals</th>
<th>.....</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>.....</td>
<td>3</td>
</tr>
<tr>
<td>Main challenges</td>
<td>.....</td>
<td>3</td>
</tr>
<tr>
<td>Why edge maps?</td>
<td>.....</td>
<td>3</td>
</tr>
<tr>
<td>Comparing edge maps</td>
<td>.....</td>
<td>4</td>
</tr>
<tr>
<td>Kolmogorov complexity and NCD</td>
<td>.....</td>
<td>5</td>
</tr>
<tr>
<td>Goals</td>
<td>.....</td>
<td>7</td>
</tr>
<tr>
<td><strong>Course of action and Implementation</strong></td>
<td>.....</td>
<td>7</td>
</tr>
<tr>
<td>Edge detector</td>
<td>.....</td>
<td>7</td>
</tr>
<tr>
<td>Compressor</td>
<td>.....</td>
<td>7</td>
</tr>
<tr>
<td>Image Database</td>
<td>.....</td>
<td>8</td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td>.....</td>
<td>9</td>
</tr>
<tr>
<td><strong>Conclusions</strong></td>
<td>.....</td>
<td>13</td>
</tr>
<tr>
<td><strong>References</strong></td>
<td>.....</td>
<td>13</td>
</tr>
</tbody>
</table>
Ideas and goals

Background

Image comparison is a complex field of research; currently the used methods are still far from practical and satisfactory. With the growing and developing Internet and usage of Intranet in organizations nowadays, we encounter large resources of data and images and thus it is very important to find a useful and robust approach of comparing images in databases.

Image comparison methods usually consist of two main elements:

1. The image features being compared.
2. The method with which the images features are to be compared.

Main challenges

In order to compare images in a robust and useful manner, I needed to think of image features that can be defined easily and which can help compare images of different forms (e.g. compare sketches or cartoons to captured photos). The image features selected were edge maps.

Why edge maps?

Edge maps can be basically defined as the “skeleton” of the image, every image has an edge map. An edge map is actually a binary two dimensional array, and therefore comparison of two edge maps can be very effective and robust.

For example; let’s say we have two images, one of them is a photo and the other is the same photo after inverting its colors:
If we would have used a comparison method that compares the images’ colors, we would have gotten an answer that these images are as different from each other as possible. But, comparing the edge maps would have resulted in a perfect match, since the edge maps are identical:

Comparing edge maps
After choosing edge maps as the compared image feature, a comparison method has to be chosen. The most common comparison methods used to find the distance between two feature vectors (in our case the feature vectors are the edge map matrix converted to a vector) are the least square method, hamming distance, correlation etc… If any of the mentioned methods would have been used to compare the edge maps in Figure 2 then we would have received a perfect match (least square and hamming distances would be 0, and the correlation would be maximal), simply due to the fact that the edge maps are...
identical. But what would happen if we tried to compare two edge maps in which an identical object would appear in different locations (as in Figure 3)?

![Edge map of line1.jpg](Edge map of line1.jpg) ![Edge map of line2.jpg](Edge map of line2.jpg)

*Figure 3 – Edge maps of different images with the same object in different locations*

Even in the 2 simple edge maps shown in Figure 3 (original images are a line in different locations) which are identical except for the fact that the object was moved, we will receive a large distance from the 3 mentioned comparison methods. Thus, a different approach needs to be considered to properly compare the edge maps.

**Kolmogorov complexity and NCD**

The Kolmogorov complexity of x given y is the length of the shortest binary program, for the reference universal prefix Turing machine, that on input y outputs x; it is denoted as $K(x|y)$. The Kolmogorov complexity of x is the length of the shortest binary program with the empty word as its’ input that outputs x; it is described as $K(x) = K(x|y)$ where y is the empty word. Basically, the Kolmogorov complexity of an object is the length of the ultimate compressed version of that object. For example, the shortest program in the C language that will describe an NxM black grayscale image would be:

```c
for (int i=0 ;i<N;i++)
    for (int j=0;j<M;j++)
        I[i][j]=0;
```

It is easy to see that a distance between 2 objects (x and y) can be evaluated using $K(x|y)$, due to the fact that the more similar the 2 objects are, it requires less actions to be made
in order to transform x into y. For example, let’s look at the following 2 binary strings: x=10101 and y=01010, in order to transform x to y all we need to do is just shift the string to the right by one step, this results in a very short distance (note the fact that hamming an least square would result in a large distance in this example).

Unfortunately, K(x|y) is not computable, thus we need a form of evaluating it. As mentioned before, Kolmogorov complexity of an object is actually the ultimate compressed version of that object, so why not try to evaluate the Kolmogorov complexity of an object by using a compressor? All compressors use redundancy in a similar form that calculating Kolmogorov complexity uses.

