Product Quantization for Nearest Neighbor Search

Hervé Jégou, Matthijs Douze, Cordelia Schmid

Product quantization for nearest neighbor search

Hervé Jégou, Matthijs Douze, Cordelia Schmid

Abstract— **This paper introduces a product quantization based approach for approximate nearest neighbor search. The idea is to decomposes the space into a Cartesian product of low dimensional subspaces and to quantize each subspace separately. A vector is represented by a short code composed of its subspace quantization indices. The Euclidean distance between two vectors can be efficiently estimated from their codes. An asymmetric version increases precision, as it computes the approximate distance between a vector and a code.**

Experimental results show that our approach searches for nearest neighbors efficiently, in particular in combination with an inverted file system. Results for SIFT and GIST image descriptors show excellent search accuracy outperforming three state-of-the-art approaches. The scalability of our approach is validated on a dataset of two billion vectors.

Index Terms— **High-dimensional indexing, image indexing, very large databases, approximate search.**

I. Introduction

Computing Euclidean distances between high dimensional vectors is a fundamental requirement in many applications. It is used, in particular, for nearest neighbor (NN) search. Nearest neighbor search is inherently expensive due to the *curse of dimensionality* [3], [4]. Focusing on the D-dimensional Euclidean space $\mathbb{R}^D$, the problem is to find the element $\text{NN}(x)$, in a finite set $\mathcal{Y} \subset \mathbb{R}^D$ of n vectors, minimizing the distance to the query vector $x \in \mathbb{R}^D$:

$$\text{NN}(x) = \arg\min_{y \in \mathcal{Y}} d(x, y). \tag{1}$$

Several multi-dimensional indexing methods, such as the popular KD-tree [5] or other branch and bound techniques, have been proposed to reduce the search time. However, for high dimensions it turns out [6] that such approaches are not more efficient than the brute-force exhaustive distance calculation, whose complexity is $\mathcal{O}(nD)$.

There is a large body of literature [7], [8], [9] on algorithms that overcome this issue by performing approximate nearest neighbor (ANN) search. The key idea shared by these algorithms is to find the NN with high probability "only", instead of probability 1. Most of the effort has been devoted to the Euclidean distance, though recent generalizations have been proposed for other metrics [10]. In this paper, we consider the Euclidean distance, which is relevant for many applications. In this case, one of the most popular ANN algorithms is the Euclidean Locality-Sensitive Hashing (E2LSH) [7], [11], which provides theoretical guarantees on the search quality with limited assumptions. It has been successfully used for local descriptors [12] and 3D object indexing [13], [11]. However, for real data, LSH is outperformed by heuristic methods, which exploit the distribution of the vectors. These methods include randomized KD-trees [14] and hierarchical k-means [15], both of which are implemented in the FLANN selection algorithm [9].

ANN algorithms are typically compared based on the trade-off between search quality and efficiency. However, this trade-off does not take into account the memory requirements of the indexing structure. In the case of E2LSH, the memory usage may even be higher than that of the original vectors. Moreover, both E2LSH and FLANN need to perform a final re-ranking step based on exact L2 distances, which requires the indexed vectors to be stored in main memory if access speed is important. This constraint seriously limits the number of vectors that can be handled by these algorithms. Only recently, researchers came up with methods limiting the memory usage. This is a key criterion for problems involving large amounts of data [16], i.e., in large-scale scene recognition [17], where millions to billions of images have to be indexed. In [17], Torralba *et al.* represent an image by a single global GIST descriptor [18] which is mapped to a short binary code. When no supervision is used, this mapping is learned such that the neighborhood in the embedded space defined by the Hamming distance reflects the neighborhood in the Euclidean space of the original features. The search of the Euclidean nearest neighbors is then approximated by the search of the nearest neighbors in terms of Hamming distances between codes. In [19], spectral hashing (SH) is shown to outperform the binary codes generated by the restricted Boltzmann machine [17], boosting and LSH. Similarly, the Hamming embedding method of Jegou *et al.* [20],

This work was partly realized as part of the Quaero Programme, funded by OSEO, French State agency for innovation. It was originally published as a technical report [1] in August 2009. It is also related to the work [2] on source coding for nearest neighbor search.

[21] uses a binary signature to refine quantized SIFT or GIST descriptors in a bag-of-features image search framework.

In this paper, we construct short codes using quantization. The goal is to estimate distances using vector-to-centroid distances, i.e., the query vector is not quantized, codes are assigned to the database vectors only. This reduces the quantization noise and subsequently improves the search quality. To obtain precise distances, the quantization error must be limited. Therefore, the total number k of centroids should be sufficiently large, e.g., $k = 2^{64}$ for 64-bit codes. This raises several issues on how to learn the codebook and assign a vector. First, the number of samples required to learn the quantizer is huge, i.e., several times k. Second, the complexity of the algorithm itself is prohibitive. Finally, the amount of computer memory available on Earth is not sufficient to store the floating point values representing the centroids.

The hierarchical k-means see (HKM) improves the efficiency of the learning stage and of the corresponding assignment procedure [15]. However, the aforementioned limitations still apply, in particular with respect to memory usage and size of the learning set. Another possibility are scalar quantizers, but they offer poor quantization error properties in terms of the trade-off between memory and reconstruction error. Lattice quantizers offer better quantization properties for uniform vector distributions, but this condition is rarely satisfied by real world vectors. In practice, these quantizers perform significantly worse than k-means in indexing tasks [22]. In this paper, we focus on product quantizers. To our knowledge, such a semi-structured quantizer has never been considered in any nearest neighbor search method.

The advantages of our method are twofold. First, the number of possible distances is significantly higher than for competing Hamming embedding methods [20], [17], [19], as the Hamming space used in these techniques allows for a few distinct distances only. Second, as a byproduct of the method, we get an estimation of the expected squared distance, which is required for ε-radius search or for using Lowe's distance ratio criterion [23]. The motivation of using the Hamming space in [20], [17], [19] is to compute distances efficiently. Note, however, that one of the fastest ways to compute Hamming distances consists in using table lookups. Our method uses a similar number of table lookups, resulting in comparable efficiency.

An exhaustive comparison of the query vector with all codes is prohibitive for very large datasets. We, therefore, introduce a modified inverted file structure to rapidly access the most relevant vectors. A coarse quantizer is used to implement this inverted file structure, where vectors corresponding to a cluster (index) are stored in the associated list. The vectors in the list are represented by short codes, computed by our product quantizer, which is used here to encode the residual vector with respect to the cluster center.

The interest of our method is validated on two kinds of vectors, namely local SIFT [23] and global GIST [18] descriptors. A comparison with the state of the art shows that our approach outperforms existing techniques, in particular spectral hashing [19], Hamming embedding [20] and FLANN [9].

Our paper is organized as follows. Section II introduces the notations for quantization as well as the product quantizer used by our method. Section III presents our approach for NN search and Section IV introduces the structure used to avoid exhaustive search. An evaluation of the parameters of our approach and a comparison with the state of the art is given in Section V.

II. Background: quantization, product quantizer

A large body of literature is available on vector quantization, see [24] for a survey. In this section, we restrict our presentation to the notations and concepts used in the rest of the paper.

A. Vector quantization

Quantization is a destructive process which has been extensively studied in information theory [24]. Its purpose is to reduce the cardinality of the representation space, in particular when the input data is real-valued. Formally, a quantizer is a function q mapping a D-dimensional vector $x \in \mathbb{R}^D$ to a vector $q(x) \in \mathcal{C} = \{c_i; i \in \mathcal{I}\}$, where the index set $\mathcal{I}$ is from now on assumed to be finite: $\mathcal{I} = 0 \dots k-1$. The reproduction values c_i are called *centroids*. The set of reproduction values $\mathcal{C}$ is the *codebook* of size k.

