COMPACT HASH CODES AND DATA STRUCTURES FOR EFFICIENT MOBILE VISUAL SEARCH

Simone Ercoli, Marco Bertini, Alberto Del Bimbo

MICC - Università degli Studi di Firenze

ABSTRACT

In this paper we present an efficient method for mobile visual search that exploits compact hash codes and data structures for visual features retrieval. The method has been tested on a large scale standard dataset of one million SIFT features, showing a retrieval performance comparable or superior to state-of-the-art methods, and a very high efficiency in terms of memory consumption and computational requirements. These characteristics make it suitable for application to mobile visual search, where devices have limited computational and memory capabilities.

Index Terms— Mobile visual search, nearest neighbor search, hashing, SIFT.

1. INTRODUCTION

The technical features of modern smartphones have greatly improved in all aspects in recent years. Regarding the computational capabilities mobile phones now have relatively fast processors and storage that is in the order of tens of GB. However certain characteristics are still quite lagging with respect to PCs. For example mobile phones have very limited memory (e.g. Apple iPhone 6 has only 1 GB RAM) and CPUs with smaller cache, and smaller clock rate, than those of PCs. Because of these limitations, algorithms designed to run on desktop computers are not suitable for smartphones and mobile devices.

Regarding the problem of visual search, methods that aim at reducing computational costs typically use feature hashing, performing nearest neighbor search using Hamming distances. These methods generally use inverted files, e.g. hash tables, that require large quantities of memory to store the hash codes of the features. Moreover hash codes are relatively long (in the order of several tens of bits) to obtain a reasonable performance in retrieval, thus requiring fairly large amounts of memory when storing large scale databases of features.

In this paper we present a novel method for feature hashing, based on k-means, that requires a very limited codebook size and that obtains good performance in retrieval even with very compact hash codes. We show also the benefit of using compact data structures to store a large database of features. The proposed approach greatly reduces memory requirements and is suitable, also in terms of computational cost, for mobile visual search applications. The proposed method is compared to state-of-the-art approaches on a standard large scale dataset, showing a retrieval performance comparable or superior to more complex state-of-the-art approaches.

This paper is organized as follows: previous works are reviewed in Sect. 2; the proposed method is presented in Sect. 3; experimental results and comparison with state-of-the-art approaches are shown in Sect. 4. Finally, conclusions are drawn in Sect. 5.

2. PREVIOUS WORK

Previous works on visual feature hashing can be classified in methods based on hashing functions, vector quantization and scalar quantization. These methods typically rely on the use of inverted files to store the hash codes and to perform retrieval. A few works have also addressed specifically the study of efficient data structures for nearest neighbor retrieval.

Hashing: Weiss *et al.* [1] have cast the problem of hashing as a particular form of graph partitioning, and then relaxing the problem. The proposed hashing algorithm, called Spectral Hashing (SH), is efficient and outperforms other methods based on hashing such as Locality Sensitive Hashing (LSH) and semantic hashing [2], based on restricted Boltzmann machine (RBM). Heo *et al.* [3] have proposed to use hyperspheres instead of hyperplanes to encode high- dimensional data points, to improve mapping of spatially coherent data points into a binary code. Paulevé *et al.* [4] have compared different types of hash functions: random projections and different types of lattices for structured quantization, and two vector quantization methods, i.e. k-means and hierarchical k-means clustering, for unstructured quantizers. Experimental results on SIFT features show that unstructured quantizers that fit better real datasets, and provide performances significantly superior to structured quantizers.

Vector quantization: Jégou *et al.* [5] have proposed to decompose the feature space into a Cartesian product of low-dimensional subspaces, that are quantized separately. This Product Quantization (PQ), originally used in source coding, is efficient in solving memory issues that arise when using vector quantization methods such as k-means, since in this case a much reduced number of centroids is needed. The method has been shown to obtain state-of-the-art results on a large scale SIFT features dataset, improving over methods such as SH [1] and Hamming Embedding [6]. Norouzi and Fleet [7] build upon the idea of compositionality of the PQ approach, proposing two variations of k-means: Orthogonal k-means and Cartesian k-means, that can be viewed as generalization of the Iterative Quantization (ITQ) [8] and Product Quantization algorithms, respectively. Chandrasekhar *et al.* [9] have compared several compression schemes for SIFT descriptors, showing that PQ approach obtains best results, when compared to other vector quantization approach, as well as against hashing and transform coding. Ge *et al.* [10] have proposed an improvement of PQ, optimizing product quantization by minimizing quantization distortions w.r.t. the space decomposition and the quantization codebooks.

