Searching with quantization: approximate nearest neighbor search using short codes and distance estimators

Hervé Jégou, Matthijs Douze, Cordelia Schmid

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Searching with quantization: approximate nearest neighbor search using short codes and distance estimators

Hervé Jégou — Matthijs Douze — Cordelia Schmid

N° 7020

August 2009

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--7020--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

Searching with quantization: approximate nearest neighbor search using short codes and distance estimators

Hervé Jégou*, Matthijs Douze*, Cordelia Schmid*

Thème COG — Systèmes cognitifs
Équipe-Projet Lear

Rapport de recherche n° 7020 — August 2009 — 25 pages

Abstract: We propose an approximate nearest neighbor search method based on quantization. It uses, in particular, product quantizer to produce short codes and corresponding distance estimators approximating the Euclidean distance between the orginal vectors. The method is advantageously used in an asymmetric manner, by computing the distance between a vector and code, unlike competing techniques such as spectral hashing that only compare codes.

Our approach approximates the Euclidean distance based on memory efficient codes and, thus, permits efficient nearest neighbor search. Experiments performed on SIFT and GIST image descriptors show excellent search accuracy. The method is shown to outperform two state-of-the-art approaches of the literature. Timings measured when searching a vector set of 2 billion vectors are shown to be excellent given the high accuracy of the method.

Key-words: nearest neighbor search, large databases, quantization

This technical report is in submission.

* firstname.lastname@inria.fr

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Quantifier pour chercher: recherche approximative par codes compacts et estimateurs de distances

Résumé : Nous proposons une méthode de recherche appproximative qui permet d'estimer la distance entre deux vecteurs en utilisant des codes courts quantifiés. Ces codes sont définis de manière conjointe avec leur estimateurs, qui approximent la distance euclidienne entre deux vecteurs. La méthode permet d'estimer la distance entre deux vecteurs à partir de leur codes respectifs. Contrairement aux techniques concurrentes, elle peut également être utilisée de manière asymétrique avec un estimateur de distance qui prend en entrée un vecteur et un code, ce qui améliore la qualité de l'estimation.

Nous montrons que notre approche offre des résultats qui sont significativement au dessus de ceux de l'état de l'art en terme du compromis entre usage mémoire et qualité de la recherche. Les temps de recherche mesurés sur une base de vecteurs de 2 milliards de vecteurs SIFT montrent l'intérêt de notre méthode en pratique.

Mots-clés : recherche de plus proches voisins, grandes bases de données, distance euclidienne, quantification

1 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* [1, 2]. Focusing on the D-dimensional Euclidean space $\mathbb{R}^D$, the problem is to find the element $\mathrm{NN}(x)$ in a finite set $\mathcal{Y} \subset \mathbb{R}^D$ minimizing the distance to the query vector $x \in \mathbb{R}^D$:

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

Several multi-dimensional indexing methods, such as the popular KD-tree [3] or branch and bound techniques, have been proposed to reduce the search time. However, for many dimensions it turns out [4] that such approaches are not more efficient than the brute-force exhaustive distance calculation, whose complexity is $\mathcal{O}(nD)$.

There is a large literature [5, 6, 7] 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 [8]. In this paper, we will only consider the Euclidean distance, which is relevant for many applications. In that case, one of the most popular ANN algorithm is the Euclidean Locality-Sensitive Hashing (E2LSH) [5, 9], which provides theoretical guarantees on the search quality with limited assumptions. It has been successfully used for local descriptors [10] and 3D object indexing [11, 9]. However, for real data, LSH is outperformed by heuristic methods [7], which exploit the distribution of the vectors better.

ANN algorithms are typically compared 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, i.e., several hundreds of bytes. Only recently, researchers have tried to design methods limiting the memory usage. This is a key criterion for problems involving large amounts of data [12], for instance in large-scale scene recognition problems [13], where millions to billions of images have to be indexed. In [13], Torralba et al. represent an image by a single global GIST descriptor [14] 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 based on the Hamming distance. In [15], spectral hashing (SH) is shown to outperform the binary codes generated by the restricted Boltzmann machine [13], boosting and LSH. Another related work is the Hamming embedding method of Jegou *et al.* [16], where the binary signature is used to refine quantized SIFT 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, only the database vectors are assigned to codes. This reduces the quantization noise and subsequently improves the search quality. This method

requires that the codebook provides a low quantization error. To obtain such a precise representation, the total number k of centroids should be high enough, e.g., $k = 2^{64}$ for codes of 64-bits. This raises several issues on how to learn the codebook and assign a vector. First, the amount of samples required to learn the quantizer should be several times k. Second, the complexity of the algorithm itself is by many orders of magnitude too large. Finally, the amount of computer memory available on earth is not sufficient to store the floating points values representing the centroids.

