Kernel Auto-Encoder for Semi-Supervised Hashing

Behnam Gholami
Computer Science Department
Rutgers University
bb510@cs.rutgers.edu

Abolfazl Hajisami
Department of Electrical and Computer Engineering
Rutgers University
hajisamik@cac.rutgers.edu

Abstract

Hashing-based approaches have gained popularity for large-scale image retrieval in recent years. It has been shown that semi-supervised hashing, which incorporates similarity/dissimilarity information into hash function learning could improve the hashing quality. In this paper, we present a novel kernel-based semi-supervised binary hashing model for image retrieval by taking into account auxiliary information, i.e., similar and dissimilar data pairs in achieving high quality hashing. The main idea is to map the data points into a highly non-linear feature space and then map the non-linear features into compact binary codes such that similar/dissimilar data points have similar/dissimilar hash codes. Empirical evaluations on three benchmark datasets demonstrate the superiority of the proposed method over several existing unsupervised and semi-supervised hash function learning methods.

1. Introduction

Given a dataset of images (images are usually encoded as feature vectors), Content-Based Image Retrieval (CBIR) systems return the nearest neighbors of a given image query in the database based on the similarity between the query and of images in that database.

Among the various methods that have been proposed for nearest neighbor search [18], hashing-based methods have gained popularity in recent years [9], [6], [12], [26], [11]. The main advantage of hashing-based methods is that they encode data into binary features that leads to low storage requirements and high computational efficiency [24], [21], [23].

Broadly speaking, hash code learning algorithms can be categorized into three main groups: Unsupervised, Semi-Supervised, and Supervised algorithms. Unsupervised hashing algorithms aim at projecting nearby points in the original space to similar binary codes in the Hamming space (the space of binary codes) [7], [8], [20], [26]. Locality-Sensitive Hashing (LSH) [7] generates linear hash functions through a set of random projection to approximate cosine similarity with high probability. Its kernelized version has recently been developed in [11].

On the other hand, supervised hashing maps the original features to compact binary codes such that label based similarities are preserved in the Hamming space [12], [13], [16], [17], [21], [23].

Using labeled data, Supervised Discrete Hashing (SDH) [19] formulates the hash code learning problem as the classification problem in which the binary codes are learned to be optimal for the classification task. By decomposing the hash code learning into sub-problems, SDH requires a carefully chosen loss function for efficiency and scalability of the classification optimization problem.

In semi-supervised hashing algorithms, the provided side information is available in binary mode (similar/dissimilar pairs) [1, 2]. These algorithms try to find a projection of the data into the space of binary codes such that the Hamming distance between the codes reflects the similarity/dissimilarity relations between the similar/dissimilar pairs in the training set [14], [23], [25], [12]. Liu et al. [12] proposed a kernel-based semi-supervised hashing model called KSH which sequentially train the hash functions one bit at a time, yielding very short yet discriminative codes. Recently, Xia et al. [27] proposed a semi-supervised hashing method called CNNH based on the Convolutional Neural Nets (CNN) for image retrieval, in which a set of hash functions and an appropriate image representation are learned simultaneously.

Very recently, Carreira-Perpinan and Raziperchikolaei [4] have proposed a learning hash method named Binary Auto-encoder (BA) based on the well-known Auto-encoder model. BA optimizes jointly over the hash functions and the binary codes using a recently proposed method of auxiliary coordinates (MAC) [5]. More precisely, BA considers a special auto-encoder consisting of an encoder $h(x) = \text{sgn}(Wx)$ (sgn(x) is the sign function applied elementwise, i.e., $\text{sgn}(x) = 1$, if $x \geq 0$ and $\text{sgn}(x) = 0$, if $x < 0$) which maps a real continuous input $x \in \mathbb{R}^D$ into a binary code vector with $L$ bits, $z \in \{0, 1\}^L$, and a
linear decoder \( f(z) = Az + b \) which maps the binary code \( z \) back to the input space \( \mathbb{R}^D \) to reconstruct \( x \) as accurate as possible. Hence, the objective function of BA given the training data \( X = (x_1, ..., x_N) \) is as follows:

\[
E_{BA}(f, h) = \sum_{n=1}^{N} \| x_n - f(h(x_n)) \|^2_2. \tag{1}
\]

BA minimizes (1) by introducing some additional parameters as the outputs of \( h \) (the binary codes for each of the \( N \) data points), and obtains the following equality-constrained problem \((n = 1, ..., N)\):

\[
\min_{h,f,Z} \sum_{n=1}^{N} \| x_n - f(z_n) \|^2_2 \quad \text{s.t.} \quad z_n = h(x_n) \in \{0,1\}^L, \tag{2}
\]

where \( Z = [z_1, z_2, ..., z_N] \). By applying the quadratic-penalty method [4], BA optimizes over \( Z \) and \((h, f)\) using the coordinate descent algorithm.