The set $\mathcal{V}_i$ of vectors mapped to a given index i is referred to as a (Voronoi) *cell*, and defined as

$$\mathcal{V}_i \triangleq \{x \in \mathbb{R}^D : q(x) = c_i\}. \tag{2}$$

The k cells of a quantizer form a partition of $\mathbb{R}^D$. By definition, all the vectors lying in the same cell $\mathcal{V}_i$ are reconstructed by the same centroid c_i. The quality of a quantizer is usually measured by the mean squared error between the input vector x and its reproduction value $q(x)$:

$$\mathrm{MSE}(q) = \mathbb{E}_X\big[d(q(x),x)^2\big] = \int p(x)\, d\big(q(x),x\big)^2\, dx, \tag{3}$$

where $d(x,y) = ||x-y||$ is the Euclidean distance between x and y, and where $p(x)$ is the probability distribution function corresponding the random variable X. For an arbitrary probability distribution function, Equation 3 is numerically computed using Monte-Carlo sampling, as the average of $||q(x)-x||^2$ on a large set of samples.

In order for the quantizer to be optimal, it has to satisfy two properties known as the Lloyd optimality conditions. First, a vector x must be quantized to its nearest codebook centroid, in terms of the Euclidean distance:

$$q(x) = \arg\min_{c_i \in \mathcal{C}} d(x, c_i). \tag{4}$$

As a result, the cells are delimited by hyperplanes. The second Lloyd condition is that the reconstruction value must be the expectation of the vectors lying in the Voronoi cell:

$$c_i = \mathbb{E}_X\big[x|i\big] = \int_{\mathcal{V}_i} p(x)\, x\, dx. \tag{5}$$

The Lloyd quantizer, which corresponds to the k-means clustering algorithm, finds a near-optimal codebook by iteratively assigning the vectors of a training set to centroids and re-estimating these centroids from the assigned vectors. In the following, we assume that the two Lloyd conditions hold, as we learn the quantizer using k-means. Note, however, that k-means does only find a local optimum in terms of quantization error.

Another quantity that will be used in the following is the mean squared distortion $\xi(q, c_i)$ obtained when reconstructing a vector of a cell $\mathcal{V}_i$ by the corresponding centroid c_i. Denoting by $p_i = \mathbb{P}\big(q(x) = c_i\big)$ the probability that a vector is assigned to the centroid c_i, it is computed as

$$\xi(q, c_i) = \frac{1}{p_i} \int_{\mathcal{V}_i} d\big(x, q(x)\big)^2\, p(x)\, dx. \tag{6}$$

Note that the MSE can be obtained from these quantities as

$$\text{MSE}(q) = \sum_{i \in \mathcal{I}} p_i\, \xi(q, c_i). \tag{7}$$

The memory cost of storing the index value, without any further processing (entropy coding), is $\lceil \log_2 k \rceil$ bits. Therefore, it is convenient to use a power of two for k, as the code produced by the quantizer is stored in a binary memory.

B. Product quantizers

Let us consider a 128-dimensional vector, for example the SIFT descriptor [23]. A quantizer producing 64-bits codes, i.e., "only" 0.5 bit per component, contains $k = 2^{64}$ centroids. Therefore, it is impossible to use Lloyd's algorithm or even HKM, as the number of samples required and the complexity of learning the quantizer are several times k. It is even impossible to store the $D \times k$ floating point values representing the k centroids.

Product quantization is an efficient solution to address these issues. It is a common technique in source coding, which allows to choose the number of components to be quantized jointly (for instance, groups of 24 components can be quantized using the powerful Leech lattice). The input vector x is split into m distinct subvectors u_j, $1 \leq j \leq m$ of dimension $D^* = D/m$, where D is a multiple of m. The subvectors are quantized separately using m distinct quantizers. A given vector x is therefore mapped as follows:

$$\underbrace{x_1, ..., x_{D^*}}_{u_1(x)}, ..., \underbrace{x_{D-D^*+1}, ..., x_D}_{u_m(x)} \rightarrow q_1\big(u_1(x)\big), ..., q_m(u_m(x)\big), \tag{8}$$

where q_j is a low-complexity quantizer associated with the j^{th} subvector. With the subquantizer q_j we associate the index set $\mathcal{I}_j$, the codebook $\mathcal{C}_j$ and the corresponding reproduction values $c_{j,i}$.

A reproduction value of the product quantizer is identified by an element of the product index set $\mathcal{I} = \mathcal{I}_1 \times \ldots \times \mathcal{I}_m$. The codebook is therefore defined as the Cartesian product

$$\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_m, \tag{9}$$

and a centroid of this set is the concatenation of centroids of the m subquantizers. From now on, we assume that all subquantizers have the same finite number k^* of reproduction values. In that case, the total number of centroids is given by

$$k = (k^*)^m. \tag{10}$$

Note that in the extremal case where $m = D$, the components of a vector x are all quantized separately. Then the product quantizer turns out to be a scalar quantizer, where the quantization function associated with each component may be different.

The strength of a product quantizer is to produce a large set of centroids from several small sets of centroids: those associated with the subquantizers. When learning the subquantizers using Lloyd's algorithm, a limited number of vectors is used, but the codebook is, to some extent, still adapted to the data distribution to represent. The complexity of learning the quantizer is m times the complexity of performing k-means clustering with k^* centroids of dimension D^*.

	memory usage	assignment complexity
k-means	$k\,D$	$k\,D$
HKM	$\frac{b_\mathrm{f}}{b_\mathrm{f}-1}(k-1)\,D$	$l\,D$
product k-means	$m\,k^*\,D^* = k^{1/m}\,D$	$m\,k^*\,D^* = k^{1/m}\,D$

TABLE I

MEMORY USAGE OF THE CODEBOOK AND ASSIGNMENT COMPLEXITY FOR DIFFERENT QUANTIZERS. HKM IS PARAMETRIZED BY TREE HEIGHT l AND THE BRANCHING FACTOR b_f.

Storing the codebook $\mathcal{C}$ explicitly is not efficient. Instead, we store the $m \times k^*$ centroids of all the sub-quantizers, i.e., $m\,D^*\,k^* = k^*\,D$ floating points values. Quantizing an element requires k^*D floating point operations. Table I summarizes the resource requirements associated with k-means, HKM and product k-means. The product quantizer is clearly the the only one that can be indexed in memory for large values of k.

In order to provide good quantization properties when choosing a constant value of k^*, each subvector should have, on average, a comparable energy. One way to ensure this property is to multiply the vector by a random orthogonal matrix prior to quantization. However, for most vector types this is not required and not recommended, as consecutive components are often correlated by construction and are better quantized together with the same subquantizer. As the subspaces are orthogonal, the squared distortion associated with the product quantizer is

$$\mathrm{MSE}(q) = \sum_j \mathrm{MSE}(q_j), \qquad (11)$$

where $\mathrm{MSE}(q_j)$ is the distortion associated with quantizer q_j. Figure 1 shows the MSE as a function of the code length for different (m,k^*) tuples, where the code length is $l = m \log_2 k^*$, if k^* is a power of two. The curves are obtained for a set of 128-dimensional SIFT descriptors, see section V for details. One can observe that for a fixed number of bits, it is better to use a small number of subquantizers with many centroids than having many subquantizers with few bits. At the extreme when $m = 1$, the product quantizer becomes a regular k-means codebook.

High values of k^* increase the computational cost of the quantizer, as shown by Table I. They also increase the memory usage of storing the centroids ($k^* \times D$ floating point values), which further reduces the efficiency if the centroid look-up table does no longer fit in cache memory. In the case where $m = 1$, we can not afford using more than 16 bits to keep this cost tractable. Using

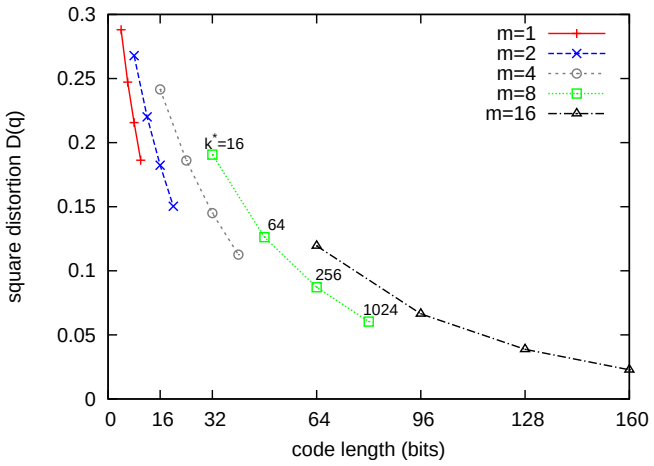

Fig. 1. SIFT: quantization error associated with the parameters m and k^*.