Scalar quantization: Zhou *et al.* [11] have proposed an approach based on scalar quantization, applying it to SIFT descriptors. Hashing is performed computing the median and the third quartile of the bins of a SIFT descriptor, then coding the value of each bin according to this subdivision. The hash code has a dimension of 256, and the first 32 bits are used to index the code in an inverted file. Ren *et al.* [12] have extended the approach of [11] including an evaluation of the reliability of bits, depending on their quantization errors. "Unreliable" bits, i.e. those with large quantization errors, are then flipped at query time to perform query expansion. Chen and Hsieh [13] have recently proposed an approach that quantizes the differences of the bins of the SIFT descriptor, using as a threshold the median computed on all the SIFT points of a training set.

Data structures: Babenko and Lempitsky [14] have proposed a data structure for efficient similarity search, called inverted multi-index, that generalizes the inverted index by replacing vector quantization inside inverted indices with product quantization, and building the multi-index as a multi-dimensional table. An efficient algorithm to produce an ordered sequence of multi-index entries for a query is also proposed. Very recently Norouzi *et al.* [15] have proposed a method to build multiple hashing tables for exact k-nearest neighbor search of hash codes, testing the method on a large scale SIFT dataset.

3. THE PROPOSED METHOD

The proposed method exploits a novel version of the k-means based hashing schema, introducing the possibility of assignment to multiple cluster centers during the quantization process, performing a sort of quantized codebook soft assignment. This approach greatly reduces the number of required cluster centers and thus, also training data. The resulting hash code is stored in a memory efficient data structure, suitable for devices with very limited RAM such as mobile devices. Most of the approaches presented in scientific literature have relied on inverted files [5, 11, 12] that are typically implemented as hash tables[1] [13, 15–17] or B-Trees. Instead we propose the use of a variant of radix tree (also known as 'patricia trie' or 'trie') for an extremely compact storage of the hash codes of visual features. The combination of these two solutions results in a greatly reduced consumption of memory and improved search speed.

3.1. Hashing

A typical approach to unstructured vector quantization is the use of k-means algorithm to compute the hash code of visual feature, since it minimizes the quantization error by satisfying the two Lloyd optimality conditions [5]. In this approach a dictionary is learned over a training set and hash codes of features are obtained by computing their distance to each cluster center. Vectors are assigned to the nearest cluster center whose code is used as hash code; considering the case of SIFT points, i.e. a 128-dimensional feature vector, this means that compressing it to 64 bits code, requires to use $k = 2^{64}$ centroids. Therefore, the computational cost of learning a k-means based quantizer becomes expensive in terms of both memory and time: there is need of large quantities of training data, e.g. several times larger than k, and the execution time of the algorithm becomes not feasible. A possible way to reduce this cost is to use hierarchical k-means (HKM), but the problem of memory usage and size of the required learning set is affecting also this approach. Since the quantizer is defined by the k centroids, the use of quantizers with a large number of centroids may not be practical or efficient: if a feature has a dimension D, there is need to store $k \times D$ values to represent the codebook of the quantizer.

A possible solution is to reduce the length of the hash signature, at the expenses of retrieval performance. Jégou *et al.* [5] have proposed the use of product k-means quantization to overcome this issue. In our approach, instead, we propose to compute a sort of soft assignment within the k-means framework, to obtain very compact signatures and dimension of the quantizer, while maintaining a retrieval performance similar to that of [5].

The proposed method, called multi-k-means, starts learning a standard k-means dictionary, using a very small number of centroids to maintain a low computational cost. The main difference resides in the assignment and creation of the hash code. The geometric mean of all the distances between the feature to be quantized and the centroids of the codebook is computed. Then the feature vector is considered as belonging to all the centroids from which its distance is below the