Although some promising results demonstrate the effectiveness of the BA model, there still exist two major issues with BA: (i) the hash function is formulated as a linear projection that may not effectively reflect the underlying relationship among the data points. (ii) The BA method cannot take advantage of the available side information (information of the similar/dissimilar pairs).

To address the above limitations, in this paper, we present a semi-supervised nonlinear hashing method named KAE, using the auto-encoder model. Specifically, to accommodate linearly inseparable data, we develop a kernel formulation of BA. Moreover, we incorporate the supervisory information into the hash function \( h \) instead of incorporating it directly into the Hamming (binary code) space to avoid nonconvexity of the optimization in that space.

### 2. Proposed Method

Let \( X = [x_1, x_2, ..., x_N] \in \mathbb{R}^{D \times N} \) and \( Z = [z_1, z_2, ..., z_N] \in \{0,1\}^{L \times N} \) be a set of observations and their corresponding binary codes. We are also given two sets of pairwise constraints which are defined as

\[
S = \{(i,j) \mid (x_i, x_j) \text{ are in the same class}\},
\]

\[
D = \{(i,j) \mid (x_i, x_j) \text{ are in two different classes}\},
\]

where \( S \) is the set of similar pairwise constraints, and \( D \) is the set of dissimilar pairwise constraints. We now consider the case when the data is kernelized. Let \( \Phi(X) = [\phi(x_1), ..., \phi(x_N)] \) be the matrix containing the transformed training data, so that each data point \( x_i (i = 1, ..., N) \) is projected into a potentially infinite dimensional vector \( \phi(x_i) = [\phi_1(x_i), \phi_2(x_i), ...]^\top \) where \( \phi : \mathbb{R}^D \to \mathcal{F} \) is a non-linear function which maps data points from input space to the high-dimensional feature space \( \mathcal{F} \), and \( K = \Phi(X)^\top \Phi(X) \) be the corresponding \( N \times N \) kernel matrix, where \( K(i,j) = k(x_i, x_j) \) and \( k(.,.) \) denotes a kernel function which satisfies Mercer’s condition [22]. We now perform our linear projection-based hash function learning in the feature space, which implicitly defines a nonlinear projection in the original data space.

### 2.1. Kernel-Based Semi-Supervised Hashing

We kernelize the BA model by replacing \( x_n \) with \( \phi(x_n) \) in Eq. 2. Hence, we define the following minimization problem to learn the hash function.

\[
\min_{h,f,Z} \sum_{n=1}^{N} \| \phi(x_n) - f(z_n) \|^2_2 \quad \text{s.t.} \quad z_n = h(\phi(x_n)), \quad z_n \in \{0,1\}^L, \quad n = 1, ..., N. \tag{3}
\]

where \( h(\phi(x_n)) = \text{sgn}(W \phi(x_n)) \), and \( f(z_n) = A z_n \) denote the encoder and the decoder respectively (parameters \( W \) and \( A \) include a bias by having an extra dimension \( \phi_0(x) = 1 \) and \( z_0 = 1 \) for each \( x \) and \( z \) respectively).

Since the objective function in (3) measures only the reconstruction accuracy, it is not discriminative. To get better discriminative ability, we add a regularization term for the encoder \( h \) such that it gives the same bits for \((x_i, x_j) \in S \) and different bits for \((x_i, x_j) \in D \). We define the following simple regularizer measuring the empirical accuracy on the labeled data.

\[
\Omega(h) = \sum_{(x_i, x_j) \in S} \| h(\phi(x_i)) - h(\phi(x_j)) \|^2_H - \sum_{(x_i, x_j) \in D} \| h(\phi(x_i)) - h(\phi(x_j)) \|^2_H,
\]

where \( \| \cdot \|_H \) denotes the Hamming distance. Adding \( \Omega(h) \) to (3) makes it difficult to solve for \( W \) due to non-convexity and non-differentiability of the regularizer. To remedy this problem, we propose to relax the regularizer by replacing the Hamming distance with the \( l_2 \) Euclidean distance, the sign of projection with the signed magnitude ( \( \text{sgn}(W \phi(x)) \) is replaced with \( W \phi(x) \) ), and the non-convex term

\[
- \sum_{(x_i, x_j) \in D} \| h(\phi(x_i)) - h(\phi(x_j)) \|^2_H
\]

with

\[
+ \sum_{(x_i, x_j) \in D} (W \phi(x_i))^\top (W \phi(x_j)).
\]