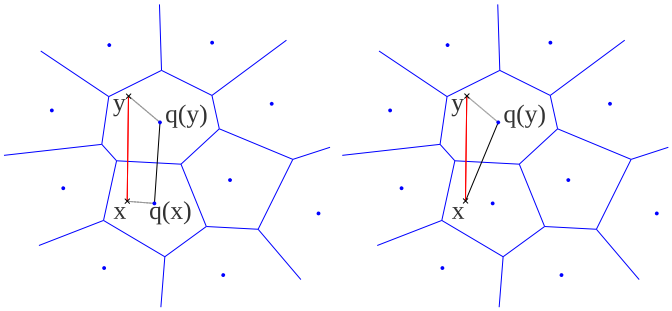

Fig. 2. Illustration of the symmetric and asymmetric distance computation. The distance $d(x, y)$ is estimated with either the distance $d(q(x), q(y))$ (*left*) or the distance $d(x, q(y))$ (*right*). The mean squared error on the distance is on average bounded by the quantization error.

$k^* = 256$ and $m = 8$ is often a reasonable choice.

III. SEARCHING WITH QUANTIZATION

Nearest neighbor search depends on the distances between the query vector and the database vectors, or equivalently the squared distances. The method introduced in this section compares the vectors based on their quantization indices, in the spirit of source coding techniques. We first explain how the product quantizer properties are used to compute the distances. Then we provide a statistical bound on the distance estimation error, and propose a refined estimator for the squared Euclidean distance.

A. Computing distances using quantized codes

Let us consider the query vector x and a database vector y. We propose two methods to compute an approximate Euclidean distance between these vectors,

a symmetric and a asymmetric one. See Figure 2 for an illustration.

Symmetric distance computation (SDC): both the vectors x and y are represented by their respective centroids $q(x)$ and $q(y)$. The distance $d(x,y)$ is approximated by the distance $\hat{d}(x,y) \triangleq d\big(q(x),q(y)\big)$ which is efficiently obtained using a product quantizer

$$\hat{d}(x,y) = d\big(q(x),q(y)\big) = \sqrt{\sum_j d\big(q_j(x),q_j(y)\big)^2}, \tag{12}$$

where the distance $d\big(c_{j,i},c_{j,i'}\big)^2$ is read from a look-up table associated with the j^{th} subquantizer. Each look-up table contains all the squared distances between pairs of centroids (i,i') of the subquantizer, or $(k^*)^2$ squared distances[1].

Asymmetric distance computation (ADC): the database vector y is represented by $q(y)$, but the query x is not encoded. The distance $d(x,y)$ is approximated by the distance $\tilde{d}(x,y) \triangleq d\big(x,q(y)\big)$, which is computed using the decomposition

$$\tilde{d}(x,y) = d\big(x,q(y)\big) = \sqrt{\sum_j d\big(u_j(x),q_j(u_j(y))\big)^2}, \tag{13}$$

where the squared distances $d\big(u_j(x),c_{j,i}\big)^2 \;:\; j = 1\ldots m, i = 1\ldots k^*$, are computed prior to the search.

For nearest neighbors search, we do not compute the square roots in practice: the square root function is monotonically increasing and the squared distances produces the same vector ranking.

Table II summarizes the complexity of the different steps involved in searching the k nearest neighbors of a vector x in a dataset $\mathcal{Y}$ of $n = |\mathcal{Y}|$ vectors. One can see that SDC and ADC have the same query preparation cost, which does not depend on the dataset size n. When n is large ($n > k^* D^*$), the most consuming operations are the summations in Equations 12 and 13. The complexity given in this table for searching the k smallest elements is the average complexity for $n \gg k$ and when the elements are arbitrarily ordered ([25], Section 5.3.3, Equation 17).

The only advantage of SDC over ADC is to limit the memory usage associated with the queries, as the query vector is defined by a code. This is most cases not relevant and one should then use the asymmetric version, which obtains a lower distance distortion for a similar complexity. We will focus on ADC in the rest of this section.

[1]In fact, it is possible to store only $k^*\,(k^*-1)/2$ pre-computed squared distances, because this distance matrix is symmetric and the diagonal elements are zeros.

	SDC	ADC
encoding x	$k^*\,D$	0
compute $d\big(u_j(x),c_{j,i}\big)$	0	$k^*\,D$
for $y \in \mathcal{Y}$, compute $\hat{d}(x,y)$ or $\tilde{d}(x,y)$	$n\,m$	$n\,m$
find the k smallest distances	$n + k\log k \log\log n$	

TABLE II
ALGORITHM AND COMPUTATIONAL COSTS ASSOCIATED WITH SEARCHING THE k NEAREST NEIGHBORS USING THE PRODUCT QUANTIZER FOR SYMMETRIC AND ASYMMETRIC DISTANCE COMPUTATIONS (SDC, ADC).

B. Analysis of the distance error

In this subsection, we analyze the distance error when using $\tilde{d}(x,y)$ instead of $d(x,y)$. This analysis does not depend on the use of a product quantizer and is valid for any quantizer satisfying Lloyd's optimality conditions defined by Equations 4 and 5 in Section II.

In the spirit of the mean squared error criterion used for reconstruction, the distance distortion is measured by the mean squared distance error (MSDE) on the distances:

$$\text{MSDE}(q) \triangleq \iint \big(d(x,y) - \tilde{d}(x,y)\big)^2\, p(x)dx\, p(y)dy. \tag{14}$$

The triangular inequality gives

$$d\big(x,q(y)\big) - d\big(y,q(y)\big) \le d(x,y) \le d\big(x,q(y)\big) + d\big(y,q(y)\big), \tag{15}$$

and, equivalently,

$$\Big(d(x,y) - d(x,q(y))\Big)^2 \le d\big(y,q(y)\big)^2. \tag{16}$$

Combining this inequality with Equation 14, we obtain

$$\text{MSDE}(q) \le \int p(x) \left(\int d\big(y,q(y)\big)^2\, p(y)\, dy \right) dx \tag{17}$$

$$\le \text{MSE}(q). \tag{18}$$

where $\text{MSE}(q)$ is the mean squared error associated with quantizer q (Equation 3). This inequality, which holds for any quantizer, shows that the distance error of our method is statistically bounded by the MSE associated with the quantizer. For the symmetric version, a similar derivation shows that the error is statistically bounded by $2\times\text{MSE}(q)$. It is, therefore, worth minimizing the quantization error, as this criterion provides a statistical upper bound on the distance error. If an exact distance

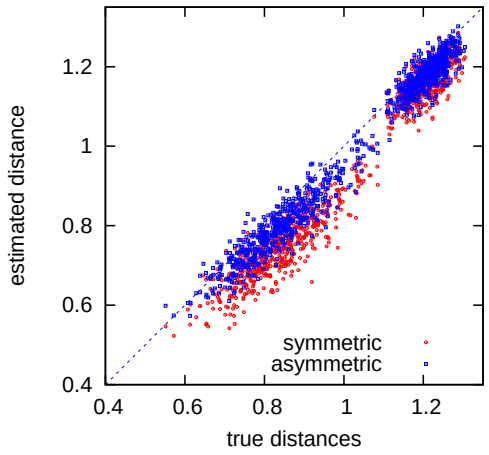

Fig. 3. Typical query of a SIFT vector in a set of 1000 vectors: comparison of the distance $d(x,y)$ obtained with the SDC and ADC estimators. We have used $m = 8$ and $k^* = 256$, i.e., 64-bit code vectors. Best viewed in color.

calculation is performed on the highest ranked vectors, as done in LSH [7], the quantization error can be used (instead of selecting an arbitrary set of elements) as a criterion to dynamically select the set of vectors on which the post-processing should be applied.

C. Estimator of the squared distance

As shown later in this subsection, using the estimations $\hat{d}$ or $\tilde{d}$ leads to underestimate, on average, the distance between descriptors. Figure 3 shows the distances obtained when querying a SIFT descriptor in a dataset of 1000 SIFT vectors. It compares the true distance against the estimates computed with Equations 12 and 13. One can clearly see the bias on these distance estimators. Unsurprisingly, the symmetric version is more sensitive to this bias.

