Hash Functions for Near Duplicate Image Retrieval

Adrien Auclair
Paris Descartes University
LIPADE

Nicole Vincent
Paris Descartes University
LIPADE

Laurent D. Cohen
Paris Dauphine University
CEREMADE

Abstract

This paper proposes new hash functions for indexing local image descriptors. These functions are first applied and evaluated as a range neighbor algorithm. We show that it obtains similar results as several state of the art algorithms. In the context of near duplicate image retrieval, we integrated the proposed hash functions within a bag of words approach. Because most of the other methods use a kmeans-based vocabulary, they require an off-line learning stage and highest performance is obtained when the vocabulary is learned on the searched database. For application where images are often added or removed from the searched dataset, the learning stage must be repeated regularly in order to keep high recalls. We show that our hash functions in a bag of words approach has similar recalls as bag of words with kmeans vocabulary learned on the searched dataset, but our method does not require any learning stage. It is thus very well adapted to near duplicate image retrieval applications where the dataset evolves regularly as there is no need to update the vocabulary to guarantee the best performance.

1. Introduction

In this paper, we are concerned by the problem of near-duplicate images retrieval in large database. Near-duplicate images are images that are deformations of a single original image. Common deformations are cropping, global modification of intensity, adding noise... One major application is to detect image copyright violation. This is not the same problem as detecting images of the same 3D scene or categorizing images. Similar solutions exist in the literature for these three applications but in this article, we are only studying the near-duplicate detection.

In the past few years, several bag-of-words methods have led to good results for this task. These methods imitate the principle of text documents retrieval [13] for images [15]. A core step is to define a visual vocabulary. In [15], the vocabulary is built using a kmeans algorithm on local image descriptors (typically SIFT descriptors [10]). This is a costly step and image retrieval effectiveness depends of the learning database. High recalls are obtained if the vocabulary is learned on the searched dataset. But if vocabulary is computed on a different set of images, effectiveness is lower. This is a major problem for applications searching images in a database where images are regularly added or removed. To ensure the best retrieval results, one must regularly update the kmeans-based visual vocabulary. This step adds parameters concerning the frequency of these updates and it may be required to stop the application while computing the kmeans.

To tackle this issue, we propose a non learned algorithm for bag-of-words. Our method is based on original hash functions inspired by several nearest neighbors algorithms. There is no learning stage and thus results do not depend on any off-line process applied to a particular dataset. As a result, a near-duplicate search engine based on our algorithm is particularly adapted to applications having regular updates of the database.

2. Related work

In a first part of this paper, we review the use of a nearest neighbors algorithm for image retrieval in a bag-of-words approach, as explained in [7]. Then, we review several approximate nearest neighbors algorithms that have inspired our work.

2.1. Nearest neighbors algorithms and bag-of-words

In a bag-of-words approach, each database image is described by several local descriptors (in our work, we use the SIFT descriptors of [10]). Each descriptor is assigned to a word within a visual vocabulary. Computing similarities between images is then casted to detecting common visual words between images. In [15], the authors use a kmeans to define the visual words and similarity scores are computed with a term-frequency inverse-document-frequency (tf-idf) model, classically used in the text retrieval domain [13].

A general expression of the bag-of-words similarity

978-1-4244-5498-3/09/$25.00 ©2009 IEEE

score is given in [7] :

$$s_{q,j} = \sum_{i=1}^{h^q} \sum_{k=1}^{h^j} f(x_i^q, p_k^j) \quad (1)$$

where x_i^q is the i^{th} descriptor of image q, p_k^j is the k^{th} descriptor of image j and f is a similarity function between two descriptors. For a classical kmeans-based vocabulary, the similarity function is :

$$f(x^q, y^j) = \delta_{c(x^q),c(y^j)} \times \left(\frac{1}{h^q h^j} \times log(\frac{n}{n_{c(y^j)}})^2 \right)$$

where c is a quantizer that associates the index of the closest class center to a descriptor, δ is the kronecker function. The second part of the similarity is the tf-idf term where h^q and h^j are respectively the number of descriptors in images q and j, and $n_{c(y^j)}$ is the number of descriptors corresponding to the word $c(y^j)$. The log term is squared as it is computed from both descriptors x^q and y^j.

