# Adaptive Object Retrieval with Kernel Reconstructive Hashing

Haichuan Yang[1] Xiao Bai[1] Jun Zhou[2] Peng Ren[3] Zhihong Zhang[4] Jian Cheng[5]

[1]School of Computer Science and Engineering, Beihang University, China
[2]School of Information and Communication Technology, Griffith University, Australia
[3]College of Information and Control Engineering, China University of Petroleum, China
[4]Software School, Xiamen University, China
[5]Institute of Automation, Chinese Academy of Sciences, China

## Abstract

*Hashing is very useful for fast approximate similarity search on large database. In the unsupervised settings, most hashing methods aim at preserving the similarity defined by Euclidean distance. Hash codes generated by these approaches only keep their Hamming distance corresponding to the pairwise Euclidean distance, ignoring the local distribution of each data point. This objective does not hold for k-nearest neighbors search. In this paper, we firstly propose a new adaptive similarity measure which is consistent with k-NN search, and prove that it leads to a valid kernel. Then we propose a hashing scheme which uses binary codes to preserve the kernel function. Using low-rank approximation, our hashing framework is more effective than existing methods that preserve similarity over arbitrary kernel. The proposed kernel function, hashing framework, and their combination have demonstrated significant advantages compared with several state-of-the-art methods.*

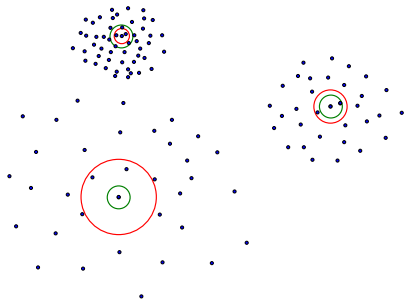


Figure 1. Illustration of the proposed normalized Gaussian kernel method (details are in Section 2). Red circle uses normalized Gaussian kernel on the retrieval radius, and green circle uses the Gaussian kernel. For a given threshold, the original Gaussian kernel gives the same retrieval radius. While using the normalized Gaussian kernel, the retrieval radius is adaptive to the surrounding distribution of the data points. This gives more consistent result for k-NN search.

## 1. Introduction

Nearest neighbor (NN) search is a fundamental tool in computer vision, machine learning and information retrieval. The complexity of NN method on a dataset of size $n$ is $O(n)$, which makes it infeasible on huge dataset. To make NN search scalable, approximate nearest neighbor (ANN) techniques have been proposed. The idea is to find the approximate nearest neighbor rather than the exact one. Locality-sensitive hashing (LSH) was introduced for this purpose [5] and has attracted lots of attention. It maps a vector $x \in R^d$ to a binary string $h \in \{0,1\}^r$, and guarantees that neighbors in the original space having similar binary codes. However, as a data-independent hash method, LSH needs large code length to reduce the false positive rate. This increases the storage costs and degrades query efficiency [22].

Recently, many data-dependent hashing methods have been proposed, which learn hash functions from a training set. They normally optimize an objective function which explicitly preserves data similarity using Hamming distance [22, 8, 12]. In the unsupervised setting, the most widely used similarity metrics include Euclidean distance [2, 3, 6, 4], or the Gaussian kernel as a monotonic function for Euclidean distance [22, 24, 12]. Several supervised hashing methods have also been proposed based on semantic similarity preservation [20, 21, 14, 11] and have shown promising results. In practice, we often need to find the k-nearest neighbors (k-NN) of a query. Because the distances to the nearest neighbors vary a lot for different data samples, using threshold based on Euclidean distance or Gaussian kernel returns different number of neighbors, as

shown in Figure 1. If we want to get consistent results, the local distribution of data should be taken into consideration. In [16], Qin *et al*. proposed an extended Euclidean distance metric by normalizing the Euclidean distance locally so as to obtain a unified degree of measurement across different queries. Inspired by their method, we propose an adaptive similarity function to replace Euclidean distance or Gaussian kernel. For different data points, this similarity measure can adapt to their local distribution.

Kernel methods enable flexible models be built for a variety of applications by adopting different kernel functions. Some hashing methods have already taken kernel function as input. Kulis and Darrell proposed a binary reconstructive embedding (BRE) approach [8] with a kernelized hashing function. The objective of BRE is minimizing the reconstruction error between the pairwise Euclidean distances in the kernel space and the Hamming distances of the binary codes in the mapped feature space. Both the original distance and the Hamming distances are scaled to the same range in $[0, 1]$. However, this objective function is non-convex, and the reconstructions are binary which makes the objective not differentiable. When a coordinate-descent optimization scheme is used [8], the objective function converges to a local minimum. Its training time is highly dependent on the amount of the input data pairs and that has constrained the scale of the training set. Besides BRE, Kernelized Locality-Sensitive Hashing (KLSH) [9] can also preserve the kernel similarity by generalizing the principle in LSH [2] to arbitrary reproducing kernel Hilbert spaces. Its time complexity is much lower than BRE, but it has the same problem as LSH in that its performance degrades with compact code. In [12], Liu *et al*. proposed a graph-based hashing method called Anchor Graph Hashing (AGH). It builds a neighborhood graph and uses graph Laplacian to solve the relaxed graph partition problem. For efficient training and indexing, this method approximates the neighborhood graph by a sparse anchor graph. Although it can take any customized similarity including kernels, the sparsity of the anchor graph limits the degree of the approximation.