[1]E.g. Yael Library `http://yael.gforge.inria.fr`

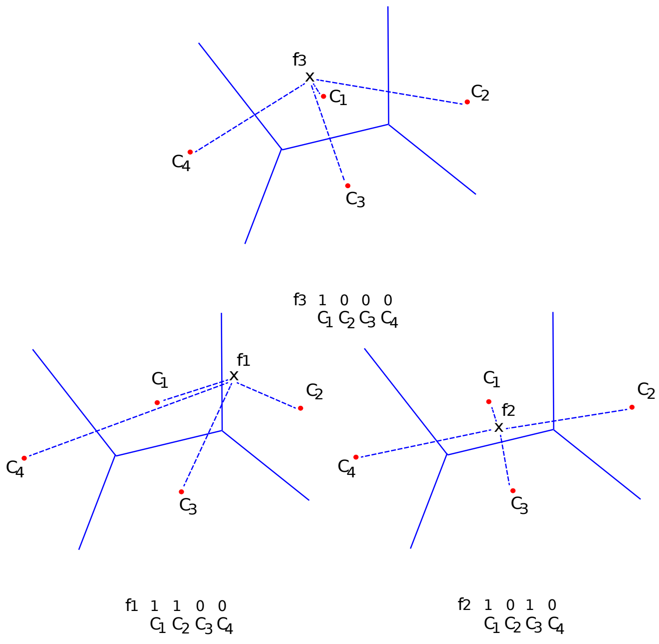

Fig. 1. Illustration of the proposed method: if a feature is assigned to a centroid the corresponding bit in the hash code is set to 1. Features that strongly belong to a cluster are assigned only to it (*top*), while those that are near to the borders of a Voronoi cell are assigned to more than one cell (*bottom*).

mean. The hash signature is then computed by concatenating the sequence of 0s, if the feature vector does not belong to the centroid, and 1s, if the feature is assigned to the centroid. This approach allows to create hash signatures using a much smaller number of centroids, than using the usual k-means baseline, since each centroid is directly associated to a bit of the hash code. This approach can be considered a quantized version of codebook soft assignment [18] and, similarly, it alleviates the problem of codeword ambiguity while reducing the quantization error. Fig. 1 illustrates quantization process and the resulting hash codes in three cases: one in which a vector strongly belongs to a single cluster and two cases in which vectors are assigned to more than one clusters.

Typically to solve the problem of ambiguous assignment to a codebook centroid (in case of vector quantization) or quantization error (e.g. in case of scalar quantization or LSH), a multi probe approach is used. This means that one or more bits of the query hash code are flipped to perform a query expansion, thus improving recall at the expense of computational cost. The need for multi probe queries is not present in this method, because of the possibility of assignment of features to more than one centroid.

3.2. Indexing

Radix tries are often used in approximate string matching algorithms [19], such as those required for spell checking. The trie can be used to implement an inverted file, where the key to search the data is stored in the position of the nodes. A memory efficient variant of radix tree is the patricia trie, a data structure that represents a space-optimized trie in which each node with only one child is merged with its parent. Fig. 2 compares a trie with a patricia trie.

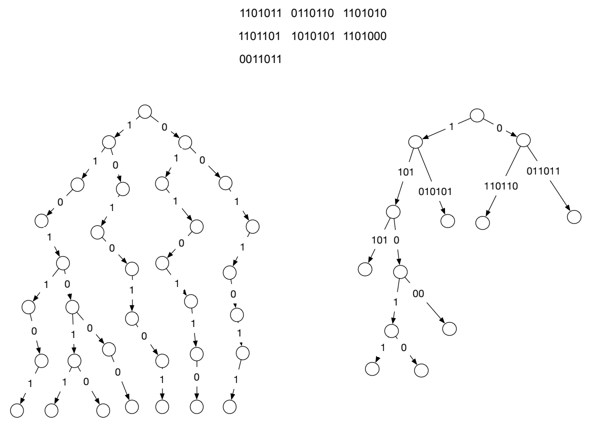

Fig. 2. Comparison of trie (*left*) vs. patricia trie (*right*), used to store 7 hash codes (*top*).

In this work we propose the use of Matching Algorithm with Recursively Implemented StorAge (MARISA) trie. MARISA trie is a recursive data structure in which a patricia trie is used to represent another patricia trie. This recursion makes the data structure more compact at the expenses of search performance. In particular, to maintain a good time-space tradeoff, we have used one level of recursion depth.

Patricia tries have a better space complexity than hash tables, while time complexity is comparable only when considering imbalanced hash tables. However, the compactness of the data structure makes it amenable to be maintained in the CPU cache, thus greatly improving its speed.

4. EXPERIMENTAL RESULTS

Our approach is compared with three methods: the state-of-the-art Product Quantization [5] approach, and two vector quantization baselines: ERC Forests [20] and standard quantization with k-means [21,22]. Experiments have been carried on a large scale standard dataset.