In [7], authors proposed that any nearest neighbors algorithm can be used as a similarity function f. For example, one can use (tf-idf weight is omitted here for simplicity) a range-neigbors algorithm :

$$f_\epsilon(x, y) = \begin{cases} 1 & \text{if } d(x, y) < \epsilon \\ 0 & \text{otherwise} \end{cases}$$

The problem of this type of functions is that exact nearest neighbors algorithms are very slow in practice for high dimensional spaces. To get fast queries, one has to replace the underlying exact nearest neighbors algorithm by an approximate one.

2.2. Approximate Nearest neighbors

There exists a large amount of algorithms for the nearest neighbors problem. As shown previously, kmeans clustering can be used as an approximate nearest algorithm. At query, only points belonging to the class of the query point will be tested as neighbors (or directly declared as neighbors). As said above, the main drawback of kmeans-based algorithm is that it is not adapted for regularly updated databases.

In [9], points are indexed by their projection on several random lines. While it obtains good performance, this approach requires much memory and for large point clouds, the indexing must be saved on disk, slowing down the querying process. An original class of solution is based on space filling curves (see [16] for example) but these approaches are rigid and offer very few tuning possibilities.

A tree-based solution is to use a non exhaustive visiting algorithm on a kd-tree (algorithm called Best-Bin-First in [10]). To be more effective, randomized kd-tree forests have been used in [12]. One problem is that there is no guarantee that parameters (number and depth of trees) will guarantee good performance if the number of images within the database evolves.

Another popular solution is Locality Sensitive Hashing [5, 3, 14]. In this approach (we note it LSH), points are projected on random lines. The obtained projections are used to compute hash keys. In order to retrieve a correct amount of a query point's neighbors, many hash tables have to be used. Thus, the major problem of this approach is the quantity of memory needed. This is a real bottleneck when required memory is larger than available main memory, leading to disk swap and large falloff in performance. Several modification of the algorithm have also been proposed in literature [6, 2], but it still requires more memory than a kmeans based approach.

In our work, we have chosen to work with hash functions for several reasons. First, it is not related to any learning dataset. It may be very fast. And retrieval performance does not evolve when modifying the database while this is not true for tree-based methods.

3. New hash functions

The main problem of current hashing (LSH or PvS framework) is that they require much memory. To reduce the memory overhead per indexed point, we want a point to be associated with a single hash key.

The underlying idea of hashing for nearest neighbors algorithm is that close points must be hashed with a similar key with high probability and far away points must be given different keys. Classically, the closeness notion is seen as the L2 distance function.

In our approach, we consider that two points are similar if they are distinctive along the same dimensions. The initial idea of our work is that if two points are far from the average value along the same dimensions, there is a good probability that they come from two similar image descriptors. Still, this requires to define what we call a distinctive dimension. We note β such a function that computes the distinctiveness of a point, for a given dimension. We tested several such functions and found that the one which obtains the best image retrieval results is :

$$\beta(x^i, j) = \left| \overline{x_j} - x_j^i \right| \sigma_j^\alpha$$

where x^i is the i^{th} point of the database, j a dimension ($j \in [1, 128]$ for the 128-D SIFT descriptor we use), x_j^i is the coordinate of x^i on dimension j, $\overline{x_j}$ is the average value of descriptors on dimension j ($\overline{x_j} = \frac{1}{n} \sum_{p=1}^{n} x_j^p$ where n is the number of points in database), σ_j is the standard deviation of points along dimension j and α is a parameter to balance the two terms. After tuning, we found that the best image retrieval results were obtained for $\alpha = 0.5$. The distinctiveness function is used to sort the dimensions for each

database point. We note $D(x^i)$ the vector of these sorted dimensions for a point x^i :

$$\beta(x^i, D(x^i)_1) > \beta(x^i, D(x^i)_2) > ... > \beta(x^i, D(x^i)_{128})$$

Then, for a point x^i, the k first values of $D(x^i)$ are used to compute a hash key :

$$key(x^i) = h(D(x^i)_1, ..., D(x^i)_k)$$

where h can be any classical hash function using a set of k integers as input. We chose to use a function similar to the one used in LSH [14] :

$$h(v_1, ..., v_k) = \left(\sum_{j=1}^{k} r_j \times v_j \right) mod\ P\ mod\ H$$

where the r_j are random integers, P is a prime number and H is the size of the hash table. In practice, there will be collisions (different integer sets may be assigned the same key). To reduce the impact of such collisions, a second hash key $g(v_1, ..., v_k)$ is computed and used as a checksum. In the hash table, for a database point x^i, an identifier (containing the id of its image and the id of the descriptor within this image) and a checksum are saved in bucket $h(D(x^i)_1, ..., D(x^i)_k)$. Thus, in the hash table, a point x^i is saved on 64 bits (32 bits for its identifier and 32 for the checksum). The checksum size can certainly be reduced but we have not investigated this option.