In this paper, we address the problem of unsupervised learning of hashing function. There are two main contributions with our method. Firstly, We explore the idea of normalization on the Gaussian kernel, which is motivated by [16], so that a new similarity function is proposed. This similarity function is proved to be a positive semi-definitive (PSD) kernel. Secondly, we present a hashing method which aims at reconstructing the kernel function using binary code. This method enables linear training time with respect to the size of the training set, and enables constant time for indexing by the proposed hash function. In the experiments, we show that both proposed similarity function and the hashing method are more effective than several baseline methods. When the normalized Gaussian kernel is combined with proposed hashing framework, our method outperforms the state-of-the-art unsupervised hashing methods.

The rest of the paper is organized as follows. We introduce a normalized Gaussian kernel in Section 2, and present the hashing framework in Section 3. In Section 4, we show the experimental results, and finally conclude the paper and discuss the future work in Section 5.

## 2. Normalized Gaussian Kernel

In this section, we introduce a normalized Gaussian kernel. We show that this kernel has positive semi-definite property.

### 2.1. Symmetric Adaptive Similarity

Euclidean distance is the most common metric to measure the similarity between local feature descriptors. In order to derive a measure that is adaptive to the query feature, Qin *et al*. [16] proposed the normalized the Euclidean distance as follows

$$dn(x_i, x_j) = \frac{d(x_i, x_j)}{E(d(x_i))} \tag{1}$$

where $d(x_i, x_j)$ and $dn(x_i, x_j)$ are the Euclidean distance between $x_i$ and $x_j$ and its normalized version, respectively. $E(d(x_i))$ is the expected Euclidean distance between $x_i$ and its non-matching features which can be randomly sampled. This normalized Euclidean distance aims at obtaining unified degree of measurement across different queries. Notice this normalized Euclidean distance is not symmetric, which means that $dn(x_i, x_j) \neq dn(x_j, x_i)$.

Motivated by the above normalized method, we propose a normalized Gaussian kernel. A Gaussian kernel can be expressed as

$$\kappa_G(x_i, x_j) = exp(-(d(x_i, x_j))^2/2\sigma^2) \tag{2}$$

where $\sigma$ is used to scale the exponential function. It maps the data points to an infinite dimensional Hilbert space, which derives a nonlinear model. Gaussian kernel also benefit from the special property of the squared exponential curve, which makes it more natural than the Euclidean distance.

In [16], normalization of Euclidean distance is performed based on the non-matching features of the query because estimating the distance distribution between the query and its correspondences is intractable. While this approach is only established for local feature retrieval, we need an adaptive similarity which considers the surrounding distribution of the data and improves the performance in k-NN search. Inspired by Bag-of-Words method [10] which uses clustering method to cluster local features as different

"visual words", we treat the points in the same cluster as correspondences to each other. If we designate the cluster center index of data point $a$ as $\iota(a)$, we can get the expected similarity $C_{\kappa_G(a)}$ of $a$ to its corresponding points by averaging the Gaussian kernel between $a$ and points in cluster $\iota(a)$. Furthermore, if the distribution of pairwise Gaussian kernel function values in the same cluster is approximately uniform, we can obtain $C_{\kappa_G(a)}$ via:

$$C_{\kappa_G(a)} = \frac{\sum_{\iota(x_i)=\iota(a),\iota(x_j)=\iota(a)} \kappa_G(x_i, x_j)}{\#(\{(x_i, x_j)|\iota(x_i) = \iota(a), \iota(x_j) = \iota(a)\})} \quad (3)$$

where $\#(\cdot)$ is the number of elements in the set.

Notice that $C_{\kappa_G(a)}$ only relies on $\iota(a)$, for simplicity, we denote it as $C_{\iota(a)}$, and $C_i = C_{\iota(a)=i}$ defines the expected similarity of cluster $i$. If we use the corresponding points to normalize the Gaussian kernel, the generated similarity function between $a$ and $b$ is $\kappa_G(a, b)/C_{\iota(a)}$, while normalized similarity between $b$ and $a$ is $\kappa_G(a, b)/C_{\iota(b)}$. This similarity is asymmetric because $C_{\iota(a)}$ and $C_{\iota(b)}$ are not always the same. To overcome this problem, we set the geometric mean $\sqrt{C_{\iota(a)}C_{\iota(b)}}$ as the denominator. Formally, if the total number of clusters is $l$, let $\delta(a) \in \{0, 1\}^{l\times 1}$ indicate which cluster that $a$ belongs to. The normalized Gaussian kernel for points $a$ and $b$ is

$$\kappa_{Gn}(a, b) = (\delta(a)^T D\delta(b))\kappa_G(a, b) \quad (4)$$

where $D$ is an $l \times l$ matrix and $D_{ij} = 1/\sqrt{C_i C_j}$. We will prove that $\kappa_{Gn}$ is a valid kernel. In this paper, we use the k-means algorithm to obtain the cluster centers and $\delta(a)$. Figure 1 shows a simple example on the adaptability of the proposed similarity function.

### 2.2. Positive Semi-definite Property

Having defined a symmetric normalized similarity measure, we prove that this kernel is positive semi-definite (PSD), which is necessary for every valid kernel function.