The hierarchical k-means (HKM) was proposed as a way of improving the efficiency the learning stage and of the corresponding assignment procedure, see the numerous references in [17] and see [18] for an application in computer vision. However, the aforementioned limitations still apply, in particular those on the memory usage and the size of the learning set. One could also consider the use of scalar quantizers. However, 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 are significantly inferior to k-means in indexing tasks [19]. In this paper, we will focus on product quantizers. To our knowledge, such a semi-structured quantizer has never been considered in any nearest neighbor search method, and is the only one that fulfills the requirements of our search algorithm.

The advantages of our method are twofold. First, the number of possible distances is significantly higher than for the competing embedding methods [16, 13, 15], for which this number is equal to the signature length plus one because it is a Hamming distance. Second, as a byproduct of the method, we get an estimation of the expected squared distance, which is interesting for ε-radius search or to use Lowe's distance ratio criterion [20]. The motivation of using the Hamming space in [16, 13, 15] is the efficient computation of distances. Note, however, that the fastest way of computing Hamming distances consists of using table lookups. Our method is implemented using such a strategy and provides comparable efficiency.

An exhaustive comparison with all codes representing the vectors is prohibitive in the context of 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, similar to [16]. The difference is that we use the codes computed by our product quantizer to encode the residual vector with respect to the cluster center.

A comparison with the state-of-the-art shows that our approach significantly outperforms existing techniques, in particular spectral hashing [15] and Hamming embedding [16].

Our paper is organized as follows. In Section 2 we introduce the notations for quantization as well as the product quantizer used by our method. Section 3 presents our approach for NN search and Section 4 introduces the structure used to avoid exhaustive search. An evaluation of the parameters of our approach and a comparison to the state-of-art is finally given in Section 5.

2 Background: quantization, product quantizer

A large literature is available on vector quantization, see [17] for a survey. In this section, we restrict our presentation to the notations and concepts used in the rest of this paper.

2.1 Vector quantization

Quantization is a destructive process which has been extensively studied in information theory [17]. 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*. Denoting by $k = |\mathcal{C}|$ its cardinality, we assume without loss of generalization that the indexes are consecutive integers ranging from 0 to $k-1$.

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 square error between the input vector x and its reproduction value $q(x)$:

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

where $d(x, y) = ||x - y||$ is the Euclidean distance between x and y, and where $p(x)$ is the probability distribution function corresponding to a generic 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, as

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

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:

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

The Lloyd quantizer, which corresponds to the k-means clustering algorithm, finds the optimal solution by iteratively assigning the vectors of a training set to centroids and re-estimating these centroids from the assigned points. In the following, we assume that the two Lloyd necessary conditions hold, as we learn the

quantizer using k-means. Note, however, that k-means does not necessarily find the global optimum, but a local one satisfying the aforementioned conditions.

Another interesting quantity that will be used afterward 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

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

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 usually stored in a binary memory.

2.2 Product quantizers

Let us consider a 128-dimensional vector, for example the SIFT descriptors [20]. 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. The number of samples and the learning complexity required to learn the quantizer should be several times k. Furthermore, it is impossible to store the $D \times k$ floating point values representing the k centroids.

Product quantizers are an efficient solution to address all these issues. 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 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

	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	$k^{1/m}\,D$	$k^{1/m}\,D$

Table 1: Memory usage of the codebook and assignment complexity for different quantizers. HKM is parameterized by tree height l and the branching factor b_f.

quantizer, where the quantization function associated with each component may be different from one component to another.

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 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^*.

A centroid of the product quantizer is obtained by concatenating m subquantizer centroids. Storing this codebook $\mathcal{C}$ explicitly is not efficient. Instead, we store the $m \times k^*$ centroids of all the subquantizers, i.e, $m\,D^*\,k^* = k^*\,D$ floating points values. Quantizing an element requires k^*D floating point operations. Table 1 summarizes the resource requirements associated with k-means, HKM and product k-means. The product quantizer is clearly the only quantizer that can be reasonably 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

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

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 5 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^* increases the computational cost of the quantizer, as shown by Table 1. This also increases the memory usage of storing the centroids ($k^* \times D$ floating point values), which by itself further reduces the efficiency if the centroid look-up table does not fit in cache memory anymore. In the case where $m = 1$, we can not afford using more than 16 bits to keep this cost tractable. Using $k^* = 256$ and $m = 8$ seems a reasonable choice.

