Adaptive Hash Retrieval with Kernel Based Similarity

Xiao Bai\textsuperscript{a}, Cheng Yan\textsuperscript{a}, Haichuan Yang\textsuperscript{a}, Lu Bai\textsuperscript{b}, Jun Zhou\textsuperscript{c}, Edwin Robert Hancock\textsuperscript{d}

\textsuperscript{a}School of Computer Science and Engineering, Beihang University, Beijing, China
\textsuperscript{b}School of Information, Central University of Finance and Economics, Beijing, China
\textsuperscript{c}School of Information and Communication Technology, Griffith University, Nathan, QLD, Australia
\textsuperscript{d}Department of Computer Science, University of York, York, UK

Abstract
Indexing methods have been widely used for fast data retrieval on large scale datasets. When the data are represented by high dimensional vectors, hashing is often used as an efficient solution for approximate similarity search. When a retrieval task does not involve supervised training data, most hashing methods aim at preserving data similarity defined by a distance metric on the feature vectors. Hash codes generated by these approaches normally maintain the Hamming distance of the data in accordance with the similarity function, but ignore the local details of the distribution of data. This objective is not suitable for k-nearest neighbor search since the similarity to the nearest neighbors can vary significantly for different data samples. In this paper, we present a novel adaptive similarity measure which is consistent with k-nearest neighbor search, and prove that it leads to a valid kernel if the original similarity function is a kernel function. Next we propose a method which calculates hash codes using the kernel function. With a low-rank approximation, our hashing framework is more effective than existing methods that preserve similarity over an arbitrary kernel. The proposed similarity function, hashing framework, and their combination demonstrate significant improvement when compared with several alternative state-of-the-art methods.

Keywords:
Hashing, K-NN, Kernel, Binary Indexing, Normalized Euclidean Distance
1. Introduction

Nearest neighbor (NN) search is a flourishing area in computer vision, machine learning and information retrieval. It can be directly used to implement simple retrieval algorithms, or be adopted as a procedure in more complex systems. The time complexity of the NN method on a dataset of size $n$ is $O(n)$, which means it is infeasible in some applications, e.g., real-time retrieval on large datasets. For data with relatively low dimensionality, this problem can be solved using tree based methods such as the binary search tree [4]. However, the dimensionality of most widely used image descriptors, for example those constructed by the bag-of-words [18] and GIST [33] methods, is very large. Moreover, it degrades the efficiency of these methods to that of exhaustive search [46]. To make the NN search scalable in this situation, approximate nearest neighbor (ANN) techniques have been proposed. The idea is to find an approximate nearest neighbor rather than the exact one. Locality-sensitive hashing (LSH) was introduced for this purpose [10]. It maps a vector $x \in \mathbb{R}^d$ to a binary string $h \in \{0,1\}^r$, and guarantees that the neighbors in the original space have similar binary codes.

Hashing methods can be classified into two types, data-independent hashing and data-dependent hashing, depending on whether or not a training set is employed to learn the hash function. Data-independent hashing does not require training data, but instead adopts specific rules to ensure that desirable mathematical properties are observed. Thus methods of this type are robust to data variations. Data-independent methods [10, 5, 17, 16] can generate an arbitrary number of hash functions without a training set, but with a decrease of efficiency [48]. Locality sensitive hashing based on a p-stable distribution (LSH) [5] is one of the most representative data independent hashing methods. In addition to LSH, a number of alternative data-independent hashing schemes have been proposed. For example, random Fourier features have been used to make the Hamming distance related to the shift-invariant kernel (e.g., Gaussian kernel) between vectors [35]. These methods usually need larger binary codes to reduce the false positive rate, which in turn increases the storage costs and reduces the query efficiency.

On the other hand, data-dependent hashing approaches use a training process. A common objective is to explicitly force the similarity measured in the original feature space to be preserved in the Hamming space [48, 27, 29, 25, 26, 37, 16, 6, 22]. One representative data-dependent hashing scheme is spectral hashing (SH) [48] which transforms the problem of finding a sim-
ilarity preserving code for a given dataset to an NP-hard graph partitioning problem that is similar to Laplacian eigenmaps [2]. SH relaxes this problem and solves it using a spectral method [48, 2]. For a novel or unseen data point, SH uses the Laplace-Beltrami eigenfunctions to obtain binary codes under the hypothesis that the data is uniformly distributed. When this hypothesis does not hold, anchor graph hashing (AGH) [24] uses an anchor graph to obtain a low-rank adjacency matrix which is a computationally tractable approximation to the similarity matrix. Anchor Graph Hashing is then used to process the low rank adjacency matrix in constant time using the Nyström sampling method [3]. Zhang et al. proposed a self-taught hashing [51] method that firstly performs Laplacian eigenmaps and then thresholds the eigen-vectors to obtain a binary code for the training set. Subsequently, it trains an SVM classifier as the hash function for each bit. Dimensionality reduction has also been used to improve the efficiency of data-dependent hashing methods, such as [8, 32, 12]. IsoHash [12] learns projection functions which can produce projected dimensions with isotropic variances. ITQ [8] formulates the problem of learning a good binary code in terms of directly minimizing the quantization error of mapping this data to vertices of the binary hypercube. Several extensions of the above methods have also been developed with various considerations in the training process [47, 21, 41, 28, 1, 20].

Both supervised and semi-supervised learning settings have been explored in data dependent hashing. Semi-supervised hashing (SSH) [44] has been introduced to search for semantically similar instances when only part of the data are labelled. It minimizes the empirical error over the labeled data, and applies an information theoretic regularizer over both the labeled and unlabeled data. Projection learning hashing method [45] has been proposed in a similar form as SSH, having both semi-supervised and unsupervised modes of operation. Beside SSH, weakly-supervised hashing [31] and kernel-based supervised hashing (KSH) [23] are both examples of supervised hashing schemes that have kernel based hash functions. Kulis et al extended LSH functions using a learned metric [17] which can also be considered as a supervised method. Zhao et al proposed a deep semantic ranking method for learning hash functions [52], however, its time complexity is very high.

The local data distribution and its impact on the retrieval performance of hashing schemes worth further attention. Along this line, manifold methods make use of the data distribution. Locally linear hashing (LLH) [7] learns manifold structure by discovering the locally linear structure at each sample point and embeds the structure into binary codes. Inductive manifold
hashing (IMH) [41] achieves significant accuracy improvement on the basis of t-SNE [43]. However, since the similarity or distance to the nearest neighbors varies considerably for different data samples, simple thresholding on the similarity function returns different numbers of neighbors, as shown in Figure 1. This does not meet the expectation that the retrieval results shall not vary significantly for classes with different data densities when the same threshold is used. Therefore, the local distribution of data should be taken into consideration in order to generate consistent results. To solve this problem, Qin et al [34] 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 this method, we propose an adaptive similarity function to replace Euclidean distance or Gaussian kernel. For different data points, this similarity measure can adapt to the local data distribution.