Intuitively, the relaxed regularizer not only desires similar points to have the same sign but also it encourages the similar points to be close to each other after the projection.
Moreover, it encourages dissimilar points not only have different signs but also be far apart from each other. It is worth noting that KSH [12] also relaxes the sign function to its magnitude for the similar and dissimilar constraints, however, a crucial difference between the relaxation of the proposed method and the KSH is that for similar pairs, KSH uses the negative sign of the inner product of the magnitude of the similar pairs that results in the non-convexity of the objective function respect to the parameter $W$, while our method uses the square Euclidean distance between the sign magnitude of the similar pairs that preserve the convexity of the objective function. With this new regularizer, the proposed optimization problem can be written as a function of $W, A, Z$.

$$
\begin{align*}
\min_{W, A, Z} & \sum_{n=1}^{N} \| \phi(x_n) - Az_n \|_2^2 \\
+ & \lambda_s \sum_{(x_i, x_j) \in S} \| W \phi(x_i) - W \phi(x_j) \|_2^2 \\
+ & \lambda_d \sum_{(x_i, x_j) \in D} (W \phi(x_i))^\top (W \phi(x_j)) \\
\text{s.t.} & \quad z_n = \text{sgn}(W \phi(x_n)), \quad z_n \in \{0, 1\}^L, \quad n = 1, \ldots, N,
\end{align*}
$$

where $\lambda_s \geq 0$ and $\lambda_d \geq 0$ denote the regularization parameters.

3. Optimization

Following [4], we use the quadratic penalty method [15] for solving (4) that results in minimizing the following objective function:

$$
\mathcal{L}(W, A, Z; \mu) = \sum_{n=1}^{N} \| \phi(x_n) - Az_n \|_2^2 \\
+ \lambda_s \sum_{(x_i, x_j) \in S} \| W \phi(x_i) - W \phi(x_j) \|_2^2 \\
+ \lambda_d \sum_{(x_i, x_j) \in D} (W \phi(x_i))^\top (W \phi(x_j)) \\
+ \mu \| z_n - \text{sgn}(W \phi(x_n)) \|_2^2 \\
\text{s.t.} & \quad z_n \in \{0, 1\}^L, \quad n = 1, \ldots, N,
$$

where $\mu$ is the constraint parameter that should be increased gradually, so the constraints are eventually satisfied [4]. Now we apply alternating optimization over $A, W, Z$ that goes as follows.

3.1. Optimization over $A$ for fixed $W$ and $Z$

This is equivalent to solving the following minimization problem:

$$
\min_A \mathcal{L}(A) = \sum_{n=1}^{N} \| \phi(x_n) - Az_n \|_2^2. \quad (6)
$$

Solving the above problem is often intractable due to the high (infinite) dimensionality of $\phi(\cdot)$. Fortunately, the following theorem states that the optimal solution for (6) can be spanned by $\{\phi(x_n)\}_{n=1}^{N}$.

**Theorem 1.** The optimal solution $A^*$ for problem (6) admits a representation of the form

$$
A^* = \Phi(X)\alpha, \quad (7)
$$

where $\alpha \in \mathbb{R}^{N \times (L+1)}$ denotes the coefficient matrix.

**Proof.** By expanding $\mathcal{L}(A)$, we have:

$$
\mathcal{L}(A) = \sum_{n=1}^{N} \phi(x_n)^\top \phi(x_n) - 2\phi(x_n)^\top Az_n + z_n^\top A^\top Az_n.
$$

By taking the derivative of $\mathcal{L}(A)$ with respect to $A$ and setting it to zero, we have:

$$
\frac{\partial \mathcal{L}(A)}{\partial A} = 2A \left( \sum_{n=1}^{N} z_n z_n^\top - 2 \sum_{n=1}^{N} \phi(x_n)z_n^\top \right) = 0
$$

$$
\Rightarrow A^* = \sum_{n=1}^{N} \phi(x_n)^\top \left( \sum_{n=1}^{N} z_n z_n^\top \right)^{-1} = \Phi(X)Z^\top (ZZ^\top)^{-1}.
$$

By setting $\alpha = Z^\top (ZZ^\top)^{-1}$, $A^*$ for problem (6) admits a representation of the form (7). \hfill \square