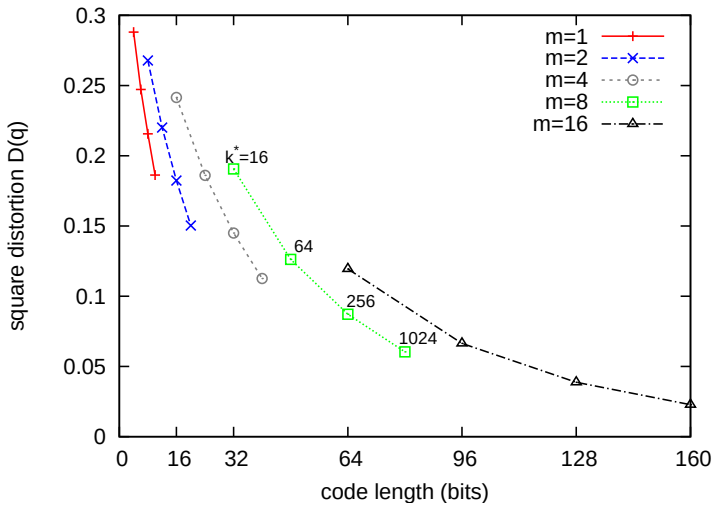
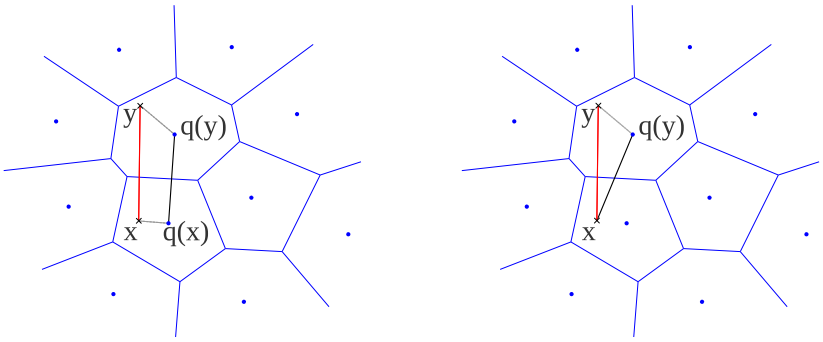


Figure 1: SIFT: quantization error associated the parameters m and k^*.



symmetric case asymmetric case

Figure 2: Principle of our method: the distance $d(x, y)$ is estimated using the distance $d(x, q(y))$. The mean squared error on the distance is bounded, on average, by the quantization error.

3 Searching with quantization

Nearest neighbor search depends solely 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. 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.

3.1 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 $d(x, y_i)$ between these vectors, a symmetric and a asymmetric one. See Figure 2 for an illustration.

	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	$n + k \log n$	$n + k \log n$

Table 2: Algorithm and computational costs associated with searching the k nearest neighbors using the product quantizer for symmetric and asymmetric distance computations (SDC, ADC).

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 as

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

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

Asymmetric distance computation (ADC): a given 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 square 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 root in practice: the square root function is monotonously increasing and the square distances produces the same vector ranking.

Table 2 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 worst case complexity [21]. For $n \gg k$ and when the elements are arbitrarily ordered, this complexity is overestimated (the behavior is closer to linear) and the search bottleneck is the distance calculation step.

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

The only advantage of SDC over ADC is to limit the memory usage associated with the queries, as in that case the query vector is completely defined by a code. In most cases, one should prefer the asymmetric version, which obtains a lower distance distortion for a similar complexity. We will focus on ADC in the rest of this section.

3.2 Analysis of the distance error

In this subsection, we analyze the error affecting the distance 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 2. The analysis is similar for the symmetric version.

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

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

where it is reasonable to assume that the joint probability distribution function $p(x,y) = p(x)\,p(y)$ is separable.

Given the triangular inequality, we have

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

and, equivalently,

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

Combining this inequality with Equation 14, we obtain

$$\begin{aligned}\mathrm{MSDE}(q) &\leq \int p(x) \left(\int d\big(y,q(y)\big)^2 \, p(y)\, dy \right) dx & (17)\\ &\leq \mathrm{MSE}(q). & (18)\end{aligned}$$

where $\mathrm{MSE}(q)$ is the mean squared error associated with quantizer q. 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 $2\times\mathrm{MSE}(q)$. It is, therefore, worth minimizing the quantization error, as this criterion provides a statistical guarantee on the error altering the distance. If an exact distance calculation is performed on the first vector, as done in LSH [5], the quantization error can be used (instead of selecting an arbitrary set of k elements) as a criterion to dynamically select the set of vectors on which the post-processing should be applied.