**Definition 2.1.** *Let $\mathcal{X}$ be a nonempty set. Function $f : (\mathcal{X} \times \mathcal{X}) \to \mathbb{R}$ is a positive semi-definite kernel if and only if $f$ is symmetric and for all $m \in \mathbb{N}, x_i, ..., x_m \in \mathcal{X}$, and $c_i \in \mathbb{R}$*

$$\sum_{i,j=1}^{m} c_i c_j f(x_i, x_j) \geq 0 \quad (5)$$

Based on this definition, we introduce a theorem which can be proved directly by the the Schur product theorem [19]:

**Theorem 2.1.** *If $\kappa_1$, $\kappa_2$ are all positive semi-definite kernels. Then their product $\kappa_1 \cdot \kappa_2$ is a positive semi-definite kernel.*

Recall matrix $D$ defined in equation (4), we rewrite it as $D = \gamma\gamma^T$ where $\gamma$ is an $l \times 1$ vector and $\gamma_i = 1/\sqrt{C_i}$. Then $\delta(a)^T D\delta(b)$ becomes $(\gamma^T\delta(a))(\gamma^T\delta(b))$. Because $\gamma^T\delta(a)$ is a feature mapping of $a$, $\delta(a)^T D\delta(b)$ can be considered as the inner product between features $\gamma^T\delta(a)$ and $\gamma^T\delta(b)$ which is obviously PSD. Since we have already known that the Gaussian kernel is PSD, by using the Theorem 2.1, the function $\kappa_{Gn}(a, b)$ defined in equation (4) is also PSD.

## 3. Kernel Reconstructive Hashing

After defining the similarity measure in a kernel form, now we show how to perform fast retrieval based on this metric. In this section, we propose a hashing scheme called Kernel Reconstructive Hashing (KRH) which can preserve the similarity defined by an arbitrary kernel using compact binary code.

### 3.1. Objective Formulation

First of all, we formulate the objective function. Let $X = \{x_1, x_2, ..., x_n\}$ be a training set of size $n$, and $\kappa(x_i, x_j)$ be a kernel function over $X$. Our objective is learning a set of $r$ hash functions which generate the binary code of $x_i$ as a $\{-1, 1\}^{1\times r}$ vector $\tilde{x}_i = [h_1(x_i), h_2(x_i), ..., h_r(x_i)]$. The Hamming distance $d_h(\tilde{x}_i, \tilde{x}_j)$ between the binary codes for all instance pairs in $X$ should has a negative correlation with the similarity represented by the kernel function $\kappa(x_i, x_j)$, *i.e.*, a smaller $d_h(\tilde{x}_i, \tilde{x}_j)$ corresponds to a larger $\kappa(x_i, x_j)$. In [11], Liu *et al*. showed that the inner product $\langle\tilde{x}_i, \tilde{x}_j\rangle = \tilde{x}_i\tilde{x}_j^T$ between binary codes has a one-to-one correspondence with the Hamming distance: $\langle\tilde{x}_i, \tilde{x}_j\rangle = r - 2d_h(\tilde{x}_i, \tilde{x}_j)$. This implies that the inner product also has a negative correlation with the Hamming distance. Therefore, our objective is to use the inner product $\langle\tilde{x}_i, \tilde{x}_j\rangle$ to reconstruct the kernel $\kappa(x_i, x_j)$. To do so, we minimize the reconstruction error:

$$\min \sum_{x_i,x_j\in X} (s \cdot \langle\tilde{x}_i, \tilde{x}_j\rangle - \kappa(x_i, x_j))^2 \quad (6)$$

where $s$ is a positive number for scaling the inner product. It should be noted that although $\langle\tilde{x}_i, \tilde{x}_j\rangle \in [-r, r]$, the kernel $\kappa(x_i, x_j)$ is not bounded by this range.

### 3.2. Real-valued Relaxation

The objective function in equation (6) is difficult to solve because optimization on the binary code is not straightforward. However, if we do not constrain the output to be binary, the problem will be much easier to solve. Here, we use a $\mathbb{R}^{1\times r}$ vector $\hat{x}_i$ to substitute $\tilde{x}_i$ in equation (6)

$$\min \sum_{x_i,x_j\in X} (\langle\hat{x}_i, \hat{x}_j\rangle - \kappa(x_i, x_j))^2 \quad (7)$$

The scaling factor $s$ in formula (6) is absorbed by $\hat{x}_i$. This is a basic metric multidimensional scaling (MDS) problem [17]. It's optimal solution can be obtained by the spectral decomposition of the $n \times n$ kernel matrix $K$ whose entries $K_{ij} = \kappa(x_i, x_j)$. If $\lambda_1, \lambda_2, ..., \lambda_r > 0$ are the top $r$ largest eigenvalues, and their corresponding eigenvectors are $v_1, v_2, ..., v_r \in \mathbb{R}^{n\times 1}$, the optimal solution is

$$(\hat{x}_i)_\alpha = \sqrt{\lambda_\alpha}(v_\alpha)_i \quad \text{for } i = 1, 2, ..., n, \alpha = 1, 2, ..., r \tag{8}$$

Using standard algorithm to perform the spectral decomposition of the $n \times n$ kernel matrix $K$ requires $O(n^3)$ time [23]. This is very infeasible for large training set. A possible solution to this problem is using the low-rank matrix approximation. We adopt the Nyström method to construct a low-rank matrix $\tilde{K}$ for $K$ [23]. Randomly choose $m$ ($m < n$) columns of $K$ to form an $n \times m$ sub-matrix $A$, let $I$ be the set of indices of the sampled columns, and $M$ be an $m \times m$ sub-matrix of $K$ such that $M_{ij} = K_{ij}, i \in I, j \in I$. Then the low-rank approximation for matrix $K$ is $\tilde{K} = AM^{-1}A^T$. If $M$ is not invertible, Moore-Penrose generalized inverse is used to substitute $M^{-1}$. In our method, the operation for the inverse and the generalized inverse are similar, so we assume $M$ is invertible. Approximation $\tilde{K}$ is exactly the same as $K$ when the rank of $K$ is equal to the rank of $M$. Using this approximation, we perform the eigen decomposition on $\tilde{K}$ instead of $K$.

