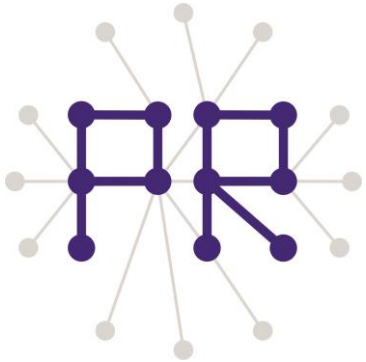
Multimedia Retrieval Exercise Course

4 Query by Example: Similarity Computation

Kimiaki Shirahama, D.E.

Research Group for Pattern Recognition
Institute for Vision and Graphics
University of Siegen, Germany



Overview of Today's Lesson

1. Loading histograms
- Vector
- Structure for storing the image information
- String
- Histogram normalization

2. Similarity computation
- Histogram intersection
- Cosine similarity

3. Sorting images based similarities

Implementing Query by Example

Pre-processing phase: Extract features from all images in the database, and store them into a file (In this course, we will use histogram-type features)

```
File f;
for each image I in the database
   histo = extractHistogram(I);
   outputHistogram(f, histo);
end
```

Last week

Retrieval phase: Compute similarities between a query image and the other images in the database, and output images with the highest similarities.

```
Query q;
q_histo = loadHistogram(q);
Result r;
For each image I in the databse
  histo = loadHistogram(I);
  sim = computeSimilarity(q_histo, histo);
  addResult(r, I, sim);
end
sortBySimilarity(r);
outputSimilarImages(r);
(Evaluation of the retrieval result)
```

This week and the next week

Taking 30 Minutes to Histogram Extraction

(My histogram file)

C:\Users\公章\Documents\MM_course\101_ObjectCategories\accordion\image_0001.jpg 21293 257 105 86 137 246 …
C:\Users\公章\Documents\MM_course\101_ObjectCategories\accordion\image_0002.jpg 13829 269 59 143 309 495 …
C:\Users\公章\Documents\MM_course\101_ObjectCategories\accordion\image_0003.jpg 14010 1 15 167 151 160 145 …
C:\Users\公章\Documents\MM_course\101_ObjectCategories\accordion\image_0004.jpg 10589 7 2 15 47 90 323 401 …
…

9,144 images (104 categories)

Bin values may be slightly different from those of your histogram file, because of the difference in OpenCV versions and versions of involved image processing libraries (libjpeg)

Loading Histograms

Two modes are required:

1. Load the histogram of the image, specified by the filename

➡ Used for a query image, which is specified at the beginning of retrieval
(Using command line variables or standard inputs is up to you)

2. Load histograms of all images

➡ Used for images for which similarities for the query image is computed

Useful Tips (1/3)

1. vector: Variable-length array (like ArrayList in java)

The array is automatically extended or shrunk if an element is added or removed

Arbitrary number of examples or arbitrary-length vectors

Rule of programming: Fewer number of constant values are desirable

- size(): Return the number of elements in a vector
- (Variable of vector)[i]: Access to the i-th element in a vector (like normal array)
- begin(): Return the *iterator* (like pointer) of the first element in a vector
  adding iterator means moving to the next element
- end(): Return the iterator representing the end of a vector

For more detail, please refer to http://www.cplusplus.com/reference/vector/vector/

2. Structure for storing the image information

```
struct ImageInfo{
    string filename; // Path to an actual image
    string label; // Category to which the image belong (e.g. accordion, airplanes, etc.)
    vector<double> hist; // Array representing the histogram
    double sim; // Variable to show the similarity to a query image
};
```

My implementation uses *a vector of ImageInfo*.

Useful Tips (2/3)

3. string: Class for storing a character sequence

Character sequences can be processed much more easily than char[].

