Explicit Embeddings for Nearest Neighbor Search with Mercer Kernels

Anthony Bourrier · **Florent Perronnin** ·
Rémi Gribonval · **Patrick Pérez** · **Hervé Jégou**

Received: 30 June 2014 / Accepted: 2 January 2015 / Published online: 22 January 2015

Abstract Many approximate nearest neighbor search algorithms operate under memory constraints, by computing short signatures for database vectors while roughly keeping the neighborhoods for the distance of interest. Encoding procedures designed for the Euclidean distance have attracted much attention in the last decade. In the case where the distance of interest is based on a Mercer kernel, we propose a simple, yet effective two-step encoding scheme: first, compute an explicit embedding to map the initial space into a Euclidean space; second, apply an encoding step designed to work with the Euclidean distance. Comparing this simple baseline with existing methods relying on implicit encoding, we demonstrate better search recall for similar code sizes with the chi-square kernel in databases comprised of visual descriptors, outperforming concurrent state-of-the-art techniques by a large margin.

This work was done while the first author was affiliated to Technicolor, Cesson-Sévigné and INRIA Rennes-Bretagne Atlantique.

A. Bourrier (✉)
Gipsa-Lab, Saint-Martin-d'Hères, France
e-mail: anthony.bourrier@gipsa-lab.grenoble-inp.fr

F. Perronnin
Xerox Research Center Europe, Meylan, France
e-mail: florent.perronnin@xrce.xerox.com

R. Gribonval · H. Jégou
INRIA Rennes-Bretagne Atlantique, Rennes, France
e-mail: remi.gribonval@inria.fr

H. Jégou
e-mail: herve.jegou@inria.fr

P. Pérez
Technicolor, Cesson-Sévigné, France
e-mail: patrick.perez@technicolor.com

Keywords Explicit embeddings · Nearest Neighbor search · Mercer kernels

1 Introduction

This paper tackles the problem of nearest neighbor (NN) search. Let Ω be a set supplied with a distance d and $\mathscr{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_L\}$ be a database of elements of Ω. Given a new element $\mathbf{q} \in \Omega$, typically called a *query*, a nearest neighbor of $\mathbf{q}$ in $\mathscr{X}$ is a solution of the problem

$$\operatorname*{argmin}_{\mathbf{x}\in\mathscr{X}} d(\mathbf{q}, \mathbf{x}). \qquad (1)$$

Such a problem arises in applications such as image search [25,26] and classification [1,5,8,20]. In such practical scenarii, one is usually interested in computing not only the nearest neighbor of a query but its k nearest neighbors. In this paper, we consider the case where we only seek the nearest neighbor of a query for simplicity, but the contributions described here can be easily extended to the case where one searches for several nearest neighbors of a query.

When the distance d is costly to compute and/or when L is large, it is necessary to rely on approximate nearest neighbor (ANN) schemes. These schemes typically rely on the computation of small-dimensional *signatures*, which are compressed representations of the elements of Ω. The ANN search of a query can be performed at a reduced cost by computing the distances between the query and the signatures, which will typically be way faster to compute than the initial distance d.

Significant progress has been achieved on ANN search in the last two decades, in particular with the substantial contribution of the Locality Sensitive Hashing (LSH) approach [4]. Variants of this method have been developed, consisting of different types of low-dimensional signatures [10,19,29].

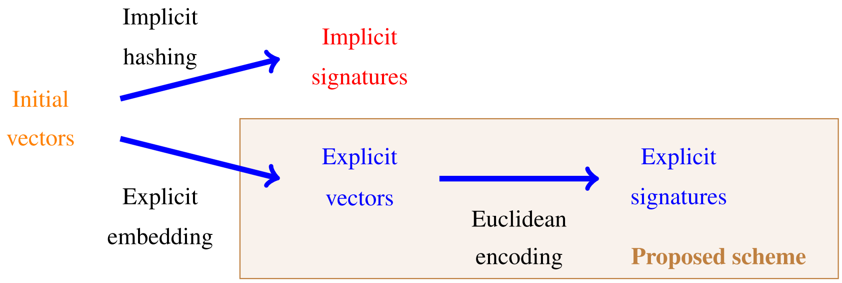

Fig. 1 Proposed approximate search scheme. As opposed as the existing *implicit* hashing methods (*top* path), we first compute explicit vectors belonging to a Euclidean space, then apply an encoding method designed for Euclidean distances (*bottom* path)

However, the aforementioned methods only consider simple metrics such as ℓ_p norms, and mainly the Euclidean norm.

In this paper, we will consider the case where the distance d derives from a Mercer kernel K, which is a broad generalization of the case where the distance is Euclidean. In this case, some methods have been proposed to compute adequate signatures by analogy with LSH, such as Kernelized Locality Sensitive Hashing (KLSH) [15] or Random Maximum Margin Hashing (RMMH) [13]. These methods essentially aim at imitating LSH in the implicit space associated to K. Our contribution is to propose a simple alternative to these methods by exploiting the notion of *explicit embedding*, which consists in approximating explicitly the kernel by a finite-dimensional scalar product. After this step, a standard Euclidean encoding scheme can be applied. This scheme is illustrated in Fig. 1. While simple, this strategy is shown to perform surprisingly well compared to the previously mentioned hashing schemes.

The paper is divided as follows: in Sect. 2, we review basics on kernels and mention some explicit embedding techniques, particularly Kernel Principal Component Analysis (KPCA), which we subsequently use. Section 3 describes existing binary hashing schemes designed for Mercer kernels. In Sect. 4, we present the proposed method to perform ANN search. In Sect. 5, we describe the datasets and report the experiments comparing our method with the previously mentioned hashing schemes. Finally, the appendix describes an exact search technique which can be derived from applying KPCA to the initial vectors. Though much more computationally costly than an approximate search scheme, this method can yield computational savings in some cases.

