Adaptive Hash Retrieval with Kernel Based Similarity

Xiao Bai[a], Cheng Yan[a], Haichuan Yang[a], Lu Bai[b], Jun Zhou[c], Edwin Robert Hancock[d]

[a]*School of Computer Science and Engineering, Beihang University, Beijing, China*
[b]*School of Information, Central University of Finance and Economics, Beijing, China*
[c]*School of Information and Communication Technology, Griffith University, Nathan, QLD, Australia*
[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 $\mathbf{x} \in R^d$ 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, such as [44, 45, 31, 23, 52, 17].

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.

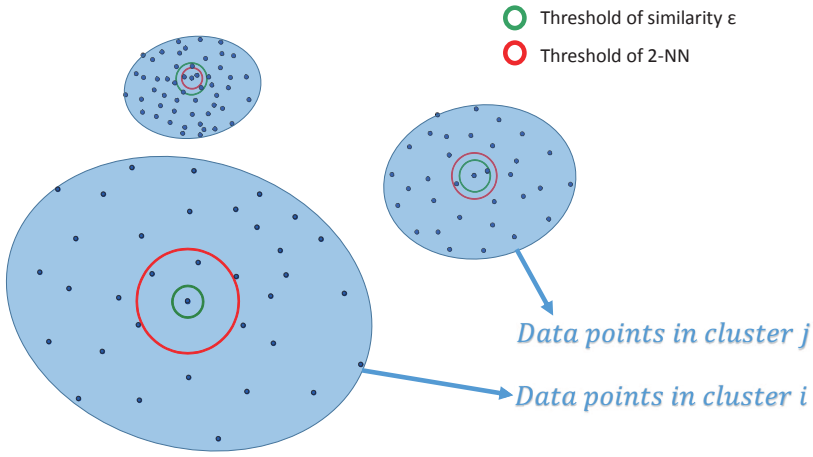


Figure 1: Simple thresholding on the similarity function returns different numbers of neighbors, since the similarity to the nearest neighbors varies a lot for different data samples.

1.1. Contributions and Overview

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

$$d_n(\mathbf{x}_i, \mathbf{x}_j) = \frac{d(\mathbf{x}_i, \mathbf{x}_j)}{E(d(\mathbf{x}_i))} \tag{1}$$

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

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

where σ 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 performance 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 $\mathbf{a}$ is designated as $\iota(\mathbf{a})$, then we can compute the adaptive similarity $C_{\kappa(\mathbf{a})}$ of $\mathbf{a}$ with its corresponding points by averaging the kernel between $\mathbf{a}$ and the samples that belong to the cluster with center $\iota(\mathbf{a})$. Furthermore, if the distribution of pairwise kernel function values in the same cluster is approximately uniform, then we can obtain $C_{\kappa(\mathbf{a})}$ via:

$$C_{\kappa(\mathbf{a})} = \frac{\sum_{\iota(\mathbf{x}_i)=\iota(\mathbf{a}),\iota(\mathbf{x}_j)=\iota(\mathbf{a})} \kappa(\mathbf{x}_i, \mathbf{x}_j)}{\#(\{(\mathbf{x}_i, \mathbf{x}_j)|\iota(\mathbf{x}_i) = \iota(\mathbf{a}), \iota(\mathbf{x}_j) = \iota(\mathbf{a})\})} \tag{3}$$

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

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

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

$$\kappa_n(\mathbf{a}, \mathbf{b}) = (\delta(\mathbf{a})^T \mathbf{D} \delta(\mathbf{b}))\kappa(\mathbf{a}, \mathbf{b}) \tag{4}$$

where κ_n refers to the normalized kernel, $\mathbf{D}$ is an $l \times l$ matrix and $\mathbf{D}_{ij} = 1/\sqrt{C_i C_j}$. 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 $\delta(\mathbf{a})$. 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 $\phi(\mathbf{x})$. 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

$$\min \sum_{j=1}^{k} \sum_{i=1}^{n} \mathcal{I}_{ij} \|\phi(\mathbf{x}_i) - \mathbf{u}_j\|^2 \tag{5}$$

$$\text{subject to: } \mathcal{I}_{ij} \in \{0, 1\}, \sum_{j=1}^{k} \mathcal{I}_{ij} = 1$$

where $\mathcal{I}_{ij}$ indicates whether $\phi(\mathbf{x}_i)$ belongs to the j-th cluster, and $\mathbf{u}_j$ is the class center of the j-th cluster.

In the original k-means algorithm $\phi(\mathbf{x}_i)$ and $\mathbf{u}_j$ 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 $\mathbf{u}_j$ is updated via

$$\mathbf{u}_j = \frac{\sum_{i=1}^{n} \mathcal{I}_{ij} \phi(\mathbf{x}_i)}{\sum_{i=1}^{n} \mathcal{I}_{ij}} \tag{6}$$

The squared Euclidean distance between $\phi(\mathbf{x}_i)$ and $\mathbf{u}_j$ is:

$$\|\phi(\mathbf{a}) - \mathbf{u}_j\|^2 = \phi(\mathbf{a}) \cdot \phi(\mathbf{a}) + \mathbf{u}_j \cdot \mathbf{u}_j - 2 \frac{\sum_{i=1}^{n} \mathcal{I}_{ij} \phi(\mathbf{x}_i) \cdot \phi(\mathbf{a})}{\sum_{i=1}^{n} \mathcal{I}_{ij}} \tag{7}$$

The dot product or inner product between the mapped vectors can be obtained from the kernel matrix directly calculated by equation 4.