3.1. Querying algorithm

The initial querying algorithm using these hash functions is very simple. All database points are fist indexed in the hash table. For a query point, its hash key and checksum are computed. The points of the bucket of its hash key are visited. For the points having the same checksum as the query, the euclidean distance is computed to decide whether or not they are neighbors.

Figure 1 shows an example of computing the hash keys. On this figure, two points are highly similar but hashed with different keys. Using the given hash function is too much distinctive. Even very similar points are very rarely hashed to identical keys. As a first correction, we propose to sort the k first values of vector $D(x^i)$ before using it in hash function h. Thus the key of a point x^i is obtained as :

$$key(x^i) = h(sort(D(x^i)_1, ..., D(x^i)_k))$$

With this little modification, the two points of figure 1 will be hashed with the same key (both will get the key $h(4, 5)$). When testing this option, points are still too much spread in buckets and the algorithm fails to retrieve most of the true neighbors. The solution we propose is to hash a database point with a single key and to compute several keys for a query point. Thus, it is memory effective as a database point is indexed only once. And when querying for neighbors of a point, several buckets are browsed.

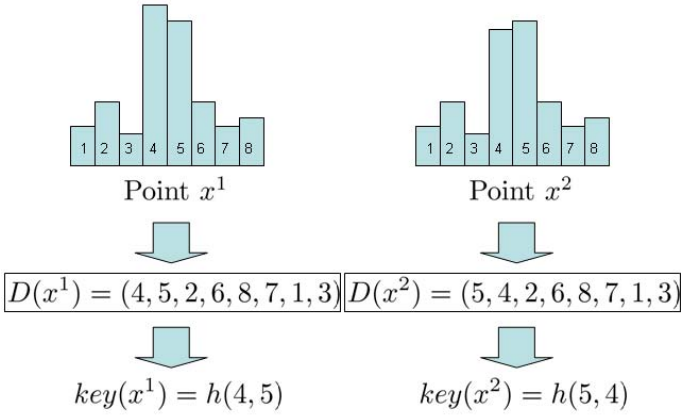

Figure 1. Example of the proposed hashing. The histograms show values of two points in a 8-dimensional space. For sake of simplicity, we consider here that $\beta(x^i_j, j) = x^i_j$. In this example, we use $k = 2$. It shows that even if x^1 and x^2 are very similar, they will be hashed with different keys.

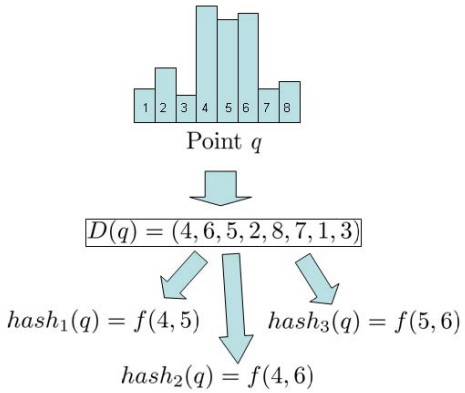

Figure 2. Hash keys computed for a query point. In this example, $k = 2$ and $n = 3$. The query point will generates three hash keys and thus three buckets of the table will be browsed to find its neighbors.

There is still to define how to generate several keys for a query point that should increase the probability to find its neighbors. For a database point x^i, only the k first dimensions of $D(x^i)$ are used to compute a key. For a query point q, we propose to use all the combinations of k dimensions among the n first values of $D(q)$. This generates $\binom{n}{k}$ hash keys for a query point. Figure 2 illustrates this principle. With this option, a query point and a database point can share a similar hash key even if their distinctive dimensions are not exactly the same. In our tests, we typically use n close to 10 and a varying k.

4. Evaluation as a nearest neighbors algorithm

Our algorithm is first tested as a range-neighbors algorithm : for a query point q, one wants to retrieve all the database points x^i such that $d(q, x^i) < R$. The points we used for our tests are 128-D SIFT (using both regions of interest detector and descriptor of [10]). Point clouds used in our experiments are publicly available on our website [1].