3.3 Estimator of the squared distance

As shown later in this subsection, using the estimations $d\big(q(x),q(y)\big)$ or $d\big(x,q(y)\big)$ leads to underestimate, on average, the distance between points. Figure 3 shows

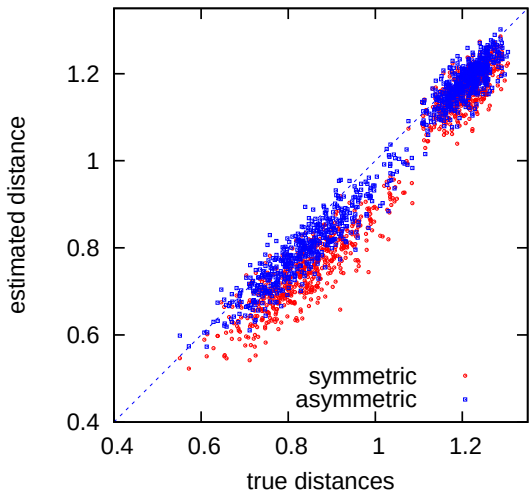


Figure 3: Typical query of a SIFT vector in a set of 1000 vectors: comparison of the distance $d(x, y)$ obtained with the SDC and ADC estimators. We have used $m = 8$ and $k^* = 256$, i.e., 64-bit code vectors. Best viewed in color.

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 square distance in order to cancel the bias. For a particular vector y, we have the quantized index $q(y)$, which in the case of the product quantizer is obtained for 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 square 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, knowing $q(Y) = q(y) = c_i$, 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

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

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 square distance estimator:

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

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. Clearly the bias of the distance estimation by Equation 13, significantly reduced in the corrected version. However, correcting the bias leads, in some cases, 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 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.

4 Non exhaustive search

The search method proposed in the previous section allows the efficient calculation of distances with a small amount of memory. Searching the nearest neighbors with a product quantizer is faster because less memory has to be visited and only m additions are required per distance calculation, but the search

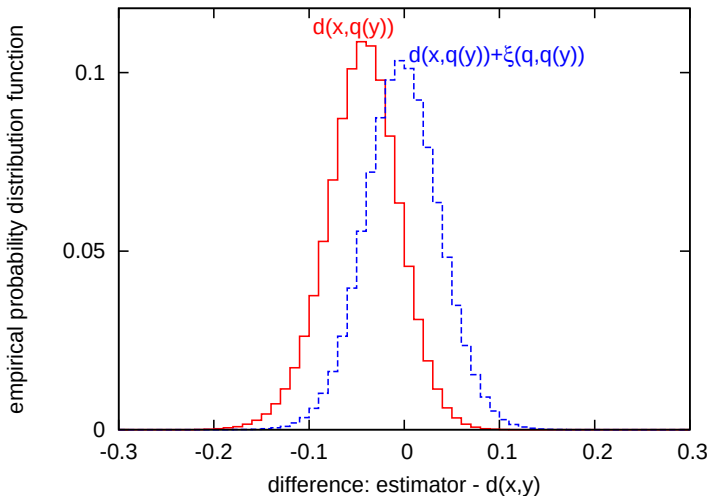


Figure 4: PDF of the error on the distance estimation $d - \tilde{d}$ for the asymmetric method, evaluated on a set of 10000 SIFT vectors with $m = 8$ and $k^* = 256$. The bias (=-0.044) of the estimator $\tilde{d}$ is corrected (=0.002) with the error quantization term $\xi\big(q, q(y)\big)$. However,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

is still exhaustive. This is possible in the context of a global descriptor [13] and [15]. However, to index billions of descriptors and perform multiple queries, as required by approaches based on local descriptors [16], an exhaustive search is prohibitive.

In this section, we propose an approach, denoted by inverted file asymmetric distance computation (IVFADC), that avoids the exhaustive search at the cost of a few additional bits/bytes per descriptor. It is built upon an inverted file structure, which has been shown successful for very large scale image search [22, 18, 16, 23]. This approach significantly accelerates the search and in addition improves its quality.

4.1 Coarse quantizer, inverted lists and multiple assignment

Similar to the so-called Video-Google approach [22], a codebook is learned using k-means, producing a quantizer q_c, referred to as the *coarse quantizer* in the following. The regular k-means is advantageously replaced by an approximate k-means and the corresponding approximate quantization strategy, as done in [18, 24, 25]. 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 this paper.

We use this coarse quantizer to implement an inverted file structure. It is 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 $\{y \in \mathcal{Y} : q_c(y) = c_i\}$.