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 $m \in \mathbb{N}, \mathbf{x}_i, ..., x_m \in \mathcal{X}$, and $c_i \in \mathbb{R}$*

$$\sum_{i,j=1}^{m} c_i c_j f(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{8}$$

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

Lemma. *If both κ_1 and κ_2 are positive semi-definite kernels, then their product $\kappa_1 \cdot \kappa_2$ is a positive semi-definite kernel.*

Recall the matrix $\mathbf{D}$ defined in equation (4). If we we rewrite it as $\mathbf{D} = \boldsymbol{\gamma}\boldsymbol{\gamma}^T$ where $\boldsymbol{\gamma}$ is an $l \times 1$ vector and $\boldsymbol{\gamma}_i = 1/\sqrt{C_i}$, then $\delta(\mathbf{a})^T\mathbf{D}\delta(\mathbf{b})$ becomes $(\boldsymbol{\gamma}^T\delta(\mathbf{a}))(\boldsymbol{\gamma}^T\delta(\mathbf{b}))$. Because $\boldsymbol{\gamma}^T\delta(\mathbf{a})$ is a feature mapping of $\mathbf{a}$, $\delta(\mathbf{a})^T\mathbf{D}\delta(\mathbf{b})$ can be considered as the inner product between features $\boldsymbol{\gamma}^T\delta(\mathbf{a})$ and $\boldsymbol{\gamma}^T\delta(\mathbf{b})$ which is obviously PSD. Since we already know that the kernel is PSD, by using the Lemma 3.1, the kernel function $\kappa_n(\mathbf{a}, \mathbf{b})$ 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 $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ be a training set of size n, and $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ be a kernel function over X. Our objective formulation is learning a set of r hash functions which generate the binary code of $\mathbf{x}_i$ as a $\{-1, 1\}^{1\times r}$ vector $\tilde{\mathbf{x}}_i = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), ..., h_r(\mathbf{x}_i)]$. The Hamming distance $d_h(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ between the binary codes for all instance pairs in X should have a negative correlation with the similarity represented by the kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j)$, i.e., a smaller $d_h(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ corresponds to a larger $\kappa(\mathbf{x}_i, \mathbf{x}_j)$. Liu *et al* [23] showed that the inner product $\langle\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\rangle = \tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_j^T$ between binary codes has a one-to-one correspondence with the Hamming distance $d_h(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$, i.e.

$$\langle\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\rangle = r - 2d_h(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) \tag{9}$$

This implies that the inner product also has a negative correlation with the Hamming distance. Therefore, our objective is to use the inner product $\langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle$ to reconstruct the kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j)$. To do so, we minimize the reconstruction error

$$\min \sum_{\mathbf{x}_i, \mathbf{x}_j \in X} (s \cdot \langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle - \kappa(\mathbf{x}_i, \mathbf{x}_j))^2 \tag{10}$$

where s is a positive number for scaling the inner product. It should be noted that although $\langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle \in [-r, r]$, the kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ 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 $\hat{\mathbf{x}}_i \in \mathbb{R}^{1 \times r}$ to substitute the binary vector $\tilde{\mathbf{x}}_i$ in equation (10), so that the optimisation problem becomes

$$\min \sum_{\mathbf{x}_i, \mathbf{x}_j \in X} (\langle \hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j \rangle - \kappa(\mathbf{x}_i, \mathbf{x}_j))^2 \tag{11}$$

The scaling factor s in formula (10) is absorbed into $\hat{\mathbf{x}}_i$. This is a basic MDS problem [38]. Its optimal solution can be obtained by the spectral decomposition of the $n \times n$ kernel matrix $\mathbf{K}$ whose entries $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. If $\lambda_1, \lambda_2, ..., \lambda_r > 0$ are the r largest eigenvalues, and their corresponding eigenvectors are $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_r \in \mathbb{R}^{n \times 1}$, then the optimal solution is

$$\begin{aligned} &(\hat{\mathbf{x}}_i)_\alpha = \sqrt{\lambda_\alpha}(\mathbf{v}_\alpha)_i \\ &\text{for } i = 1, 2, ..., n, \alpha = 1, 2, ..., r \end{aligned} \tag{12}$$

Using the standard numerical algorithm to perform the spectral decomposition of the $n \times n$ kernel matrix $\mathbf{K}$ 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 $\tilde{\mathbf{K}}$ of $\mathbf{K}$ [49]. We randomly select m ($m < n$) columns of $\mathbf{K}$ to form an $n \times m$ sub-matrix $\mathbf{A}$. Let I be the set of indices of the sampled columns, and $\mathbf{M}$ be an $m \times m$ submatrix of $\mathbf{K}$ such that $\mathbf{M}_{ij} = \mathbf{K}_{ij}, i \in I, j \in I$. The low-rank approximation for matrix $\mathbf{K}$ is $\tilde{\mathbf{K}} = \mathbf{A}\mathbf{M}^{-1}\mathbf{A}^T$. If $\mathbf{M}$ is not invertible, then the Moore-Penrose generalized inverse is used to substitute $\mathbf{M}^{-1}$. In our method, the

operations for the inverse and the generalized inverse are similar, so we assume $\mathbf{M}$ is invertible. The approximation $\tilde{\mathbf{K}}$ is exactly the same as $\mathbf{K}$ when the rank of $\mathbf{K}$ equals the rank of $\mathbf{M}$. Using this approximation, we perform the eigen decomposition on $\tilde{\mathbf{K}}$ instead of $\mathbf{K}$.