4.1. Dataset

For our experiments we used a dataset of 1 million SIFT descriptors [5, 23], commonly used to compare methods for visual feature hashing [5, 7, 10, 14, 15, 23]. The dataset is composed by three different subsets: a learning set, a query set and base set, along with the ground truth results for each query. Each of these data sets is coming from publicly available images and they are also publicly available[2]. In particular, query and base descriptors are extracted from the INRIA Holidays images [24], while the learning set is extracted from Flickr 1M dataset. Tab. 1 summarizes the characteristics of the dataset. For each query the ground truth reports the 100 nearest SIFT points in the database set, computed using an exhaustive Euclidean distance calculation.

Table 1. Dataset dimensions

vector dataset	SIFT
descriptor dimensionality D	128
learning set vectors	100,000
database set vectors	1,000,000
queries set vectors	10,000

4.2. Data structure

In the first experiment we evaluate the performance of MARISA trie w.r.t. the data structures typically used to implement inverted files, i.e. hash table and binary tree. The 10^6 feature vectors of the database set have been coded using the proposed multi-k-means quantization, then stored using C++ implementations of the data structures. The STL versions of `unordered_multimap` and `multimap` provided by GCC C++ compiler have been used for hash tables and B-Tree, respectively. Results reported in Tab. 2 show that MARISA trie obtains a dramatic improvement both in terms of speed and size. In particular hash table and B-Tree would occupy a very large percentage of RAM in a mobile phone (e.g. $\sim 10\%$ in an Apple iPhone 6), while the trie fits the L1 cache of an ARM CPU commonly used in mobile phones.

Table 2. Comparison of data structures used in the experiments to store 1 million SIFT hash codes (computed with the proposed approach). Search time measures the time required to perform the series using all the 10,000 query vectors of the dataset.

	MARISA trie	Hash table	Binary Tree
Dimension (Kb)	**19**	90,112	93,184
Search Time (ms)	**25.3**	245.31	370

4.3. Comparison

The methods used in the experiments have different types of parameters that can be changed to obtain the best performances.

Our method uses the number of centroids k in standard k-means to compute the hash code. The final hash code length is equal to the number of k means centroids.
The standard quantization with k-means is characterized by number of centroids *k* and quantization approach produces an hash code of length $log_2 k$. Due to the noise in quantization, this standard approach should have a number of quantizers *k* sufficiently large, but with a large number of centroids the complexity is prohibitive. Therefore we should have a big amount of memory to store the codes; because of this we use, for standard k-means approach, an hash code of maximum length 16.

The product quantizer presented in [5] uses two different distance computations: *Asymmetric Distance Computation* (ADC) and *Inverted File with Asymmetric Distance Computation* (IVFADC).
ADC is characterized by the number of sub vectors *m* and the number of quantizers per sub vectors *k**, and produces a code of length *m* x $log_2 k^*$.
IVFADC is characterized by the codebook size *k'* (number of centroids associated to each quantizer), the number of neighbouring cells *w* visited during the multiple assignment, the number of sub vectors *m* and the number of quantizers per sub vectors k^* which is in this case fixed to *k**=256. The length of the final code is given by *m* x $log_2 k^*$.

ERC Forests [20] is characterized by the number of estimators, which indicates the number of trees in the forest and it produces a code of length at most $\#estimators \times 2^{tree\_depth}$, where *tree_depth* value indicates the maximum depth of each tree in the forest. In our experiments we used *tree_depth=3*.

Table 3. Method parameters for signature length

methods	signature length (bits)
k-means	$log_2 k$
ERC Forests [20]	$\#estimators \times 2^{tree\_depth}$
ADC/IVFADC [5]	*m* x $log_2 k^*$
m-k-means	k

Retrieval performance is measured following two measures: *i) recall@R* reported in Table 4; *ii) precision@R* reported in Table 5 and Fig. 3.

recall@R is the average rate of queries for which the 1-nearest neighbor from ground truth is ranked in the top *R* positions. This metric has been used in [5]. This value *R* is an output of the method. Following the experimental setup of [5] we have used only the first nearest neighbor. *precision@R* is the average rate of queries for which the nearest neighbor computed by a method is ranked in the first R positions of the ground truth set, and was used to evaluate approximate nearest neighbor in [25]. This value indicates the fraction of queries for which the nearest neighbor is correctly retrieved among the base vectors. In the case of *R=1* we have that *recall@R* and *precision@R* are the same measure.