Figure 1: Simple thresholding on the similarity function returns different numbers of neighbors, since the similarity to the nearest neighbors varies a lot for different data samples.

There are three original contributions in this paper. Firstly, we explore the idea of normalizing the Gaussian kernel, and use this to construct a new similarity function which gives consistent retrieval results. The normalization takes the local distribution of data into consideration, and is suitable
for the k-nearest neighbour search. This new similarity function is proved to give a positive semi-definitive (PSD) kernel. Secondly, we present two unsupervised hashing methods which aim at reconstructing the kernel function using binary codes. The first method takes the global similarity of measure given by the kernel function into consideration. The second method is local and focuses on the similarity of pairs of data, and can better capture semantically meaningful manifold structure. Both hashing methods enables linear training time with respect to the size of the training set, and gives a constant time for indexing using the proposed hash function. Moreover, we present a supervised hashing scheme based on subspace learning to improve semantic retrieval performance. Our third contribution is using supervised information (class labels) based on the former unsupervised hashing framework to achieve semantically better results. In the experiments, we show that both the proposed similarity function and the hashing methods are more effective than several baseline methods.

This paper is a significantly extended from a previous published conference paper [50]. The extensions include a new objective formulation for the case of supervised ground truth data (Section 4.2), development of a new supervised hashing scheme (Section 5), and experiments on two newly added methods in Section 6.

The remainder of this paper is organized as follows. A review of related work is given in Section 2. The proposed adaptive similarity measure is described in Section 3. A detailed description of the kernel based unsupervised hashing methods are given in Section 4. An extension of our hashing methods to the supervised setting is introduced in Section 5. We present the experimental results in Section 6, and then draw conclusions and discuss directions for future work in Section 7.

2. Related Work

In this section, we review some kernel based hashing methods and an adaptive distance measure. These methods are related to our hashing schemes which are based on normalized Gaussian kernel and hence provide the prerequisite material for our study.

2.1. Kernel Based Hashing Scheme

Kernel methods enable flexible models be constructed for a variety of applications by adopting different types of kernel functions. Some hashing
methods take a kernel function as input. For instance, Kulis and Darrell proposed a binary reconstructive embedding (BRE) approach [14] with a kernelized hashing function. The objective of BRE is to minimize the reconstruction error between 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 over the interval $[0, 1]$. However, this objective function is non-convex, and the reconstruction is binary which makes the objective none differentiable. When a coordinate-descent optimization scheme is used [14], the objective function converges to a local minimum. The corresponding training time is highly dependent on the number of input data pairs, which constrains the scale associated with the training set. Besides BRE, the kernelized locality-sensitive hashing (KLSH) method [15] can also preserve the kernel similarity by generalizing the principle underlying LSH [5] to arbitrary reproducing kernel Hilbert spaces. The SSH method [26] exploits shared structures across hash functions. The time complexity of KLSH is much lower than BRE, but the method suffers from the same problem as LSH, i.e., its performance degrades with compact codes. Liu et al [24] proposed an anchor graph hashing (AGH) method, which constructs a neighborhood graph and uses the graph Laplacian to solve a relaxed graph partition problem. For efficient training and indexing, this method approximates the neighborhood graph by a sparse anchor graph. Although it can cope with any customized similarity function including kernelized versions, the sparsity of the anchor graph limits the accuracy of the approximation.

Existing kernel hashing schemes use kernels only in the process of hash code calculation. In this paper, we propose a novel kernel which takes the distribution of data into account, and we show how this can be used for the purpose of hashing.

2.2. Query Adaptive Distance

Euclidean distance is the most widely used metric for measuring the similarity between local feature descriptors. In order to derive a measure that is adaptive to the query feature, Qin et al [34] proposed to use the normalized Euclidean distance, defined as follows

$$d_n(x_i, x_j) = \frac{d(x_i, x_j)}{E(d(x_i))}$$

(1)
where \( \mathbf{x}_i \) and \( \mathbf{x}_j \) are two different data vectors from the same dataset. \( d(\mathbf{x}_i, \mathbf{x}_j) \) is the Euclidean distances between \( \mathbf{x}_i \) and \( \mathbf{x}_j \) and \( dn(\mathbf{x}_i, \mathbf{x}_j) \) is the normalized version of the distance. \( E(d(\mathbf{x}_i)) \) is the expected Euclidean distance between \( \mathbf{x}_i \) and its non-matching features, which can be randomly sampled. This normalized Euclidean distance aims at obtaining unified degree of measurement across different queries. Note that the normalized Euclidean distance is not symmetric, which means that \( dn(\mathbf{x}_i, \mathbf{x}_j) \neq dn(\mathbf{x}_j, \mathbf{x}_i) \). Qin et al discussed feature similarity from a probabilistic perspective, with the aim of obtaining better retrieval results, which they have achieved on several image datasets [34].

3. Normalized Kernel Similarity

In this section, we propose a normalized Gaussian kernel. We show that this kernel is positive semi-definite.

The kernel trick allows methods defined in linear pattern spaces to be transformed to nonlinear pattern spaces by adopting nonlinear kernel functions. There are two key components in a kernel method. The first one is a data mapping rule defined using the kernel function. The second component is a learning algorithm, which is linear in the mapped feature space. The best known example of the kernel methods is the support vector machine (SVM) [11].

In this paper, we propose a retrieval framework which takes the kernel function as the basis for measuring the similarity of two samples. For example, a Gaussian kernel can be used to calculate the similarity in the following manner

\[
\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-d(\mathbf{x}_i, \mathbf{x}_j)^2 / 2\sigma^2)
\]  

(2)

where \( \sigma \) is used to scale the exponential function which defines a nonlinear model that can accommodate linearly inseparable data. The Gaussian kernel also benefits from the properties of the squared exponential function. If the value of \( \kappa(\mathbf{x}_i, \mathbf{x}_j) \) is large, then we consider \( \mathbf{x}_i \) and \( \mathbf{x}_j \) to be similar, otherwise they are considered relatively dissimilar. In k-NN search, the Gaussian kernel is equivalent to the Euclidean distance. Data pairs with smaller Euclidean distance have larger kernel function values, which means that they are similar to each other, and vice versa.

The most commonly used application of hashing techniques is hash lookup which uses binary codes as the addresses required to index data for retrieval.
Given a query, all samples with the same or similar hash codes are normally
returned as the results. If the objective of a hashing method is to preserve
the similarity conveyed by the kernel function, then it is equivalent to k-NN
search. For different queries, those samples which fall into the same hash
bucket may not have the same level of similarity because of the different
distribution of neighborhoods for the queries. If we wish to obtain a better
result for any query by hash lookup, then we should take into consideration
the local distribution of the different samples.