Here we show how to solve the eigenvalues and eigenvectors on $\tilde{K}$ efficiently. Firstly we perform the eigen decomposition $M = Z\Sigma Z^T$, where $\Sigma$ is a diagonal matrix containing the eigenvalues of $M$, and columns of $Z$ are the corresponding eigenvectors. Since $M$ is a positive semi-definite matrix and is invertible, we have $M^{-1} = Z\Sigma^{-1}Z^T = Z\Sigma^{-1/2}\Sigma^{-1/2}Z^T$. Furthermore, introducing an $m \times m$ matrix $B = Z\Sigma^{-1/2}$, we have $M^{-1} = BB^T$. Designating $AB$ as an $n \times m$ matrix $F$, we can get $\tilde{K} = FF^T$. Introducing $E = F^TF$, $E$ is an $m \times m$ matrix whose eigenvalues and eigenvectors can be computed in $O(m^3)$. Directly perform eigen decomposition on $E$

$$E = U\Lambda U^T \tag{9}$$

where $\Lambda$ and $U$ are eigenvalues and eigenvectors of $E$. Based on the singular value decomposition (SVD) of $F$, we know that $FF^T$ has the same nonzero eigenvalues as $F^TF$, so the $m$ positive eigenvalues of $\tilde{K}$ are diagonal entries of $\Lambda$: $\lambda_1, \lambda_2, ..., \lambda_m$, and the rest $n - m$ eigenvalues are all zero. If $V$ is an $n \times m$ matrix whose columns are eigenvectors of $\tilde{K}$ corresponding to $\lambda_1, \lambda_2, ..., \lambda_m$, by SVD, we have:

$$V = FU\Lambda^{-1/2} = ABU\Lambda^{-1/2} \tag{10}$$

From equations (9) and (10), we can obtain the $m$ positive eigenvalues and their corresponding eigenvectors of $\tilde{K}$. Pick $r$ largest ones, $\hat{x}_i$ for $x_i \in X$ can be computed according to equation (8).

This procedure only generates the $r$-dimension real-valued reconstructions for the training data. For an arbitrary input, we generalize the eigenvectors to the eigenfunctions $\phi_\alpha(\cdot), \alpha = 1, 2, ..., r$ that for $x_i$, it gives the $i$-th entry of the eigenvector for eigenvalue $\lambda_\alpha$. Given a novel input $y$, we also obtain the eigenfunctions using the Nyström method [1]

$$\phi_\alpha(y) = \sum_{i=1}^{n} \kappa(y, x_i)(v_\alpha)_i/\lambda_\alpha \tag{11}$$

This eigenfunction costs $O(n)$ time for each data point, we can decrease the time complexity to $O(m)$ by Nyström approximation. Let $\varphi(y)$ be a $1 \times n$ vector that $\varphi(y)_i = \kappa(y, x_i)$, and $\varphi(y)_I$ be a $1 \times m$ vector whose entries correspond to the sampled indices set $I$, then the Nyström approximation of $\varphi(y)$ is $\varphi(y)_I M^{-1}A^T$. Putting them together, we get the approximate eigenfunction $\tilde{\phi}_\alpha(y)$:

$$\begin{aligned}\tilde{\phi}_\alpha(y) &= (\varphi(y)_I M^{-1}A^T V\Lambda^{-1})_\alpha \\ &= (\varphi(y)_I BB^T A^T ABU\Lambda^{-3/2})_\alpha \\ &= (\varphi(y)_I BEU\Lambda^{-3/2})_\alpha \\ &= (\varphi(y)_I BU\Lambda^{-1/2})_\alpha \end{aligned} \tag{12}$$

Let $J$ be the set of $\alpha$ that $\lambda_\alpha$ is in the $r$ largest eigenvalues. According to equations (8) and (12), we get the $1 \times r$ real-valued reconstruction for $y$:

$$\hat{y} = (\varphi(y)_I BU)_J \tag{13}$$

Notice that this formula is established no matter $y$ is a training data or a novel input.

### 3.3. Minimizing the Quantization Loss

We have already got the optimal solution for the problem in equation (7). Now we need to transform the real-valued result to be binary. The most common method is using $\tilde{y} = sign(\hat{y})$. However, sometimes, directly using the $sign(\cdot)$ may make $\hat{y}$ be largely deviated from $\tilde{y}$. To address this problem, we define the following quantization loss $Q(\cdot)$

$$Q(G) = \|G - \tilde{s} \cdot sign(G)\|_F^2 \tag{14}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. $G$ is an arbitrary real-valued matrix, $sign(G)$ is a same sized matrix as $G$ that $sign(G_{ij}) = 1$ for $G_{ij} \geq 0$, and $sign(G_{ij}) = -1$ otherwise. Scaling factor $\tilde{s}$ is introduced to match the range between the real-valued matrix and the binary matrix. A similar loss function was adopted in [3], which, however, does not have the scaling factor. This is not optimal because the range of $G$ and $sign(G)$ may be very different.