2 Mercer Kernels and Explicit Embeddings

2.1 Mercer Kernels

Let us first recall standard definitions and properties of Mercer kernels [22]. A positive semi-definite (PSD) kernel on a space Ω is an application $K : \Omega^2 \to \mathbb{R}$ such that for any collection $\mathscr{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_L\}$ of elements of Ω, the Gram matrix

$$\mathbf{K}(\mathscr{X}) = \begin{pmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \dots & K(\mathbf{x}_1, \mathbf{x}_L) \\ \vdots & & \vdots \\ K(\mathbf{x}_L, \mathbf{x}_1) & \dots & K(\mathbf{x}_L, \mathbf{x}_L) \end{pmatrix} \tag{2}$$

is symmetric semi-definite positive. For any such kernel, there exists a unique (up to isomorphism) Hilbert space $\mathscr{H}$ and application $\boldsymbol{\Psi} : \Omega \to \mathscr{H}$ such that $\forall \mathbf{x}, \mathbf{y} \in \Omega$,

$$K(\mathbf{x}, \mathbf{y}) = \langle \boldsymbol{\Psi}(\mathbf{x}), \boldsymbol{\Psi}(\mathbf{y}) \rangle_{\mathscr{H}}, \tag{3}$$

where $\langle \cdot, \cdot \rangle_{\mathscr{H}}$ is the scalar product of $\mathscr{H}$. Space $\mathscr{H}$ is usually named *implicit space*, since it is a space which allows the implicit interpretation of the action of K as a scalar product, provided one knows the *implicit* function $\boldsymbol{\Psi}$ which embeds Ω into $\mathscr{H}$. Usually, one does not know this function but some interesting properties can still be derived, relying on the so-called *kernel trick*: one does not need to explicitly know the values of $\boldsymbol{\Phi}(\mathbf{x})$ and $\boldsymbol{\Phi}(\mathbf{y})$ to compute their scalar product in $\mathscr{H}$, since this scalar product is equal to $K(\mathbf{x}, \mathbf{y})$.

In particular, K induces a distance d_K on Ω by posing

$$d_K(\mathbf{x}, \mathbf{y}) = K(\mathbf{x}, \mathbf{x}) + K(\mathbf{y}, \mathbf{y}) - 2K(\mathbf{x}, \mathbf{y}). \tag{4}$$

Let us notice that if we define

$$S_K(1) = \{\mathbf{x} \in \Omega : K(\mathbf{x}, \mathbf{x}) = 1\}, \tag{5}$$

then for all $\mathbf{x}, \mathbf{y} \in S_K(1)$,

$$d_K(\mathbf{x}, \mathbf{y}) = 2\,(1 - K(\mathbf{x}, \mathbf{y}))\,. \tag{6}$$

This property ensures that when considering data $\mathscr{X} \subset S_K(1)$ and a query $\mathbf{q} \in S_K(1)$, the problem (1) with $d = d_K$ is equivalent to the following problem:

$$\underset{\mathbf{x} \in \mathscr{X}}{\operatorname{argmax}}\ K(\mathbf{q}, \mathbf{x}), \tag{7}$$

that is searching for the nearest neighbor in the sense of d_K is equivalent to searching for the most correlated vector in the sense of K. This constitutes a good incentive to normalize data, that is project it onto $S_K(1)$, when it is possible: it allows one to consider kernels instead of distances. This will be used in our contribution, where we will perform our experiments on normalized data.

In the particular case where Ω is a compact Hausdorff space provided with a finite measure μ with support Ω, the kernel K is a Mercer kernel [14], that is $\mathscr{H} = \ell^2(\mathbb{R})$ and for all $i \in \mathbb{N}$, there exists $\lambda_i \in \mathbb{R}$ and $\boldsymbol{\Psi}_i : \Omega \to \mathbb{R}$ satisfying

$$\boldsymbol{\Psi}(\mathbf{x}) = \left[\sqrt{\lambda_i}\boldsymbol{\Psi}_i(\mathbf{x})\right]_{i=0}^{\infty}, \tag{8}$$

with $\lambda_i \downarrow 0$ and $\int_\Omega \boldsymbol{\Psi}_i \boldsymbol{\Psi}_j \, d\mu = 1$ if $i = j$ and 0 if $i \neq j$. The λ_i's and $\boldsymbol{\Psi}_i$'s are, respectively, the *eigenvalues* and the *eigenfunctions* of K. In this case, the embedding $\boldsymbol{\Psi}$, therefore, transforms elements of Ω into square-summable sequences. This is particularly important in order to approximate the kernel values, as we will see in the next section.

2.2 Explicit Embeddings

Performing an explicit embedding consists in approximating the unknown function $\boldsymbol{\Psi}$ by an explicit function $\tilde{\boldsymbol{\Psi}}$ which maps Ω to the Euclidean space $\mathbb{R}^E$ ($E > 0$ is the *embedding dimension*), and satisfying

$$\left\langle \boldsymbol{\Psi}(\mathbf{q}), \boldsymbol{\Psi}(\mathbf{x}_i)\right\rangle_{\mathscr{H}} \approx \left\langle \tilde{\boldsymbol{\Psi}}(\mathbf{q}), \tilde{\boldsymbol{\Psi}}(\mathbf{x}_i)\right\rangle, \tag{9}$$

where $\langle \cdot, \cdot \rangle$ is the usual scalar product in $\mathbb{R}^E$. Let us provide a quick review of the standard KPCA method, as well as a brief list of other embedding methods which have been proposed.

2.2.1 Kernel Principal Component Analysis

KPCA was introduced in [23] as a way to perform explicitly Principal Component Analysis (PCA) in the implicit space $\mathscr{H}$, thus being a generic way to compute a finite-dimensional embedding of the data satisfying the condition (9).

Performing PCA on the set of vectors $\boldsymbol{\Psi}(\mathscr{Y}) = \{\boldsymbol{\Psi}(\mathbf{y}_i)\}_{i=1}^M$ in implicit space $\mathscr{H}$ consists in projecting them onto the E-dimensional subspace V that minimizes the following mean-squared error:

$$\sum_{i,j=1}^{M} \left(K(\mathbf{y}_i, \mathbf{y}_j) - \langle P_W \boldsymbol{\Psi}(\mathbf{y}_i), P_W \boldsymbol{\Psi}(\mathbf{y}_j)\rangle_{\mathscr{H}}\right)^2 \tag{10}$$

over all E-dimensional subspaces W (P_W being the orthogonal projection onto W).

It is possible to compute explicitly this projection for any vector of the form $\boldsymbol{\Psi}(\mathbf{y})$ by exploiting the Gram matrix $\mathbf{K}(\mathscr{Y}) = (K(\mathbf{y}_i, \mathbf{y}_j))_{i,j=1,\ldots M}$. Let us denote its E largest eigenvalues by $(\sigma_i^2)_{i=1}^E$ and by $(\mathbf{u}_i)_{i=1}^E$ the corresponding eigenvectors. Let $K(\mathbf{y}, :)$ be the vector $[K(\mathbf{y}, \mathbf{y}_1), \ldots, K(\mathbf{y}, \mathbf{y}_M)]^T$. Then if $(\mathbf{v}_i)_{i=1}^E$ denote the E first eigenvectors of the covariance operator of $\boldsymbol{\Psi}(\mathscr{X})$, one has [23]:

$$\varphi_i(\mathbf{x}) = \left\langle P_V \boldsymbol{\Psi}(\mathbf{x}), \mathbf{v}_i\right\rangle_{\mathscr{H}} = \frac{1}{\sigma_i}\left\langle K(\mathbf{x}, :), \mathbf{u}_i\right\rangle. \tag{11}$$

