Sub-image Searching Through Intersection of Local Descriptors

Tomas Homola
Faculty of Informatics
Masaryk University
Botanicka 68a, 602 00 Brno
Czech Republic
xhomola@fi.muni.cz

Vlastislav Dohnal
Faculty of Informatics
Masaryk University
Botanicka 68a, 602 00 Brno
Czech Republic
dohnal@fi.muni.cz

Pavel Zezula
Faculty of Informatics
Masaryk University
Botanicka 68a, 602 00 Brno
Czech Republic
zezula@fi.muni.cz

1. INTRODUCTION

The complexity of search in current business intelligence systems, academic research, or even the home audiovisual databases grows up rapidly. Users require searching by the content of their data. For example, the user sees a cathedral while watching a movie and by taking a snapshot, his or her private collection of holiday photos can be searched for images containing that cathedral, as demonstrated in Figure 1. In practice, it is not sufficient to store data and search in it by exact match but rather by means of similarity, i.e. retrieving data items similar to a query item. Similarity searching is especially requested in multimedia databases, digital right management systems, computer aided diagnosis, but also in natural sciences and psychology. In these fields, theoretical primal background for querying by similarity is already defined.

In this demonstration, we focus on similarity searching from the perspective of identifying and locating an object depicted in a query image in a collection of images. In particular, we have implemented an algorithm that is based on local affine-invariant descriptors and that is capable of retrieving database images containing the object in the query image. The algorithm's capabilities are demonstrated on a collection consisting of 15,337 company logos from which 2×10^6 SIFT local descriptors were extracted.

2. SUB-IMAGE SEARCH ALGORITHM

Local descriptors, such as SIFT [5], extracted from images usually consist of three distinguished parts, called a *locator*, *descriptor* and *scale*. To avoid misunderstandings, we denote these local descriptors (SIFTs) as *features*. The locator defines the position of the feature within the image, while the descriptor itself summarizes the position's proximity in terms of color, texture, edges, etc. The scale attribute represents the importance of the feature (e.g., the spatial size of the surroundings the descriptor characterizes). This leads to the idea that for performance reasons we do not need to check all features extracted from an image, but for many applications it is sufficient to inspect several most-important ones to obtain high recall. Spurious false positives can later be filtered out using the complete sets of features. Finally, each feature has also a *unique identification* of the corresponding image.

In the following, we introduce an algorithm for locating a query object in the database. A formal specification of this algorithm has been proposed in [3] but for the needs of this demonstration it has been modified for performance reasons. The pseudo-code in Alg. 1 specifies the procedure of locating a query Q in database images. After extracting the most-important features from Q, several range queries with the radius ε (Line 4) are executed to obtain a candidate set. Notice that throughout the algorithm we use pairs of features, because the spatial correspondence of local features in the query and a database image is important for final ranking (in our case) or for filtering out non-matching images. After removing 'duplicate' pairs (more pairs between the same images), candidates containing fewer pairs of features than $limit$ get eliminated. Each set in the result corresponds to a database image and it contains the pairs of matching features. So, by taking the image's features from the pairs, the specific location of the query in the image is formed by computing a bounding rectangle over these features' positions. The final ranking is done on Lines 17-23 by checking the order of features in both the coordinates (x, y) independently.

3. DEMONSTRATION

We demonstrate the effectiveness of Algorithm 1 on a collection of $15,337$ company logos from which on average 152 local features (SIFT) were extracted. Thus, $2,359,839$ features were obtained in total. These features along with original image IDs were organized in the M-tree [2], so the range queries (see Line 4 of Alg. 1) were evaluated efficiently. Next, a sequential file ordered by image ID was created to store other meta-data, such as filename, about all the images. The algorithm is implemented in Java 6 with the implementation of M-tree available at http://mufin.fi.muni.cz/trac/mtree/. The demo is available at **http://mufin.fi.muni.cz/subimages/** and it is run on a single server equipped with two quad-core Intel Xeon 2 GHz CPUs, 14 GiB RAM and a six-disk RAID5 array. To speed up the retrieval in the M-tree, eight queries were executed concur-

Figure 1: Example of a sub-image searching for Sacré-Cœur cathedral.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SISAP '10, September 18-19, 2010, Istanbul, Turkey.
Copyright 2010 ACM 978-1-4503-0420-7/10/09 ...$10.00.

Algorithm 1 Locating Sub-images `locateQuery`

Input: a query image Q, a number nof of important features, a threshold ε, and a limit on the number of features ($limit$).
Output: $Result$ consisting of images containing the query.

