MyFinder: Near-Duplicate Detection for Large Image Collections

Xin Yang  Qiang Zhu  Kwang-Ting Cheng
Dept. of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA
xinyang@umail.ucsb.edu  qzhu@ece.ucsb.edu  timcheng@ece.ucsb.edu

ABSTRACT
The explosive growth of multimedia data poses serious challenges to data storage, management and search. Efficient near-duplicate detection is one of the required technologies for various applications. In this paper, we introduce MyFinder, an image near-duplicate detection system for large image collections. MyFinder consists of three major components: 1) a local-feature-based image representation utilizing the proposed LDP (Local-Difference-Pattern) feature, 2) the Locality-Sensitive-Hashing (LSH) as the core indexing structure to assure the most frequent data access occurred in the main memory, and 3) multi-step verification for queries to best exclude false positives and to increase the precision.

Categories and Subject Descriptors: I.4.9 [Image Processing and Computer Vision]: Application

General Terms: Algorithms, Experimentation

Keywords
Near-Duplicate Detection, Image Feature, Database Retrieval

1. INTRODUCTION
Several applications demand an efficient system for identifying image duplicates. These applications include:

- **Image copyright protection**: A non-trivial portion of the data on the internet infringes the copyrights and is used illegally. In fact, any image used on a website potentially infringes copyrights unless it is a personal photo, a graphic file generated by the owner of the source, or is acquired from a photo bank. Detection of duplicates in conjunction with a web crawler can be used to automate the detection of online copyright infringements.

- **Image spam detection**: A portion of the recent increase in undetected spam can be attributed to the emergence of new and more sophisticated forms of image spam. Early anti-spam technique is based on a cryptographic hash for images in spam e-mails, which can be easily cracked if spammers generate a unique variant for each email from the source spam images. Identifying duplicates can therefore address the image spam detection problem.

1.1 Related Work
Early attempts use watermarking techniques by inserting non-visual information into images to establish image ownership. However, such methods usually crash nowadays since watermark is easily changed by variant transformations of images. Some studies also employ the traditional Content Based Image Retrieval (CBIR) framework for detecting near-duplicate in a large image collection. However, CBIR-based systems are proven to be insufficient to detect near-duplicates which are produced by applying exotic digital transformations to the source image.

The framework proposed in [1] achieves a high accuracy in detecting near-duplicates. Its success lies in: 1) a local-feature-based image representation utilizing the PCA-SIFT feature, 2) the use of Locality-Sensitive Hashing (LSH) [2] for handling a large number of local image patches, 3) applying the L2 distance and RANSAC [1] as the post-verification steps to further eliminate false positives. Their study concluded that a local-feature-based image representation is much more effective than the global-feature-based methods for the target application.

A similar conclusion was drawn from a recent study [5] which evaluated methods on data representations and indexing structures:

- **Data Representation**: Local-feature-based methods consistently outperform global-feature-based methods, especially for near-duplicates resulting from complex transformations. However, the local-feature-based methods, typically using hundreds of features to describe an image, require greater storage space and longer processing time.

- **Indexing Structure**: An efficient indexing structure is one of the most important factors for addressing the scalability of a database design.

1.2 Our System
The proposed detection system for near-duplicates, MyFinder, differs from the existing solutions in two aspects:

- **Image Feature**: Our unique feature, Local-Difference-Pattern (LDP), results in significant reduction in both computation cost and storage space. The LDP feature is encoded using only 72 binary-bits, which facilitates very efficient hashing indexing. In contrast, the feature used in Ke’s system [1] requires 9180 binary-bits for hashing.

- **Indexing Structure**: Based on the LSH technique, the hierarchical indexing structure assures that the most frequent data access to the hash table occurs in the main memory. Only a small subset of “candidate” features are selected from the feature set for further L2/RANSAC verification which typically involves hard-disk access. This design greatly reduces I/O costs yet still capable of finding duplicates accurately.

Copyright is held by the author/owner(s).
MM’09, October 19–24, 2009, Beijing, China.
2. SYSTEM IMPLEMENTATION

2.1 Core Components
We illustrate our duplicate detection system in Figure 1 which consists of two key operations: database construction and query.

Figure 1: System framework for MyFinder demo

In our demo, the default database consists of 150 original source images randomly selected from CGFA [3]. We applied 40 predefined transformations [1] to each source image to produce a total of 6000 near-duplicates. These near-duplicates, along with their source images and remaining images from CGFA, were included in the gallery to create a dataset of 13115 images in total. During database construction, the following techniques are applied:

1) **LDP-Feature Extraction**: For each image we first detect a set of interest points using Difference of Gaussian (DOG) [4], and then we extract the LDP feature for each detected image patch by the following processes: 1) Divide each image patch into a $3 \times 3$ equal-sized grid and compute the sum of the differences (SD) between the corresponding pixels for each pair of grids. 2) Construct a symmetric $9 \times 9$ difference matrix, each entry of which corresponds to the average difference derived from a pair of grids and this step can be further sped up by using the Integral Image calculation. 3) Concatenate the upper part of the matrix into a 36-d feature vector as our LDP feature.

2) **Quantization and Encoding**: We quantize each of the 36 feature value into two binary-bits with respect to a threshold $T$: if the feature value is between $T$ and 255, it will be encoded as ‘11’. Similarly, it will be encoded as ‘10’, ‘01’ and ‘00’ if the feature value is in the ranges of $(0, T)$, $(0, -T)$, and $(-T, -255)$ respectively. As a result, we encode the 36-dimensional descriptor into a 72-bit binary string.

3) **Hierarchical Indexing Structure**: We organize our database in three tables: 1) The File Table stores a list of file names to locate the images in the hard disk. 2) The Feature Table stores all local LDP descriptors together with their geometry information (X and Y locations, scale, and orientation). Both File and Feature tables are stored in hard disk to assure the system scalability for larger databases in real applications. 3) The Hash Table is constructed by the Locality Sensitive Hashing (LSH) technique. We apply a family of hash functions on the encoded 72-bit binary string, and hash similar objects into the same bucket in one hash table. The concept of LSH is described in great details in [2] and our hieratical indexing structure is mostly inspired by the work published in [1].

Our query process follows the following steps: 1) For each query image, we extract a set of 72-bit LDP features. 2) We perform LSH hashing to obtain a set of candidate features and acquire them from the disk to the main memory through the feature table. 3) We apply L2-distance and RANSAC geometric verification to minimize the false matches.

2.2 User Interface
Figure 2 illustrates the main UI of our system. In the left-top area, the user can browse images stored in the database. In the right-top area, we display a selected image in its full-size, followed by its brief description. We can either search the near-duplicates of this image or delete it from the database. The lower part of the page provides the interface for the user to upload his/her own samples from the hard disk, either for adding them to the database or as a query image for search. We display the returned image set in the left-top area after the query.

Figure 2: The UI of MyFinder.

3. REFERENCES