Let $\hat{X}$ be an $n \times r$ matrix whose rows are the real-valued reconstructions $\hat{x}_i$, then $Q(\hat{X})$ is the quantization loss for

our solution. Recall that $\hat{x}_i, i = 1, 2, ..., n$ target at approximating the kernel $\kappa(x_i, x_j)$ by inner product $\hat{x}_i \hat{x}_j^T$. Because multiplying an arbitrary orthogonal matrix $R$ does not change the inner product $\hat{x}_i \hat{x}_j^T = \hat{x}_i R (\hat{x}_j R)^T$, $\hat{X}R$ has the same effect as $\hat{X}$. Then we need to find the matrix $R$ which gives the minimum $Q(\hat{X}R)$. A similar problem was proposed and solved in [3] by iteratively updating $R$ using the classic orthogonal procrustes method [18]. In our method, we iteratively update both $R$ and $\tilde{s}$. In each iteration $i$, we solve the optimal orthogonal matrix $R_{(i)}$ by

$$\arg\min_{R_{(i)}} \|\hat{X}R_{(i)} - \tilde{s}_{(i-1)} \cdot sign(\hat{X}R_{(i-1)})\|_F^2 \qquad (15)$$

This is a classic Orthogonal Procrustes problem [18] and can be solved by singular value decomposition: if the SVD of $\hat{X}^T(\tilde{s}_{(i-1)} sign(\hat{X}R_{(i-1)}))$ is $\bar{U}S\bar{V}^T$, then $R_{(i)}$ should be $\bar{U}\bar{V}^T$. The optimized $\tilde{s}_{(i)}$ can be obtained by setting the partial derivative $\partial Q(\hat{X}R_{(i)})/\partial \tilde{s} = 0$, such that

$$\tilde{s}_{(i)} = \frac{tr(sign(\hat{X}R_{(i)})^T \hat{X}R_{(i)})}{tr(sign(\hat{X}R_{(i)}) sign(\hat{X}R_{(i)})^T)} \qquad (16)$$

We initialize $R_{(0)}$ as a random orthogonal matrix and $\tilde{s}_{(0)}$ as 1. Minimizing the quantization loss makes $\tilde{s} \cdot sign(\hat{X}R)$ approximate $\hat{X}R$, then $\tilde{s}^2 \langle sign(\hat{x}_i R), sign(\hat{x}_j R) \rangle$ is approximate to inner product $\langle \hat{x}_i, \hat{x}_j \rangle$, which is an approximation of kernel $\kappa(x_i, x_j)$. Therefore, we get an approximate solution of objective function in (6): $\tilde{x}_i = sign(\hat{x}_i R)$ and $s = \tilde{s}^2$.

### 3.4. Complexity Analysis

Here we analysis the time complexity of KRH. In the training process, if $A$ is given, $M^{-1}$ and its eigen decomposition can be computed in $O(m^3)$. Matrix multiplication for obtaining $E$ needs $O(n \times m^2)$, and performing eigen decomposition for $E$ costs $O(m^3)$. So obtaining $BU$ in equation (13) needs at most $O(n \times m^2)$. In the procedure of minimizing quantization loss, each iteration needs $O(r^2 \times n)$. In the experiments, we have found that 50 iterations is usually enough to get the quantization loss be converged. In the search phase, for each query $y$, computing $\hat{y}$ in equation (13) costs $O(m \times r)$, and the orthogonal transformation needs $O(r^2)$. Therefore, the training time is linear with the size of training set, and binary code can be generated in constant time.

The proposed KRH method is summarized in Algorithm 1.

**Algorithm 1:** Kernel Reconstructive Hashing

**Data**: training data $X$, kernel function $\kappa(\cdot)$ , code length $r$
**Result**: $r$ hash functions $h_p(\cdot), p = 1, 2, ..., r$
Randomly sample $m$ indices $I$ to generate sub-matrices $A$, $M$;
Eigen decomposition: $M = Z\Sigma Z^T$;
Assign $B = Z\Sigma^{-1/2}$, $F = AB$;
Assign $E = F^T F$, solve its eigenvectors for the $r$ largest eigenvalues, build $m \times r$ matrix $U_J$;
Assign $\hat{X} = ABU_J$, assign $R_{(0)}$ as random orthogonal matrix, and $\tilde{s}_{(0)} = 1$;
**while** $R, \tilde{s}$ *is not converged* **do**
  Update $R$ by equation (15);
  Update $\tilde{s}$ by equation (16);
**end**
Define row vector $\varphi(y)$ as in Section 3.2, and hash functions $[h_1(y), h_2(y), ..., h_r(y)] = \varphi(y)_I BU_J R$

## 4. Experiment Results

We evaluate the performance of our method on two datasets. The first one is SIFT-1M from [21], which contains 1 million local features represented as 128 dimensional SIFT descriptors [13]. We selected $10,000$ random samples as queries and left the others as the target database. The second is CIFAR-10 dataset [7] that consists of 60,000 32×32 color images in 10 classes. Figure 2 shows some sample images in this dataset. We used 384 dimensional GIST descriptor [15] to represent each image. For retrieval, $1,000$ images were randomly sampled as queries and the rest were taken as the database.