Fig. 3 reports *precision@R* of the best performing methods of Tab. 5, for R varying between 1 and 100. For each

[2]BIGANN dataset - `http://corpus-texmex.irisa.fr`

Table 4. *Recall@1* comparison between our method, two baselines (k-means and ERC Forests), and the Product Quantization method [5].

method	parameters	code length	recall@1
k-means	k=256	8	0,012
k-means	k=1024	10	0,0007
k-means	k=4096	12	0,0004
k-means	k=65536	16	0
ERC Forests	3 estimators	23	0,535
ADC	m=2 k*=256	16	0,00023
ADC	m=4 k*=256	32	0,054
ADC	m=8 k*=256	64	0,224
ADC	m=16 k*=256	128	0,457
ADC	m=16 k*=4096	192	0,633
IVFADC	m=16 k'=1024 w=64	128	0,467
IVFADC	m=16 k'=8192 w=64	128	0,496
m-k-means	k=23	23	**0,724**

method we show only the configuration that allows to obtain the best performance.

In all these experiments we use a MARISA trie data structure for matching, with a minimal recursion to maintain a good balance between dimension and search time. Results for ADC and IVFADC versions of Product Quantization have been obtained using original code of the authors of the method.

Results in Tab. 4 show that the proposed approach has the best performance in terms of *recall@1*, and that the second best method results in a much larger code length, thus requiring a much larger memory to store the database. Results in Tab. 5 show that the proposed approach has the best performance for *precision@10*. Regarding the value of *precision@100* both variants of Product Quantization obtain the best performance, although the difference with the performance of the proposed method is negligible. In all the cases it has to be noted that the code length of the competing methods is much larger than that of multi-k-means.

Table 5. *Precision@10* and *Precision@100* comparison between our method, two baselines (k-means and ERC Forests), and the Product Quantization method [5].

method	parameters	code length	precision@10	precision@100
k-means	k=256	8	0,034	0,083
k-means	k=1024	10	0,025	0,065
k-means	k=4096	12	0,001	0,0027
k-means	k=65536	16	0,0001	0,0023
ERC Forests	3 estimators	23	0,8871	0,983
ADC	m=2 k*=256	16	0,02	0,092
ADC	m=4 k*=256	32	0,191	0,483
ADC	m=8 k*=256	64	0,579	0,902
ADC	m=16 k*=256	128	0,884	**0,996**
IVFADC	m=16 k'=1024 w=64	128	0,894	**0,996**
IVFADC	m=16 k'=8192 w=64	128	0,872	0,991
m-k-means	k=23	23	**0,966**	0,993

Figure 3 shows how our method with a signature of 23 bits outperforms other methods (in the best configuration) for *R*=1 and *R*=10 while for *R*=*100* results are almost equal.

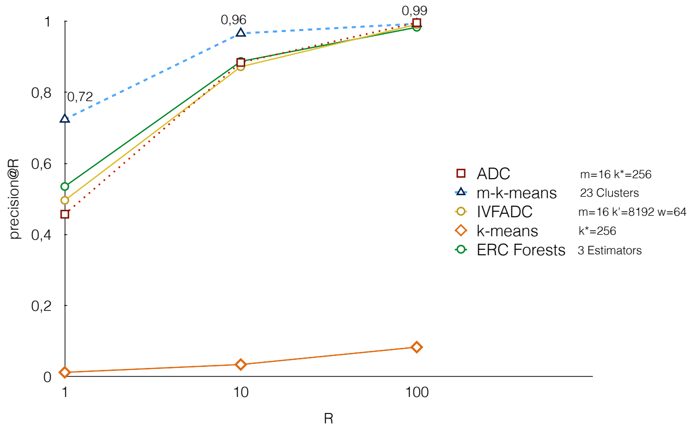

Fig. 3. Evaluation in terms of Precision@R using the best configurations of all the methods (ADC: m=16 k^*=256, IVFADC: m=16 k'=8192 w=64, ERC Forests: 3 $Estimators$, k-means: k^*=256, m-k-means: k = 23 clusters) for different values of R.