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

$$\mathbf{E} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \tag{13}$$

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

$$\mathbf{V} = \mathbf{F}\mathbf{U}\mathbf{\Lambda}^{-1/2} = \mathbf{A}\mathbf{B}\mathbf{U}\Lambda^{-1/2} \tag{14}$$

From equations (13) and (14), we can obtain the m positive eigenvalues and their corresponding eigenvectors of $\tilde{\mathbf{K}}$. Selecting the r largest values, $\hat{\mathbf{x}}_i$ for $\mathbf{x}_i \in \mathbf{X}$ 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 $\mathbf{x}_i$ gives the i-th entry of the eigenvector corresponding to the eigenvalue λ_α. Given a novel input $\mathbf{y}$, we also obtain the eigenfunction using the Nyström method [3]

$$\phi_\alpha(\mathbf{y}) = \sum_{i=1}^{n} \kappa(\mathbf{y}, \mathbf{x}_i)(\mathbf{v}_\alpha)_i / \lambda_\alpha \tag{15}$$

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 $\varphi(\mathbf{y})$ be a $1 \times n$ vector such that $\varphi(\mathbf{y})_i = \kappa(\mathbf{y}, \mathbf{x}_i)$, and $\varphi(\mathbf{y})_I$ be a

$1 \times m$ vector whose entries correspond to the sampled index set I, then the Nyström approximation of $\varphi(\mathbf{y})$ is $\varphi(\mathbf{y})_I \mathbf{M}^{-1}\mathbf{A}^T$. As a result the approximate eigenfunction $\tilde{\phi}_\alpha(\mathbf{y})$ is given by

$$\begin{aligned}\tilde{\phi}_\alpha(\mathbf{y}) &= (\varphi(\mathbf{y})_I \mathbf{M}^{-1}\mathbf{A}^T\mathbf{V}\mathbf{\Lambda}^{-1})_\alpha \\ &= (\varphi(\mathbf{y})_I \mathbf{B}\mathbf{B}^T\mathbf{A}^T\mathbf{A}\mathbf{B}\mathbf{U}\mathbf{\Lambda}^{-3/2})_\alpha \\ &= (\varphi(\mathbf{y})_I \mathbf{B}\mathbf{E}\mathbf{U}\mathbf{\Lambda}^{-3/2})_\alpha \\ &= (\varphi(\mathbf{y})_I \mathbf{B}\mathbf{U}\mathbf{\Lambda}^{-1/2})_\alpha \end{aligned} \tag{16}$$

Let J be the set of indices α such that λ_α 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 $\mathbf{y}$ satisfying

$$\hat{\mathbf{y}} = (\varphi(\mathbf{y})_I \mathbf{B}\mathbf{U})_J \tag{17}$$

Note that this formula holds irrespective of whether $\mathbf{y}$ is training data or a novel input.

We now turn our attention to analysing the time complexity of KRH. In the training phase, if $\mathbf{A}$ is given then $\mathbf{M}^{-1}$ and its eigen decomposition can be computed in $O(m^3)$ time. The matrix multiplication for constructing $\mathbf{E}$ requires $O(n \times m^2)$ time, and performing the eigen decomposition of $\mathbf{E}$ costs $O(m^3)$ time, so obtaining $\mathbf{B}\mathbf{U}$ 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 $\mathbf{y}$, computing $\hat{\mathbf{y}}$ 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

$$\begin{aligned} &\min \sum_{i,j=1}^{n} \mathbf{K}_{ij} \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|^2 \\ &subject\ to: \tilde{\mathbf{x}}_i \mathbf{1} = 0, \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_j^T = nI \end{aligned} \tag{18}$$

where $\mathbf{K}$ is the reconstructive kernel matrix containing pairwise similarity values and $\mathbf{1} = [1, 1, ..., 1]^T \in \mathbb{R}^n$. 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 $\mathbf{u}_j$ ($1 \leqslant j \leqslant c$) whose centres give the anchor point positions. We then define a truncated similarity $\mathbf{Z}$

$$\mathbf{Z}_{ij} = \begin{cases} \frac{\kappa(\mathbf{x}_i, \mathbf{u}_j)}{\sum_{j' \in \langle i \rangle} \kappa(\mathbf{x}_i, \mathbf{u}_j)}, & \forall j \in \langle i \rangle \\ 0, & otherwise \end{cases} \tag{19}$$

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

Based on the similarity matrix $\mathbf{Z}$, the adjacency matrix $\mathbf{A}$ can be approximated as $\hat{\mathbf{A}} = \mathbf{Z}\mathbf{\Lambda}^{-1}\mathbf{Z}^T$ where $\mathbf{\Lambda} = diag(\mathbf{Z}^T\mathbf{1})$. Matrix $\mathbf{M} = \mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{Z}^T\mathbf{Z}\mathbf{\Lambda}^{-\frac{1}{2}}$ has the same decomposition as $\hat{\mathbf{A}}$, giving the eigenvector-eigenvalue pairs of the r largest eigenvalues $\{(\mathbf{v}_k, \sigma_k)\}_{k=1}^{r}$ ($1 > \sigma_1 \geqslant ... \geqslant \sigma_r > 0$). Let the eigenvector matrix be $\mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_r] \in \mathbb{R}^{c \times r}$ and the eigenvalue matrix be $\mathbf{\Sigma} = diag(\sigma_1, \sigma_2, ..., \sigma_r) \in \mathbb{R}^{r \times r}$. After the spectral composition of $\hat{\mathbf{A}}$, the r-dimensional real-valued embedding of $\mathbf{X}$ can be represented as

$$\mathbf{Y} = [\hat{\mathbf{y}_1}, \hat{\mathbf{y}_2}, ..., \hat{\mathbf{y}_n}]^T = \sqrt{n}\mathbf{Z}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}\mathbf{\Sigma}^{-\frac{1}{2}} = \mathbf{Z}\mathbf{W} \tag{20}$$

where $\mathbf{W} = \sqrt{n}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}\mathbf{\Sigma}^{-\frac{1}{2}} = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_r] \in \mathbb{R}^{c \times r}$.