To evaluate approximate range-neighbors algorithms, one can consider them as filtering algorithms. For a given query point, the database point cloud is filtered in a way that only a small subset of points are tested as potential neighbors (i.e. L2 distance with the query is computed for these points). Thus, an algorithm will be evaluated with two criteria : the size of the returned subset and the number of true neighbors found in this subset. The first criterion is evaluated by the ratio of the returned subset size on the number of points in database (noted RatioFilter in figures). A RatioFilter of 0.01 means that only 1% of the full point cloud is returned, and thus the corresponding algorithm can be at best 100 times faster than a linear search. The second criterion is measured as a classical recall. These two measures are equivalent to the more traditional recall-precision evaluation (precision can be expressed as a function of recall and RatioFilter) but as speed is a requirement, we found that RatioFilter is a better indication than precision.

Some results are given on figure 3. On this test, our algorithm is only slightly inferior to the best algorithm which is the modified version of LSH of [2]. Still, our algorithm is much more memory effective and will be better if disk swap is required by LSH. The kmeans-based algorithms (either flat or hierarchical) which are memory effective are clearly outperformed by our hashing on this dataset. Our hashing is also slightly better than LSH of [5] and [3] (noted LSH2).

Another important point to keep in mind is that these approximate algorithms have to be compared to an exhaustive linear search. For the sake of comparison, we implemented an exact linear search using the computational power of GPU (we used a Nvidia 8800 GT GPU). On the same point cloud, we obtain an algorithm 20 times faster than a linear search executed on a 2.66GHz processor. In order to be 20 times faster than linear search on CPU, an approximate algorithm requires a RatioFilter below 0.05. This is only an indication as one can use multi-cores processors or also more recent GPU, one can also use several GPU... But this test shows that an approximate algorithm having a RatioFilter above 0.05 is not interesting compared to a GPU implementation.

In the previous paragraphs, our algorithm has proven to compare very well to several state of the art approximate range-neighbors algorithms. In the next section, we use it in a bag-of-words image retrieval framework.

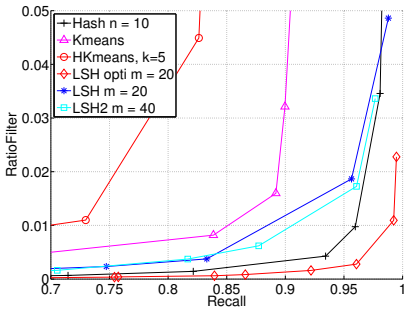

Figure 3. Comparison of several algorithms for a range-neighbors search on a 265.000 points dataset extracted as 128-D SIFT descriptors of various images. We used a range of 200 as it is the value we used to associate SIFT descriptors for similar details on our images. Hash is our algorithm, KMeans use a kmeans clustering of the point cloud, HKmeans is a hierarchical kmeans, LSH is the Locality Sentitive Hashing of [5], LSH-Opti is the slightly modified LSH of [2] and LSH2 is our implementation of [3] (we also tested results obtained using the E2LSH package implemented by authors of [3], results were very similar to those of our implementation). For our algorithm, each point is obtained by varying k. On the KMeans curve, each point corresponds to a different number of classes. For HKmeans, the branch factor is set to 5 and the number of levels varies. For LSH algorithms, the number of hash tables is fixed (noted m) and the number of hyperplans varies.

5. Image retrieval with our functions

Integrating our range-neighbors algorithm within a bag-of-words is straightforward. In the score expression given in equation 1, we use the following similarity function :

$$f(x^q, y^j) = \delta'_{keys(x^q),key(y^j)} \times \left(\frac{1}{h^q h^j} \times log(\frac{n}{n_{key(y^j)}})^2 \right)$$

where x^q is a descriptor from the query image, y^j a database descriptor, $keys(x^q)$ is the list of keys computed from x^q , $key(y^j)$ is the single key of y^j, $n_{key(y^j)}$ is the number of database points in the bucket of $key(y^j)$, and δ' is a modified kronecker operator :

$$\delta'_{keys(x),key(y)} = \begin{cases} 1 & \text{if } key(y) \in keys(x) \\ 0 & \text{otherwise} \end{cases}$$

With this formulation, it is important to notice that for a database point, the 128 dimensional descriptors are not required as no euclidean distance is computed. These descriptors are only used initially to compute hash keys and thus the approach does not require much memory.