Hereafter, we compute the expectation of the squared distance in order to cancel the bias. The approximation $q(y)$ of a given vector y is obtained, in the case of the product quantizer, from the subquantizers indexes $q_j\big(u_j(y)\big), j = 1 \dots m$. The quantization index identifies the cells $\mathcal{V}_i$ in which y lies. We can then compute the expected squared distance $\tilde{e}\big(x, q(y)\big)$ between x, which is fully known in our asymmetric distance computation method, and a random variable Y, subject to $q(Y) = q(y) = c_i$, which represents all the hypothesis on y knowing its quantization index.

$$\tilde{e}(x,y) \triangleq \mathbb{E}_Y\big[(x-Y)^2 | q(Y) = c_i\big] \tag{19}$$

$$= \int_{\mathcal{V}_i} (x-y)^2 \; p(y|i)\, dy, \tag{20}$$

$$= \frac{1}{p_i} \int_{\mathcal{V}_i} (x - c_i + c_i - y)^2 \; p(y)\, dy. \tag{21}$$

Developing the squared expression and observing, using Lloyd's condition of Equation 5, that

$$\int_{\mathcal{V}_i} (y - c_i)\, p(y)\, dy = 0, \tag{22}$$

Equation 21 simplifies to

$$\tilde{e}(x,y) = \big(x - q(y)\big)^2 + \int_{\mathcal{V}_i} (x-y)^2 \; p\big(y|q(y) = c_i\big)\, dy \tag{23}$$

$$= \tilde{d}(x,y)^2 + \xi\big(q, q(y)\big) \tag{24}$$

where we recognize the distortion $\xi\big(q, q(y)\big)$ associated with the reconstruction of y by its reproduction value.

Using the product quantizer and Equation 24, the computation of the expected squared distance between a vector x and the vector y, for which we only know the quantization indices $q_j\big(u_j(y)\big)$, consists in correcting Equation 13 as

$$\tilde{e}(x,y) = \tilde{d}(x,y)^2 + \sum_j \xi_j(y) \tag{25}$$

where the correcting term, i.e., the average distortion

$$\xi_j(y) \triangleq \xi\left(q_j, q_j\big(u_j(y)\big)\right) \tag{26}$$

associated with quantizing $u_j(y)$ to $q_j(y)$ using the j^{th} subquantizer, is learned and stored in a look-up table for all indexes of $\mathcal{I}_j$.

Performing a similar derivation for the symmetric version, i.e., when both x and y are encoded using the product quantizer, we obtain the following corrected version of the symmetric squared distance estimator:

$$\hat{e}(x,y) = \hat{d}(x,y)^2 + \sum_j \xi_j(x) + \sum_{j'} \xi_{j'}(y). \tag{27}$$

Discussion: Figure 4 illustrates the probability distribution function of the difference between the true distance and the ones estimated by Equations 13 and 25. It has been measured on a large set of SIFT descriptors. The bias of the distance estimation by Equation 13 is significantly reduced in the corrected version. However, we observe that correcting the bias leads, in this case, to a higher variance of the estimator, which is a common phenomenon in statistics. Moreover, for the nearest neighbors, the correcting term is likely to be higher

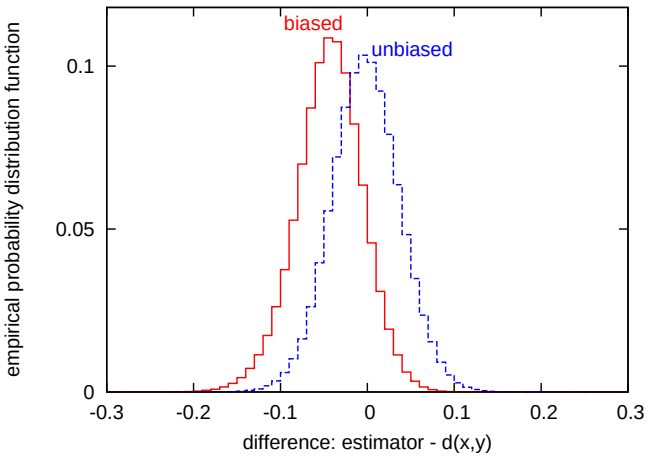

Fig. 4. PDF of the error on the distance estimation $d - \tilde{d}$ for the asymmetric method, evaluated on a set of 10000 SIFT vectors with $m = 8$ and $k^* = 256$. The bias (=-0.044) of the estimator $\tilde{d}$ is corrected (=0.002) with the error quantization term $\xi\big(q, q(y)\big)$. However, the variance of the error increases with this correction: $\sigma^2(d - \tilde{e}) = 0.00155$ whereas $\sigma^2(d - \tilde{d}) = 0.00146$.

than the measure of Equation 13, which means that we penalize the vectors with rare indexes. Note that the correcting term is independent of the query in the asymmetric version.

In our experiments, we observe that the correction returns inferior results on average. Therefore, we advocate the use of Equation 13 for the nearest neighbor search. The corrected version is useful only if we are interested in the distances themselves.

IV. Non exhaustive search

Approximate nearest neighbor search with product quantizers is fast (only m additions are required per distance calculation) and reduces significantly the memory requirements for storing the descriptors. Nevertheless, the search is exhaustive. The method remains scalable in the context of a global image description [17], [19]. However, if each image is described by a set of local descriptors, an exhaustive search is prohibitive, as we need to index billions of descriptors and to perform multiple queries [20].

To avoid exhaustive search we combine an inverted file system [26] with the asymmetric distance computation (IVFADC). An inverted file quantizes the descriptors and stores image indices in the corresponding lists, see the step "coarse quantizer" in Figure 5. This allows rapid access to a small fraction of image indices and was shown successful for very large scale search [26]. Instead of storing an image index only, we add a small code for each descriptor, as first done in [20]. Here, we encode the difference between the vector and its corresponding coarse centroid with a product quantizer, see Figure 5. This approach significantly accelerates the search at the cost of a few additional bits/bytes per descriptor. Furthermore, it slightly improves the search accuracy, as encoding the residual is more precise than encoding the vector itself.

A. Coarse quantizer, locally defined product quantizer

Similar to the "Video-Google" approach [26], a codebook is learned using k-means, producing a quantizer q_{c}, referred to as the *coarse quantizer* in the following. For SIFT descriptors, the number k' of centroids associated with q_{c} typically ranges from $k' = 1\,000$ to $k' = 1\,000\,000$. It is therefore small compared to that of the product quantizers used in Section III.

In addition to the coarse quantizer, we adopt a strategy similar to that proposed in [20], i.e., the description of a vector is refined by a code obtained with a product quantizer. However, in order to take into account the information provided by the coarse quantizer, i.e., the centroid $q_{\mathrm{c}}(y)$ associated with a vector y, the product quantizer q_{p} is used to encode the residual vector

$$r(y) = y - q_{\mathrm{c}}(y), \tag{28}$$

corresponding to the offset in the Voronoi cell. The energy of the residual vector is small compared to that of the vector itself. The vector is approximated by

$$\ddot{y} \triangleq q_{\mathrm{c}}(y) + q_{\mathrm{p}}\big(y - q_{\mathrm{c}}(y)\big). \tag{29}$$

It is represented by the tuple $\big(q_{\mathrm{c}}(y), q_{\mathrm{p}}(r(y))\big)$. By analogy with the binary representation of a value, the coarse quantizer provides the most significant bits, while the product quantizer code corresponds to the least significant bits.

The estimator of $d(x, y)$, where x is the query and y the database vector, is computed as the distance $\ddot{d}(x, y)$ between x and $\ddot{y}$:

$$\ddot{d}(x, y) = d(x, \ddot{y}) = d\Big(x - q_{\mathrm{c}}(y), q_{\mathrm{p}}\big(y - q_{\mathrm{c}}(y)\big)\Big). \tag{30}$$

Denoting by q_{p_j} the j^{th} subquantizer, we use the following decomposition to compute this estimator efficiently:

$$\ddot{d}(x, y)^2 = \sum_j d\Big(u_j\big(x - q_{\mathrm{c}}(y)\big), q_{\mathrm{p}_j}\big(u_j(y - q_{\mathrm{c}}(y))\big)\Big)^2. \tag{31}$$

Similar to the ADC strategy, for each subquantizer q_{p_j} the distances between the partial residual vector $u_j\big(x - q_{\mathrm{c}}(y)\big)$ and all the centroids $c_{j,i}$ of q_{p_j} are preliminarily computed and stored.