Using only the index obtained with q_c is an imprecise representation of a vector. The vector description can be further improved by adding a binary signature [16], that refines the vector location in a cell of the coarse quantizer. It is stored jointly with the vector identifier in $\mathcal{L}_i$. An entry is then defined as

field	length (bits)
identifier	8–32
code	l

where l is the length of the binary code associated with each descriptor. 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 the vectors of the same image have the same identifier. Therefore, a 20-bit field is sufficient to identify an image among one million images. This memory cost can be further reduced by the use of index compression [26, 27], 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 in [16] and [26].

Given an inverted list, the nearest neighbor y of a query vector x may not be quantized to $q_c(x)$. To address this problem, we use the multiple assignment strategy of [28]. The query[3] is assigned to w indexes instead of only one, which correspond to the w nearest neighbors of x in the codebook q_c. All the corresponding inverted lists are scanned.

4.2 Locally defined product quantizer codes

We adopt a strategy similar to that proposed in [16], i.e., the description of a vector is refined by a short code obtained with a product quantizer. However, in order to take into account the information provided by the coarse quantizer, i.e, the centroid $q_c(x)$ associated with the vector x, the product quantizer is used to encode the residual vector

$$r(x) = x - q_c(x), \tag{28}$$

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

Denoting by q_p the product quantizer used to encode the residual vector, a vector x is then represented by the tuple $\big(q_c(x), q_p(r(x))\big)$, where $q_p(r(x))$ is stored in the inverted list entry associated with x. 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 formally computed as the distance $\ddot{d}(x, y)$ between x and the approximation of y by

$$\ddot{y} \triangleq q_c(y) + q_p\big(y - q_c(y)\big). \tag{29}$$

[2] An average cost of 11 bits is reported in [26] using delta encoding and Huffman codes.

[3] Multiple assignment is not applied to database vectors, as this would severely increase the memory usage.

It can be re-written as

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

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

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

Similar to the ADC strategy, for each subquantizer q_{p_i} the distances between the partial residual vector $u_j\big(x - q_{\mathrm{c}}(x)\big)$ and all the centroids $c_{j,i}$ of q_{p_i} are preliminarily computed and stored. This improves the efficiency of the distance calculation when the query x is compared with a large set of vectors in the inverted list.

The product quantizer is determined 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 an 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 is probably inferior 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'.

4.3 Indexing structure and search algorithm

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 is done for the product quantizer by assigning $u_j(y)$ to $q_j(u_j(y))$, $j = 1 \ldots m$.
4. store the vector (or image) identifier and the binary code representing the product quantizers indexes in an entry of the inverted list.

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 step 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 square distance $d\big(u_j(r(x)), c_{j,i}\big)^2$ for each subquantizer j and each of its centroids $c_{j,i}$;

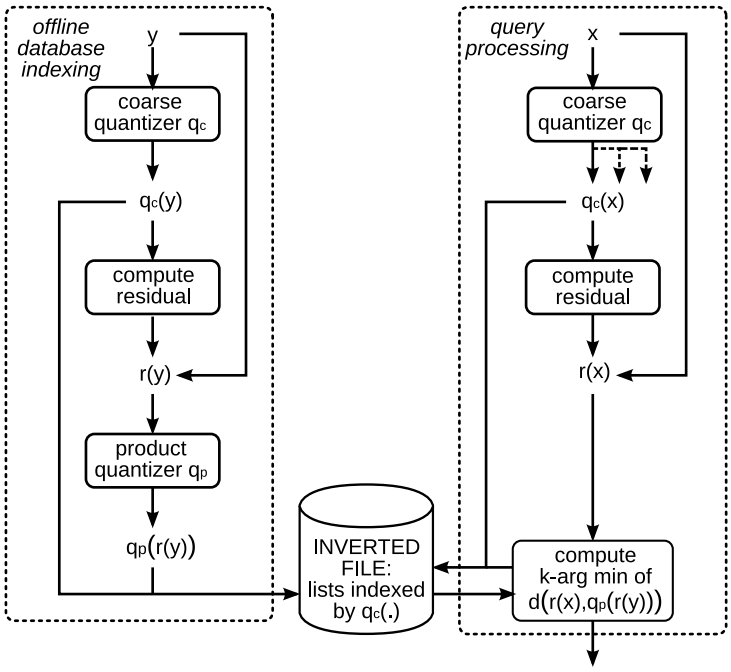


Figure 5: Overview of the *inverted file with asymmetric distance computation (IVFADC)* indexing system. Left: insertion of a vector. Right: search.