Here we propose a normalized kernel for local feature retrieval. The goal
is to achieve an adaptive similarity measure which not only considers the
neighborhood distribution of the data samples, but also improves the perfor-
mance of k-NN search. Motivated by the Bag-of-Words method [18] which
uses a clustering method to cluster local features as different “visual words”,
we treat the data samples in the same cluster as being in correspondence to
each other. If the cluster center index of data sample \(a\) is designated as \(\iota(a)\),
then we can compute the adaptive similarity \(C_{\kappa(a)}\) of \(a\) with its corresponding
points by averaging the kernel between \(a\) and the samples that belong to the
cluster with center \(\iota(a)\). Furthermore, if the distribution of pairwise kernel
function values in the same cluster is approximately uniform, then we can
obtain \(C_{\kappa(a)}\) via:

\[
C_{\kappa(a)} = \frac{\sum_{i(x_i)=\iota(a),j(x_j)=\iota(a)} \kappa(x_i, x_j)}{\#(\{(x_i, x_j)|i(x_i) = \iota(a), j(x_j) = \iota(a)\})}
\]  

(3)

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

Since \(C_{\kappa(a)}\) only depends on \(\iota(a)\), for simplicity, we denote it as \(C_{\iota(a)}\), and
\(C_i = C_{\iota(a)=i}\) defines the adaptive similarity of cluster \(i\). If the corresponding
points are used to normalize the kernel, the generated similarity function
between \(a\) and \(b\) is \(\kappa(a, b)/C_{\iota(a)}\), while the normalized similarity between \(b\)
and \(a\) is \(\kappa(a, b)/C_{\iota(b)}\). This similarity is asymmetric because \(C_{\iota(a)}\) and \(C_{\iota(b)}\)
are not always the same. To overcome this problem, the geometric mean
\(\sqrt{C_{\iota(a)}C_{\iota(b)}}\) is used as the normalising denominator.

If the total number of clusters is \(l\), let \(\delta(a) \in \{0, 1\}^{l \times 1}\) indicates to which
cluster center the data point \(a\) belongs. The normalized kernel for points \(a\)
and \(b\) is

\[
\kappa_n(a, b) = (\delta(a)^T D \delta(b)) \kappa(a, b)
\]

(4)

where \(\kappa_n\) refers to the normalized kernel, \(D\) is an \(l \times l\) matrix and \(D_{ij} = 1/\sqrt{C_i C_j}\). A simple example on the adaptability of the proposed similarity
function is shown in Figure 1.

For the clustering step, the kernel k-means algorithm is used to obtain the cluster centers and $\delta(a)$. Kernel k-means is the kernel based version of the k-means algorithm. In contrast to the original k-means algorithm, kernel k-means takes the kernel matrix as input and represents each cluster centroid as a linear combination of the mapped data $\phi(x)$. The objective function of kernel k-means is almost identical to that of k-means, except that in this case all samples are mapped and as a result

$$\min \sum_{j=1}^{k} \sum_{i=1}^{n} I_{ij} \|\phi(x_i) - u_j\|^2$$

subject to: $I_{ij} \in \{0, 1\}$, $\sum_{j=1}^{k} I_{ij} = 1$ \hspace{1cm} (5)

where $I_{ij}$ indicates whether $\phi(x_i)$ belongs to the $j$-th cluster, and $u_j$ is the class center of the $j$-th cluster.

In the original k-means algorithm $\phi(x_i)$ and $u_j$ can be updated iteratively. For the kernel k-means algorithm, on the other hand, the original calculation of vectors (e.g., Euclidean distance) has to be replaced by the kernel function. At each iteration, the implicit vector $u_j$ is updated via

$$u_j = \frac{\sum_{i=1}^{n} I_{ij} \phi(x_i)}{\sum_{i=1}^{n} I_{ij}}$$

\hspace{1cm} (6)

The squared Euclidean distance between $\phi(x_i)$ and $u_j$ is:

$$\|\phi(a) - u_j\|^2 = \phi(a) \cdot \phi(a) + u_j \cdot u_j - 2 \sum_{i=1}^{n} I_{ij} \phi(x_i) \cdot \phi(a)$$

\hspace{1cm} (7)

The dot product or inner product between the mapped vectors can be obtained from the kernel matrix directly.

3.1. Positive Semi-definiteness

Having defined a symmetric normalized similarity measure, we prove that this kernel is positive semi-definite (PSD), which is necessary for a valid kernel function.
Definition 3.1. 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 $m \in \mathbb{N}, x_i, \ldots, x_m \in \mathcal{X}$, and $c_i \in \mathbb{R}$

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

(8)

Based on this definition, we introduce a theorem which can be proved directly by the Schur product theorem [40]:

The Lemma. If both $\kappa_1$ and $\kappa_2$ are positive semi-definite kernels, then their product $\kappa_1 \cdot \kappa_2$ is a positive semi-definite kernel.

Recall the matrix $D$ defined in equation (4). If we 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 already know that the kernel is PSD, by using the Lemma 3.1, the kernel function $\kappa_n(a, b)$ defined in equation (4) is also PSD.

4. 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 two unsupervised hashing methods that preserve the similarity defined by an arbitrary kernel using a compact binary code. The first method is Kernel Reconstructive Hashing (KRH) which uses Multidimensional Scaling (MDS) [38] to preserve the similarity in a compact binary code. KRH is effective in retrieving neighbours for which the metric distance is a good global distance measure. The second method addresses the need in many real-world applications where the data reside on a low-dimensional manifold. In this situation accurately capturing the nearest neighbor relationships on the corresponding manifold is crucial. We use the Laplacian Eigenmap to preserve the manifold structure of the dataset, and the resulting hashing algorithm is referred to as KRHs.
4.1. KRH Hashing Scheme

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 formulation 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 have 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 $\kappa(x_i, x_j)$. Liu et al [23] 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 $d_h(\tilde{x}_i, \tilde{x}_j)$, i.e.

$$\langle \tilde{x}_i, \tilde{x}_j \rangle = r - 2d_h(\tilde{x}_i, \tilde{x}_j)$$

(9)

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

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

(10)

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 in this range.