The approach we use here to evaluate our algorithm is to retrieve images that are synthetic deformations of a set of images. This is well adapted to the application of detecting copyright violation where one wants to detect if a given image is a modification of a database image. We established a list of transformations that our image retrieval

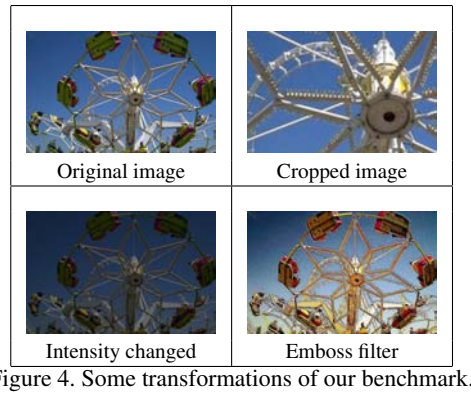

Figure 4. Some transformations of our benchmark.

algo	Recall	Time (ms)
Our hashing, n,k=10,8	0.974	18
KMeans,k=4000	0.900	275
KMeans,k=16000	0.927	310
KMeans,k=32000	0.935	385
KMeans,k=64000	0.943	593
HKMeans, k,L=10,4	0.844	567
HKMeans, k,L=10,5	0.952	107
HKMeans, k,L=10,6	0.972	15
HKMeans, k,L=12,6	0.961	24
HKMeans-other, k,L=10,4	0.840	592
HKMeans-other, k,L=10,5	0.918	92
HKMeans-other, k,L=10,6	0.948	44
HKMeans-other, k,L=12,6	0.952	37

Table 1. Recall using a bag-of-words algorithm (no affine verification is applied) using either our range-neighbors algorithm or kmeans-based vocabularies. Database is DB_{32k}. For each query image, only the 53 most similar images are returned. KMeans uses a flat kmeans vocabulary, learned on a different dataset. HKMeans is the hierarchical kmeans of [11] where vocabulary is learned on DB_{32k}. HKMeans-other is the same algorithm but vocabulary is learned on a different dataset. For kmeans based algorithms, assignation of a word to a descriptor of the query image is done using an exact linear search. For HKMeans, flat scoring is used.

engine should handle. This list was obtained by merging the lists of transformations used in [4, 8, 9]. The 53 transformations we use are (numbers in brackets are the number of transformations per category) : Colorizing (3), Changing contrast (4), Cropping (7), Despeckling (1), Applying emboss filter (1), Converting to GIF (1), Rotating (3), Scaling (6), Changing saturation (5), Changing intensity (6), Applying sharpness filter (2), Converting to JPEG (2), Applying median filter (1), Rotating and Cropping (2), Rotating and Scaling (2), Shearing (4), Applying a Gaussian Blur (3), Adding Gaussian Noise (2). Our test framework is to pick 50 random images from Internet and to apply the 53 deformations to these images. The obtained 2650 (50×53) images are a first dataset. For space limits reasons, we do not give here all the details about the transformations we used but the full set of images is publicly available at [1].

The 50 original images will be used as queries. For each one, the algorithm should only return the 53 deformations of the image. We also add other images as perturbators in the database : a first set of 30k images or a set of 510k images. Figure 4 shows some examples of difficult transformations we used. The database of 32650 images is noted DB_{32k} and the database of 512650 images is noted DB_{512k}. For each image, 256 descriptors (of type 128-D SIFT) are extracted. We limit the number of descriptors as it has been noticed in [4] that the recall almost does not decrease when passing from 1000 features per image to 256. We also confirmed these results in our tests. DB_{32k} contains 8.1M descriptors while DB_{512k} has more than 128M points.

5.1. Results on comparing vocabularies

We first applied the algorithm on the DB_{32k} dataset. Table 1 shows the performance of our algorithm and those of kmeans-based approaches. A first conclusion is that our algorithm has a higher recall than the kmeans-based vocabularies. When using a flat kmeans, even with a 64k vocabulary, recall is lower and it is slower than our hashing. In order to achieve similar query times or to use larger vocabularies, one should use an approximate nearest neighbors algorithm to assign words to query descriptors. In [12], authors show that using a randomized kd-tree forest for this task does not generate any loss in the recall while being much faster. This solution may bring kmeans vocabulary and our hashing to similar query time. But on the given dataset, the recall obtained by our hashing would still be higher for the tested vocabulary sizes. Another advantage of our approach is its simplicity. Coupling kmeans clustering and a randomized forest leads to more parameters and thus to more tuning. In table 1, we also indicate performance of hierarchical kmeans. When the vocabulary is learned on the DB_{32k} dataset, using a 1M vocabulary (k,L=10,5) obtains performance similar to our hashing. When using a different dataset for learning, our algorithm performs better.