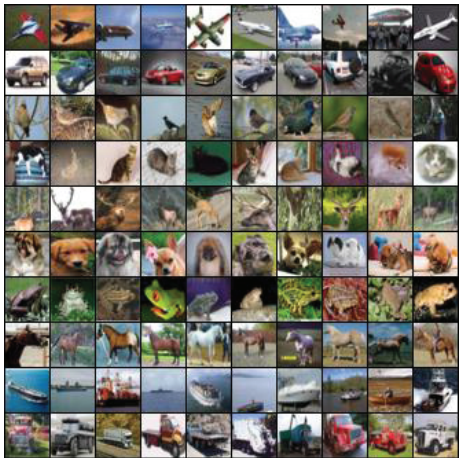
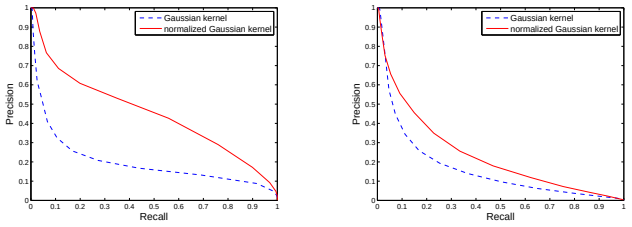


Figure 2. Sample images in CIFAR-10 dataset. Each row contains 10 images of the same class.

To evaluate the performance of different methods under comparison, we have used precision-recall and recall curves, in which the precision and recall are calculated by

$$\text{precision} = \frac{\text{Number of retrieved relevant pairs}}{\text{Total number of retrieved pairs}} \qquad (17)$$




(a) SIFT-1M (b) CIFAR-10

Figure 3. Precision-recall curves of k-NN search on SIFT-1M and CIFAR-10 when different distance metrics are compared.

$$\text{recall} = \frac{\text{Number of retrieved relevant pairs}}{\text{Total number of all relevant pairs}} \tag{18}$$

In the comparison with state-of-the-art hashing methods, we also use the recall curves. All the evaluations are based on hash lookup usage [3, 6].

In our method, there are some parameters to be set. For the proposed kernel, the length $l$ of $\delta(a)$ in equation (4) and the constant $\sigma$ in the Gaussian kernel function in equation (2) can be selected empirically. In the experiments, we found that the accuracy is relatively stable when $l \geq 30$, and large $l$ makes $\kappa_{Gn}$ complex, so we set $l = 30$ in all the experiments. For fair comparison, we set $\sigma$ as the averaged Euclidean distance for all methods which use Gaussian kernel. In KRH, there is nearly no difference in performance when the sample number $m \geq 1000$, so we set $m = 1000$. For parameters in other methods, we set them the same values as in their papers correspondingly.

### 4.1. Find k-NN Using Normalized Gaussian Kernel

In this section, we show the superiority of the proposed normalized Gaussian kernel in the k-NN search task. For every query, we took the 100 nearest neighbors in Euclidean distance as the ground-truth. By decreasing the threshold of similarity measures, we get various precision and recall values by equations (17) and (18), and generate the precision-recall curves. In this setting, Gaussian kernel and the Euclidean distance is equivalent.

Figure 3 shows the results using Gaussian kernel and normalized Gaussian kernel as the similarity function. It can be seen from the figure that the proposed normalized Gaussian kernel has significantly outperformed the non-normalized counterpart.

### 4.2. Performance of Kernel Reconstructive Hashing

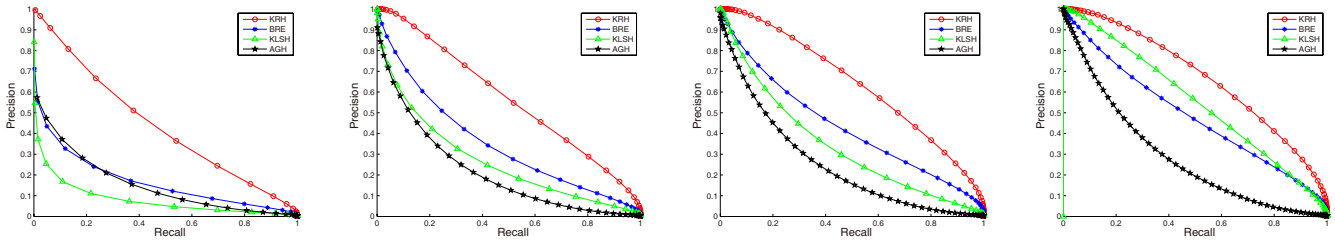
Note that our KRH can use arbitrary kernel function, we compared it with several similar hashing methods: BRE [8], KLSH [9] and AGH [12]. For all methods, we used the same Gaussian kernel as in equation (2). Here we wanted to evaluate the degree of similarity preserving based on the kernel, therefore, we defined $\tau$ as the average Euclidean distance between queries and their 50th nearest neighbors. For each query $y$, we set all points $x_i$ whose kernel $\kappa_G(y, x_i) \geq exp(-\tau^2/2\sigma^2)$ as the true neighbors. The precision-recall curves under different code length are illustrated in Figures 4 and 5. We find that our method has significant advantages over the competitors on the SIFT-1M dataset. On the CIFAR-10 dataset, our method leads the other method with great margin when the precision is high. It should be mentioned that KLSH is quite sensitive to code length.