By plugging (7) into (6), and replacing $\Phi(X)^\top \Phi(X)$ with $K$ we have

$$
\alpha^* = \arg \min_{\alpha} \sum_{n=1}^{N} \| \phi(x_n) - \Phi(X)\alpha z_n \|_2^2
$$

$$
\alpha^* = \arg \min_{\alpha} \sum_{n=1}^{N} (\phi(x_n) - \Phi(X)\alpha z_n)^\top (\phi(x_n) - \Phi(X)\alpha z_n)
$$

$$
\alpha^* = \arg \min_{\alpha} \sum_{n=1}^{N} z_n^\top K\alpha z_n - 2K_n^\top \alpha z_n, \quad (8)
$$

where $K_n$ denotes the $n$-th row of the kernel matrix $K$. So, by taking the derivative of (8) with respect to $\alpha$ and setting it to zero, $\alpha^*$ can be computed analytically as

$$
\alpha^* = K^{-1}(\sum_{n=1}^{N} z_n K_n)^\top (\sum_{n=1}^{N} z_n z_n^\top).$$
3.2. Optimization over $W$ for fixed $A$ and $Z$

This has the following form:

$$
\min_{W} \mathcal{L}(W) = \sum_{n=1}^{N} \|z_n - \text{sgn}(W\phi(x_n))\|_2^2
+ \lambda_s \sum_{i,j \in \mathcal{S}} ||W\phi(x_i) - W\phi(x_j)||_2^2
+ \lambda_d \sum_{l \in \mathcal{D}} (W\phi(x_l))^\top (W\phi(x_l))
= \sum_{l=1}^{L} \min_{w_l} \left( \sum_{n=1}^{N} (z_{l_n} - \text{sgn}(w_l^\top \phi(x_n)))^2
+ \lambda_s \sum_{i,j \in \mathcal{S}} (w_l^\top \phi(x_i) - w_l^\top \phi(x_j))^2
+ \lambda_d \sum_{i,j \in \mathcal{D}} (w_l^\top \phi(x_i) - w_l^\top \phi(x_j))^2 \right),
$$

where $w_l^\top$ and $z_{l_n}$ denote the $l$-th row of the matrix $W$ and the $l$-th entry of the vector $z_n$, respectively. As can be seen, $\mathcal{L}(W)$ separates for each of the $L$ single-bit hash functions. Hence, optimizing each $w_l$ is independent of other $W_{-l}$ and is represented as:

$$
w_l^* = \arg \min_{w_l} \mathcal{L}(w_l) = \sum_{n=1}^{N} (z_{l_n} - \text{sgn}(w_l^\top \phi(x_n)))^2
+ \lambda_s \sum_{i,j \in \mathcal{S}} (w_l^\top \phi(x_i) - w_l^\top \phi(x_j))^2
+ \lambda_d \sum_{i,j \in \mathcal{D}} (w_l^\top \phi(x_i) - w_l^\top \phi(x_j))^2,
$$

The above problem can be treated as a linear binary classification problem in the feature space where $w_l$ is the parameter of the linear classifier, $\{\phi(x_n)\}_{n=1}^{N}$ and $\{z_n\}_{n=1}^{N}$ are the training samples and their corresponding labels respectively. So, the first term in (9) can be considered as the misclassification error of 0/1 loss function, and the remaining terms as the regularizers of the classifier’s parameter $w_l$. Since the 0/1 loss function is not convex nor differentiable, we use a quadratically smoothed hinge loss function $[\max(1 - z_{l_n}w_l^\top \phi(x_n), 0)]^2$ introduced by [28] as an approximation to 0/1 loss function. Using quadratic hinge loss, we can recast the above optimization problem as:

$$
w_l^* = \arg \min_{w_l} \mathcal{L}(w_l) = \sum_{n=1}^{N} [\max(1 - z_{l_n}w_l^\top \phi(x_n), 0)]^2
+ \lambda_s \sum_{i,j \in \mathcal{S}} (w_l^\top \phi(x_i) - w_l^\top \phi(x_j))^2
+ \lambda_d \sum_{i,j \in \mathcal{D}} \phi(x_i)\phi(x_j)^T w_l, \ l = 1, ..., L.
$$

It should be noted that for optimizing each $w_l$, we replace zero-entries of the binary codes with $-1$ (in this step the binary codes are fixed), and for optimizing $Z$ and $A$, we use 0 back. Again, because of high (infinite) dimensionality of $\phi(.)$, solving the above problem is intractable. However, inspired by the representer theorem [22], the following theorem states that the optimal solution for (10) can be spanned by $\{\phi(x_n)\}_{n=1}^{N}$.