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

$$z(\mathbf{y})_i = \begin{cases} \frac{\kappa(\mathbf{y}, \mathbf{u}_i)}{\sum_{j \in \langle z \rangle} \kappa(\mathbf{y}, \mathbf{u}_j)}, & \forall i \in \langle z \rangle \\ 0, & otherwise \end{cases} \tag{21}$$

Then the real-valued embedding of $\mathbf{y}$ is

$$\hat{\mathbf{y}} = \mathbf{z}(\mathbf{y})\mathbf{W} \tag{22}$$

The time complexity of the k-means algorithm with T iterations is $O(dcnT)$, and of the construction of matrix $\mathbf{Z}$ is $O(dcn)$. The matrix $\mathbf{M}$ is computed using matrix multiplication, which costs $O(nc^2)$. The eigen-decomposition of $\mathbf{M}$ costs $O(c^3)$. The calculation of the matrix $\mathbf{Y}$ 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 $\tilde{\mathbf{y}} = sign(\hat{\mathbf{y}})$. However, sometimes, directly using the $sign(\cdot)$ function may cause $\hat{\mathbf{y}}$ to deviate significantly from $\tilde{\mathbf{y}}$. To address this problem, we define the following quantization loss $Q(\cdot)$

$$Q(\mathbf{G}) = \|\mathbf{G} - \tilde{s} \cdot sign(\mathbf{G})\|_F^2 \tag{23}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. $\mathbf{G}$ is an arbitrary real-valued matrix, $sign(\mathbf{G})$ has the same size as $\mathbf{G}$ and $sign(\mathbf{G}_{ij}) = 1$ for $\mathbf{G}_{ij} \geq 0$, and $sign(\mathbf{G}_{ij}) = -1$ 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 $\mathbf{G}$ and $sign(\mathbf{G})$ may be very different.

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

$$\arg\min_{\mathbf{R}_{(i)}} \|\hat{\mathbf{X}}\mathbf{R}_{(i)} - \tilde{s}_{(i-1)} \cdot sign(\hat{\mathbf{X}}\mathbf{R}_{(i-1)})\|_{\mathbf{F}}^2 \tag{24}$$

This is a classic Orthogonal Procrustes problem [39] which can be solved by singular value decomposition. If the SVD of $\hat{\mathbf{X}}^T(\tilde{s}_{(i-1)}sign(\hat{\mathbf{X}}\mathbf{R}_{(i-1)}))$ is $\bar{\mathbf{U}}S\bar{\mathbf{V}}^T$, then $\mathbf{R}_{(i)}$ should be $\bar{\mathbf{U}}\bar{\mathbf{V}}^T$. The optimized $\tilde{s}_{(i)}$ can be obtained by setting the partial derivative $\partial Q(\hat{\mathbf{X}}\mathbf{R}_{(i)})/\partial\tilde{s} = 0$, such that

$$\tilde{s}_{(i)} = \frac{tr(sign(\hat{\mathbf{X}}\mathbf{R}_{(i)})^T\hat{\mathbf{X}}\mathbf{R}_{(i)})}{tr(sign(\hat{\mathbf{X}}\mathbf{R}_{(i)})sign(\hat{\mathbf{X}}\mathbf{R}_{(i)})^T)} \tag{25}$$

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

Algorithm 1: Kernel Reconstructive Hashing (Related to Section 4.1 and 4.3)

Data: training data $\mathbf{X}$, 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 $\mathbf{A}$, $\mathbf{M}$;
Eigen decomposition: $\mathbf{M} = \mathbf{Z}\mathbf{\Sigma}\mathbf{Z}^T$;
Assign $\mathbf{B} = \mathbf{Z}\mathbf{\Sigma}^{-1/2}$, $\mathbf{F} = \mathbf{A}\mathbf{B}$;
Assign $\mathbf{E} = \mathbf{F}^T\mathbf{F}$, solve its eigenvectors for the r largest eigenvalues, build $m \times r$ matrix $\mathbf{U}_J$;
Assign $\hat{\mathbf{X}} = \mathbf{A}\mathbf{B}\mathbf{U}_J$, assign $\mathbf{R}_{(0)}$ as random orthogonal matrix, and $\tilde{s}_{(0)} = 1$;
while $\mathbf{R}, \tilde{s}$ *is not converged* **do**
    Update $\mathbf{R}$ by Equation (24);
    Update $\tilde{s}$ by Equation (25);
end
Define row vector $\varphi(\mathbf{y})$ as in Section 4.1, and hash functions $[h_1(\mathbf{y}), h_2(\mathbf{y}), ..., h_r(\mathbf{y})] = sgn(\varphi(\mathbf{y})_I\mathbf{B}\mathbf{U}_J\mathbf{R})$.

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

Algorithm 2: KRHs (Related to Section 4.2 and 4.3)

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

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 $\mathbf{u}_j (1 \leqslant j \leqslant c)$ as anchor points.;

Calculate truncated similarity matrix $\mathbf{Z}$ by Equation (19);

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

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

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

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

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

  Update $\mathbf{R}$ by Equation (24);

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

end

Calculate $z(\mathbf{y})$ by Equation (21);

The r hash functions $h_i(\mathbf{y}) = z(\mathbf{y})\mathbf{w}_i\mathbf{R} (1 \leqslant i \leqslant r)$.

maximises the ratio of the between class and within class dispersion. This gives well separated and compact classes. The objective formulation of LDA is

$$\max_{\mathbf{P}} J = \frac{\mathbf{P}^T \mathbf{S}_B \mathbf{P}}{\mathbf{P}^T \mathbf{S}_\omega \mathbf{P}} \tag{26}$$

where $\mathbf{S}_\omega = \sum_{i=1}^{C} \sum_{\mathbf{x} \in \omega_i} (\mathbf{x} - \mu_i)(\mathbf{x} - \mu_i)^T$ is the within-class variance, $\mathbf{S}_B = \sum_{i=1}^{C} \frac{N_i}{N} (\mu_i - \mu)(\mu_i - \mu)^T$ is the between class variance, and $\mathbf{P}$ is a linear projection matrix (μ_i is the mean of class ω_i and μ is the mean of μ_i ($1 \leqslant i \leqslant C$)). To solve this problem, eigen decomposition is performed on $\mathbf{S}_\omega^{-1} \mathbf{S}_B$ and the eigenvector matrix $\mathbf{P} \in \mathbb{R}^{r \times n}$ 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 $\mathbf{X}$, label vector $\boldsymbol{l}$, normalized kernel function $\kappa_n(\cdot)$, code length r, number of anchor points c, a novel input $\mathbf{y}$
Result: r hash functions $h_p(\cdot), p = 1, 2, ..., r$
Perform LDA on the training data X with label $\mathbf{l}$, and get the projection matrix $\mathbf{P}$;
Assign the low-dimensional embedding $\mathbf{E} = \mathbf{P}^T \mathbf{X} = [\boldsymbol{e}_1, ..., \boldsymbol{e}_n]$;
Calculate kernel matrix $\mathbf{K}$ by normalized kernel function $\mathbf{K}_{ij} = \kappa_n(\boldsymbol{e}_i, \boldsymbol{e}_j)(1 \leqslant i, j \leqslant n)$;
Execute unsupervised hashing schemes in Algorithm1.