### 4.3. Comparison with the State-of-the-arts Hashing Methods

Finally, we incorporate the KRH scheme with the normalized Gaussian kernel, and compare it with several state-of-the-art unsupervised hashing methods. These methods are locality sensitive hashing (LSH) [2], spectral hashing (SH) [22], unsupervised sequential projection learning hashing (USPLH) [21], iterative quantization (ITQ) [3], Isotropic Hashing (IsoHash) [6] and K-means Hashing (KMH) [4]. As in [21], the top 2 percentile nearest neighbors in Euclidean distance are taken as the true positive.

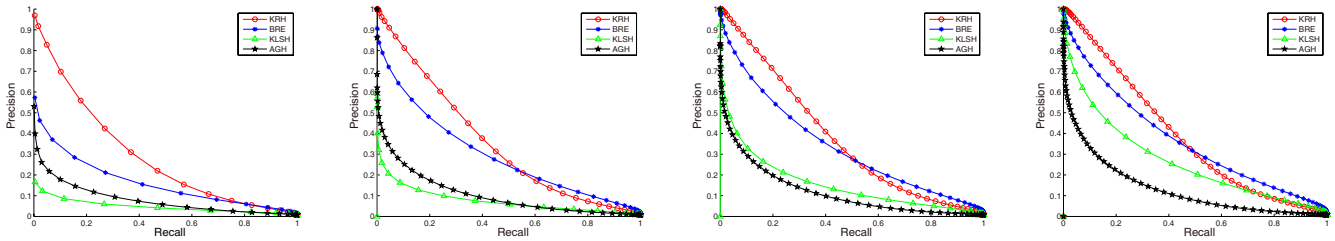
The resulting recall curves are shown in Figures 6 and 7. Since we follow the hash lookup usage, the recall values are obtained by evaluating the recall for the average first $N$ Hamming neighbors of all queries. Due to the space limitation, we do not show the precision-recall curves but substitute them with a more succinct measurement, mean averaged precision (mAP) which is the area under the precision-recall curve, as shown in Figure 8. We can see that the performance from various methods being compared on the two datasets follows the same trend. The proposed method has outperformed all the other methods with clear margin. It can be observed from Figure 8 that the advantage of our method become larger when the code length increase. This is because large code length gives more precise approximation of the normalized Gaussian kernel, which is reasonable for k-NN search. Among all other methods, ITQ is the most competitive. Performance of LSH grows fast with the increase of code length, while USPLH performs worse with longer code. We also found that our method can retrieve more semantically similar images. Figure 9 shows some sample retrieved images on CIFAR-10.

To compare the efficiency of various methods, we show the training time on CIFAR-10 in Table 1. All experiments were implemented with MATLAB code and ran on a PC with Intel Core-i7 3.4GHz CPU, 16GB RAM. Since BRE and KLSH cannot work on large training set, the size of training set for them were based on the settings in [8, 9]. For methods using kernelized data, we do not count the time of kernel computation in this training time. We can see that BRE, USPLH and KMH are relatively time consuming compared with other methods.




(a) 32 bits (b) 64 bits (c) 96 bits (d) 128 bits

Figure 4. Precision-recall curves for kernel similarity preserving on SIFT-1M.




(a) 32 bits (b) 64 bits (c) 96 bits (d) 128 bits

Figure 5. Precision-recall curves for kernel similarity preserving on CIFAR-10.

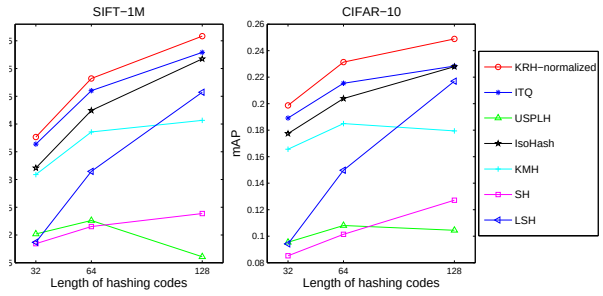



Figure 8. Mean averaged precision of several unsupervised hashing methods on SIFT-1M and CIFAR-10.

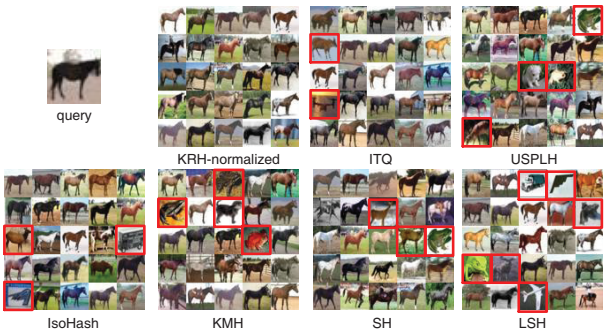


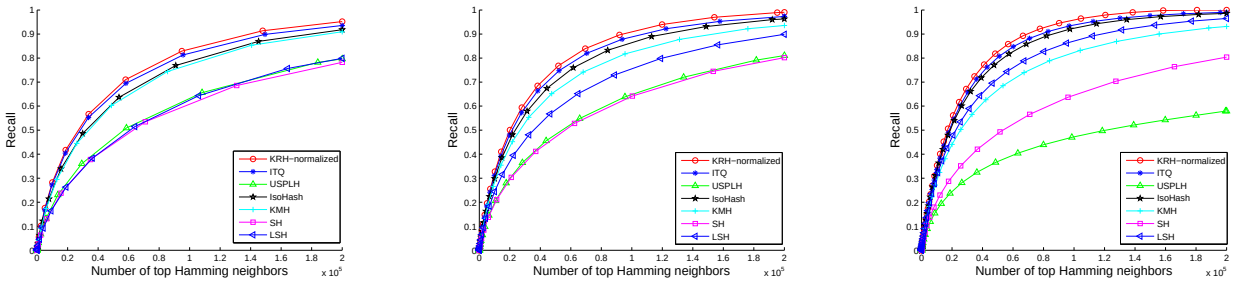

Figure 9. Qualitative results on CIFAR-10. We used 64-bit hashing codes, and show the false positives in red rectangle.