The product quantizer is learned on a set of residual vectors collected from a learning set. Although the vectors are quantized to different indexes by the coarse quantizer, the resulting residual vectors are used to learn a unique product quantizer. We assume that the same product quantizer is accurate when the distribution of the residual is marginalized over all the Voronoi cells. This probably gives inferior results to the approach consisting of learning and using a distinct product quantizer per Voronoi cell. However, this would be computationally expensive and would require storing k' product quantizer codebooks, i.e., $k' \times d \times k^*$ floating points values, which would be memory-intractable for common values of k'.

B. Indexing structure

We use the coarse quantizer to implement an inverted file structure as an array of lists $\mathcal{L}_1 \dots \mathcal{L}_{k'}$. If $\mathcal{Y}$ is the vector dataset to index, the list $\mathcal{L}_i$ associated with the centroid c_i of q_c stores the set $\{y \in \mathcal{Y} : q_\mathrm{c}(y) = c_i\}$.

In inverted list $\mathcal{L}_i$, an entry corresponding to y contains a vector identifier and the encoded residual $q_\mathrm{p}(r(y))$:

field	length (bits)
identifier	8–32
code	$m\lceil \log_2 k^* \rceil$

The identifier field is the overhead due to the inverted file structure. Depending on the nature of the vectors to be stored, the identifier is not necessarily unique. For instance, to describe images by local descriptors, image identifiers can replace vector identifiers, i.e., all vectors of the same image have the same identifier. Therefore, a 20-bit field is sufficient to identify an image from a dataset of one million. This memory cost can be reduced further using index compression [27], [28], which may reduce the average cost of storing the identifier to about 8 bits, depending on parameters[2]. Note that some geometrical information can also be inserted in this entry, as proposed in [20] and [27].

C. Search algorithm

The inverted file is the key to the non-exhaustive version of our method. When searching the nearest neighbors of a vector x, the inverted file provides a subset of $\mathcal{Y}$ for which distances are estimated: only the inverted list $\mathcal{L}_i$ corresponding to $q_\mathrm{c}(x)$ is scanned.

However, x and its nearest neighbor are often not quantized to the same centroid, but to nearby ones. To address this problem, we use the multiple assignment strategy of [29]. The query x is assigned to w indexes instead of only one, which correspond to the w nearest neighbors of x in the codebook of q_c. All the corresponding inverted lists are scanned. Multiple assignment is not applied to database vectors, as this would increase the memory usage.

[2]An average cost of 11 bits is reported in [27] using delta encoding and Huffman codes.

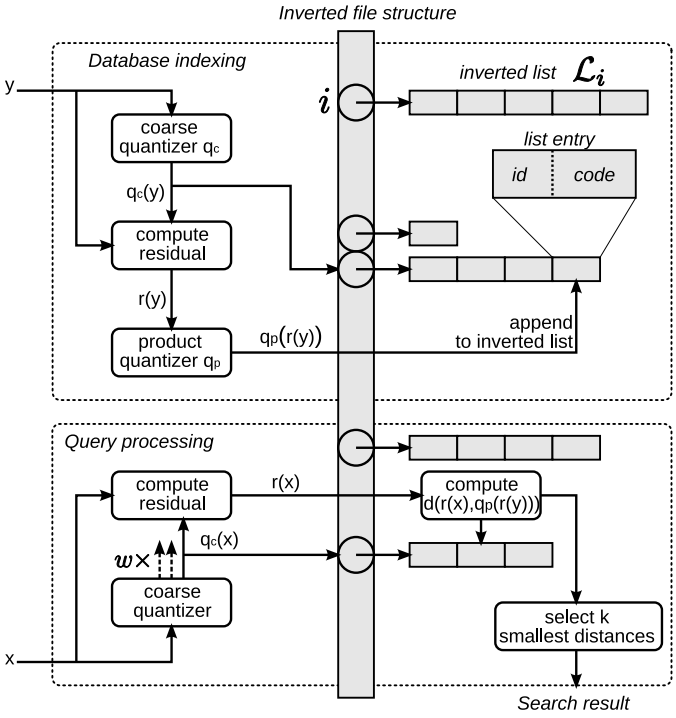

Fig. 5. Overview of the *inverted file with asymmetric distance computation* (IVFADC) indexing system. *Top*: insertion of a vector. *Bottom*: search.

Figure 5 gives an overview of how a database is indexed and searched.

Indexing a vector y proceeds as follows:

1) quantize y to $q_\mathrm{c}(y)$
2) compute the residual $r(y) = y - q_\mathrm{c}(y)$
3) quantize $r(y)$ to $q_\mathrm{p}(r(y))$, which, for the product quantizer, amounts to assigning $u_j(y)$ to $q_j(u_j(y))$, for $j = 1 \dots m$.
4) add a new entry to the inverted list corresponding to $q_\mathrm{c}(y)$. It contains the vector (or image) identifier and the binary code (the product quantizer's indexes).

Searching the nearest neighbor(s) of a query x consists of

1) quantize x to its w nearest neighbors in the codebook q_c;

   For the sake of presentation, in the two next steps we simply denote by $r(x)$ the residuals associated

with these w assignments. The two steps are applied to all w assignments.

2) compute the squared distance $d\big(u_j(r(x)), c_{j,i}\big)^2$ for each subquantizer j and each of its centroids $c_{j,i}$;
3) compute the squared distance between $r(x)$ and all the indexed vectors of the inverted list. Using the subvector-to-centroid distances computed in the previous step, this consists in summing up m looked-up values, see Equation 31;
4) select the K nearest neighbors of x based on the estimated distances. This is implemented efficiently by maintaining a Maxheap structure of fixed capacity, that stores the K smallest values seen so far. After each distance calculation, the point identifier is added to the structure only if its distance is below the largest distance in the Maxheap.

Only Step 3 depends on the database size. Compared with ADC, the additional step of quantizing x to $q_c(x)$ consists in computing k' distances between D-dimensional vectors. Assuming that the inverted lists are balanced, about $n \times w/k'$ entries have to be parsed. Therefore, the search is significantly faster than ADC, as shown in the next section.

V. Evaluation of NN search

In this section, we first present the datasets used for the evaluation[3]. We then analyze the impact of the parameters for SDC, ADC and IVFADC. Our approach is compared to three state-of-the-art methods: spectral hashing [19], Hamming embedding [20] and FLANN [9]. Finally, we evaluate the complexity and speed of our approach.

A. Datasets

We perform our experiments on two datasets, one with local SIFT descriptors [23] and the other with global color GIST descriptors [18]. We have three vector subsets per dataset: learning, database and query. Both datasets were constructed using publicly available data and software. For the SIFT descriptors, the learning set is extracted from Flickr images and the database and query descriptors are from the INRIA Holidays images [20]. For GIST, the learning set consists of the first 100k images extracted from the tiny image set of [16]. The database set is the Holidays image set combined with Flickr1M used in [20]. The query vectors are from the Holidays image queries. Table III summarizes the number of descriptors extracted for the two datasets.

vector dataset:	SIFT	GIST
descriptor dimensionality d	128	960
learning set size	100,000	100,000
database set size	1,000,000	1,000,991
queries set size	10,000	500

TABLE III
SUMMARY OF THE SIFT AND GIST DATASETS.

The search quality is measured with recall@R, i.e., the proportion of query vectors for which the nearest neighbor is ranked in the first R positions. This measure indicates the fraction of queries for which the nearest neighbor is retrieved correctly, if a short-list of R vectors is verified using Euclidean distances. Furthermore, the curve obtained by varying R corresponds to the distribution function of the ranks, and the point R=1 corresponds to the "precision" measure used in [9] to evaluate ANN methods.

In practice, we are often interested in retrieving the K nearest neighbors ($K > 1$) and not only the nearest neighbor. We do not include these measures in the paper, as we observed that the conclusions for K=1 remain valid for $K > 1$.