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.

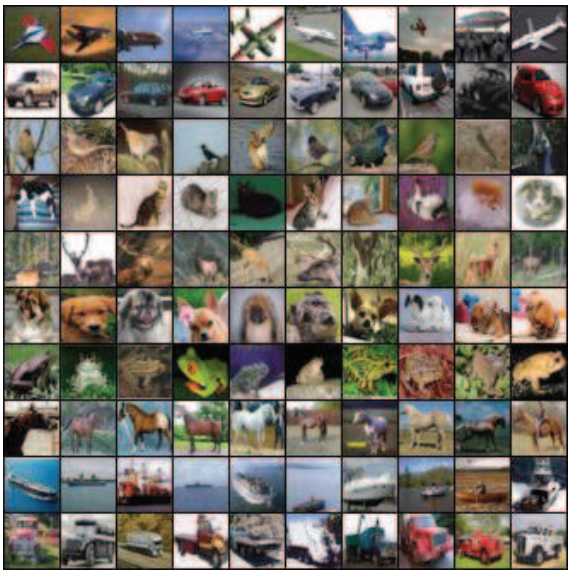


Figure 2: Sample images in CIFAR-10 dataset. Each row contains 10 images of the same class.

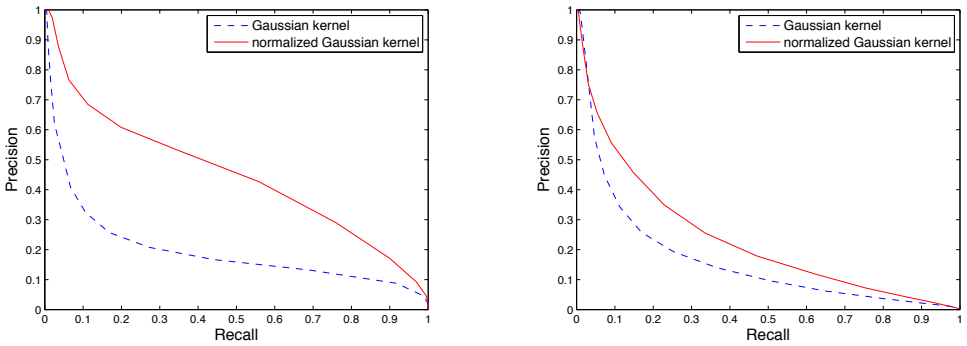
To evaluate the performance of different methods under comparison, we used precision-recall and recall curves. The precision and recall are calculated by

$$\text{precision} = \frac{\text{Number of retrieved relevant pairs}}{\text{Total number of retrieved pairs}} \tag{27}$$

$$\text{recall} = \frac{\text{Number of retrieved relevant pairs}}{\text{Total number of all relevant pairs}} \tag{28}$$

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 $\delta(\mathbf{a})$ in Equation (4) and the constant σ in the Gaussian kernel



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

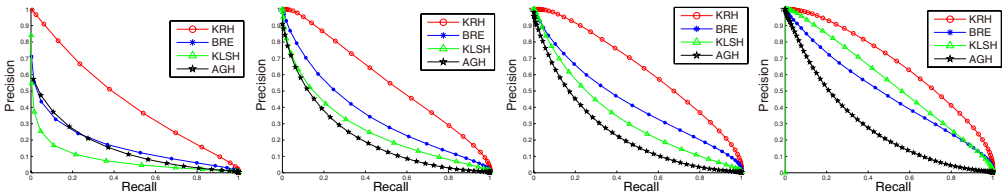
Figure 3: Precision-recall curves of k-NN search on SIFT-1M and CIFAR-10 when different distance metrics are compared.

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 κ_n complex valued, and so we set $l = 30$ in all the experiments. For a fair comparison, we set σ 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.

6.1. Finding k-NN using Normalized Kernel Similarity

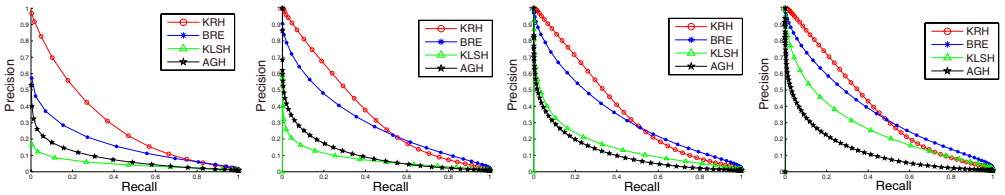
In this section, we demonstrate that the proposed normalized Gaussian kernel in the k-NN search task is superior in performance to the alternatives studied. For each query, we took the 100 nearest neighbors from Euclidean distance as the ground-truth. By decreasing the similarity threshold, we obtained 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.



(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.

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 τ as the average Euclidean distance between queries and their 50th nearest neighbors. For each query $\mathbf{y}$, we set all points $\mathbf{x}_i$ whose kernel satisfies the condition $\kappa(\mathbf{y}, \mathbf{x}_i) \geq exp(-\tau^2/2\sigma^2)$ as the "true" neighbors. The 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.