| #bit | 32 bits | 64 bits | 96 bits | 128 bits |
|---|---|---|---|---|
| KRH | 5.63 | 8.27 | 11.02 | 14.20 |
| BRE | 53.61 | 279.23 | 492.81 | 931.54 |
| KLSH | 7.83 | 8.21 | 8.58 | 8.75 |
| AGH | 4.26 | 4.43 | 4.97 | 5.33 |
| ITQ | 2.14 | 4.04 | 6.41 | 9.14 |
| USPLH | 35.38 | 72.24 | 110.62 | 143.94 |
| IsoHash | 0.35 | 0.51 | 0.72 | 1.61 |
| KMH | 289.85 | 313.47 | 338.21 | 358.42 |
| SH | 0.54 | 0.58 | 0.70 | 0.85 |

Table 1. Training time (seconds) on CIFAR-10.

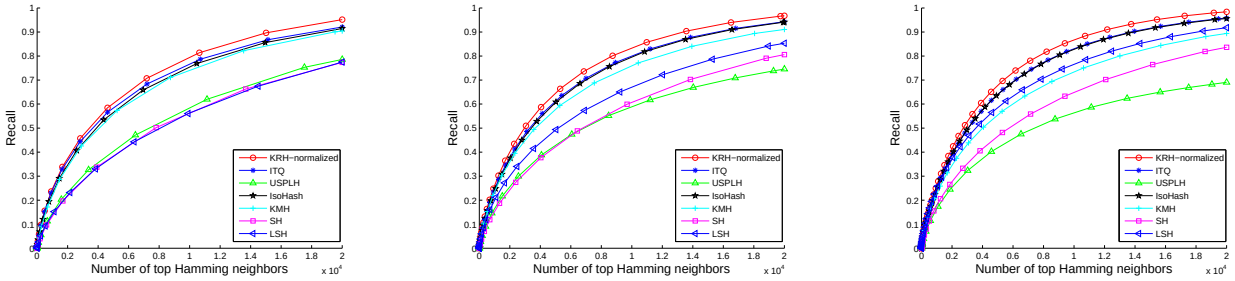
## 5. Conclusion and Future Work

We have presented a novel normalized Gaussian kernel function and a kernel reconstructive hashing framework KRH which can reconstruct the kernel between data points using binary code. By considering the local distribution around data point, the proposed kernel function is consistent with the k-NN search. Incorporating this kernel in KRH, we can improve the ANN performance under the k-NN protocol. KRH costs linear time to train the efficient hash function with respect to the size of the training set by low-rank approximation. Their effectiveness have been validated on two public datasets. In the future, we will try more kernel based models with the proposed normalized Gaussian kernel. Since our method can reconstruct the pairwise value of kernel function by binary codes, we believe its usage is not constrained on ANN search, for example, it can be useful for some scenarios where fast kernel computation is required.




(a) 32 bits (b) 64 bits (c) 128 bits

Figure 6. Recall curves of several unsupervised hashing methods on SIFT-1M.




(a) 32 bits (b) 64 bits (c) 128 bits

Figure 7. Recall curves of several unsupervised hashing methods on CIFAR-10.

**Acknowledgements** This work was supported by the NSFC projects (No. 61370123, 61105002 and 61105005), the Australian Research Councils DECRA Projects funding scheme (project ID DE120102948) and the Open Project Program of the National Laboratory of Pattern Recognition (NLPR).

## References

[1] Y. Bengio, O. Delalleau, N. Le Roux, J.-F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. *Neural Computation*, 16(10):2197–2219, 2004.

[2] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, 2004.

[3] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[4] K. He, F. Wen, and J. Sun. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In *CVPR*, 2013.

[5] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998.

[6] W. Kong and W.-J. Li. Isotropic hashing. In *NIPS*, 2012.

[7] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[8] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, 2009.

[9] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.

[10] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006.

[11] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, 2012.

[12] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *ICML*, 2011.

[13] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[14] M. Norouzi and D. M. Blei. Minimal loss hashing for compact binary codes. In *ICML*, 2011.

[15] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42(3):145–175, 2001.

[16] D. Qin and C. W. L. van Gool. Query adaptive similarity for large scale object retrieval. In *CVPR*, 2013.

[17] L. K. Saul, K. Q. Weinberger, J. H. Ham, F. Sha, and D. D. Lee. Spectral methods for dimensionality reduction. *Semisupervised learning*, pages 293–308, 2006.

[18] P. H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

[19] J. Schur. Bemerkungen zur theorie der beschränkten bilinearformen mit unendlich vielen veränderlichen. *Journal für die reine und Angewandte Mathematik*, 140:1–28, 1911.

[20] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, 2010.

[21] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *ICML*, 2010.

[22] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2008.

[23] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *NIPS*, 2001.

[24] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *SIGIR*, 2010.