B. *Memory* vs *search accuracy: trade-offs*

The product quantizer is parametrized by the number of subvectors m and the number of quantizers per subvector k^*, producing a code of length $m \times \log_2 k^*$. Figure 6 shows the trade-off between code length and search quality for our SIFT descriptor dataset. The quality is measured for recall@100 for the ADC and SDC estimators, for $m \in \{1, 2, 4, 8, 16\}$ and $k^* \in \{2^4, 2^6, 2^8, 2^{10}, 2^{12}\}$. As for the quantizer distortion in Figure 1, one can observe that for a fixed number of bits, it is better to use a small number of subquantizers with many centroids than to have many subquantizers with few bits. However, the comparison also reveals that MSE underestimates, for a fixed number of bits, the quality obtained for a large number of subquantizers against using more centroids per quantizer.

As expected, the asymmetric estimator ADC significantly outperforms SDC. For m=8 we obtain the same accuracy for ADC and k^*=64 as for SDC and k^*=256. Given that the efficiency of the two approaches is equivalent, we advocate not to quantize the query when possible, but only the database elements.

[3]Both the software and the data used in these experiments are available at http://www.irisa.fr/texmex/people/jegou/ann.php.

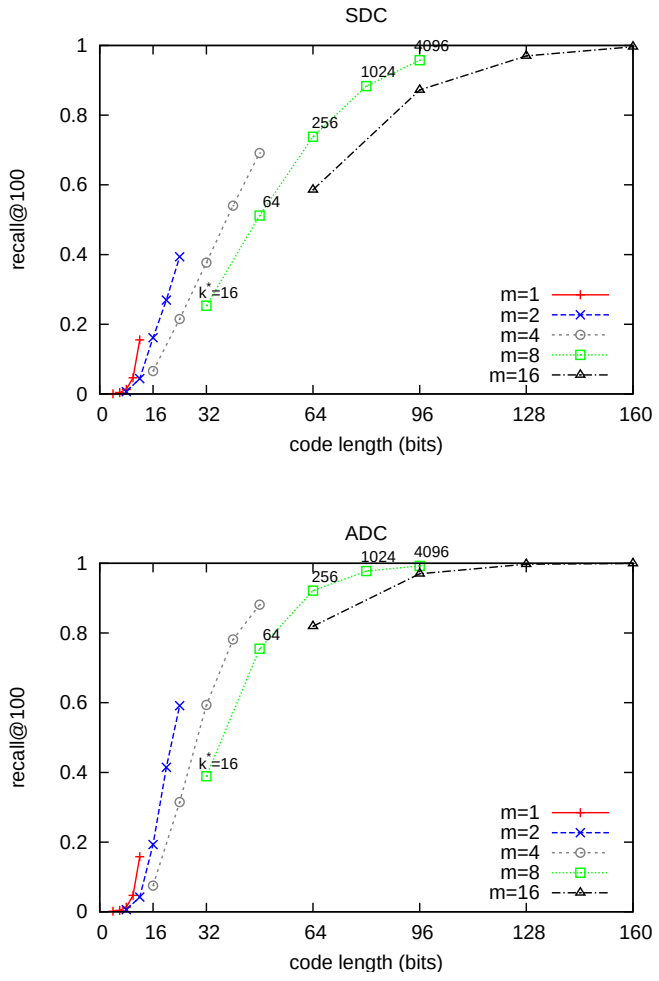

Fig. 6. SDC and ADC estimators evaluated on the SIFT dataset: recall@100 as a function of the memory usage (code length=$m \times \log_2 k^*$) for different parameters (k^*=16,64,256,...,4096 and m=1,2,4,8,16). The missing point (m=16,k^*=4096) gives recall@100=1 for both SDC and ADC.

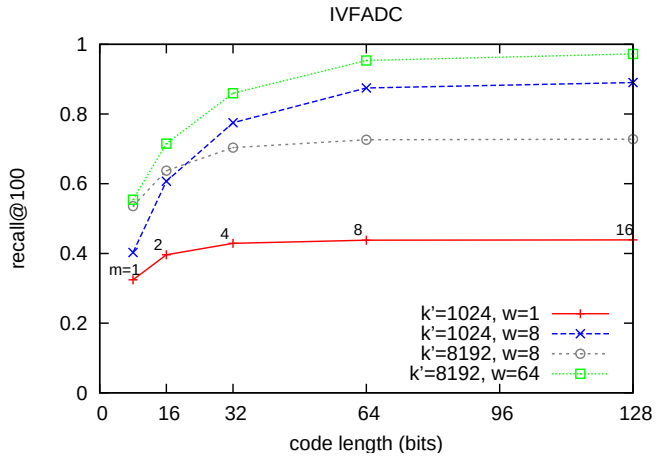

Fig. 7. SIFT dataset: recall@100 for the IVFADC approach as a function of the memory usage for k^*=256 and varying values of $m = \{1, 2, 4, 8, 16\}$, $k' = \{1024, 8192\}$ and $w = \{1, 8, 64\}$.

Figure 7 evaluates the impact of the parameters for the IVFADC method introduced in Section IV. For this approach, we have, in addition, to set the codebook size k' and the number of neighboring cells w visited during the multiple assignment. We observe that the recall@100 strongly depends on these parameters, and that increasing the code length is useless if w is not big enough, as the nearest neighbors which are not assigned to one of the w centroids associated with the query are definitely lost.

This approach is significantly more efficient than SDC and ADC on large datasets, as it only compares the query to a small fraction of the database vectors. The proportion of the dataset to visit is roughly linear in w/k'. For a fixed proportion, it is worth using higher values of k', as this increases the accuracy, as shown by comparing, for the tuple (m, w), the parameters $(1024, 1)$ against $(8192, 8)$ and $(1024, 8)$ against $(8192, 64)$.

C. Impact of the component grouping

The product quantizer defined in Section II creates the subvectors by splitting the input vector according to the order of the components. However, vectors such as SIFT and GIST descriptors are structured because they are built as concatenated orientation histograms. Each histogram is computed on grid cells of an image patch. Using a product quantizer, the bins of a histogram may end up in different quantization groups.

The *natural order* corresponds to grouping consecutive components, as proposed in Equation 8. For the SIFT descriptor, this means that histograms of neighboring grid cells are quantized together. GIST descriptors are composed of three 320-dimension blocks, one per color channel. The product quantizer splits these blocks into parts.

	SIFT		GIST
m	4	8	8
natural	0.593	0.921	0.338
random	0.501	0.859	0.286
structured	0.640	0.905	0.652

TABLE IV
IMPACT OF THE DIMENSION GROUPING ON THE RETRIEVAL PERFORMANCE OF ADC (RECALL@100, k^*=256).

To evaluate the influence of the grouping, we modify the u_j operators in Equation 8, and measure the impact of their construction on the performance of the ADC method. Table IV shows the effect on the search quality,

measured by recall@100. The analysis is restricted to the parameters k^*=256 and $m \in \{4, 8\}$.

Overall, the choice of the components appears to have a significant impact of the results. Using a random order instead of the natural order leads to poor results. This is true even for GIST, for which the natural order is somewhat arbitrary.

The "structured" order consists in grouping together dimensions that are related. For the $m = 4$ SIFT quantizer, this means that the 4×4 patch cells that make up the descriptor [23] are grouped into 4 2×2 blocks. For the other two, it groups together dimensions that have have the same index modulo 8. The orientation histograms of SIFT and most of GIST's have 8 bins, so this ordering quantizes together bins corresponding to the same orientation. On SIFT descriptors, this is a slightly less efficient structure, probably because the natural order corresponds to spatially related components. On GIST, this choice significantly improves the performance. Therefore, we use this ordering in the following experiments.

Discussion: A method that automatically groups the components could further improve the results. This seems particularly important if we have no prior knowledge about the relationship between the components as in the case of bag-of-features. A possible solution is the minimum sum-squared residue co-clustering [30] algorithm.

D. Comparison with the state of the art

Comparison with Hamming embedding methods: We compare our approach to spectral hashing (SH) [19], which maps vectors to binary signatures. The search consists in comparing the Hamming distances between the database signatures and the query vector signature. This approach was shown to outperform the restricted Boltzmann machine of [17]. We have used the publicly available code. We also compare to the Hamming embedding (HE) method of [20], which also maps vectors to binary signatures. Similar to IVFADC, HE uses an inverted file, which avoids comparing to all the database elements.