3. compute the square 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 distance is the sum of m looked-up values, see Equation 31;

4. select the k-nearest neighbors of x based on the estimated distance using the Maxheap algorithm. Note that, for more efficiency, this step is done jointly with the distance calculation, which avoids storing all the distances.

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 for 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 subsection.

5 Evaluation of NN search

In this section, we introduce the datasets used for the evaluation. We first analyze the impact of the parameters for SDC, ADC and IVFADC. Our approach is then compared to two state-of-the-art methods: spectral hashing [15] and Hamming embedding [16]. Finally, we evaluate the complexity and speed of our approach.

5.1 Datasets

In this section we use two datasets, one with local SIFT descriptors [20] and the other with global color GIST descriptors [14]. The learning stage is performed on separate sets of vectors. Therefore, 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[4], the learning is set extracted from Flickr images and the database and query descriptors are from the INRIA Holidays dataset [16]. For GIST, the learning set consists of the first 100k images extracted from the tiny image dataset [12]. The database set is the Holidays dataset combined with the Flickr1M dataset used in [16]. The query images are the Holidays queries. Table 3 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 3: Summary of the SIFT and GIST datasets.

The search quality is measured by the recall@R, i.e., the proportion of query vectors, for which the nearest neighbor is ranked in the first R positions. This measures 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.

5.2 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^*, which corresponds to a code length of $m \times \log_2 k^*$. 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 two estimators ADC and SDC, 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, we 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, we can also see 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 the efficiency of the two approaches is equivalent, we advocate not to quantize the query, but only the database elements.

Figure 7 is an evaluation of the parameters for the IVFADC method introduced in Section 4. We can observe that the recall@100 depends on the codebook size k' as well as the number of neighboring cells w visited during the multiple assignment. We can observe that increasing the code length is useless

[4]This dataset is available at http://lear.inrialpes.fr/people/jegou/data.php

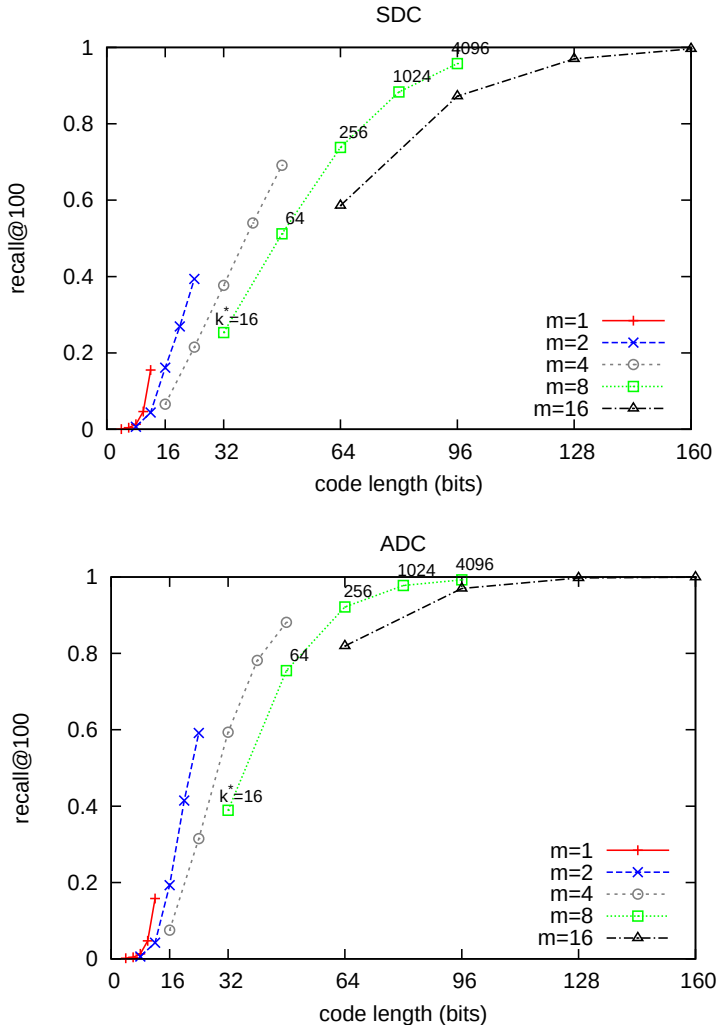


Figure 6: SIFT dataset: recall@100 as a function of the memory usage (code length) for different parameters and the SDC and ADC estimators