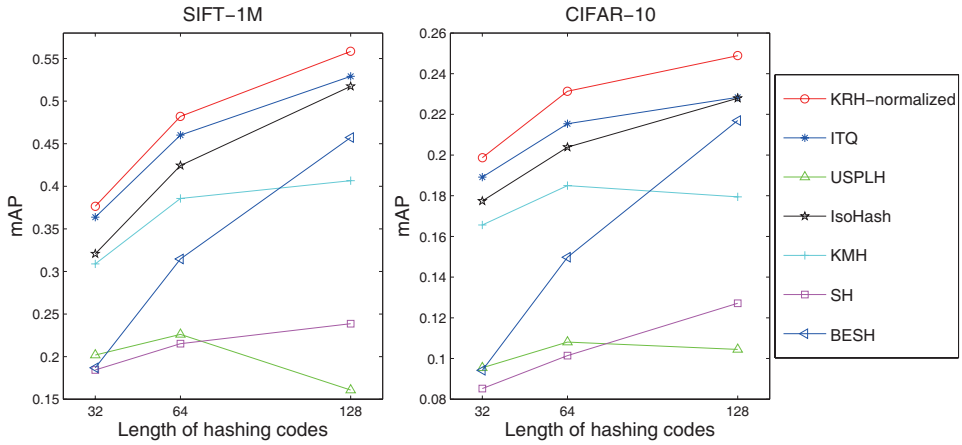


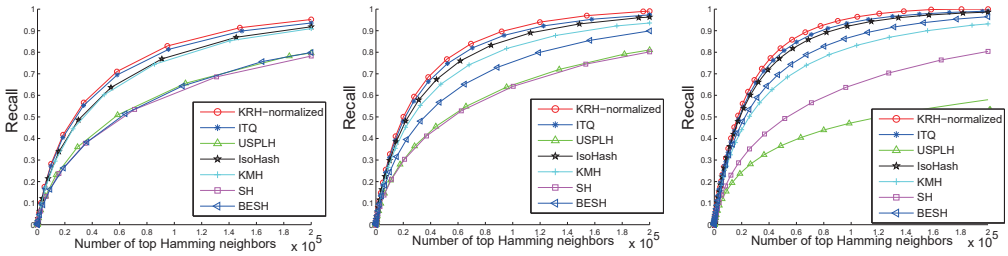
Figure 6: Mean averaged precision of several unsupervised hashing methods on SIFT-1M and CIFAR-10.

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.

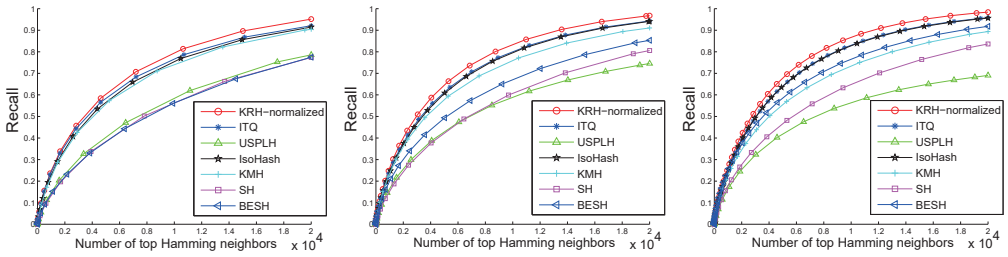
With respect to KRH, the top 2 percentile nearest neighbors in the Euclidean distance were taken as the true positives as in the original paper [45]. The resulting recall curves are shown in Figures 7 and 8. Since we used hash lookup, the recall values were obtained by evaluating the recall for the average 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 Fig-



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

Figure 7: Recall curves for several unsupervised hashing methods on SIFT-1M.



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

Figure 8: Recall curves of several unsupervised hashing methods on CIFAR-10.

ure 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 category information for the two labeled datasets. We compared the KRHs with

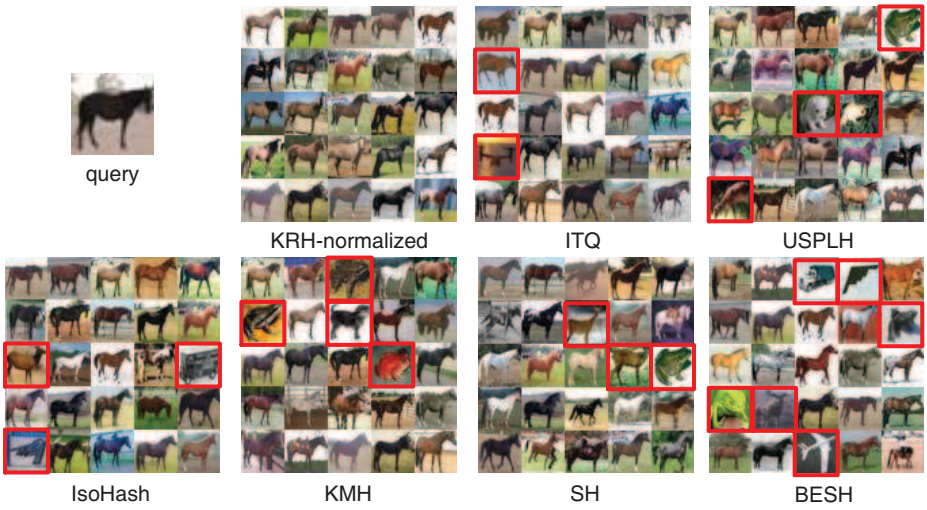


Figure 9: Qualitative results on CIFAR-10. We use 64-bit hashing codes, and show the false positives in red rectangle.

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.

Table 1: MAPs of different methods for varying code lengths on CIFAR-10.

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

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

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

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

Table 3: MAPs of different methods for varying code lengths on Imagenet.