Figures 8 and 9 show, respectively for the SIFT and the GIST datasets, the rank repartition of the nearest neighbors when using a signature of size 64 bits. For our product quantizer we have used $m = 8$ and $k^* = 256$, which give similar results in terms of run time. All our approaches significantly outperform spectral hashing[4] on the two datasets. To achieve the same recall as spectral hashing, ADC returns an order of magnitude less vectors.

The best results are obtained by IVFADC, which for low ranks provides an improvement over ADC, and significantly outperforms spectral hashing. This strategy avoids the exhaustive search and is therefore much faster, as discussed in the next subsection. This partial scan explains why the IVFADC and HE curves stop at some point, as only a fraction of the database vectors are ranked. Comparing these two approaches, HE is significantly outperformed by IVFADC. The results of HE are similar to spectral hashing, but HE is more efficient.

[4]In defense of [17], [19], which can be learned for arbitrary distance measures, our approach is adapted to the Euclidean distance only.

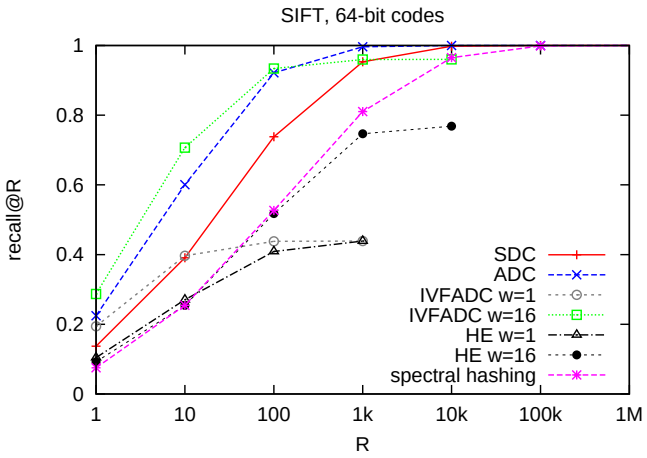

Fig. 8. SIFT dataset: recall@R for varying values of R. Comparison of the different approaches SDC, ADC, IVFADC, spectral hashing [19] and HE [20]. We have used m=8, k^*=256 for SDC/ADC and k'=1024 for HE [20] and IVFADC.

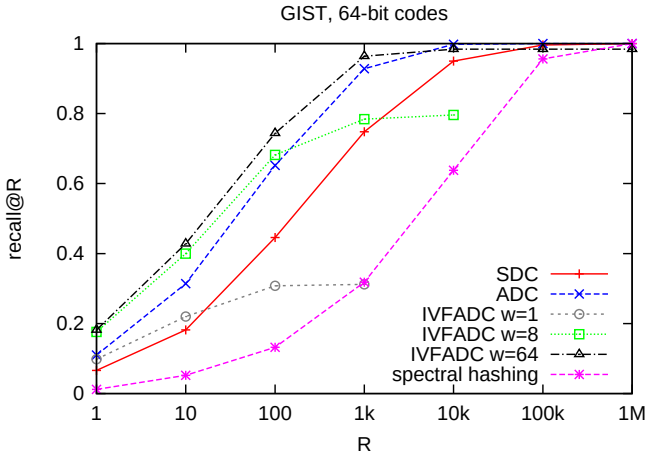

Fig. 9. GIST dataset: recall@R for varying values of R. Comparison of the different approaches SDC, ADC, IVFADC and spectral hashing [19]. We have used m=8, k^*=256 for SDC/ADC and $k' = 1\,024$ for IVFADC.

Comparison with FLANN: The approximate nearest-neighbor search technique of Muja & Lowe [9] is based on hierarchical structures (KD-trees and hierarchical k-means trees). The software package FLANN automatically selects the best algorithm and parameters for a given dataset. In contrast with our method and spectral hashing, all vectors need to remain in RAM as the method includes a re-ranking stage that computes the real distances for the candidate nearest neighbors.

The evaluation is performed on the SIFT dataset by measuring the 1-recall@1, that is, the average proportion of true NNs ranked first in the returned vectors. This measure is referred to as *precision* in [9].

For the sake of comparison with FLANN, we added a verification stage to our IVFADC method: IVFADC queries return a shortlist of R candidate nearest neighbors using the distance estimation. The vectors in the shortlist are re-ordered using the real distance, as done in [7], [9], and the closest one is returned. Note that, in this experimental setup, all the vectors are stored in main memory. This requirement seriously limits the scale on which re-ordering can be used.

The IVFADC and FLANN methods are both evaluated at different operating points with respect to precision and search time. For FLANN, the different operating points are obtained with parameters generated automatically for various target precisions. For IVFADC, they are obtained by varying the number k' of coarse centroids, the number w of assignments and the short-list size R. The product quantizer is generated using k^*=256 and m=8, i.e., 64-bit codes. This choice is probably not optimal for all operating points.

Figure 10 shows that the results obtained by IVFADC are better than those of FLANN for a large range of operating points. Moreover, our method has a much smaller memory footprint than FLANN: the indexing structure occupies less than 25 MB, while FLANN requires more than 250 MB of RAM. Note, however, that both are negligible compared to the memory occupied by the vectors in the case of large datasets. On such a scale, the re-ranking stage is not feasible and only memory-aware approaches (HE, SH and our methods) can be used.

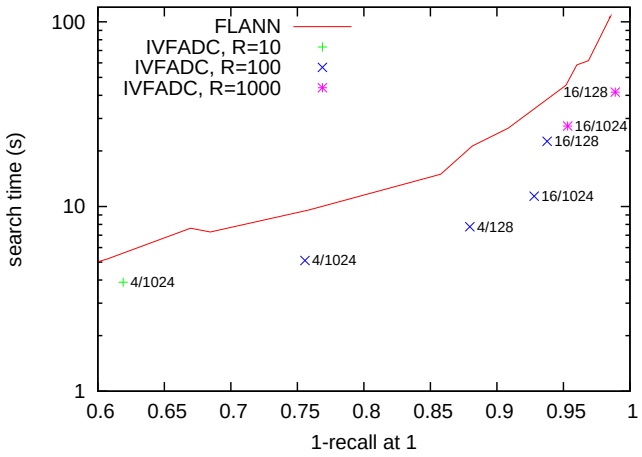

Fig. 10. IVFADC *vs* FLANN: trade-offs between search quality (1-recall@1) and search time. The IVFADC method is parametrized by the shortlist size R used for re-ranking the vector with the L2 distance, and the two parameters w and k' of the inverted file, which correspond to the number of assignments and to the number of coarse centroids.

E. Complexity and speed

Table V reports the search time of our methods. For reference, we report the results obtained with the spectral hashing algorithm of [19] on the same dataset and machine (using only one core). Since we use a separate learning set, we use the out-of-sample evaluation of this algorithm. Note that for SH we have re-implemented the Hamming distance computation in C in order to have the approaches similarly optimized. The algorithms SDC, ADC and SH have similar run times. IVFADC significantly improves the performance by avoiding an exhaustive search. Higher values of k' yield higher search efficiencies for large datasets, as the search benefits from parsing a smaller fraction of the memory. However, for small datasets, the complexity of the coarse quantizer may be the bottleneck if $k' \times D > n/k'$ when using a exhaustive assignment for q_c. In that case the ADC variant may be preferred. For large datasets and using an efficient assignment strategy for the coarse quantizer, higher values of k' generally lead to better efficiency, as first shown in [15]. In this work, the authors propose a hierarchical quantizer to efficiently assign descriptors to one million centroids.

F. Large-scale experiments

To evaluate the search efficiency of the product quantizer method on larger datasets we extracted about 2 billion SIFT descriptors from one million images. Search is performed with 30 000 query descriptors from ten images. We compared the IVFADC and HE methods with similar parameters. In particular, the amount of memory that is scanned for each method and the cost of the coarse quantization are the same.