The objective function in equation (10) is difficult to solve because optimization with respect to binary code is not straightforward. However, if we relax the problem and do not constrain the output to be binary, it will be much easier to solve. Here, we use a vector $\hat{x}_i \in \mathbb{R}^{1 \times r}$ to substitute the binary vector $\tilde{x}_i$ in equation (10), so that the optimization problem becomes

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

(11)

The scaling factor $s$ in formula (10) is absorbed into $\hat{x}_i$. This is a basic MDS problem [38]. Its 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 $r$ largest eigenvalues, and their corresponding eigenvectors are $v_1, v_2, ..., v_r \in \mathbb{R}^{n \times 1}$, then the optimal solution is

$$\hat{x}_i = \sqrt{\lambda_\alpha} (v_\alpha)_i$$

(12)

for $i = 1, 2, ..., n, \alpha = 1, 2, ..., r$
Using the standard numerical algorithm to perform the spectral decomposition of the \( n \times n \) kernel matrix \( K \) requires \( O(n^3) \) time [49]. This cubic method is intractable for large training sets. A possible solution to this problem is to use a low-rank approximation of the kernel matrix. Here we use the Nyström method to construct a low-rank approximation \( \tilde{K} \) of \( K \) [49]. We randomly select \( m \) \( (m < n) \) 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 \( M_{ij} = K_{ij}, i \in I, j \in I \). The low-rank approximation for matrix \( K \) is \( \tilde{K} = AM^{-1}A^T \). If \( M \) is not invertible, then the Moore-Penrose generalized inverse is used to substitute \( M^{-1} \). In our method, the operations for the inverse and the generalized inverse are similar, so we assume \( M \) is invertible. The approximation \( \tilde{K} \) is exactly the same as \( K \) when the rank of \( K \) equals the rank of \( M \). Using this approximation, we perform the eigen decomposition on \( \tilde{K} \) instead of \( K \).

To solve for the eigenvalues and eigenvectors of \( \tilde{K} \) efficiently, we adopt the following procedure: Firstly we perform the eigen decomposition \( M = ZZ^T \), where \( \Sigma \) is a diagonal matrix containing the eigenvalues of \( M \), and the columns of \( Z \) are the corresponding eigenvectors. Since \( M \) is a positive semi-definite matrix and is invertible, we have \( M^{-1} = ZZ^{-1/z} = ZZ^{-1/2}Z^T \). Furthermore, introducing an \( m \times m \) matrix \( B = Z\Sigma^{-1/2} \), we have \( M^{-1} = BB^T \). If let \( F = AB \) be an \( n \times m \) matrix, we can obtain \( \tilde{K} = FF^T \). By letting \( E = F^TF \) where \( E \) is an \( m \times m \) matrix whose eigenvalues and eigenvectors can be computed in \( O(m^3) \), we can directly perform eigen decomposition on \( E \)

\[
E = U\Lambda U^T
\]  

(13)

where \( \Lambda \) and \( U \) are eigenvalue and eigenvector matrices 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 \( m \) positive eigenvalues of \( \tilde{K} \) are diagonal entries of \( \Lambda \): \( \lambda_1, \lambda_2, \ldots, \lambda_m \), and the remaining \( n - m \) eigenvalues are all zero. If \( V \) is an \( n \times m \) matrix whose columns are the eigenvectors of \( \tilde{K} \) corresponding to \( \lambda_1, \lambda_2, \ldots, \lambda_m \), by SVD we have:

\[
V = FUA\Lambda^{-1/2} = ABU\Lambda^{-1/2}
\]  

(14)

From equations (13) and (14), we can obtain the \( m \) positive eigenvalues and their corresponding eigenvectors of \( \tilde{K} \). Selecting the \( r \) largest values, \( \hat{x}_i \) for \( x_i \in X \) can then be effected using equation (12).
This procedure only generates the \( r \)-dimensional real-valued reconstructions for the training data. For an arbitrary input, we generalize the eigenvectors to be the eigenfunctions \( \phi_\alpha(\cdot), \alpha = 1, 2, ..., r \) such that \( x_i \) gives the \( i \)-th entry of the eigenvector corresponding to the eigenvalue \( \lambda_\alpha \). Given a novel input \( y \), we also obtain the eigenfunction using the Nyström method \[3\]

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

(15)

Calculating this eigenfunction has time cost \( O(n) \) for each data point. We can decrease the time complexity to \( O(m) \) using the Nyström approximation. Let \( \phi(y) \) be a \( 1 \times n \) vector such that \( \phi(y)_i = \kappa(y, x_i) \), and \( \phi(y)_I \) be a \( 1 \times m \) vector whose entries correspond to the sampled index set \( I \), then the Nyström approximation of \( \phi(y) \) is \( \phi(y)_I M^{-1} A^T \). As a result the approximate eigenfunction \( \tilde{\phi}_\alpha(y) \) is given by

\[
\tilde{\phi}_\alpha(y) = (\phi(y)_I M^{-1} A^T V \Lambda^{-1})_\alpha
\]

\[
= (\phi(y)_I B B^T A^T A B U \Lambda^{-3/2})_\alpha
\]

\[
= (\phi(y)_I B E U \Lambda^{-3/2})_\alpha
\]

\[
= (\phi(y)_I B U \Lambda^{-1/2})_\alpha
\]  

(16)

Let \( J \) be the set of indices \( \alpha \) such that \( \lambda_\alpha \) is one of the \( r \) largest eigenvalues. According to equations (12) and (16), we obtain a \( 1 \times r \) vector which is a real-valued reconstruction for \( y \) satisfying

\[
\hat{y} = (\phi(y)_I B U)_J
\]  

(17)

Note that this formula holds irrespective of whether \( y \) is training data or a novel input.

We now turn our attention to analysing the time complexity of KRH. In the training phase, if \( A \) is given then \( M^{-1} \) and its eigen decomposition can be computed in \( O(m^3) \) time. The matrix multiplication for constructing \( E \) requires \( O(n \times m^2) \) time, and performing the eigen decomposition of \( E \) costs \( O(m^3) \) time, so obtaining \( B U \) in equation (17) requires at most \( O(n \times m^2) \) time. For minimizing the quantization loss, each iteration requires \( O(r^2 \times n) \) time. In our experiments, we have found that 50 iterations are usually sufficient to obtain convergence. In the search phase, for each query \( y \), computing \( \hat{y} \) in equation (17) costs time \( O(m \times r) \). Therefore, the training time is linear with the size of training set, and the binary code can be generated in constant time.
4.2. KRHs Hashing Scheme

The objective formulation of KRHs is

$$\min \sum_{i,j=1}^{n} K_{ij} \|\tilde{x}_i - \tilde{x}_j\|^2$$  \hspace{1cm} (18)

subject to: $\tilde{x}_1 = 0, \tilde{x}_j^T = nI$

where $K$ is the reconstructive kernel matrix containing pairwise similarity values and $1 = [1, 1, ..., 1]^T \in \mathbb{R}^n$. Hashing based on the Laplacian eigenmap for large training sets can be solved by using the anchor graph in AGH [24]. We also adopt real-valued relaxation for finding the solution. First, we perform $k$-means clustering on the training set to obtain $c$ ($c \ll n$) clusters $u_j$ ($1 \leq j \leq c$) whose centres give the anchor point positions. We then define a truncated similarity $Z$

$$Z_{ij} = \begin{cases} \frac{\kappa(x_i, u_j)}{\sum_{j' \in \langle i \rangle} \kappa(x_i, u_{j'})}, & \forall j \in \langle i \rangle \\ 0, & otherwise \end{cases}$$  \hspace{1cm} (19)

where $\langle i \rangle \subset [1 : c]$ denotes the set of indices of the $s$ ($s \ll c$) nearest anchors of $x_i$. For $s \ll c$, $Z \in \mathbb{R}^{n \times c}$ is a sparse matrix.