1: $Result = \emptyset$ and $Temp = \emptyset$.
2: $F_Q = \{q_1, \ldots, q_{nof}\}$. // Extract local features from query
3: **for all** $q_i \in F_Q$ **do**
4: $RQ = rangeSearch(q_i, \varepsilon)$.
5: Add the pairs (q_i, f_k) for each $f_k \in RQ$ to $Temp$.
6: **end for**
7: // Group the pairs in $Temp$ by image identification of f_ks:
8: $Result = \{G_{\mathbb{E}_1}, \ldots, G_{\mathbb{E}_n}\}$, where $G_{\mathbb{E}_i} = \{(q_v, f_w), \ldots, (q_x, f_y)\}$ and all f_s have the same image ID.
9: **for all** $G_{\mathbb{E}_j} \in Result$ **do**
10: **for all** $q_i \in F_Q$ **do**
11: $G = \{(q_i, f_x) | (q_i, f_x) \in G_{\mathbb{E}_i}\}$. // Take pairs with q_i
12: Remove G from $G_{\mathbb{E}_i}$.
13: Add (q_i, f_k) to $G_{\mathbb{E}_i}$, where $(q_i, f_k) = argmin_{(q_i, f_x) \in G}(distance(q_i|\text{Desc}, f_x|\text{Desc}))$.
14: **end for**
15: **end for**
16: Remove all $G_{\mathbb{E}_i}$ from $Result$ which contain $< limit$ pairs.
17: **for all** coordinates α of the position coordinates (x,y) **do**
18: **for all** $G_{\mathbb{E}_i} \in Result$ **do**
19: Sort the pairs $(q_v, f_w) \in G_{\mathbb{E}_i}$ by the coordinate α of f_s.
20: Compute the α-rank of $G_{\mathbb{E}_i}$ // Number of swapped features w.t.r. the query (ORD function in [3])
21: **end for**
22: **end for**
23: Sort $Result$ by $\frac{\sum_\alpha \alpha\text{-rank of } G_{\mathbb{E}_i}}{nof}$.

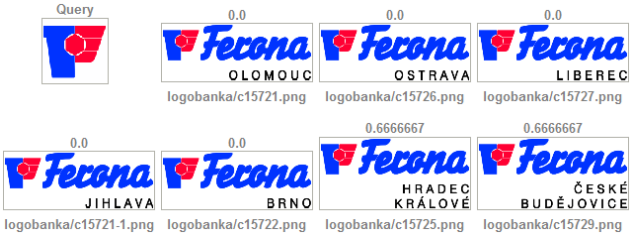

Figure 2: Result of search for the example logo (left-top image).

rently, so all cores were loaded. Figure 2 depicts a retrieval example, where the numbers above the images depict their dis-similarity to the query (as computed on Line 23 of Alg. 1). Below the images, there are the corresponding file names, which exemplifies the fact that no textual information was used during retrieval. As for the algorithm's parameters, we used $\varepsilon = 250$, $limit = 6$ and $nof = 18$, where these values were empirically obtained as good ones. In the demonstration, these parameters can be modified. The average query execution time is below 3 seconds, where nearly 2 seconds. are spent in the M-tree by evaluating all 18 queries and 1 second is needed to compare the feature pairs and ranks the images obtained.

4. RELATED WORK

There are many systems for content-base image retrieval, for example MUFIN, ALIPR, ImBrowse, idée and Gazopa. However, these systems focus on searching for images similar to a query image as whole, thus searching for sub-images is not supported.

Approaches to search for sub-images are based on locality sensitive hashing or encoding image features to visual words, which is hashing in fact too. Many approaches, e.g. [4], compare only the hash codes of features' descriptors without tackling their spatial distribution, which leads to false positives in the result set. The Geometric min-Hash (GmH) [1] converts image features to visual words too, but it extends the resulting hash code by including also image features in close neighborhood. From the searching point of view, it identifies regions in an image in which the identical or almost-identical groups of features with respect to the query image occur. It eliminates the features that are encoded to one visual word, which can cause problems when processing repeating patterns or when the queried sub-image occurs in the original image multiple times. By analogy to our approach, GmH optimizes the retrieval process by selecting some important features (anchors). A disadvantage of GmH is that only small spatial surrounding is considered instead of taking into account spatial order of features. So, it is limited to small query images only.

In contrary to the approaches mentioned above, the authors of [6] does not use local descriptors for the searching, but MPEG-7 global features are rather taken. In particular, database images as well as the query image are segmented to chunks of pre-defined size and the global features are extracted. Searching procedure then finds correspondence between the database images and the query image.

5. CONCLUSIONS

By exploiting the spatial information (locator property) of local image features, an effective algorithm for searching for objects contained in the query image has been implemented. An online application demonstrates the properties of this algorithm on a collection of 15 thousand images of company logos from which more than 2 million SIFT local features were extracted. The system exhibits query response time under three seconds while it operates on a moderate HW infrastructure.

Acknowledgements

This research was supported by Czech Science Foundation – projects GACR P103/10/0886, GACR 201/09/0683 – and a national project MUNI/A/0922/2009. The access to the METACentrum supercomputing facilities provided under the research intent MSM-6383917201 is highly appreciated too. We used this infrastructure for feature extraction. We also thank the LogoOpen company for providing the logo data set.

6. REFERENCES

[1] O. Chum, M. Perdoch, and J. Matas. Geometric min-hashing: Finding a (thick) needle in a haystack. *CVPR*, pages 17–24, 2009.

[2] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435. Morgan Kaufmann, 1997.

[3] T. Homola, V. Dohnal, and P. Zezula. Proximity-based order-respecting intersection for searching in image databases. In *Proceedings of the* 8^{th} *International Workshop on Adaptive Multimedia Retrieval (AMR 2010)*, 2010.

[4] Y. Ke, R. Sukthankar, L. Huston, Y. Ke, and R. Sukthankar. Efficient near-duplicate detection and sub-image retrieval. In *In ACM Multimedia*, pages 869–876, 2004.

[5] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[6] M.-S. Ryu, S.-J. Park, and C. S. Won. Image retrieval using sub-image matching in photos using mpeg-7 descriptors. In *AIRS*, pages 366–373, 2005.