The main conclusion of these tests is that our algorithm performs better than kmeans based vocabularies when visual words are not learned from the searched dataset. In the case of kmeans vocabularies learned on the searched dataset, results of our algorithm are similar to those of a 1M words hierarchical kmeans-based vocabulary. Still, the advantage of our algorithm is that performance does not depend of any learning dataset. Using a kmeans-based vocabulary, if the database is created incrementally (e.g. users add images from time to time), one has to update the visual words frequently to guarantee optimal performance. This is not required by our algorithm.

5.2. Scalability

As each database point is indexed only once in a single hash table, our algorithm is well adapted to large databases.

For the DB_{512k} dataset, the hash table requires 1GB of memory. Using a maximum of 256 descriptors per image, the table of all database descriptors requires 1GB. For each descriptor, only position, orientation and scale are saved (it is compacted to require only 64bits per descriptor). Thus, our implementation does not require any disk access for a query within a database of 512.000 images.

We applied our algorithm to the same image retrieval benchmark as described in the previous paragraph, but on the DB_{512k}. When only 53 images are returned, recall is 0.743 but this is a very difficult problem as only 53 images are chosen among 512.000 possibilities. Much better results are obtained when more images are kept by the bag-of-words filtering (according to the bag-of-words score) and then affine verification filters out outliers (using a RANSAC algorithm as in [10]). Using this second approach, if our algorithm selects the 4000 images most similar to the query and then applies affine verification, it obtains a recall of 0.965 and a precision of 0.971. With this configuration, the total time of a query is 360ms.

6. Conclusions

In this article, we were concerned by the problem of near duplicate detection in image databases. Our main contribution is a new range-neighbors algorithm that is straightforward to use in a bag-of-words framework. It is based on detecting distinctive dimensions to compute hash keys for each local descriptor. These hash functions are fast to compute and the indexing does not require as much memory as LSH approaches do. As a pure approximate range-neighbors algorithm, our algorithm compares very well to several others. But the main result is that, for near duplicate retrieval, it performs better than kmeans-based vocabulary if the kmeans is not applied on the searched dataset. This makes our algorithm a particularly suitable solution for applications where the image database evolves. Using a kmeans approach would require to update the visual vocabulary regularly to avoid degraded performance (and to define when to do these updates). Our algorithm does not need such updates and provides retrieval results as good as those of kmeans based vocabulary learned on the searched dataset.

References

[1] http://www.math-info.univ-paris5.fr/ auclair/dataset.html.

[2] A. Auclair, L. D. Cohen, and N. Vincent. How to use SIFT vectors to analyze an image with database templates. In *Proceedings of the 5th International Workshop on Adaptive Multimedia Retrieval (AMR), Paris, France*, July 2007.

[3] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, New York, NY, USA, 2004. ACM.

[4] J. J. Foo and R. Sinha. Pruning sift for scalable near-duplicate image matching. In J. Bailey and A. Fekete, editors, *Eighteenth Australasian Database Conference (ADC 2007)*, volume 63 of *CRPIT*, pages 63–71, Ballarat, Australia, 2007. ACS.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *The VLDB Journal*, pages 518–529, 1999.

[6] H. Jegou, L. Amsaleg, C. Schmid, and P. Gros. Query-adaptative locality sensitive hashing. In *International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 2008.

[7] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Proceedings of the European Conference on Computer Vision (ECCV), Marseille, France*, LNCS. Springer, oct 2008.

[8] Y. Ke, R. Sukthankar, and L. Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 869–876, New York, NY, USA, 2004. ACM Press.

[9] H. Lejsek, F. H. Ásmundsson, B. T. Jónsson, and L. Amsaleg. Efficient and effective image copyright enforcement. In *BDA, 21èmes Journées Bases de Données Avancées, BDA 2005, Saint Malo, 17-20 octobre 2005, Actes (Informal Proceedings)*, 2005.

[10] D. G. Lowe. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision (IJCV)*, volume 20, pages 91–110, 2004.

[11] D. Nistér and H. Stewénius. Scalable recognition with a vocabulary tree. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), New York, USA*, pages 2161–2168, 2006.

[12] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Minneapolis, USA*, 2007.

[13] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[14] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing)*. The MIT Press, 2006.

[15] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision (ICCV), Nice, France*, volume 2, pages 1470–1477, Oct. 2003.

[16] E. Valle, M. Cord, and S. Philipp-Foliguet. High-dimensional descriptor indexing for large multimedia databases. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 739–748, New York, NY, USA, 2008. ACM.