Based on the similarity matrix $Z$, the adjacency matrix $A$ can be approximated as $\hat{A} = Z\Lambda^{-\frac{1}{2}}Z^T$ where $\Lambda = diag(Z^T 1)$. Matrix $M = \Lambda^{-\frac{1}{2}}Z^T Z\Lambda^{-\frac{1}{2}}$ has the same decomposition as $A$, giving the eigenvector-eigenvalue pairs of the $r$ largest eigenvalues $\{(v_k, \sigma_k)\}_{k=1}^r$ ($1 > \sigma_1 \geq ... \geq \sigma_r > 0$). Let the eigenvector matrix be $V = [v_1, ..., v_r] \in \mathbb{R}^{c \times r}$ and the eigenvalue matrix be $\Sigma = diag(\sigma_1, \sigma_2, ..., \sigma_r) \in \mathbb{R}^{r \times r}$. After the spectral composition of $\hat{A}$, the $r$-dimensional real-valued embedding of $X$ can be represented as

$$Y = [\hat{y}_1, \hat{y}_2, ..., \hat{y}_n]^T = \sqrt{nZ}\Lambda^{-\frac{1}{2}}V\Sigma^{-\frac{1}{2}} = ZW$$  \hspace{1cm} (20)

where $W = \sqrt{n}\Lambda^{-\frac{1}{2}}V\Sigma^{-\frac{1}{2}} = [w_1, w_2, ..., w_r] \in \mathbb{R}^{c \times r}$.

For a novel or unseen input $y$, first a truncated similarity vector $z(y)$ is needed, which can be regarded as a row in $Z$. The elements of this vector are given by

$$z(y)_i = \begin{cases} \frac{\kappa(y, u_i)}{\sum_{j' \in \langle i \rangle} \kappa(y, u_{j'})}, & \forall i \in \langle z \rangle \\ 0, & otherwise \end{cases}$$  \hspace{1cm} (21)
Then the real-valued embedding of \( y \) is

\[
\hat{y} = z(y)W
\]  

(22)

The time complexity of the k-means algorithm with \( T \) iterations is \( O(dcnT) \), and of the construction of matrix \( Z \) is \( O(dcn) \). The matrix \( M \) is computed using matrix multiplication, which costs \( O(nc^2) \). The eigen-decomposition of \( M \) costs \( O(c^3) \). The calculation of the matrix \( Y \) requires \( O(ncr) \) time. For \( r \ll n \), the construction of the Anchor Graph is linear of \( n \).

### 4.3. Minimizing the Quantization Loss

The methods described in Section 4.1 and Section 4.2 both give real-valued results from input data (from equations (17)(20)(21)). We now need to transform the real-valued result into a binary one. The simplest way to do this is using the sign function \( \hat{y} = \text{sign}(\hat{y}) \). However, sometimes, directly using the \( \text{sign}() \) function may cause \( \hat{y} \) to deviate significantly from \( \tilde{y} \). To address this problem, we define the following quantization loss \( Q(\cdot) \)

\[
Q(G) = \| G - \tilde{s} \cdot \text{sign}(G) \|_F^2
\]

(23)

where \( \| \cdot \|_F \) denotes the Frobenius norm. \( G \) is an arbitrary real-valued matrix, \( \text{sign}(G) \) has the same size as \( G \) and \( \text{sign}(G_{ij}) = 1 \) for \( G_{ij} \geq 0 \), and \( \text{sign}(G_{ij}) = -1 \) otherwise. The scaling factor \( \tilde{s} \) is introduced to match the ranges between the real-valued matrix and the binary matrix. A similar loss function was adopted in [8], which, however, does not have the scaling factor. This is not optimal because the ranges of \( G \) and \( \text{sign}(G) \) may be very different.

Let \( \hat{X} \) be an \( n \times r \) matrix whose rows are the real-valued reconstruction vectors \( \hat{x}_i \), then \( Q(\hat{X}) \) is the quantization loss for our solution. Recall that \( \hat{x}_i, i = 1, 2, ..., n \) aim to approximate the kernel \( \kappa(x_i, x_j) \) by the inner product \( \hat{x}_i\hat{x}_j^T \). Because multiplying a vector by an arbitrary orthogonal matrix \( R \) does not change the value of the inner product \( \hat{x}_i\hat{x}_j^T = \hat{x}_iR(\hat{x}_jR)^T \), \( \hat{X}R \) has the same effect as \( \hat{X} \). As a result we need to find the matrix \( R \) which gives the minimum value of \( Q(\hat{XR}) \). A similar problem was proposed and solved in [8] by iteratively updating \( R \) using the classic orthogonal Procrustes method [39]. In our method, we iteratively update both \( R \) and \( \tilde{s} \). At each iteration \( i \), we estimate the optimal orthogonal matrix \( R_{(i)} \) by

\[
\arg\min_{R_{(i)}} \| \hat{XR}_{(i)} - \tilde{s}_{(i-1)} \cdot \text{sign}(\hat{XR}_{(i-1)}) \|_F^2
\]

(24)
This is a classic Orthogonal Procrustes problem [39] which can be solved by singular value decomposition. If the SVD of \( \hat{X}^T(\tilde{s}_{(i-1)}\text{sign}(\hat{X}R_{(i-1)})) \) is \( \bar{U}\Sigma\bar{V}^T \), 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

\[
\tilde{s}_{(i)} = \frac{\text{tr}(\text{sign}(\hat{X}R_{(i)})^T\hat{X}R_{(i)})}{\text{tr}(\text{sign}(\hat{X}R_{(i)})\text{sign}(\hat{X}R_{(i)})^T)}
\tag{25}
\]

We initialize \( R_{(0)} \) as a random orthogonal matrix and \( \tilde{s}_{(0)} \) as 1. \( \tilde{s} \cdot \text{sign}(\hat{X}R) \) approximates \( \hat{X}R \) by minimizing the quantization loss. \( \tilde{s}^2\langle\text{sign}(\hat{x}_i R), \text{sign}(\hat{x}_j R)\rangle \) is equivalent to the inner product \( \langle\hat{x}_i, \hat{x}_j\rangle \), which in turn is an approximation of the kernel \( \kappa(x_i, x_j) \). Therefore, we obtain an approximate minimiser of the objective function in (10): \( \hat{x}_i = \text{sign}(\hat{x}_i R) \) and \( s = \tilde{s}^2 \). The time complexity of computing the orthogonal transformation is \( O(r^2) \).

Two proposed unsupervised hashing methods with the minimization of quantization loss are summarized in Algorithms 1 and 2.