This result allows us to define by PCA approximation an explicit embedding $\tilde{\boldsymbol{\Psi}} : \Omega \to \mathbb{R}^E$ such that $\tilde{\boldsymbol{\Psi}}(\mathbf{x}) = [\varphi_i(\mathbf{x})]_{i=1}^E$ and an approximate kernel $\tilde{K}(\mathbf{x}, \mathbf{y}) = \langle \tilde{\boldsymbol{\Psi}}(\mathbf{x}), \tilde{\boldsymbol{\Psi}}(\mathbf{y})\rangle$. We therefore have

$$\tilde{K}(\mathbf{x}, \mathbf{y}) = \left\langle P_V \boldsymbol{\Psi}(\mathbf{x}), P_V \boldsymbol{\Psi}(\mathbf{y})\right\rangle_{\mathscr{H}}. \tag{12}$$

Note that KPCA is *adaptive*, that is the explicit embedding $\tilde{\boldsymbol{\Phi}}$ it produces is learned on training data $\mathscr{Y}$, and thus depends on it (as well as on the chosen E).

2.2.2 Other Explicit Embeddings

Other embedding methods have been proposed for specific kernels in a classification context. In [17], the authors describe an explicit embedding for the histogram intersection kernel, defined for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ by

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \min(x_i, y_i). \tag{13}$$

In [27,28], the authors propose a nonadaptive (thus requiring no input data nor learning step) embedding algorithm for additive or multiplicative kernels, that is kernels defined on $\mathbb{R}^n$ as

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} k(x_i, y_i)$$

or

$$K(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{n} k(x_i, y_i),$$

where k is a kernel on $\mathbb{R}$.

In [20], the authors suggest applying KPCA independently in each dimension if the kernel K is additive. This yields a computational gain at the cost of a loss of precision in the embedding approximation.

As we will see, the methods we propose heavily relies on explicit embeddings. In all our experiments, we used KPCA as the embedding method, mainly because we are aiming at a generic kernel-independent method. In that sense, the KPCA has the benefit that it does not make any assumption on K other than the fact it is a Mercer kernel. Moreover, KPCA allows one to derive an embedding in $\mathbb{R}^E$ without constraints on E, whereas for instance, [17,20,27] require the output dimensionality to be greater than or equal to that of the initial space. Finally, KPCA yields in practice good approximations on real datasets.

3 ANN Hashing Schemes for Kernels

In this section, we describe two previously existing methods developed in order to compute signatures specifically aimed at solving problem (1) when d is a Mercer kernel distance. Both methods produce binary signatures, which are then compared with respect to the Hamming distance in the signature space. Note that other methods aimed at producing small-size signatures for kernels exist, such as [7,9], but were not considered in our work because the framework presented in these papers is different from ours and closer to a classification context.

3.1 Kernelized Locality Sensitive Hashing

In LSH, a typical choice for the hash functions for n-dimensional vectors are functions of the type $h : \mathbf{x} \mapsto \operatorname{sign}(\langle \mathbf{x}, \mathbf{r} \rangle)$, where $\mathbf{r}$ is usually chosen as a random draw with respect to a normal distribution $\mathscr{N}(0, \mathbf{I})$. KLSH [15] aims at imitating this Gaussian random choice in the implicit space $\mathscr{H}$.

To this end, the authors select M data points

$$\mathscr{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_M\}$$

among $\mathscr{X}$. The hash functions are of the following form:

$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{M} w_i K(\mathbf{x}, \mathbf{y}_i)\right), \tag{14}$$

where $\mathbf{w} = [w_i]_{i=1}^{M}$ is a weight vector chosen as

$$\mathbf{w} = \mathbf{K}(\mathscr{Y})^{-\frac{1}{2}} \mathbf{e}_S, \tag{15}$$

where $\mathbf{e}_S$ is a vector with $Q < M$ entries equal to 1 and selected at random, and the others set to 0. This particular choice indeed approximates a standard Gaussian draw in the implicit space [15].

3.2 Random Maximum Margin Hashing

Random Maximum Margin Hashing (RMMH) [13] aims at finding hash functions of the form

$$h : \mathbf{x} \mapsto \operatorname{sign}(\langle \boldsymbol{\Psi}(\mathbf{x}), \mathbf{r} \rangle_{\mathscr{H}} + b),$$

where $\mathbf{r}$ is once again a vector of $\mathscr{H}$ and b is a real number. The authors express the problem of finding such hash functions as an support vector machine (SVM) problem.

Each hash function is chosen as follows: an even number M of vectors $\mathbf{y}_1, \ldots, \mathbf{y}_M$ is randomly drawn from $\mathscr{X}$ and half of these vectors are labeled with label $+1$, the other half with label -1. The corresponding hash function is given by

$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{M} \alpha_i \epsilon_i K(\mathbf{x}, \mathbf{y}_i) + b\right), \tag{16}$$

where ϵ_i is the label of $\mathbf{y}_i$ and $(\alpha_i)_{i=1}^{M}$ and b maximize the usal C-SVM cost:

$$\frac{1}{2} \sum_{i,j=1}^{M} \epsilon_i \epsilon_j \alpha_i \alpha_j K(\mathbf{y}_i, \mathbf{y}_j) + C \sum_{i=1}^{M} \xi_i, \tag{17}$$

subject to

$$\forall i,\ \ \xi_i \geq 0 \text{ and } 1 - \xi_i \leq \epsilon_i \left(\sum_{j=1}^{M} \epsilon_j \alpha_j K(\mathbf{y}_i, \mathbf{y}_j) + b\right). \tag{18}$$

Intuitively, this formulation finds hash functions which prevent close neighbors from having a different image by the hash function, since the hyperplane is chosen by margin maximization. This method has been presented as yielding better precision than KLSH for approximate search.

4 Proposed Approximate Search Method

4.1 General Description

The binary schemes described in Sect. 3 are *implicit* schemes: they aim at finding the encoding function directly as a hashing function in the implicit space $\mathscr{H}$, expressing it as a function of the kernel K. Instead, the approximate scheme we propose relies on two separate steps:

1. The data are embedded in a Euclidean space $\mathbb{R}^E$ using a function $\tilde{\boldsymbol{\Psi}}$ corresponding to KPCA.[1]
2. The signatures are computed by applying a Euclidean signature method on the vectors $\tilde{\boldsymbol{\Psi}}(\mathbf{x}_i)$ in $\mathbb{R}^E$.

Note that at step 2, if the data are normalized with respect to the kernel K, any compression technique which aims at approximating the dot-product, the Euclidean distance, or the

[1] Any other explicit embedding method can be used if applicable, but we used KPCA in our experiments.