5. CONCLUSIONS

We have proposed a new version of the k-means based hashing schema called multi-k-means which uses a small number of centroids and guarantees a low computational cost. Our compact hash signature, in conjunction with the recursive data structure MARISA trie, provides best results in terms of memory requirements and search time compared to the traditional approaches represented by hash table and binary tree. It also reaches top performances compared with two baselines methods and state-of-the-art approaches based on product quantization. Future work will focus on evaluation of this approach on a bigger dataset to study its scalability and will concentrate on its applicability to a real world image retrieval context.

Acknowledgments The authors thank Hervé Jégou for his help in running the code of his method. This work is partially supported by the "Social Museum and Smart Tourism" project (CTN01_00034_231545) and IARPA's JANUS program (GLAIVE project - 2014-14071600011).

6. REFERENCES

[1] Yair Weiss, Antonio Torralba, and Rob Fergus, "Spectral hashing," in *Proc. of Neural Information Processing Systems (NIPS)*, 2009.

[2] Ruslan Salakhutdinov and Geoffrey Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969 – 978, 2009, Special Section on Graphical Models and Information Retrieval.

[3] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon, "Spherical hashing," in *Proc. of IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2012.

[4] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1348 – 1358, 2010.

[5] Hervé Jégou, Matthijs Douze, and Cordelia Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[6] Mihir Jain, Hervé Jégou, and Patrick Gros, "Asymmetric Hamming embedding: Taking the best of our bits for large scale image search," in *Proc. of ACM Multimedia (ACM MM)*, New York, NY, USA, 2011, MM '11, pp. 1441–1444, ACM.

[7] M. Norouzi and D.J. Fleet, "Cartesian k-means," in *Proc. of IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2013, pp. 3017–3024.

[8] Yunchao Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. of IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2011, pp. 817–824.

[9] Vijay Chandrasekhar, Mina Makar, Gabriel Takacs, David Chen, Sam S Tsai, Ngai-Man Cheung, Radek Grzeszczuk, Yuriy Reznik, and Bernd Girod, "Survey of SIFT compression schemes," in *Proc. International Workshop Mobile Multimedia Processing*, 2010.

[10] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proc. of IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[11] Wengang Zhou, Yijuan Lu, Houqiang Li, and Qi Tian, "Scalar quantization for large scale image search," in *Proc. of ACM Multimedia (ACM MM)*, 2012.

[12] Guangxin Ren, Junjie Cai, Shipeng Li, Nenghai Yu, and Qi Tian, "Scalable image search with reliable binary code," in *Proc. of ACM Multimedia (ACM MM)*, 2014.

[13] Chun-Che Chen and Shang-Lin Hsieh, "Using binarization and hashing for efficient SIFT matching," *Journal of Visual Communication and Image Representation*, , no. 0, pp. –, 2015.

[14] A. Babenko and V. Lempitsky, "The inverted multi-index," in *Proc. of IEEE Computer Vision and Pattern Recognition (CVPR)*, June 2012, pp. 3069–3076.

[15] M. Norouzi, A. Punjani, and D.J. Fleet, "Fast exact search in hamming space with multi-index hashing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 6, pp. 1107–1119, June 2014.

[16] J. Sivic and A. Zisserman, "Video Google: a text retrieval approach to object matching in videos," in *Proc. of IEEE Computer Vision and Pattern Recognition (CVPR)*, Oct 2003, pp. 1470–1477 vol.2.

[17] Antonio Torralba, Rob Fergus, and Yair Weiss, "Small codes and large image databases for recognition," in *Proc. of CVPR*, 2008.

[18] J.C. van Gemert, C.J. Veenman, A.W.M. Smeulders, and J.-M. Geusebroek, "Visual word ambiguity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 7, pp. 1271–1283, July 2010.

[19] Alfred V. Aho and Margaret J. Corasick, "Efficient string matching: An aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[20] F. Moosmann, B. Triggs, and F. Jurie, "Fast discriminative visual codebooks using randomized clustering forests," in *Proc. of Neural Information Processing Systems (NIPS)*, 2007.

[21] James MacQueen et al., "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Oakland, CA, USA., 1967, vol. 1, pp. 281–297.

[22] Robert M. Gray and David L. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[23] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg, "Searching in one billion vectors: rerank with source coding," in *Proc. of ICASSP*, 2011.

[24] Herve Jegou, Matthijs Douze, and Cordelia Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Proc. of European Conference on Computer Vision (ECCV)*, pp. 304–317. Springer, 2008.

[25] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proc. VISAPP*, 2009.