The query times per descriptor are shown on Figure 11. The cost of the extra quantization step required by IVFADC appears clearly for small database sizes. For larger scales, the distance computation with the database vectors become preponderant. The processing that is applied to each element of the inverted lists is

method	parameters	search time (ms)	average number of code comparisons	recall@100
SDC		16.8	1 000 991	0.446
ADC		17.2	1 000 991	0.652
IVFADC	k'= 1 024, w=1	1.5	1 947	0.308
	k'= 1 024, w=8	8.8	27 818	0.682
	k'= 1 024, w=64	65.9	101 158	0.744
	k'= 8 192, w=1	3.8	361	0.240
	k'= 8 192, w=8	10.2	2 709	0.516
	k'= 8 192, w=64	65.3	19 101	0.610
SH		22.7	1 000 991	0.132

TABLE V

GIST DATASET (500 QUERIES): SEARCH TIMINGS FOR 64-BIT CODES AND DIFFERENT METHODS. WE HAVE USED m=8 AND k^*=256 FOR SDC, ADC AND IVFADC.

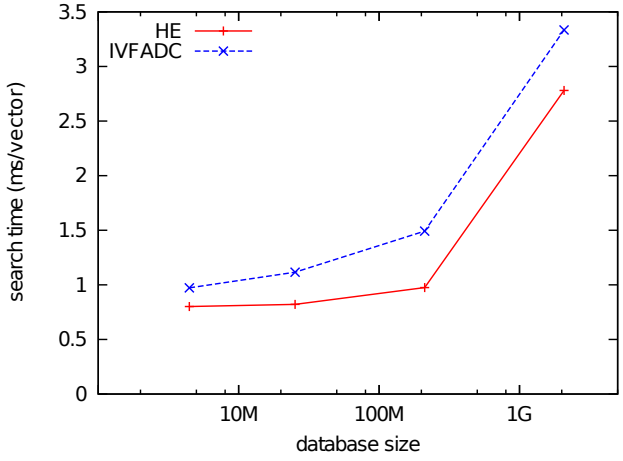

Fig. 11. Search times for SIFT descriptors in datasets of increasing sizes, with two search methods. Both use the same 20 000-word codebook, $w = 1$, and 64-bit signatures.

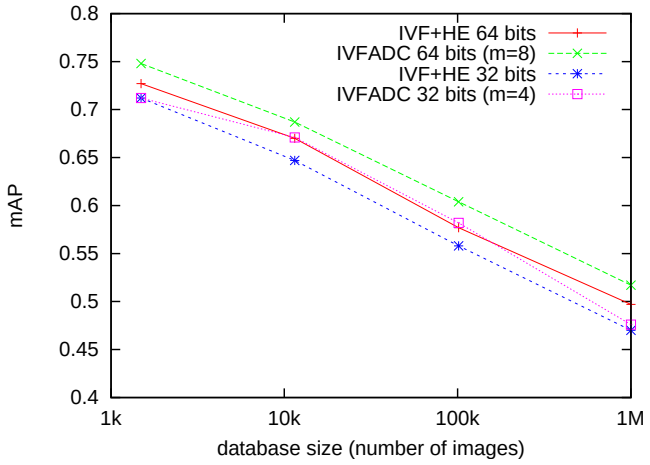

Fig. 12. Comparison of IVFADC and the Hamming Embedding method of [20]. mAP for the Holidays dataset as function of the number of distractor images (up to 1 million).

approximately as expensive in both cases. For HE, it is a Hamming distance computation, implemented as 8 table lookups. For IVFADC it is a distance computation that is also performed by 8 table lookups. Interestingly, the floating point operations involved in IVFADC are not much more expensive than the simple binary operations of HE.

G. Image search

We have evaluated our method within an image search system based on local descriptors. For this evaluation, we compare our method with the HE method of [20] on the INRIA Holidays dataset, using the pre-processed set of descriptors available online. The comparison is focused on large scale indexing, i.e., we do not consider the impact of a post-verification step [23], [31] or geometrical information [20].

Figure 12 shows the search performance in terms of mean average precision as a function of the size of the dataset. We have used the same coarse quantizer (k'=20,000) and a single assignment strategy (w=1) for both the approaches, and fixed k^*=256 for IVFADC. For a given number of bits (32 or 64), we have selected the best choice of the Hamming threshold for HE. Similarly, we have adjusted the number of nearest neighbors to be retrieved for IVFADC.

One can observe that the gain obtained by IVFADC is significant. For example, for one million distractors, the mAP of 0.497 reported in [20] with 64-bit signatures is improved to 0.517.

VI. Conclusion

We have introduced product quantization for approximate nearest neighbor search. Our compact coding scheme provides an accurate approximation of the Euclidean distance. Moreover, it is combined with an inverted file system to avoid exhaustive search, resulting in high efficiency. Our approach significantly outperforms the state of the art in terms of the trade-off between search quality and memory usage. Experimental results for SIFT and GIST image descriptors are excellent and show that grouping the components based on our prior knowledge of the descriptor design further improves the results. The scalability of our approach is validated on a dataset of two billion vectors.

References

[1] H. Jégou, M. Douze, and C. Schmid, "Searching with quantization: approximate nearest neighbor search using short codes and distance estimators," Tech. Rep. RR-7020, INRIA, August 2009.

[2] H. Sandhawalia and H. Jégou, "Searching with expectations," in *IEEE International Conference on Acoustics Speech and Signal Processing*, Signal Processing, pp. 1242–1245, IEEE, March 2010.

[3] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?," in *Proceedings of the International Conference on Database Theory*, pp. 217–235, August 1999.

[4] C. Böhm, S. Berchtold, and D. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, vol. 33, pp. 322–373, October 2001.

[5] J. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transaction on Mathematical Software*, vol. 3, no. 3, pp. 209–226, 1977.

[6] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proceedings of the International Conference on Very Large DataBases*, pp. 194–205, 1998.

[7] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the Symposium on Computational Geometry*, pp. 253–262, 2004.

[8] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimension via hashing," in *Proceedings of the International Conference on Very Large DataBases*, pp. 518–529, 1999.

[9] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proceedings of the International Conference on Computer Vision Theory and Applications*, 2009.

[10] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Proceedings of the International Conference on Computer Vision*, October 2009.

[11] G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*, ch. 3. MIT Press, March 2006.

[12] Y. Ke, R. Sukthankar, and L. Huston, "Efficient near-duplicate detection and sub-image retrieval," in *ACM International conference on Multimedia*, pp. 869–876, 2004.

[13] B. Matei, Y. Shan, H. Sawhney, Y. Tan, R. Kumar, D. Huber, and M. Hebert, "Rapid object indexing using locality sensitive hashing and joint 3D-signature space estimation," IEEE *Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, pp. 1111 – 1126, July 2006.

[14] C. Silpa-Anan and R. Hartley, "Optimized kd-trees for fast image descriptor matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[15] D. Nistér and H. Stewénius, "Scalable recognition with a vocabulary tree," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2161–2168, 2006.

[16] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: a large database for non-parametric object and scene recognition," IEEE *Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, pp. 1958–1970, November 2008.

[17] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large databases for recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[18] A. Oliva and A. Torralba, "Modeling the shape of the scene: a holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[19] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems*, 2008.

[20] H. Jégou, 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*, October 2008.

[21] M. Douze, H. Jégou, H. Singh, L. Amsaleg, and C. Schmid, "Evaluation of GIST descriptors for web-scale image search," in *Proceedings of the International Conference on Image and Video Retrieval*, 2009.

[22] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Lost in quantization: Improving particular object retrieval in large scale image databases," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[23] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[24] R. M. Gray and D. L. Neuhoff, "Quantization," IEEE *Transactions on Information Theory*, vol. 44, pp. 2325–2384, Oct. 1998.

[25] D. E. Knuth, *The Art of Computer Programming, Sorting and Searching*, vol. 3. Addison Wesley, 2 ed., 1998.

[26] 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*, pp. 1470–1477, 2003.

[27] M. Perdoch, O. Chum, and J. Matas, "Efficient representation of local geometry for large scale object retrieval," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2009.

[28] H. Jégou, M. Douze, and C. Schmid, "Packing bag-of-features," in *Proceedings of the International Conference on Computer Vision*, September 2009.

[29] H. Jégou, H. Harzallah, and C. Schmid, "A contextual dissimilarity measure for accurate and efficient image search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[30] H. Cho, I. Dhillon, Y. Guan, and S. Sra, "Minimum sum-squared residue co-clustering of gene expression data," in *SIAM International Conference on Data Mining*, pp. 114–125, April 2004.

[31] 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*, 2007.