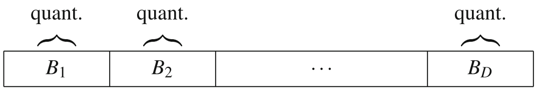

Fig. 2 Illustration of the PQ procedure on one vector. The vector is considered as a product of D subvectors of equal length, and each subvector is quantized separately using a k-means algorithm on the database $\mathscr{X}$

cosine similarity could be employed. This includes among others LSH [3,4] which approximates the cosine or Spectral Hashing [29] and Locality-Sensitive Binary Code [21] which approximate the Euclidean distance. In this section, we particularly focus on a recently proposed technique known as Product Quantization (PQ) which has been shown to be state-of-the-art for the Euclidean distance [10].

4.2 Product Quantization and Variations

Most of the techniques based on short signatures for ANN search consider binary codes [6,13,15]. In [10], the authors proposed instead the PQ algorithm which produces non-binary codes allowing the estimation of Euclidean distances in the signature domain. For this purpose, a product quantizer is used to decompose the feature space into a Cartesian product of D subspaces. A given database vector $\mathbf{x}$ is decomposed into D subvectors of equal lengths, each of which is quantized separately using a k-means quantizer with a limited number of centroids. This number is typically set to 256 such that each subvector is quantized as a 1-byte code (8 bits) and therefore the vector is represented by a D-bytes code. This quantization is illustrated in Fig. 2. The estimation of $d(\mathbf{q}, \mathbf{x})$ is done in an asymmetrical manner, *i.e.*, the query $\mathbf{q}$ is not quantized to prevent it from being approximated. Computing a distance estimate requires D table look-ups and additions, which remains competitive compared with operations on binary vectors at a given signature size.

In our experiments, we also add a pre-processing step to this standard PQ framework. It is described in Sect. 5.2.1.

4.3 Summary of the Proposed Indexing and Search Procedure

We now summarize the whole search procedure. The offline steps are as follows:

1. Embedding of the database
   Using KPCA requires to learn it from the data. To this end, we select M vectors at random from $\mathscr{X}$ and learn the embedding $\tilde{\boldsymbol{\Psi}}$ by KPCA. One may consider using another explicit embedding which does not require learning. Once $\tilde{\boldsymbol{\Psi}}$ has been determined, the vectors $\tilde{\boldsymbol{\Psi}}(\mathbf{x})$ are computed for each $\mathbf{x} \in \mathscr{X}$.
2. Encoding of the database
   PQ is learned using the vectors $\tilde{\boldsymbol{\Psi}}(\mathscr{X}) = \{\tilde{\boldsymbol{\Psi}}(x), x \in \mathscr{X}\}$. Note that other Euclidean coding schemes may not require this learning step. The signatures of the vectors $\tilde{\boldsymbol{\Psi}}(\mathbf{x})$ are computed.

Given a query $\mathbf{q}$, the online steps are as follows:

1. Embedding/encoding of the query
   The embedding of the query $\tilde{\boldsymbol{\Psi}}(\mathbf{q})$ is computed. If the chosen Euclidean encoding is symmetric (like LSH), the signature of $\tilde{\boldsymbol{\Psi}}(\mathbf{q})$ needs to be computed. If it is asymmetric (like PQ), it needs not.
2. Full nearest neighbor search with the signatures
   The distances between $\mathbf{q}$ (or its signature) and the database signatures are computed. The $N \ll L$ corresponding (approximate) nearest neighbors are identified. These are the potential (true) nearest neighbors.
3. Reranking of the potential nearest neighbors
   The true distances between the query and the N considered vectors are computed to rerank them to respect to the distance of interest.

5 Experiments

In this section, we describe the experiments performed to compare the ANN search scheme proposed in Sect. 4 with the methods mentioned in Sect. 3.

5.1 Datasets and Experimental Protocol

For the sake of reproducibility, we have used publicly available datasets of vectors that are large enough to evaluate the interest of ANN search techniques. The three benchmarks consist of visual descriptor extracted from real images:

- *SIFT1M* This benchmark introduced in [10] consists of one million local SIFT descriptors [16] computed on patches extracted with the Hessian–Affine detector [18].
- *BIGANN* [12] This set provides a NN ground-truth for subsets of increasing size, from one million to one billion vectors. These descriptors were extracted using the Difference of Gaussians (DoG) detector and are, therefore, not equivalent to those of the SIFT1M benchmark.
- *Imagenet* [5]: This set contains the pre-computed 1,000-dimensional bag-of-words (BOW) vectors provided with the images of the 2010 Large-Scale Visual Recognition (ILSVRC'2010). For our search problem, the larger "train" set is used as the database to be searched, the "val" set to learn the parameters, and we randomly selected a subset of 1,000 queries from the "test" set to act as queries to be processed by our methods.

Table 1 Datasets used for the evaluation of nearest neighbor search

Dataset	Number of vectors	Dim	Descriptor	URL
SIFT1M	1M	128	HesAff + SIFT	http://corpus-texmex.irisa.fr
BIGANN	1M–1G	128	DOG + SIFT	http://corpus-texmex.irisa.fr
Imagenet	1.261M	1,000	Bag-of-words	http://www.image-net.org

Table 1 summarizes the main characteristics of the three datasets.

In our case, each of these datasets is separated into three disjoint sets: a query set, a training set, and a validation set, this separation being random for SIFT1M and BIGANN which do not have such explicit prior separation of the full dataset.

For these three datasets, we are looking for the most similar vectors with respect to the χ^2 kernel, as done in related works [13,28]. The χ^2 kernel is defined on $\Omega = \mathbb{R}^n$ by

$$K(\mathbf{x}, \mathbf{y}) = K_{\chi^2}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \frac{x_i y_i}{x_i + y_i}. \tag{19}$$

In our experiments, all the data vectors were ℓ_1-normalized. This is the normalization for which $K_{\chi^2}(\mathbf{x}, \mathbf{x}) = 1$, and thus it is the required normalization to get the equivalence between nearest neighbor and highest kernel value, as discussed in Sect. 4.

In the experiments we describe, the search performance is measured by the Recall@R, which measures the rate of queries for which the nearest neighbor with respect to the χ^2 kernel in input space is present among the R closest neighbors in the Euclidean signature/embedded space. Note that the higher the Recall@R, the lower the complexity of the third online step of the search procedure described in Sect. 4.3, because fewer vectors need to be reranked to get the true nearest neighbor.

5.2 Experiments on Approximate Search

Let us now discuss a particular pre-processing which has been chosen for the PQ technique in our experiments.