If we would’ve used a compressor on the previously mentioned x and y (x=10101 and y=01010), the length of C(x) and C(y) (C(a) represents the length of the compressed version of a) would probably be very close as they both have similar redundancies (repeating patterns of 01 or 10). What would be the length of C(xy) (xy representing the binary string resulting from concating x and y – 1010101010) ? The length of C(xy) would be very much close to the length of C(x) and C(y) due to the fact that the compressor use the same redundancies and patterns in both objects, and these are very similar. Thus, it is easy to see that we can evaluate the distance of x and y by their Normalized Compression Distance (NCD) [see Clustering by Compression by Rudi Cilibrasi and Paul M.B. Vitányi for further explanation]:

\[
NCD(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}
\]

Where NCD(x,y)=0 represents identity and NCD(x,y)=1 represents the biggest difference.

Thus, I’ll use the NCD of 2 image edge maps to compute the images’ similarity.
**Goals**

My project goals are to test empirically whether the method of computing the similarity of image edge maps using NCD will result in proper matches. An additional goal is to find a good compressor for the edge maps.

**Course of action and Implementation**

The first part of the project was to decide which edge detection method and which compressor.

**Edge detector**

I’ve decided to use the Canny edge detector mainly due to the fact that the edge maps received are thin lines (1 pixel wide) and therefore all of the contours that will be compared will be of the same width. I’ve used the Canny edge detector from Matlab’s libraries.

**Compressor**

One of the most important elements of getting a good approximation of the NCD is the compressor being used. There are many commercial compressors such as LZ, PPM, BZIP2 and many more which are good for compressing the edge maps. Even tough there is a wide variety of compressors that I could choose from I’ve decided to use a compressor which uses an ROBDD (reduced ordered binary decision diagram). This compressor is designated to compress Boolean functions (basically binary strings); therefore it is excellent for my needs for compressing edge maps which are metrics of binary values.

Here’s a brief summary on the compression method; after receiving an edge map of an image, we take the edge map columns and reset them as a vector. Now, this vector is processed into a binary tree, with the 1s and 0s as leaves. The binary tree is now processed and reduced by removing redundancies and isomorphic sub trees. The now reduced tree is processed to the compressed string. [See *Universal Lossless Data Compression Via Binary Decision Diagrams* by J. Kiefer, P. Flajolet, and E-h. Yang. for further explanation on the compression method].
I’ve implemented the ROBDD compressor with Java, see the website for the source code.

**Image Database**

I’ve constructed an image database of 20 images (see Figure 4), containing a variety of images – sketches, cartoons, partial photos and actual photos. The variety allows testing whether the comparison is as robust as expected. I’ve compared each pair from the database and clustered the results.

*Figure 4 – The image database*
Results

The following are examples of the results received by the experiment:

1. Apple1 vs. Apple2

The 2 apple images are the same object but with different location in the image, the result was excellent as the images distance was minimal for both.

![apple1.jpg](apple1.jpg) ![apple2.jpg](apple2.jpg)

*Edge map of apple1.jpg*  *Edge map of apple2.jpg*

*Figure 5*
2. Penguin1 vs. Penguin2
   - The 2 penguin images are 2 photographs taken in a small time difference (the penguins moved between shots), the results here were good too as the distance was minimal for both images.

   ![penguin1.jpg](image)
   ![penguin2.jpg](image)

   ![Edge map of penguin1.jpg](image)
   ![Edge map of penguin2.jpg](image)

   Figure 6
3. Car vs. Apple2

- This is an example of 2 images with almost no edge map similarity that resulted in maximal distance.

![car.jpg](car.jpg)  ![apple2.jpg](apple2.jpg)

*Figure 7*
4. Buenos vs. Simpson1

- This is an example of a bad result, as these are 2 images with not much in common yet the distance from buenos.jpg to all of the other images was minimal when compared to simpson1.jpg
- This kind of result (false similarity) was received when comparing images with very complex edge maps such as the one buenos.jpg has.

*Figure 8*
Conclusions

Overall the results of the project were to my satisfaction and were as expected:

- Images which had similar edge maps had minimal distances between them.
- Comparison of images with complex edge maps resulted with incorrect distance evaluation.

The problem of comparison of complex edge maps might be researched further by using different edge detection method and/or different compressor.

In addition it might be worth researching a comparison method in which both the edge maps and the color histogram of the image take consideration together; this sort of method might give better results when comparing complex edge maps.

References

Thanks to Uri Shaham who showed me the magic of Kolmogorov complexity.

1. J. Kiefer, P. Flajolet, and E-h. Yang. *Universal Lossless Data Compression Via Binary Decision Diagrams*
2. Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul M. B. Vitányi. *The Similarity Metric*
3. Rudi Cilibrasi, Paul Vitányi and Ronald de Wolf. *Algorithmic Clustering of Music Based on String Compression*
4. Xin-Jing Wang, Lei Zhang, Feng Jing and Wei-Ying Ma. *Image Annotation Using Search and Mining Technologies*