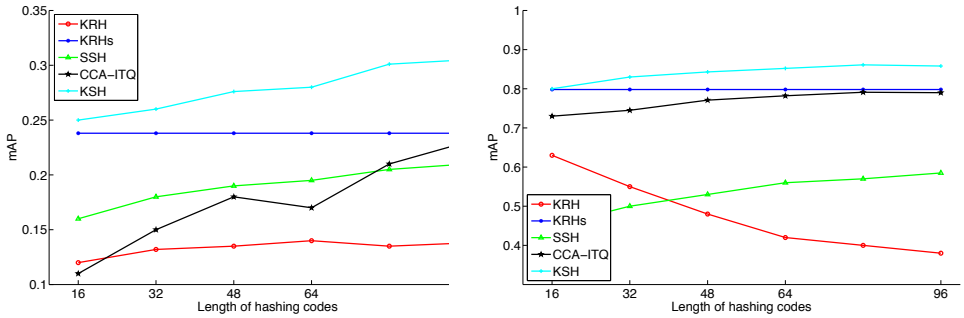
#bit	32 bits	48 bits	64 bits	96 bits	128 bits
KRHs	0.350	0.337	0.325	0.310	0.299
BRE [14]	0.143	0.190	0.228	0.243	0.272
PCAH [5]	0.108	0.106	0.106	0.105	0.104
AGH [24]	0.305	0.293	0.271	0.265	0.253
ITQ [8]	0.157	0.188	0.229	0.267	0.301
SpH [45]	0.096	0.125	0.156	0.178	0.193
SH [48]	0.103	0.105	0.109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

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

#bit	32 bits	64 bits	96 bits	128 bits
KRH	5.63	8.27	11.02	14.20
BRE [14]	53.61	279.23	492.81	931.54
KLSH [16]	7.83	8.21	8.58	8.75
AGH [24]	4.26	4.43	4.97	5.33
ITQ [8]	2.14	4.04	6.41	9.14
USPLH [45]	35.38	72.24	110.62	143.94
IsoHash [12]	0.35	0.51	0.72	1.61
KMH [9]	289.85	313.47	338.21	358.42
SH [48]	0.54	0.58	0.70	0.85
KRHs	9.51	12.67	15.31	19.29

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



(a) CIFAR-10 (b) MNIST

Figure 10: Evaluation of KRH with supervised learning by LDA(LDA-KRH) on the MNIST and the CIFAR datasets. Since there are only 10 classes in both datasets, the reduced dimensionality by LDA (thereby the binary code length) is set to 9.

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 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

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

[1] Xiao Bai, Haichuan Yang, Jun Zhou, and Peng Ren. Data-dependent hashing based on p-stable distribution. *IEEE Transactions on Image Processing*, 23(12):5033–5045, 2014.

[2] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in Neural Information Processing Systems*, pages 585–591, 2002.

[3] Yoshua Bengio, Olivier Delalleau, Nicolas Le Roux, Jean-François Paiement, Pascal Vincent, and Marie Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. *Neural Computation*, 16(10):2197–2219, 2004.

[4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

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

[6] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2475–2483, 2015.

[7] Irie Go, Li Zhenguo, Wu Xiao-Ming, and Chang Shih-Fu. Locally linear hashing for extracting non-linear manifolds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2115–2122, 2014.

[8] Yunchao Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2013.

[9] Kaiming He, Fang Wen, and Jian Sun. K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2938–2945, 2013.

[10] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Symposium on Theory of Computing*, pages 604–613, 1998.

[11] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[12] Weihao Kong and Wu-Jun Li. Isotropic hashing. In *Proceedings of the Neural Information Processing Systems Conference*, pages 1655–1663, 2012.

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

[14] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *Proceedings of the Neural Information Processing Systems Conference*, 2009.

[15] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *International Conference on Computer Vision*, 2009.

[16] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1092–1104, 2012.

[17] Brian Kulis, Prateek Jain, and Kristen Grauman. Fast similarity search for learned metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2143–2157, 2009.

[18] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[19] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The MNIST database of handwritten digits.

[20] Cong Leng, Jiaxiang Wu, Jian Cheng, Xiao Bai, and Hanqing Lu. Online sketching hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2503–2511, 2014.

[21] Peng Li, Meng Wang, Jian Cheng, Changsheng Xu, and Hanqing Lu. Spectral hashing with semantically consistent graph for image indexing. *IEEE Transactions on Multimedia*, 15(1):141–152, 2013.

[22] Zijia Lin, Guiguang Ding, Mingqing Hu, and Jianmin Wang. Semantics-preserving hashing for cross-view retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3864–3872, 2015.

[23] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[24] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *International Conference on Machine Learning*, 2011.

[25] Xianglong Liu, Cheng Deng, Bo Lang, Dacheng Tao, and Xuelong Li. Query-adaptive reciprocal hash tables for nearest neighbor search. *IEEE Transactions on Image Processing*, 25(2):907–919, 2016.

[26] Xianglong Liu, Junfeng He, and Bo Lang. Multiple feature kernel hashing for large-scale visual search. *Pattern Recognition*, 47(2):748–757, 2014.

[27] Xianglong Liu, Lei Huang, Cheng Deng, Bo Lang, and Dacheng Tao. Query-adaptive hash code ranking for large-scale multi-view visual search. *IEEE Transactions on Image Processing*, 25(10):4514–4524, 2016.

[28] Xianglong Liu, Yadong Mu, Bo Lang, and Shih-Fu Chang. Mixed image-keyword query adaptive hashing over multilabel images. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 10(2):22, 2014.

[29] Xianglong Liu, Yadong Mu, Danchen Zhang, Bo Lang, and Xuelong Li. Large-scale unsupervised hashing with shared structure learning. *IEEE Transactions on Cybernetics*, 45(9):1811–1822, 2015.

[30] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[31] Y. Mu, J. Shen, and S. Yan. Weakly-supervised hashing in kernel space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3344–3351, 2010.