5.2.1 Pre-Processing for Energy Balance

Since we are applying PQ after a KPCA, the energy is not equally distributed in the embedded vectors $\tilde{\boldsymbol{\Psi}}(\mathbf{x}_i)$: the first components have high energy, while the last components have smaller values. This has a negative impact on PQ since the last subvectors will have less variance than the first, while the k-means clustering still produces the same number of centroids for each subvector. Intuitively, one should find a way to balance energy between the subvectors so that all the quantizers have a subvector of comparable energy to encode. Note that alternatively, one could also consider using more centroids to encode subvectors with higher variance.

In [11], the authors faced the same phenomenon after a regular PCA. In this context, they argue that the PQ technique is more effective if a rotation matrix is applied on the vectors produced by the PCA and prior to PQ encoding. The drawback is that the individual product quantizers partially encode the same information, since the decorrelation performed by PCA is lost.

In our contribution, we consider the use of a random permutation of the components of the vectors $\tilde{\boldsymbol{\Psi}}(\mathbf{x}_i)$ as an alternative to applying a random rotation. The random permutation is cheaper to compute and it ensures that the component decorrelation achieved by KPCA is retained.

In order to choose the appropriate pre-processing for PQ, let us study the impact of three pre-processing techniques:

(i) No pre-processing is done at the output of the KPCA.
(ii) A random rotation is applied to the $\tilde{\boldsymbol{\Psi}}(\mathbf{x}_i)$.
(iii) The components of the $\tilde{\boldsymbol{\Psi}}(\mathbf{x}_i)$ are randomly permuted.

Table 2 shows that the choice of the pre-processing stage has an impact on the search quality in terms of Recall@R. On SIFT1M, the random permutation is clearly the best choice. On Imagenet, the difference is less clear but the random permutation still yields better results. We will therefore adopt this choice in all subsequent experiments.

5.2.2 Comparison with the State-of-the-Art

We now compare the proposed coding scheme based on KPCA+PQ with the state-of-the-art techniques described

Table 2 Choice of the pre-processing: performance of KPCA+PQ (Recall@R) with no pre-processing [10], Random rotation (RR) [11] and Random permutation (RP)

Dataset	SIFT1M			Imagenet		
Pre-processing	None	RR	RP	None	RR	RP
Recall@1	0.17	0.15	**0.19**	0.04	0.04	**0.05**
Recall@10	0.42	0.39	**0.51**	0.12	0.12	**0.14**
Recall@100	0.75	0.74	**0.85**	0.30	0.33	**0.37**
Recall@1000	0.96	0.97	**0.99**	0.60	0.67	**0.69**

Parameters: $M = 1{,}024$, $E = 64$ and $D = 8$ for the two datasets considered. Averages on 10 experiments. In each experiment, any random draw of the whole algorithm is performed again: selection of the learning vectors, PQ encoding and RR/RP

The pre-processing technique which gives the best recall is reported in bold

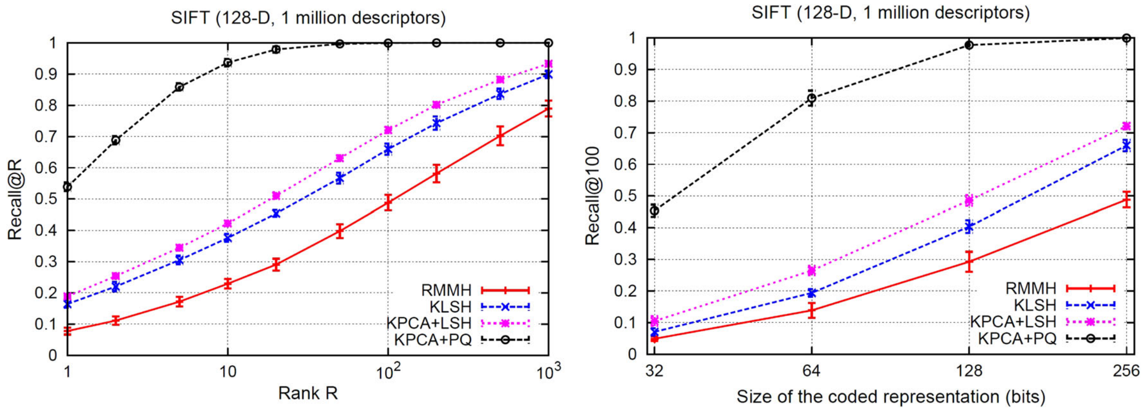

Fig. 3 Comparison with the state-of-the-art: search quality for the χ^2 kernel with coded representations for 1M SIFT. *Left* recall@R *curves* for a fixed size of $B = 256$ bits. *Right* recall@100 as a function of the number of bits (32–256). Parameters: KPCA+LSH → $E = B$ and $M = 1{,}024$; KPCA+PQ → $E = 128$ and $M = 1{,}024$

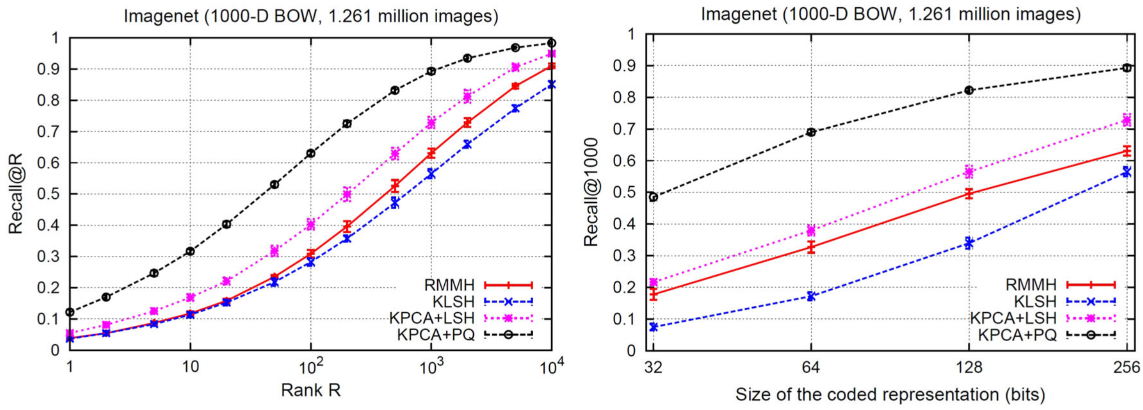

Fig. 4 Comparison with the state-of-the-art: search quality for the χ^2 kernel with coded representations for 1.2M BOW. *Left* recall@R *curves* for a fixed size of $B = 256$ bits. *Right* recall@1000 as a function of the number of bits (32–256). Parameters: KPCA+LSH → $E = B$ and $M = 1{,}024$; KPCA+PQ → $E = 128$ and $M1, 024$

earlier: KLSH [15] and RMMH [13]. We note that the proposed approach differs in two significant ways from KLSH and RMMH:

(i) KLSH and RMMH rely on an implicit embedding, while we rely on an explicit KPCA embedding.
(ii) The signatures produced in KLSH and RMMH are binary, while PQ encodes an embedded vector $\tilde{\boldsymbol{\Psi}}(\mathbf{x}_i)$ as a succession of bytes.