**Theorem 2.** The optimal solution $w_l^*$ for the problem (10) admits a representation of the form

$$
w_l^* = \Phi(X)\beta_l + v,
$$

where $\Phi(X)^\top v = 0$. Since

$$
w_l^* \phi(x_n) = (\Phi(X)\beta_l + v)^\top \phi(x_n) = \beta_l^\top \Phi(X)^\top \phi(x_n),
$$

the first term of $\mathcal{L}(w_l)$ is independent of $v$. By replacing $w_l$ with $\phi(X)\beta_l + v$ in the second, and the third term of $\mathcal{L}(w_l)$, and using the fact that $\Phi(X)^\top v = 0$, we have

$$
(\phi(X)\beta_l + v)^\top \left( \lambda_s \sum_{i,j \in \mathcal{S}} (\phi(x_i) - \phi(x_j))^2 (\phi(x_i) - \phi(x_j))^T + \lambda_d \sum_{i,j \in \mathcal{D}} \phi(x_i)\phi(x_j)^T \right) (\phi(X)\beta_l + v)
= \beta_l^\top \Phi(X)^\top \left( \lambda_s \sum_{i,j \in \mathcal{S}} (\phi(x_i) - \phi(x_j))^2 (\phi(x_i) - \phi(x_j))^T + \lambda_d \sum_{i,j \in \mathcal{D}} \phi(x_i)\phi(x_j)^T \right) (\phi(X)\beta_l) + v^\top v.
$$

Since $v^\top v \geq 0$ for all $v$, we have $\mathcal{L}(w_l^*) \leq \mathcal{L}(w_l)$ for all $w_l$. Hence, the solution of (10) admits a representation of the form (11).

By plugging $w_l = \Phi(X)\beta_l$ into $\mathcal{L}(w_l)$, and replacing $\Phi(X)^\top \Phi(X)$ with $K$, we have:

$$
\mathcal{L}(\beta_l) = \sum_{n=1}^{N} \max\{1 - z_{l_n}K_n^\beta_l, 0\}^2 + \beta_l^\top [\lambda_s K_S + \lambda_d K_D] \beta_l,
$$

where

$$
K_S = \sum_{i,j \in \mathcal{S}} (K_i - K_j)^\top (K_i - K_j),
$$

and

$$
K_D = \sum_{i,j \in \mathcal{D}} \phi(x_i)\phi(x_j)^T.
$$
\[ K_D = \sum_{i,j \in D} (K_i^T K_j) \]

We minimize the unconstrained objective function (12) using Newton Algorithm [3] that goes as follows.

We say that a point \( \phi(x_n) \) is a support vector if \( z_{ln} K_n \beta_l < 1 \), i.e. if the loss on this point is non-zero. Now, by reordering the training points such that the first \( n_{sv} \) points are support vectors, the gradient of (12) with respect to \( \beta_l \) is

\[
\nabla \mathcal{L}(\beta_l) = 2(\lambda_s K_S + \lambda_d K_D) \beta_l + 2 \sum_{n=1}^{n_{sv}} z_{ln} K_n^T (z_{ln} K_n \beta_l - 1) \\
= 2(\lambda_s K_S + \lambda_d K_D) \beta_l + 2 k_{sv} K \beta_l - 2 \sum_{n=1}^{n_{sv}} z_{ln} K_n^T \\
= 2 \left( k_{sv} K + \lambda_s K_S + \lambda_d K_D \right) \beta_l - 2 \sum_{n=1}^{n_{sv}} z_{ln} K_n^T ,
\]

where

\[ k_{sv} = \sum_{n=1}^{n_{sv}} K_{nn}, \]

and \( K_{ij} \) denote the \((i,j)\)-th element of the kernel matrix \( K \). The Hessian of (12) with respect to \( \beta_l \) is

\[
\nabla^2 \mathcal{L}(\beta_l) = 2 \left( k_{sv} K + \lambda_s K_S + \lambda_d K_D \right).
\]

By plugging (14) into (13), we obtain:

\[
\nabla \mathcal{L}(\beta_l) = \nabla^2 \mathcal{L}(\beta_l) \beta_l - 2 \sum_{n=1}^{n_{sv}} z_{ln} K_n^T .
\]