[32] Mohammad Norouzi and David M Blei. Minimal loss hashing for compact binary codes. In *International Conference on Machine Learning*, 2011.

[33] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[34] Danfeng Qin, C. Wengert, and L. Van Gool. Query adaptive similarity for large scale object retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1610–1617, 2013.

[35] Maxim Raginsky and Svetlana Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. *Proceedings of the Neural Information Processing Systems Conference*, 22:1509–1517, 2009.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[37] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[38] Lawrence K Saul, Kilian Q Weinberger, Jihun H Ham, Fei Sha, and Daniel D Lee. Spectral methods for dimensionality reduction. *Semisupervised learning*, pages 293–308, 2006.

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

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

[41] Fumin Shen, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and Zhenmin Tang. Inductive hashing on manifolds. 2013.

[42] Christoph Strecha, Alexander M Bronstein, Michael M Bronstein, and Pascal Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):66–78, 2012.

[43] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

[44] Jun Wang, Sanjiv Kumar, and S Chang. Semi-supervised hashing for large scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2393–2406, 2012.

[45] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In *International Conference on Machine Learning*, pages 1127–1134, 2010.

[46] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *International Conference on Very Large Data Bases*, volume 98, pages 194–205, 1998.

[47] Yair Weiss, Rob Fergus, and Antonio Torralba. Multidimensional spectral hashing. In *European Conference on Computer Vision - Volume Part V*, pages 340–353, 2012.

[48] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Proceedings of the Neural Information Processing Systems Conference*, 2008.

[49] Christopher Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Proceedings of the Neural Information Processing Systems Conference*, 2001.

[50] Haichuan Yang, Xiao Bai, Jun Zhou, Peng Ren, Zhihong Zhang, and Jian Cheng. Adaptive object retrieval with kernel reconstructive hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1955–1962, 2014.

[51] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. Self-taught hashing for fast similarity search. In *SIGIR*, 2010.

[52] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1556–1564, 2015.

Xiao Bai received the B.Eng. degree in computer science from Beihang University of China, Beijing, China, in 2001, and the Ph.D. degree from the University of York, York, U.K., in 2006. He was a Research Officer (Fellow, Scientist) in the Computer Science Department, University of Bath, until 2008. He is currently a Full Professor in the School of Computer Science and Engineering, Beihang University. He has published more than eighty papers in high quality international journals and refereed conferences. His current research interests include pattern recognition, image processing and remote sensing image analysis. He has been awarded New Century Excellent Talents in University by Ministry of Education of China in 2012 and Excellency Scientist by China Association for Science and Technology in 2016. He is the guest editor for Pattern Recognition special issue “Pattern Recognition for High Performance Imaging”.

Cheng Yan received the Bachelors degree in Computer Science from Nanjing University of Information Science & Technology, Nanjing, China, in 2012, and Master Degree in Computer Science from Beihang University, Beijing, 2016. He is currently pursuing his Ph.D. degree under the supervision of Prof. Xiao Bai. His current research interests include computer vision and image processing.

Haichuan Yang received the Bachelor Degree in Software Engineering from the Sun Yat-sen University, Guangzhou, China, in 2012, and Master Degree in Computer Science from Beihang University, Beijing, in 2015. His current research interests include computer vision and image processing.

LU Bai received the the Ph.D. degree from the University of York, York, U.K., in 2015. He is currently an Associate Professor, School of Information, Central University of Finance and Economics. He has published more than fifty papers in journals and refereed conferences. His current research interests include pattern recognition, machine learning and computer vision.

Jun Zhou received the B.S. degree in computer science from Nanjing University of Science and Technology, Nanjing, China, in 1996. He received the M.S. degree in computer science from Concordia University, Montreal, Canada, in 2002, and the Ph.D. degree from the University of Alberta, Edmonton, Canada, in 2006, respectively. He is a senior lecturer in the School of Information and Communication Technology at Griffith University, Nathan, Australia. Previously, he had been a research fellow in the Research School of Computer Science at the Australian National University, Canberra, Australia, and a researcher in the Canberra Research Laboratory, NICTA, Australia. His research interests include pattern recognition, computer vision and spectral imaging with their applications to remote sensing and environmental informatics. He is the guest editor for Pattern Recognition special issue “Pattern Recognition for High Performance Imaging”.

Edwin R. Hancock received the B.Sc., Ph.D., and D.Sc. degrees from the University of Durham, Durham, U.K. He is now a Professor of Computer Vision in the Department of Computer Science, University of York, York, U.K. He has published nearly 150 journal articles and 550 conference papers. Prof. Hancock was the recipient of a Royal Society Wolfson Research Merit Award in 2009. He has been a member of the editorial board of the IEEE Transactions on Pattern Analysis and Machine Intelligence, Pattern Recognition, Computer Vision and Image Understanding, and Image and Vision Computing. His awards include the Pattern Recognition Society Medal in 1991, outstanding paper awards from the Pattern Recognition Journal in 1997, and the best conference best paper awards from the Computer Analysis of Images and Patterns Conference in 2001, the Asian Conference on Computer Vision in 2002, the International Conference on Pattern Recognition (ICPR) in 2006, British Machine Vision Conference (BMVC) in 2007, and the International Conference on Image Analysis and Processing in 2009. He is a Fellow of the International Association for Pattern Recognition, the Institute of Physics, the Institute of Engineering and Technology, and the British Computer Society. He was appointed as the founding Editor-in-Chief of the Institute of Engineering & Technology Computer Vision Journal in 2006. He was a General Chair for BMVC in 1994 and the Statistical, Syntactical and Structural Pattern Recognition in 2010, Track Chair for ICPR in 2004, and Area Chair at the European Conference on Computer Vision in 2006 and the Computer Vision and Pattern Recognition in 2008. He established the energy minimization methods in the Computer Vision and Pattern Recognition Workshop Series in 1997. He is current the editor-in-chief of journal Pattern Recognition. He is also the Second-vice President of IAPR.