To better understand the respective impact of these two differences, we also evaluate a coding scheme based on KPCA+LSH, which performs an explicit embedding of the data (as in the first step of the proposed scheme) but performs a standard binary coding instead of PQ final encoding. Comparing this scheme with KLSH and RMMH allows one to measure exactly the gain of adding the explicit embedding step since the produced binary codes are very similar to those produce by KLSH and RMMH.

The experiments are carried out with different signature sizes (noted B in bits). We compare the encoding schemes in terms of Recall@R at a given signature size: this is a good measure of the complexity of the reranking step for a similar search cost in the signature space and signature storage cost.

The number of learning vectors for the KPCA is set to M =1; 024, so that the number of kernel computations when processing a query is limited. The same number of learning vectors is consistently used for the KLSH and RMMH algorithms, so that the variety of the encoding functions which can be built is the same. For the embedding dimension E, we set it differently depending on the scheme. For KPCA+PQ, we set $E = 128$. For KPCA+LSH, we set $E = B$ which led to optimal or near optimal results in preliminary experiments.

We report results for SIFT1M on Fig. 3 and for Imagenet on Fig. 4. We repeated all experiments 10 times, with different random draws for any step that relies on randomness. We show the average and standard deviation: selection of the M learning vectors and choice of the encoding functions. We show the average and standard deviation.

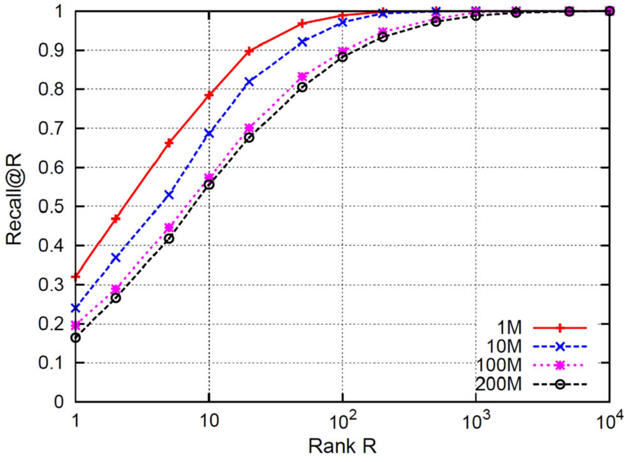

Fig. 5 Performance as a function of the database size (1M–200M vectors). Parameters for KPCA+PQ: $D = 16$, $E = 64$ and $M = 1{,}024$

First observe that, as reported in [13], RMMH outperforms KLSH on the BOW Imagenet features. However, for SIFT features, KLSH performs significantly better than RMMH. For instance, for $B = 128$ bits, the improvement in recall@100 is on the order of 10 % absolute. Second, we can see that the simple KPCA+LSH scheme outperforms KLSH andR-MMH at all ranks and code sizes. This difference is significant (on the order of 10 % absolute on SIFT1M for 128 bits). This shows the practical superiority of explicit embedding over implicit embedding for approximate search on these datasets. Finally, we can observe that KPCA+PQ performs significantly better than KPCA+LSH, which confirms the superiority of a non-binary encoding scheme over a binary coding scheme [10].

Overall, the improvement of KPCA+PQ with respect to KLSH and RMMH can be considerable, especially for small codes. For instance, for SIFT1M and for $B = 64$, the recall@100 is about 80 % for KPCA+PQ while it is 20 % for KLSH. For Imagenet and for the same number B of bits, the recall@1000 of RMMH is somewhat above 30 % while KPCA+PQ achieves on the order of 70 % (Fig. 3, right plot).

5.2.3 Large-Scale Experiments

Figure 5 reports the Recall@R measured on subsets of 1M–200M SIFT descriptors taken from the BIGANN dataset. These results show that the proposed KPCA+PQ scheme can scale to very large datasets without a drastic loss in performance.

6 Conclusion

This study shows that when dealing with a distance derived from a Mercer kernel, the use of explicit embeddings can benefit approximate nearest neighbor search. The proposed approximate scheme, despite its simplicity, performs better than other state-of-the-art encoding methods in this framework. Performing KPCA followed by a standard binary LSH encoding yields better precision than standard methods which directly mimic binary LSH in the implicit space, thus giving a first incentive to use explicit embeddings. Furthermore, mapping the initial elements into a Euclidean space allows one to use a wide range of schemes designed for computing signatures, and especially Product Quantization, which performs way better than binary schemes at an equivalent signature size. These cases are summarized in Table 3.

Table 3 Summary of the compared schemes

	Implicit	Explicit
Binary	KLSH/RMMH	KPCA+LSH
Non-binary		KPCA+PQ

Experimentally, we have shown better Recall@R by replacing implicit binary schemes (KLSH/RMMH) similar to LSH by an explicit embedding followed by LSH The Recall@R can be improved further by replacing LSH by PQ

Using an explicit embedding prior to an efficient Euclidean coding scheme should, therefore, be considered as a baseline when measuring the performance of ANN search methods with small signatures in a kernelized framework.

Appendix: Exact NN Search with KPCA

In this appendix, we mention another benefit of using explicit embedding by KPCA for NN search with a Mercer kernel: it allows one to derive precision bounds on the approximation that can be used in a procedure ensuring exact recovery of the nearest neighbor.

Error Bound for the KPCA Embedding

Precision bounds on the KPCA that are provided in the literature are usually probabilistic bounds on the error between the empirical eigenvalues/ eigenfunctions computed from the matrix $\mathbf{K}(\mathscr{X})$ (denoted σ_i^2 and $\mathbf{u}_i$ in Sect. 2) and their continuous counterparts. Such bounds are difficult to obtain and they are not tight or require oracle knowledge about the implicit embedding [2,24].