Iteration \((t + 1)\) of the Newton algorithm consists of the following update,

\[
\beta_l^{(t+1)} = \beta_l^{(t)} - \gamma \nabla^2 \mathcal{L}(\beta_l^{(t)}) \nabla \mathcal{L}(\beta_l^{(t)}),
\]

where \( \gamma \) is the step size found by line search or backtracking. In our experiments, we noticed that the default value of \( \gamma = 1 \) did not result in any convergence problem. Hence, by setting \( \gamma = 1 \) and plugging (15) into (16), we find that after each Newton update,

\[
\beta_l^{(t+1)} = \beta_l^{(t)} - \nabla^2 \mathcal{L}(\beta_l^{(t)}) \left[ \nabla^2 \mathcal{L}(\beta_l^{(t)}) \beta_l^{(t)} - 2 \sum_{n=1}^{n_{sv}} z_{ln} K_n^T \right] \\
= \left( k_{sv} K + \lambda_s K_S + \lambda_d K_D \right)^{-1} \sum_{n=1}^{n_{sv}} z_{ln} K_n^T .
\]

It is worth noting that in iteration \((t + 1)\), we consider \( \phi(x_n) \) as a support vectors if \( z_{ln} K_n \beta_l^{(t)} < 1 \).

3.3. Optimization over \( Z \) for fixed \( A \) and \( W \)

As can be seen from Eq. 5, this is a binary optimization over \( NL \) variables, but it separates into \( N \) independent optimizations each on only \( L \) variables:

\[
\mathcal{L}(z_n; \mu) = \| \phi(x_n) - A z_n \|^2 + \mu \| z_n - \text{sgn}(W \phi(x_n)) \|^2_2 \quad \text{s.t. } z_n \in \{0, 1\}^L, \quad n = 1, ..., N
\]

By plugging (7) and (11) into the above equation, (17) can be recasted as:

\[
\mathcal{L}(z_n; \mu) = z_n^T (\mu I + \alpha^T K \alpha) z_n - 2 (K_n \alpha + \mu \text{sgn}(K_n \beta)) z_n \quad \text{s.t. } z_n \in \{0, 1\}^L, \quad n = 1, ..., N,
\]

where \( I \) denotes the Identity Matrix. Using cholesky decomposition of \( \mu I + \alpha^T K \alpha = LL^\top \) (\( L \) denotes a \( L \times L \) lower triangular matrix), we can compute each \( z_n \) by solving the following binary optimization problem.

\[
z_n^* = \arg \min_{z_n} \| L^\top z_n - L^{-1} (K_n \alpha + \mu \text{sgn}(K_n \beta)) \|^2_2 \\
\text{s.t. } z_n \in \{0, 1\}^L.
\]

Since the above problem is NP-complete, following [4], we solve it approximately using alternating optimization over groups of \( l \) bits (where we enumerate over each \( l \) bit group). As can be seen from (18) and (19), (using the cholesky decomposition) we have replaced a \( N \times L \) matrix with a \( L \times L \) matrix that reduces the computational complexity of the enumeration.

3.4. Practical Considerations

While effective in practice, our method, as any kernel method, becomes computationally expensive when dealing with large datasets. To address this limitation, following [12], we approximate \( A \) and \( W \) with a weighted sum of a subset of the training set. More precisely, we replace \( \Phi(X) \) with \( \Phi(\tilde{X}) \) in (7) and (11), where

\[
\Phi(\tilde{X}) = [\phi(x_{i(1)}), ..., \phi(x_{i(M)})] \quad \text{and} \quad x_{i(1)}, ..., x_{i(M)} \text{ are } M \text{ samples uniformly selected at random from the training set } (M \ll N).
\]

Consequently, \( K \in \mathbb{R}^{N \times N}, K_n \in \mathbb{R}^{1 \times N}, K_S \in \mathbb{R}^{N \times N} \) and \( K_D \in \mathbb{R}^{N \times N} \) are approximated by \( \tilde{K} \in \mathbb{R}^{M \times M}, \tilde{K}_n \in \mathbb{R}^{1 \times M}, \tilde{K}_S \in \mathbb{R}^{M \times M} \) and \( \tilde{K}_D \in \mathbb{R}^{M \times M} \) respectively, where \( K = \Phi(\tilde{X})^\top \Phi(\tilde{X}), \tilde{K}_S = \phi(x_n)^\top \Phi(\tilde{X}), \tilde{K}_D = \phi(x_n)^\top \Phi(\tilde{X}), \)

\[
\tilde{K}_S = \sum_{i,j \in S} (\tilde{K}_i - \tilde{K}_j)^\top (\tilde{K}_i - \tilde{K}_j),
\]

and,

\[
\tilde{K}_D = \sum_{i,j \in D} (\tilde{K}_i^\top \tilde{K}_j).
\]
4. Experimental Results

We evaluate the proposed method on three benchmark datasets: (1) The CIFAR-10 dataset consists of 60K 32 × 32 color tiny images which are categorized into 10 classes. (2) The NUS-WIDE dataset contains about 270K high-resolution color images. Following [12], we only use the images associated with the 21 most frequent concept classes. (3) The MNIST dataset consists of 70K images of handwritten digits from ‘0’ to ‘9’.