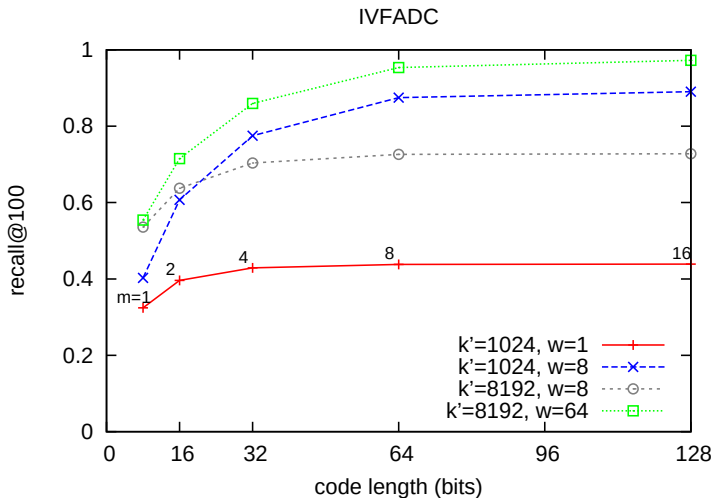


Figure 7: SIFT dataset: recall@100 for the IVFADC approach as a function of the memory usage for k^*=256 and varying values of $m = \{1, 2, 4, 8, 16\}$, $k' = \{1024, 8192\}$ and $w = \{1, 8, 64\}$.

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.

We have, in addition, to set the codebook size k' for the IVFADC approach. Recall that 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 $(8192, 64)$.

5.3 Impact of the component grouping

The product quantizer defined in Section 2 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 stemming from 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.

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

	SIFT	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 4: Impact of the dimension grouping on the retrieval performance of ADC (recall@100, k^*=256).

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 [20] 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.

5.4 Comparison with state-of-the-art

Our methods are compared with the spectral hashing of Weiss et al. [15], 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 [13]. We have used the publicly available code for SH. We also compare to the Hamming embedding (HE) method of [16], 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 shows, respectively for the SIFT and the GIST datasets, the rank repartition of the nearest neighbors when using a signature of sizes 64 bits. For our product quantizer we have used $m = 8$ and $k^* = 256$, which gives similar results in terms of run time. All our approaches significantly outperform spectral hashing on the two datasets. To achieve the same recall as spectral hashing, ADC returns an order of magnitude less vectors.

Best results are obtained by IVFADC, which for low ranks provides a significant improvement. Recall that this strategy avoids the exhaustive search and is therefore significantly faster, as discussed in the next section. 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,

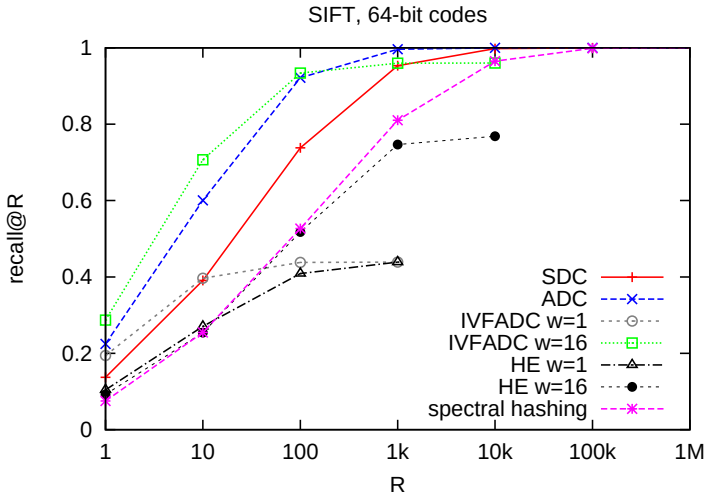


Figure 8: SIFT dataset: recall@R for varying values of R. Comparison of the different approaches SDC, ADC, IVFADC, spectral hashing [15] and HE [16]. We have used m=8, k^*=256 for SDC/ADC. The coarse quantizer contains k'=1024 centroids for HE [16] and IVFADC which do not perform exhaustive search.

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 5: 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.

HE is significantly outperformed by IVFADC. The results of HE are similar to spectral hashing, but HE is more efficient[5].

5.5 Complexity and speed

Table 5 evaluates the search time of our methods. For reference, we report the results obtained with the spectral hashing algorithm of [15] on the same dataset and machine (using only one core). Since we use a separate learning

[5]In defense of spectral hashing, which can be used for arbitrary distance measures, the other approaches are adapted to the Euclidean distance only.