However, it is possible to get tighter bounds for our problem by exploiting the data we consider. As shown in Sect. 2, KPCA is an orthogonal projection onto a subspace V in the implicit space $\mathscr{H}$. The approximation error $\epsilon(\mathbf{x}, \mathbf{y}) = K(\mathbf{x}, \mathbf{y}) - \tilde{K}(\mathbf{x}, \mathbf{y})$ can thus be written as a scalar product in the implicit space:

$$\begin{aligned}\epsilon(\mathbf{x},\mathbf{y}) &= K(\mathbf{x},\mathbf{y}) - \tilde{K}(\mathbf{x},\mathbf{y})\\ &= \left\langle \boldsymbol{\Psi}(\mathbf{x}), \boldsymbol{\Psi}(\mathbf{y})\right\rangle_{\mathscr{H}} - \left\langle P_V\boldsymbol{\Psi}(\mathbf{x}), P_V\boldsymbol{\Psi}(\mathbf{y})\right\rangle_{\mathscr{H}}\\ &= \langle P_{V^\perp}\boldsymbol{\Psi}(\mathbf{x}), P_{V^\perp}\boldsymbol{\Psi}(\mathbf{y})\rangle_{\mathscr{H}}, \end{aligned} \tag{20}$$

where $P_{V^\perp}$ is the orthogonal projection on the subspace orthogonal to V. Let us denote

$$R(\mathbf{x}) = ||P_{V^\perp}\boldsymbol{\Psi}(\mathbf{x})||_{\mathscr{H}} = \left(K(\mathbf{x},\mathbf{x}) - ||\tilde{\boldsymbol{\Psi}}(\mathbf{x})||^2\right)^{\frac{1}{2}}. \tag{21}$$

This quantity can be easily computed in practice. The Cauchy–Schwarz inequality provides a bound on $\epsilon(\mathbf{x},\mathbf{y})$ in terms of $R(\mathbf{x})$ and $R(\mathbf{y})$:

$$|\epsilon(\mathbf{x},\mathbf{y})| \leq R(\mathbf{x})R(\mathbf{y}). \tag{22}$$

This bound can be used for nearest neighbor search. Despite its simplicity, we believe it has never been used in such a context. Let us assume that we look for the nearest neighbor of a query $\mathbf{q}$ in the vector dataset $\mathscr{X}$ and let us define

$$\mathbf{x}_1 = \underset{\mathbf{x}\in\mathscr{X}}{\operatorname{argmax}}\ K(\mathbf{q},\mathbf{x}) \quad \text{and} \quad \tilde{\mathbf{x}}_1 = \underset{\mathbf{x}\in\mathscr{X}}{\operatorname{argmax}}\ \tilde{K}(\mathbf{q},\mathbf{x}), \tag{23}$$

that is respectively the nearest neighbor and approximate nearest neighbor of the query $\mathbf{q}$. We can get a lower bound for $K(\mathbf{q},\mathbf{x}_1)$ by noticing

$$K(\mathbf{q},\mathbf{x}_1) \geq K(\mathbf{q},\tilde{\mathbf{x}}_1) \geq \tilde{K}(\mathbf{q},\tilde{\mathbf{x}}_1) - R(\mathbf{q})R(\tilde{\mathbf{x}}_1). \tag{24}$$

For any vector $\mathbf{x} \in \mathscr{X}$, we can get an upper bound on $K(\mathbf{q},\mathbf{x})$:

$$K(\mathbf{q},\mathbf{x}) \leq \tilde{K}(\mathbf{q},\mathbf{x}) + R(\mathbf{q})R(\mathbf{x}). \tag{25}$$

Combining (24) and (25), we obtain that, if for some $\mathbf{x}$ we have

$$\tilde{K}(\mathbf{q},\mathbf{x}) + R(\mathbf{q})R(\mathbf{x}) < \tilde{K}(\mathbf{q},\tilde{\mathbf{x}}_1) - R(\mathbf{q})R(\tilde{\mathbf{x}}_1), \tag{26}$$

then $K(\mathbf{q},\mathbf{x}) < K(\mathbf{q},\mathbf{x}_1)$ and $\mathbf{x}$ cannot be the nearest neighbor of $\mathbf{q}$. The following section describes a procedure that uses this result to avoid kernel computations while still being sure to find the nearest neighbor of a query.

Exact Search Procedure: Description and Illustration

Assuming that the quantities $R(\mathbf{x})$ are computed and stored while performing the embedding of all vectors in dataset $\mathscr{X}$, the nearest neighbor of the query $\mathbf{q}$ is retrieved exactly using the following procedure, which principle is illustrated in Fig. 6:

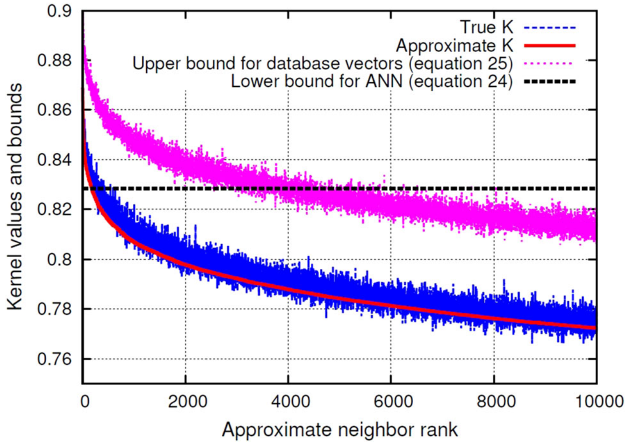

Fig. 6 Exact nearest neighbor search for a given query in a database of 10M 128-dimensional SIFT vectors (extracted from the BIGANN database), embedded in dimension 128 with KPCA, on χ^2 kernel. The reranked vectors are the neighbors which have an *upper* bound value higher than the *lower* bound of the approximate nearest neighbor

1. Compute $\tilde{\boldsymbol{\Psi}}(\mathbf{q})$ and $R(\mathbf{q})$ using Equation (11).
2. Compute $\tilde{K}(\mathbf{q},\mathbf{x}) = \langle\tilde{\boldsymbol{\Psi}}(\mathbf{q}), \tilde{\boldsymbol{\Psi}}(\mathbf{x})\rangle$ for all $\mathbf{x} \in \mathscr{X}$.
3. Find $\tilde{\mathbf{x}}_1 = \underset{\mathbf{x}\in\mathscr{X}}{\operatorname{argmax}}\ \tilde{K}(\mathbf{q},\mathbf{x})$
4. Find all elements $\mathbf{y} \in \mathscr{X}$ such that: $\tilde{K}(\mathbf{q},\tilde{\mathbf{x}}_1) - R(\mathbf{q})R(\tilde{\mathbf{x}}_1) \leq \tilde{K}(\mathbf{q},\mathbf{y}) + R(\mathbf{q})R(\mathbf{y})$. Let us denote by $\mathscr{Y}$ this set of vectors.
5. The true nearest neighbor of the query is $\mathbf{x}_1 = \underset{\mathbf{y}\in\mathscr{Y}}{\operatorname{argmax}}\ K(\mathbf{q},\mathbf{y})$.

Complexity Analysis and Experiments

Let L be the size of the base $\mathscr{X}$, E be the embedding dimension, M be the number of vectors used for KPCA, N be the cardinality of $\mathscr{Y}$, C_K be the cost of computing the kernel between two vectors in the initial space and $C_{\tilde{K}}$ be the cost of computing the approximate kernel between two embedded vectors (which is the cost of a scalar product in the embedded space). The cost of the exact search in the initial space has complexity $\mathscr{O}(C_K L)$.