We compare the performance of the proposed KAE against five state-of-the-art hashing methods, including two unsupervised methods BA [4], and Kernelized Locality-Sensitive Hashing (KLSH) [11], and three semi-supervised method CNNH [27]. Supervised Hashing with kernels (KSH) [12], and Iterative Quantization with supervised embedding (ITQ-S) [8].

We use the same Gaussian RBF kernel \( k(x, y) = \exp(-\|x - y\|^2/2\sigma^2) \) for all the kernel-based approaches KAE, KSH, and KLSH. The kernel parameter \( \sigma \) and the regularization parameters \( \lambda_s \) and \( \lambda_d \) are tuned to an appropriate value on each dataset. For tuning the penalty parameter \( \mu \), we start with \( \mu_1 = 0.05 \) and double it after each \( Z \), \( \alpha \) and \( \beta \) update. We will also stop the iterations after no further changes occur in \( Z \).

Following [4], for initializing \( Z \), we solve the relaxed problem on \( Z \) (the constraints are \( z_n \in [0, 1]^K \) rather than \( z_n \in \{0, 1\}^K \) in early iterations, and we use the code found in the previous iteration’s \( Z \) step for subsequent iterations. It should be noted that the relaxed problem is a quadratic program (QP) in \( L \) variables and its unique minimizer can be found efficiently.

For CIFAR-10, we randomly select 50K images for training and 10K images as test set; for MNIST, we use 60K images for training and 10K images as test set, and for NUS-WIDE, following [27], we randomly select 100 images from each of the 21 classes to form the test set.

Since CNNH jointly learns hash codes and the image features, we directly use the image pixels as input for this method. For all other methods, we represent each image in CIFAR-10 by a 320 dimensional GIST feature vector; we represent each image in NUS-WIDE by a 128 dimensional wavelet feature vector and we represent each image in MNIST by a 512 dimensional PCA feature vector. We also set \( M = 500 \) (the number of randomly selected samples for approximating the kernel) for all datasets.

We need to generate side information in the forms of pairwise training instances. We sample similar pairs by randomly picking up two instances from the same class and dissimilar pairs by choosing two instances from different classes. We randomly sample about 200K similar pairs and 200K dissimilar pairs from the training set of each dataset.

For comparison purposes, we report the results of different methods in two metrics: 1) mean average precision (mAP) and 2) precision of Hamming radius \( r \) (PR) (if no images satisfy, we report zero precision). Moreover, the ground truths are defined by the class labels from the datasets.

As can be seen from Tables 1-3, and Fig. 1, three major points can be inferred from the results.

(i) Not surprisingly, the proposed KAE outperforms significantly the BA method. The improvement in performance compared to BA method is because of two reasons. First, KAE captures nonlinear structure of inputs but BA uses linear model for hash function learning. Second, BA does not have the similarity preserving property while KAE can utilize the available similarity/dissimilarity side information.

(ii) The KAE and CNNH consistently achieves the best results for all code lengths on all datasets. This is due to the fact that both CNNH and KAE capture the complex structure of the data points using deep neural network with nonlinear activation functions and kernel framework respectively. The superiority of KAE over other kernel-based hashing methods KSH and KLSH is that unlike KSH and KLSH which relax the binary constraint of codes during the learning process, KAE introduce the auxiliary variables allowing it to efficiently solve binary constraints without any relaxation.

(iii) Generally (though not always), KAE has better performance than CNNH. The reason is that CNNH is a filter approach [10] which means it follows a two-step procedure: in the first step, a hash function is learned ignoring the binary constraints and in the second step, CNNH binarizes the output of the resulting hash function. On the other hand, similar to BA, KAE is a rapper approach [10] which means it optimizes the objective jointly over the hash function and the hash codes that takes into account the binary constraints while learning the hash function. In other words, optimiz-

Table 1. mAP results of different methods on CIFAR-10 dataset (the mAP values was calculated within the top 100 returned neighbors).