\textbf{Algorithm 1:} Kernel Reconstructive Hashing (Related to Section 4.1 and 4.3)

\begin{itemize}
  \item \textbf{Data:} training data \( X \), kernel function \( \kappa(\cdot) \), code length \( r \),
  \item \textbf{Result:} \( r \) hash functions \( h_p(\cdot), p = 1, 2, ..., r \)
  \item Randomly sample \( m \) indices \( I \) to generate sub-matrices \( A, M; \)
  \item Eigen decomposition: \( M = Z\Sigma Z^T; \)
  \item Assign \( B = Z\Sigma^{-1/2}, F = AB; \)
  \item Assign \( E = F^TF \), solve its eigenvectors for the \( r \) largest eigenvalues,
  \item build \( m \times r \) matrix \( U_J; \)
  \item Assign \( \hat{X} = ABU_J, \) assign \( R_{(0)} \) as random orthogonal matrix, and \( \tilde{s}_{(0)} = 1; \)
  \item while \( R, \tilde{s} \) is not converged do
    \begin{itemize}
      \item Update \( R \) by Equation (24);
      \item Update \( \tilde{s} \) by Equation (25);
    \end{itemize}
  \item end
  \item Define row vector \( \varphi(y) \) as in Section 4.1, and hash functions \([h_1(y), h_2(y), ..., h_r(y)] = sgn(\varphi(y)_JBU_JR).\)
\end{itemize}
Algorithm 2: KRHs (Related to Section 4.2 and 4.3)

Data: training data $X$, kernel function $\kappa(\cdot)$, code length $r$, number of anchor points $c$, a novel input $y$

Result: $r$ hash functions $h_p(\cdot), p = 1, 2, ..., r$

Perform K-means on training data to get $c$ clusters and their class centres $u_j(1 \leq j \leq c)$ as anchor points.;

Calculate truncated similarity matrix $Z$ by Equation (19);

Assign $\Lambda = diag(Z^T 1) \tilde{A} = Z \Lambda^{-1} Z^T$;

Assign $M = \Lambda^{-\frac{1}{2}} Z^T Z \Lambda^{-\frac{1}{2}}$;

Perform eigen decomposition on $M$, getting $r$ largest eigenvalues and their corresponding eigenvectors, forming the eigenvalue matrix $\Sigma$ and eigenvector matrix $V$; Calculate projection matrix $W$ and embedding $Y$ by Equation (20);

Assign $R_{(0)}$ as random orthogonal matrix, and $\tilde{s}_{(0)} = 1$;

while $R, \tilde{s}$ is not converged do

    Update $R$ by Equation (24);

    Update $\tilde{s}$ by Equation (25);

end

Calculate $z(y)$ by Equation (21);

The $r$ hash functions $h_i(y) = z(y)w_i R (1 \leq i \leq r)$. 
5. KRH with Supervised Information

In this section, we aim to incorporate supervised information, i.e., the class labels of the data, into our hashing scheme. A straightforward supervised extension to the proposed KRH algorithm is presented in this section. We use Linear Discriminant Analysis (LDA) to pre-process the dataset. LDA is widely used in statistics, pattern recognition, and machine learning to find a linear projection of features that well separates two or more classes of features. The main goal of LDA is to find a linear projection of data that maximises the ratio of the between class and within class dispersion. This gives well separated and compact classes. The objective formulation of LDA is

$$
\max_P J = \frac{P^T S_B P}{P^T S_\omega P}
$$

where

$$
S_\omega = \sum_{i=1}^{C} \sum_{x \in \omega_i} (x - \mu_i)(x - \mu_i)^T
$$

is the within-class variance, 

$$
S_B = \sum_{i=1}^{C} \frac{N_i}{N}(\mu_i - \mu)(\mu_i - \mu)^T
$$

is the between class variance, and 

$$
P
$$

is a linear projection matrix (\(\mu_i\) is the mean of class \(\omega_i\) and \(\mu\) is the mean of \(\mu_i\) (1 \(\leq i \leq C\))). To solve this problem, eigen decomposition is performed on 

$$
S_\omega^{-1}S_B
$$

and the eigenvector matrix 

$$
P \in \mathbb{R}^{r \times n}
$$

gives the optimal projection matrix.

LDA is directly applied to the data before we attempt KRH. This leads to a supervised hashing scheme (LDA-KRH). The training process with LDA is summarised in Algorithm 3. With the LDA preprocessing, the distribution of data points in the Euclidean space is aligned in the direction of maximum separation of the class labels. As a result the hashing scheme is adapted to class labels, and consequently it gives better results.

---

**Algorithm 3: LDA-KRH**

**Data:** training data \(X\), label vector \(l\), normalized kernel function \(\kappa_n(\cdot)\), code length \(r\), number of anchor points \(c\), a novel input \(y\)

**Result:** \(r\) hash functions \(h_p(\cdot), p = 1, 2, ..., r\)

Perform LDA on the training data \(X\) with label \(l\), and get the projection matrix \(P\);

Assign the low-dimensional embedding \(E = P^T X = [e_1, ..., e_n]\);

Calculate kernel matrix \(K\) by normalized kernel function \(K_{ij} = \kappa_n(e_i, e_j)(1 \leq i, j \leq n)\);

Execute unsupervised hashing schemes in Algorithm 1.
6. Experiments

We evaluate the performance of the proposed methods on four datasets. The first dataset is SIFT-1M [45], which contains 1 million local features represented as a vector of 128 dimensional SIFT descriptors [30]. The second dataset is the CIFAR-10 dataset [13] that consists of 60,000 32×32 color images belonging to 10 classes. Figure 2 shows some sample images in this dataset. We used a 384 dimensional GIST descriptor [33] to represent each image. The third dataset is MNIST from [19], which has 70000 size-normalized and centered images of handwritten digits. The fourth dataset is ILSVRC2012 which is a subset of Imagenet. This dataset contains over 1.2 million images belonging to 1,000 categories. Here we used a 4096 dimensional feature-vector computed by a convolutional neural network [36] on this dataset.

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 used precision-recall and recall curves. The precision and recall are calculated by

\[
\text{precision} = \frac{\text{Number of retrieved relevant pairs}}{\text{Total number of retrieved pairs}}
\]

(27)
recall = \frac{\text{Number of retrieved relevant pairs}}{\text{Total number of all relevant pairs}} \quad (28)

We also used the Mean Average Precision (MAP) which is the mean of the average precision scores for each query. All the evaluations were based on hash lookup usage [8, 12].

Some parameters have to be set in our methods. 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 $l \geq 30$. A large value of $l$ makes $\kappa_n$ complex valued, and so we set $l = 30$ in all the experiments. For a fair comparison, we set $\sigma$ as the averaged Euclidean distance for all the methods that make use of a Gaussian kernel. In KRH, there is nearly no difference in performance when the sample number $m \geq 1000$, and so we set $m = 1000$. We set the parameters in remaining methods to the values that achieve the best performance on the different datasets.