The exact search in the embedded space as described above has complexity $\mathscr{O}(MC_K)$ for step 1 (scalar products are negligible if $E \ll M$), $\mathscr{O}(NC_{\tilde{K}})$ for steps 2 to 4 (steps 3 and 4 have constant complexity per data vector, which is negligible compared to $C_{\tilde{K}}$ for high-dimensional data), and $\mathscr{O}(NC_K)$ for step 5.

Hence, this method is guaranteed to find the nearest neighbor of $\mathbf{q}$ with complexity

$$\mathscr{O}\left((M+N)C_K + LC_{\tilde{K}}\right). \tag{27}$$

If $C_{\tilde{K}} \ll C_K$ (which is likely to be the case, especially for complicated kernels), and if $M + N \ll L$, we can be sure to retrieve the nearest neighbor while performing much fewer calculations.

This exact search method was tested on 10^6 SIFT descriptors taken from the dataset BIGANN. Experiments show that it reduces the number of kernel values computed. Indeed, for $M = 2,048$, 90 % of the tested queries require less than 15,000 kernel computations, which is a comparison with less than 0.15 % of the base with respect to the distance we are interested in. These comparisons are of negligible complexity, the exact search process in the embedded space thus roughly requires 10^6 scalar products in dimension 128 instead of 10^6 χ^2 kernel computations in dimension 128 for the exact search in the initial space, which leads to a speedup considering that scalar products have more optimized implementations than the χ^2 kernel.

On the other hand, this approach does not perform well on the Imagenet database. For this dataset, the Cauchy–Schwarz inequality is too pessimistic to provide a good selection of the vectors to be reranked. Indeed, experiments showed that the number of reranked vectors reprensented 30 % of the database. In this case, it is still possible to select fewer vectors by relaxing the Cauchy–Schwarz inequality and considering that

$$\alpha R(\mathbf{x})R(\mathbf{y}) \leq \epsilon(\mathbf{x},\mathbf{y}) \leq \beta R(\mathbf{x})R(\mathbf{y}), \qquad (28)$$

where α and β are simply lower and upper bounds for $\cos(\mathbf{x},\mathbf{y})$. By empirically estimating values for α and β that work for many vectors, one can use concentration of measure inequalities to derive theoretical guarantees learned from data. However, one is not able then to precisely ensure the retrieval of the true nearest neighbor of a vector.

References

1. Boiman, O., Shechman, E., Irani, M.: In defense of nearest neighbor based image classification. In: Proceedings of the Computer Vision and Pattern Recognition (CVPR) (2008)
2. Braun, M.: Accurate error bounds for the eigenvalues of the kernel matrix. J. Mach. Learn. Res. **7**, 2303–2328 (2006)
3. Charikar, M.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the ACM Symposium on Theory of Computing, STOC (2002)
4. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Symposium on Computational Geometry, pp. 253–262 (2004)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: Proceedings of the Computer Vision and Pattern Recognition CVPR (2009)
6. Gong, Y., Lazebnik, S.: Iterative quantization: A procrustean approach to learning binary codes. In: Proceedings of the Computer Vision and Pattern Recognition CVPR (2011)
7. Gorisse, D., Cord, M., Precioso, F.: Locality-sensitive hashing for chi2 distance. IEEE Trans. Pattern Anal. Mach. Intell. **34**(2), 402–409 (2012)
8. Grauman, K., Darrell, T.: The pyramid match kernel: Discriminative classification with sets of image features. In: Proceedings of the International Conference on Computer Vision ICCV (2005)
9. He, J., Liu, W., Chang, S.F.: Scalable similarity search with optimized kernel hashing. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '10, pp. 1129–1138. ACM, New York (2010)
10. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. Trans. PAMI **33**(1), 117–128 (2011)
11. Jégou, H., Douze, M., Schmid, C., Pérez, P.: Aggregating local descriptors into a compact image representation. In: Proceedings of the Computer Vision Pattern Recognition CVPR (2010)
12. Jégou, H., Tavenard, R., Douze, M., Amsaleg, L.: Searching in one billion vectors: re-rank with source coding. In: International Conference on Acoustics, Speech and Signal Processing ICASSP. Prague Czech Republic (2011)
13. Joly, A., Buisson, O.: Random maximum margin hashing. In: Proceedings of the Computer Vision and Pattern Recognition CVPR (2011)
14. Knig, H.: Eigenvalues of compact operators with applications to integral operators. Linear Algebra Appl. **84**, 111–122 (1986)
15. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: Proceedings of the International Conference on Computer Vision ICCV (2009)
16. Lowe, D.: Distinctive image features from scale-invariant keypoints. IJCV **60**(2), 91–110 (2004)
17. Maji, S., Berg, A.: Max-margin additive models for detection. In: Proceedings of the International Conference on Computer Vision ICCV (2009)
18. Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., Gool, L.V.: A comparison of affine region detectors. IJCV **65**(1/2), 43–72 (2005)
19. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: VISAPP International Conference on Computer Vision Theory and Applications (2009)
20. Perronnin, F., Sánchez, J., Liu, Y.: Large-scale image categorization with explicit data embedding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition CVPR (2010)
21. Raginsky, M., Lazebnik, S.: Locality-sensitive binary codes from shift-invariant kernels. In: Proceedings of the Advances in Neural Information Processing Systems NIPS (2010)
22. Schölkopf, B., Smola, A.: Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond. MIT Press, Cambridge, MA (2002)
23. Schölkopf, B., Smola, A., Müller, K.R.: Nonlinear component analysis as a kernel eigenvalue problem. Neural Comput., **10**(5), 1299–1319 (1998)
24. Shawe-Taylor, J., Williams, C., Cristianini, N., Kandola, J.: On the eigenspectrum of the gram matrix and the generalization error of kernel pca. IEEE Trans. Inform. Theory **51**(7), 2510–2522 (2005)
25. Sivic, J., Zisserman, A.: Video Google: A text retrieval approach to object matching in videos. In: Proceedings of the International Conference on Computer Vision ICCV (2003)
26. Torralba, A., Fergus, R., Weiss, Y.: Small codes and large databases for recognition. In: Proceedings of the Computer Vision and Pattern Recognition CVPR (2008)
27. Vedaldi, A., Zisserman, A.: Efficient additive kernels via explicit feature maps. In: Proceedings of the Computer Vision and Pattern Recognition CVPR (2010)
28. Vedaldi, A., Zisserman, A.: Efficient additive kernels via explicit feature maps. Trans. PAMI **34**(3), 480–492 (2012)
29. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: Proceedings of the Neural Information Processing Systems NIPS (2008)