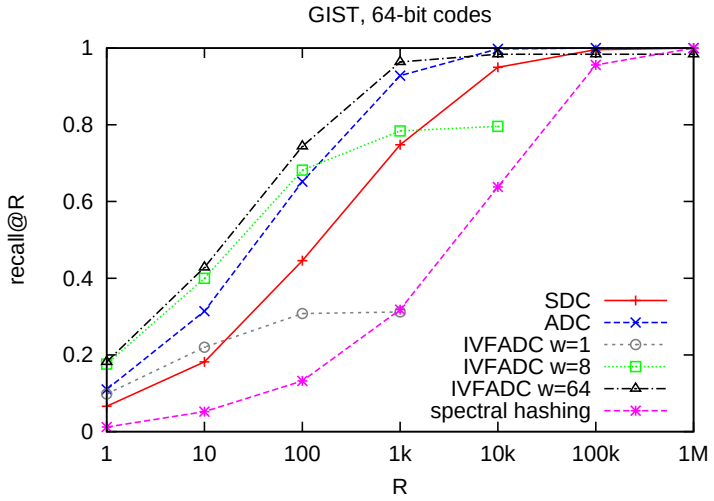


Figure 9: GIST dataset: recall@R for varying values of R. Comparison of the different approaches SDC, ADC, IVFADC and spectral hashing [15]. We have used m=8, k^*=256 for SDC/ADC and $k' = 1\,024$ for IVFADC.

set, we use the out-of-sample evaluation of this algorithm. Note that we have re-implemented the Hamming distance computation in C in order to have the approaches similarly optimized. The algorithms SDC, ADC and SH provide similar efficiencies. IVFADC significantly improves the performance by avoiding 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 regular k-means for q_c. 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 [18]. In this work, the authors propose a hierarchical quantizer to efficiently assign descriptors to the centroids in a codebook of size one million.

5.6 Large-scale experiments

To evaluate the search efficiency of the product quantizer method on larger datasets we extracted SIFT descriptors from one million images. Searches are 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 10. 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 approximately as expensive in both cases. For HE, it is a Hamming distance computation, implemented as 8 table lookups. For IVFADC it is a

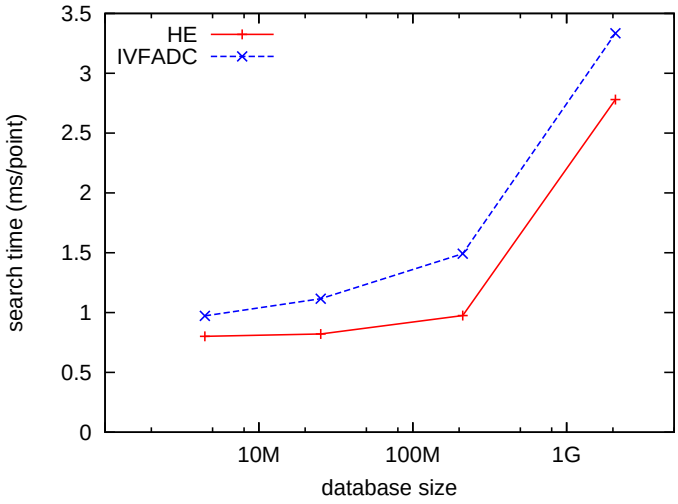


Figure 10: Search times for SIFT descriptors in datasets of increasing sizes, with two search methods. Both use the same 20 000-word codebook, $w = 1$, and 64-bit signatures.

distance computation that also boils down to 8 table lookups. Interestingly, the floating point operations involved in IVFPQ are not much more expensive than the simple binary operations of HE.

6 Conclusion

In this paper, we have introduced a product quantizer for nearest neighbor search. Our coding scheme permits to approximate the Euclidean distance accurately as well as memory efficiently. It is shown to significantly outperform the comparable state-of-the-art approaches [16, 15] in terms of the trade-off between search quality and memory usage.

Acknowledgements

We would like to thank the search engine project Quaero as well as the ANR project Gaia for their financial support.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *ICCV*, October 2009.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS*, 2008.

[16] 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.

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

[18] 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.

[19] 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.

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

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

[22] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," in *ICCV*, pp. 1470–1477, 2003.

[23] M. Douze, H. Jégou, H. Singh, L. Amsaleg, and C. Schmid, "Evaluation of GIST descriptors for web-scale image search," in *civr*, 2009.

[24] 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.

[25] H. Jégou, M. Douze, and C. Schmid, "Improving bag-of-features for large scale image search," *International Journal of Computer Vision*, 2009. to appear.

[26] 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.

[27] H. Jégou, M. Douze, and C. Schmid, "Packing bag-of-features," in *ICCV*, sep 2009. to appear.

[28] 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.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399