- "+": Concatenation of character sequences
- find: Return the index of a certain character (index starts with "0")
- substr: Split a string
- c_str(): Convert a string into char*

For more detail, refer to http://www.cplusplus.com/reference/string/basic_string/

4. How to split one text line into tokens, which are suitable for initializing "ImageInfo"

If you want, you can use the following split function:

```
vector<string> split(string org, string separator){
        vector<string> res;
        string::size_type pos = 0, ppos = 0;
        while((pos = org.find(separator, pos)) != string::npos){
                if( pos - ppos > 0 )
                        res.push_back(org.substr(ppos, pos - ppos));
                ppos = pos + 1;
                pos = ppos;
        }
        if(ppos < org.length())
                res.push_back(org.substr(ppos));
        return res;
}
```

Useful Tips (3/3)

5. Each histogram should be normalized so that the sum of bin values is equal to “1” (i.e. Probabilistic representation)
   This normalization makes histograms independent of image sizes

6. File organization is up to you.

7. The most important thing is to implement a code with **NO BUG!**
   - Please check very carefully each of your implemented functions
   - It seems good to only process a subset of all images, when implementing a code. After checking the code has no bug for the subset, please go for all images.

Similarity Computation

1. Histogram Intersection: Overlapping region between two histograms

$$Sim(\mathbf{h}_1, \mathbf{h}_2) = \sum_i \min(h_{1i}, h_{2i})$$

(Some variants)

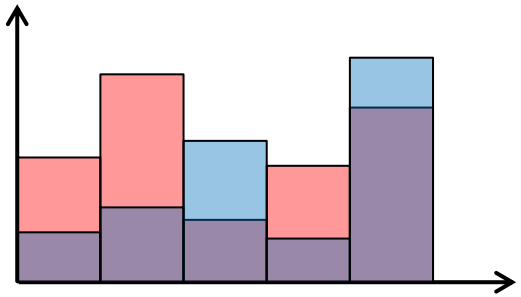
$$Sim(\mathbf{h}_1, \mathbf{h}_2) = \frac{\sum_i \min(h_{1i}, h_{2i})}{\max(\mathbf{h}_1, \mathbf{h}_2)}$$

Normalize by the larger histogram (the sum of bin values is larger)

We use already normalized histograms

$$Sim(\mathbf{h}_1, \mathbf{h}_2) = \sum_i \frac{\min(h_{1i}, h_{2i})}{\max(h_{1i}, h_{2i})}$$

Normalize values in each bin

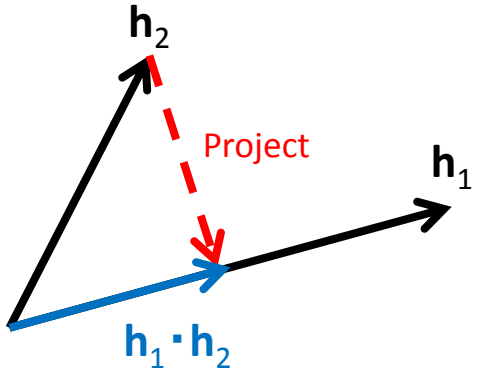


2. Cosine similarity: Difference of angles represented by histograms
(A histogram can be considered as a vector in a multi-dimensional space)

$$Sim(\mathbf{h}_1, \mathbf{h}_2) = \frac{\mathbf{h}_1 \bullet \mathbf{h}_2}{|\mathbf{h}_1| \ |\mathbf{h}_2|} = \frac{\sum h_{1i} h_{2i}}{\sqrt{\sum h_{1i}^2} \sqrt{\sum h_{2i}^2}}$$

Normalize $\mathbf{h}_1 \cdot \mathbf{h}_2$ by vector lengths

Our current normalization is not based on the vector length (L2 norm)
It is based on the sum of (absolute) values (L1 norm)



Sorting Images based on Similarities

Don't try to implement a sorting method by yourself!

```
#include <algorithm>

bool sim_descend(const ImageInfo &i1, const ImageInfo &i2){
      return i1.sim > i2.sim;
}

// Assume we have vector<ImageInfo> imageInfos;
sort( imageInfos.begin(), imageInfos.end(), sim_descend);
```

That's all ☺

Next week, we will evaluate the developed query by example system, and visualize a retrieval result in HTML format.