<table>
<thead>
<tr>
<th>Method</th>
<th>( L = 8 )</th>
<th>( L = 16 )</th>
<th>( L = 24 )</th>
<th>( L = 32 )</th>
<th>( L = 48 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>KAE</td>
<td>0.394</td>
<td>0.450</td>
<td>0.469</td>
<td>0.477</td>
<td>0.480</td>
</tr>
<tr>
<td>CNNH</td>
<td>0.371</td>
<td>0.419</td>
<td>0.440</td>
<td>0.441</td>
<td>0.444</td>
</tr>
<tr>
<td>KSH</td>
<td>0.292</td>
<td>0.310</td>
<td>0.321</td>
<td>0.348</td>
<td>0.353</td>
</tr>
<tr>
<td>ITQ-S</td>
<td>0.299</td>
<td>0.318</td>
<td>0.330</td>
<td>0.351</td>
<td>0.352</td>
</tr>
<tr>
<td>BA</td>
<td>0.187</td>
<td>0.218</td>
<td>0.250</td>
<td>0.276</td>
<td>0.273</td>
</tr>
<tr>
<td>KLSH</td>
<td>0.185</td>
<td>0.209</td>
<td>0.254</td>
<td>0.258</td>
<td>0.254</td>
</tr>
</tbody>
</table>

Table 2. mAP results of different methods on MNIST dataset (the mAP values was calculated within the top 100 returned neighbors).

<table>
<thead>
<tr>
<th>Method</th>
<th>L = 8</th>
<th>L = 16</th>
<th>L = 24</th>
<th>L = 32</th>
<th>L = 48</th>
</tr>
</thead>
<tbody>
<tr>
<td>KAE</td>
<td>0.845</td>
<td>0.883</td>
<td>0.914</td>
<td>0.919</td>
<td>0.919</td>
</tr>
<tr>
<td>CNNH</td>
<td>0.819</td>
<td>0.859</td>
<td>0.905</td>
<td>0.912</td>
<td>0.916</td>
</tr>
<tr>
<td>KSH</td>
<td>0.732</td>
<td>0.752</td>
<td>0.774</td>
<td>0.801</td>
<td>0.793</td>
</tr>
<tr>
<td>ITQ-S</td>
<td>0.669</td>
<td>0.684</td>
<td>0.715</td>
<td>0.742</td>
<td>0.734</td>
</tr>
<tr>
<td>BA</td>
<td>0.316</td>
<td>0.355</td>
<td>0.364</td>
<td>0.362</td>
<td>0.358</td>
</tr>
<tr>
<td>KLSH</td>
<td>0.312</td>
<td>0.353</td>
<td>0.365</td>
<td>0.351</td>
<td>0.349</td>
</tr>
</tbody>
</table>

Table 3. mAP results of different methods on NUS-WIDE dataset (the mAP values was calculated within the top 5000 returned neighbors).

<table>
<thead>
<tr>
<th>Method</th>
<th>L = 8</th>
<th>L = 16</th>
<th>L = 24</th>
<th>L = 32</th>
<th>L = 48</th>
</tr>
</thead>
<tbody>
<tr>
<td>KAE</td>
<td>0.551</td>
<td>0.566</td>
<td>0.572</td>
<td>0.577</td>
<td>0.571</td>
</tr>
<tr>
<td>CNNH</td>
<td>0.545</td>
<td>0.558</td>
<td>0.567</td>
<td>0.576</td>
<td>0.572</td>
</tr>
<tr>
<td>KSH</td>
<td>0.480</td>
<td>0.486</td>
<td>0.491</td>
<td>0.492</td>
<td>0.488</td>
</tr>
<tr>
<td>ITQ-S</td>
<td>0.472</td>
<td>0.480</td>
<td>0.487</td>
<td>0.471</td>
<td>0.471</td>
</tr>
<tr>
<td>BA</td>
<td>0.465</td>
<td>0.471</td>
<td>0.475</td>
<td>0.478</td>
<td>0.472</td>
</tr>
<tr>
<td>KLSH</td>
<td>0.456</td>
<td>0.470</td>
<td>0.476</td>
<td>0.478</td>
<td>0.470</td>
</tr>
</tbody>
</table>

5. Conclusion

In this paper, we have proposed a new hash function learning method for semi-supervised image retrieval by kernelizing auto-encoder models. The key idea of our hash mechanism which is easy to implement and parallelizes well is to exploit input embedding that captures non-linear relationships between the training data points and we use them to learn better hash functions for images. Experiment results confirmed the improvements of our method over previous works in the search accuracy of three image retrieval benchmark datasets.

References