\subsection*{6.1. Finding k-NN using Normalized Kernel Similarity}

In this section, we demonstrate that the proposed normalized Gaussian kernel in the k-NN search task is superior in performance to the alternatives studied. For each query, we took the 100 nearest neighbors from Euclidean distance as the ground-truth. By decreasing the similarity threshold, we obtained various precision and recall values using equations (27) and (28), and then constructed precision-recall curves from them. In this setting, the Gaussian kernel and the Euclidean distance are equivalent.

Figure 3 shows the results using both the Gaussian kernel and the normalized Gaussian kernel as the similarity function. It can be seen that the proposed normalized Gaussian kernel has significantly outperformed the non-normalized counterpart in the precision recall curves.

\subsection*{6.2. Performance of Kernel Reconstructive Hashing}

The proposed KRH method can in principle make use of an arbitrary kernel function. We compared the method with several similar hashing methods including BRE [14], KLSH [15] and AGH [24]. For all the methods studied, we used the same Gaussian kernel as defined in equation (2). We aim to evaluate the degree of similarity preservation based on the kernel. To this end 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 satisfies the condition $\kappa(y, x_i) \geq \exp(-\tau^2/2\sigma^2)$ as the “true” neighbors. The
Figure 3: Precision-recall curves of k-NN search on SIFT-1M and CIFAR-10 when different distance metrics are compared.

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

precision-recall curves under different code lengths are illustrated in Figures 4 and 5. The results show that our method has significant advantages over the competitors on the SIFT-1M dataset. On the CIFAR-10 dataset, our method outperforms the alternative methods by a significant margin when the precision is high. It should also be noted that KLSH is relatively sensitive to the code length.

6.3. Comparison with State-of-the-art Hashing Methods

In this experiment, we focus on the KRH and KRHs methods with the normalized Gaussian kernel, and compare them with several state-of-the-art unsupervised hashing methods. These methods are locality sensitive hashing (LSH) [5], spectral hashing (SH) [48], unsupervised sequential projection
learning hashing (USPLH) [45], iterative quantization (ITQ) [8], Isotropic Hashing (IsoHash) [12] and k-means Hashing (KMH) [9]. We did not include AGH in this experiment because it performs better in a supervised setting.

With respect to KRH, the top 2 percentile nearest neighbors in the Euclidean distance were taken as the true positives as in the original paper [45]. The resulting recall curves are shown in Figures 7 and 8. Since we used hash lookup, the recall values were obtained by evaluating the recall for the average first $N$ Hamming neighbors for all the queries. Due to space limitations, we do not show the precision-recall curves. Instead we use a more succinct measurement, namely the mean averaged precision (mAP) which is the area under the precision-recall curve, as shown in Figure 6.

The performance of the various methods being compared on the two datasets follows the same trend. The proposed method has outperformed all alternative methods by a clear margin. It can be observed from Figure 6 that the advantage offered by our method becomes larger when the code length increases. This is because a large code length gives more precise approximation of the normalized Gaussian kernel, which is reasonable for k-NN search. Among the alternative methods, ITQ is the most competitive. The performance of LSH grows rapidly with increasing code length. On the other hand, USPLH performs more poorly with longer code-length. We also found that our method can retrieve images with greater semantic similarity. Figure 9 shows some sample retrieved images from CIFAR-10.

We evaluated the KRHs method in a supervised setting because the combined AGH [24] method tends to capture semantic neighborhoods. Data points that are close in the Hamming space, produced by AGH, tend to share similar semantic labels. The MNIST, CIFAR and Imagenet datasets were used for this experiment and the ground truths were defined by the cat-
egory information for the two labeled datasets. We compared the KRHs with several state-of-the-art unsupervised hashing methods. These methods are (PCAH) [5], spectral hashing (SH) [48], (SpH) [45] and iterative quantization (ITQ) [8]. Tables 1, 2 and 3 show the results.

It is clear that KRHs gives good performance in its supervised mode, especially with short hashing code lengths. Among the alternative methods, ITQ is the most competitive. The performance of LSH grows quickly with increasing code length, however, USPLH performs worse with longer code-lengths.

To compare the efficiency of various methods, we show the training time on CIFAR-10 in Table 4. All experiments were implemented using MATLAB on a PC with Intel Core-i7 3.4GHz CPU, 16GB RAM. Since BRE and KLSH cannot work on large training sets, the size of training set used for them were based on the settings reported in [14, 15]. For the methods that use kernelized data, we did not count the time for kernel computation as part of the training time. It can be seen from Table 4 that BRE, USPLH and KMH are relatively time consuming compared with the remaining methods.

Figure 6: Mean averaged precision of several unsupervised hashing methods on SIFT-1M and CIFAR-10.
6.4. Experiments with supervised information

The proposed KRH algorithm has been shown to work well without using label information. In this section, we demonstrate its performance on labelled datasets. Linear discriminant analysis (LDA [42]) is taken as a baseline to verify the efficacy of the proposed KRH algorithm. The proposed method is also compared with several recently proposed supervised hashing methods, such as semi-supervised hashing (SSH [44]) with sequential projection learning, ITQ with Canonical Correlation Analysis (CCA-ITQ [8]) and kernel-based supervised hashing (KSH [23]).

Experiments were performed on the MNIST and CIFAR datasets. In this experiment, 1000 labelled samples were randomly selected for supervised learning with SSH, KSH and the proposed KRHs, and all the available labelled training data were used for the linear CCA-ITQ method. Since there are only 10 classes with both datasets, the reduced dimensionality facilitated
by LDA (and thereby the binary code length) in the proposed KRHs was fixed to 9. The results are reported in Figure 10.

It is clear that the proposed KRH with LDA has significantly improved over the original KRH method. Among the supervised hashing methods, KSH obtains the highest MAPs on both datasets. However, it requires much larger binary code lengths to achieve comparable performance to the proposed method. The proposed LDA-KRH has better performance than all of the supervised methods other than KSH. The proposed supervised hashing framework can therefore effectively leverage supervised information to improve performance.

7. Conclusion

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 the data points, the proposed kernel function is consistent with k-NN search. The proposed adaptive kernel is more suitable for various datasets. By incorporating this kernel into KRH, the performance of approx-
Table 1: MAPs of different methods for varying code lengths on CIFAR-10.

<table>
<thead>
<tr>
<th>#bit</th>
<th>32 bits</th>
<th>48 bits</th>
<th>64 bits</th>
<th>96 bits</th>
<th>128 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRHs</td>
<td>0.168</td>
<td>0.162</td>
<td>0.158</td>
<td>0.153</td>
<td>0.145</td>
</tr>
<tr>
<td>BRE [14]</td>
<td>0.156</td>
<td>0.148</td>
<td>0.158</td>
<td>0.161</td>
<td>0.163</td>
</tr>
<tr>
<td>PCAH [5]</td>
<td>0.132</td>
<td>0.129</td>
<td>0.123</td>
<td>0.120</td>
<td>0.118</td>
</tr>
<tr>
<td>AGH [24]</td>
<td>0.158</td>
<td>0.150</td>
<td>0.146</td>
<td>0.140</td>
<td>0.137</td>
</tr>
<tr>
<td>ITQ [8]</td>
<td>0.168</td>
<td>0.173</td>
<td>0.175</td>
<td>0.179</td>
<td>0.180</td>
</tr>
<tr>
<td>SpH [45]</td>
<td>0.152</td>
<td>0.158</td>
<td>0.162</td>
<td>0.165</td>
<td>0.168</td>
</tr>
<tr>
<td>SH [48]</td>
<td>0.125</td>
<td>0.129</td>
<td>0.126</td>
<td>0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>KRH</td>
<td>0.124</td>
<td>0.128</td>
<td>0.130</td>
<td>0.132</td>
<td>0.135</td>
</tr>
</tbody>
</table>

Table 2: MAPs of different methods for varying code lengths on MNIST.

<table>
<thead>
<tr>
<th>#bit</th>
<th>32 bits</th>
<th>48 bits</th>
<th>64 bits</th>
<th>96 bits</th>
<th>128 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRHs</td>
<td>0.510</td>
<td>0.450</td>
<td>0.400</td>
<td>0.380</td>
<td>0.360</td>
</tr>
<tr>
<td>BRE [14]</td>
<td>0.380</td>
<td>0.400</td>
<td>0.410</td>
<td>0.430</td>
<td>0.430</td>
</tr>
<tr>
<td>PCAH [5]</td>
<td>0.250</td>
<td>0.220</td>
<td>0.210</td>
<td>0.190</td>
<td>0.180</td>
</tr>
<tr>
<td>AGH [24]</td>
<td>0.480</td>
<td>0.420</td>
<td>0.400</td>
<td>0.380</td>
<td>0.350</td>
</tr>
<tr>
<td>ITQ [8]</td>
<td>0.440</td>
<td>0.440</td>
<td>0.450</td>
<td>0.460</td>
<td>0.470</td>
</tr>
<tr>
<td>SpH [45]</td>
<td>0.310</td>
<td>0.310</td>
<td>0.320</td>
<td>0.370</td>
<td>0.380</td>
</tr>
<tr>
<td>SH [48]</td>
<td>0.275</td>
<td>0.250</td>
<td>0.220</td>
<td>0.230</td>
<td>0.220</td>
</tr>
<tr>
<td>KRH</td>
<td>0.282</td>
<td>0.303</td>
<td>0.337</td>
<td>0.385</td>
<td>0.396</td>
</tr>
</tbody>
</table>

Approximate nearest neighbor search can be improved under the k-NN protocol. We also modify our objective formulation from using multi-dimensional scaling to using the Laplacian Eigenmap for supervised ground truth. We also extend KRH to be a supervised hashing scheme using preprocessing based on linear discriminant analysis and have good performance with supervised information.

Though the adaptive kernel can handle the nonlinear data better, there is loss in the hash process, which limits the final accuracy. We will try to reduce the loss of embedding process to improve the performance in the future. We will also try more kernel based models with the proposed normalized Gaussian kernel. Since our method can reconstruct the pairwise similarities required by a kernel function using binary codes, we believe its usage is not
Table 3: MAPs of different methods for varying code lengths on Imagenet.

<table>
<thead>
<tr>
<th>#bit</th>
<th>32 bits</th>
<th>48 bits</th>
<th>64 bits</th>
<th>96 bits</th>
<th>128 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRHs</td>
<td>0.350</td>
<td>0.337</td>
<td>0.325</td>
<td>0.310</td>
<td>0.299</td>
</tr>
<tr>
<td>BRE [14]</td>
<td>0.143</td>
<td>0.190</td>
<td>0.228</td>
<td>0.243</td>
<td>0.272</td>
</tr>
<tr>
<td>PCAH [5]</td>
<td>0.108</td>
<td>0.106</td>
<td>0.106</td>
<td>0.105</td>
<td>0.104</td>
</tr>
<tr>
<td>AGH [24]</td>
<td>0.305</td>
<td>0.293</td>
<td>0.271</td>
<td>0.265</td>
<td>0.253</td>
</tr>
<tr>
<td>ITQ [8]</td>
<td>0.157</td>
<td>0.188</td>
<td>0.229</td>
<td>0.267</td>
<td>0.301</td>
</tr>
<tr>
<td>SpH [45]</td>
<td>0.096</td>
<td>0.125</td>
<td>0.156</td>
<td>0.178</td>
<td>0.193</td>
</tr>
<tr>
<td>SH [48]</td>
<td>0.103</td>
<td>0.105</td>
<td>0.109</td>
<td>0.110</td>
<td>0.110</td>
</tr>
<tr>
<td>KRH</td>
<td>0.125</td>
<td>0.139</td>
<td>0.165</td>
<td>0.173</td>
<td>0.189</td>
</tr>
</tbody>
</table>

Table 4: Training time (seconds) on CIFAR-10.

<table>
<thead>
<tr>
<th>#bit</th>
<th>32 bits</th>
<th>64 bits</th>
<th>96 bits</th>
<th>128 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRH</td>
<td>5.63</td>
<td>8.27</td>
<td>11.02</td>
<td>14.20</td>
</tr>
<tr>
<td>BRE [14]</td>
<td>53.61</td>
<td>279.23</td>
<td>492.81</td>
<td>931.54</td>
</tr>
<tr>
<td>KLSH [16]</td>
<td>7.83</td>
<td>8.21</td>
<td>8.58</td>
<td>8.75</td>
</tr>
<tr>
<td>AGH [24]</td>
<td>4.26</td>
<td>4.43</td>
<td>4.97</td>
<td>5.33</td>
</tr>
<tr>
<td>USPLH [45]</td>
<td>35.38</td>
<td>72.24</td>
<td>110.62</td>
<td>143.94</td>
</tr>
<tr>
<td>IsoHash [12]</td>
<td>0.35</td>
<td>0.51</td>
<td>0.72</td>
<td>1.61</td>
</tr>
<tr>
<td>KMH [9]</td>
<td>289.85</td>
<td>313.47</td>
<td>338.21</td>
<td>358.42</td>
</tr>
<tr>
<td>SH [48]</td>
<td>0.54</td>
<td>0.58</td>
<td>0.70</td>
<td>0.85</td>
</tr>
<tr>
<td>KRHs</td>
<td>9.51</td>
<td>12.67</td>
<td>15.31</td>
<td>19.29</td>
</tr>
</tbody>
</table>

constrained on ANN search. For example, it can also be useful in scenarios where fast kernel computation is an imperative.


[8] Yunchao Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale


