Linear Distance Preserving Pseudo-Supervised and Unsupervised Hashing

Min Wang[†], Wengang Zhou[†], Qi Tian[‡], Zhengjun Zha[†], Houqiang Li[†]
[†] CAS Key Laboratory of Technology in Geo-spatial Information Processing and Application System, University of Science and Technology of China
[‡] Computer Science Department, University of Texas at San Antonio
wm123@mail.ustc.edu.cn, {zhwg, zhazj, lihq}@ustc.edu.cn, qitian@cs.utsa.edu

ABSTRACT

With the advantage in compact representation and efficient comparison, binary hashing has been extensively investigated for approximate nearest neighbor search. In this paper, we propose a novel and general hashing framework, which simultaneously considers a new linear pair-wise distance preserving objective and point-wise constraint. The direct distance preserving objective aims to keep the linear relationships between the Euclidean distance and the Hamming distance of data points. Based on different point-wise constraints, we propose two methods to instantiate this framework. The first one is a pseudo-supervised hashing method, which uses existing unsupervised hashing methods to generate binary codes as pseudo-supervised information. The second one is an unsupervised hashing method, in which quantization loss is considered. We validate our framework on two large-scale datasets. The experiments demonstrate that our pseudo-supervised method achieves consistent improvement for the state-of-the-art unsupervised hashing methods, while our unsupervised method outperforms the state-of-the-art methods.

Keywords

Distance preserving hashing; approximate nearest neighbor search; learning to hash

1. INTRODUCTION

Approximate nearest neighbors (ANN) search is a fundamental research problem in computer vision and multimedia fields. Given a query sample, ANN search aims to identify those nearest data points with high probability from a large corpus with a sub-linear, or even constant time complexity. To solve this problem efficiently, lots of methods have been proposed, such as tree-based approaches [1] and hashing algorithms [31, 2, 30, 17, 32, 29, 33, 22]. With the advantage in compact representation and efficient comparison, binary hashing techniques have become more and more popular.

MM '16, October 15-19, 2016, Amsterdam, Netherlands

DOI: http://dx.doi.org/10.1145/2964284.2964334

Existing binary hashing methods can be classified into two categories: data-independent and data-dependent methods according to whether the training data is involved. As the representative data-independent methods, Locality Sensitive Hashing (LSH) [4] and its variants [12, 10] have been widely explored. On the other hand, lots of research efforts are devoted to data-dependent methods to learn more compact hash codes with the training dataset. The data-dependent methods can be divided into three categories according to whether they use labels or class information: unsupervised methods [5, 23, 7, 18, 6, 9, 34], semi-supervised methods [20], and supervised methods [11, 14, 16, 26]. Although in most literature, supervised and semi-supervised methods are demonstrated with much better performance than unsupervised ones, it is a non-trivial issue to assume the availability of labels or class information for training data in many scenarios. In this paper, we only focus on unsupervised binary hashing.

The fundamental of unsupervised hashing is to achieve distance preservation across the original data space and Hamming space, which intuitively results in a pair-wise constraint. Some hashing methods implicitly achieve distance preserving with point-wise constraints, which explore the intrinsic properties of expected binary codes (such as balance and independence) or the data distribution information. For instance, Iterative Quantization (ITQ) [5] learns an orthogonal rotation matrix by minimizing the quantization loss while mapping the data generated by PCA projections to binary codes. It realizes the independence properties of binary codes by the orthogonal basis vectors of PCA. Spherical Hashing (SpH) [7] exploits hypersphere-based hashing function maps spatially coherent data points to similar binary codes. SpH aims to preserve the balance and independence properties of binary code. Although these unsupervised methods achieve promising performance, improvement can be expected by further imposing the additional explicit distance-preserving constraint.

On the other hand, deep-learning based hashing methods become more and more popular. Many supervised hashing methods [13, 24, 28, 27] adopt the deep neural networks or deep convolutional neural networks to generate hash code, which fully explore the pair-wise or triplet-wise constraints denoted by the class labels. But for unsupervised hashing, the methods based on deep neural networks [19, 15, 3] only involve the point-wise constraints, due to the lack of labels or class information. For instance, in [15], a novel neural network is developed to seek multiple hierarchical non-linear transformations to learn binary codes to preserve the non-

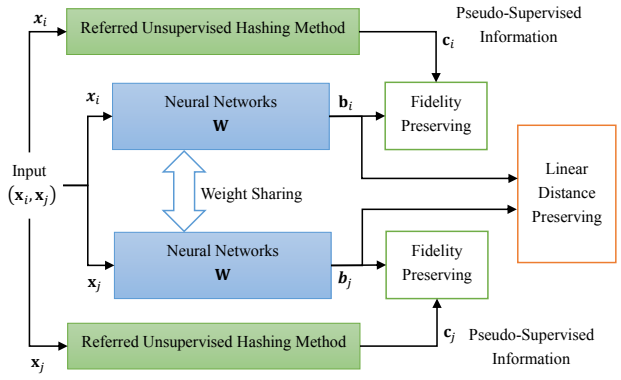

Figure 1: The basic idea of our pseudo-supervised hashing method. The input of the dual neural networks is a pair of data points, $\mathbf{x}_i$ and $\mathbf{x}_j$. $\mathbf{c}_i$ and $\mathbf{c}_j$ are the binary codes generated by the referred unsupervised hashing method, which are used as the pseudo-supervised information for training the dual neural networks in the Fidelity Preserving term. The binarized outputs of the dual neural networks are $\mathbf{b}_i$ and $\mathbf{b}_j$. We use the Hamming distance $h_{i,j}$ between $\mathbf{b}_i$ and $\mathbf{b}_j$ in the Linear Distance Preserving term to train our dual neural networks.

linear relationship of samples. The model in [15] is learnt by minimizing the quantization loss between the original real-valued feature descriptor and the learnt binary code, which is typically a point-wise constraint. Since the point-wise constraint is not essential to ensure the distance-preserving fundamental of unsupervised hashing, the unsupervised methods based on neural networks [19, 15, 3] usually do not achieve much better performance than the classical methods.

In this paper, we propose a novel and general framework which involves a new pair-wise linear distance preserving objective and point-wise constraint. In this framework, our distance preserving objective aims to preserve the linear transformation between the Euclidean distance and the Hamming distance for pairs of data. Based on different point-wise constraints, we propose two methods to instantiate the framework. The first one is a pseudo-supervised hashing method, in which we generate the pseudo-supervised information by an existing unsupervised hashing method. The motivation of this method is to apply pair-wise distance preserving objective based on neural networks to improve the performances of the referred unsupervised hashing methods. We preserve the new distance preserving objective, and simultaneously keep the fidelity of the selected unsupervised hashing method in this method. This method is solved by dual neural networks. The basic idea of this method is shown in Fig. 1. Note that the pseudo-supervised method is a general boosting method, which can be applied to existing unsupervised hashing methods. The second one is an unsupervised hashing method, in which we minimize the quantization loss between the projected data and the binarized data. By fully exploring pair-wise and point-wise constraints at the same time, our unsupervised hashing method demonstrates excellent performance.

There are three main contributions in this paper:

- A novel and general linear distance preserving hashing framework is proposed, in which a new pair-wise linear distance preserving objective and point-wise constraint are considered simultaneously.
- The framework is instantiated by a pseudo-supervised hashing method, which provides a general paradigm to boost existing unsupervised hashing methods. The experiments on two large-scale datasets validate that our pseudo-supervised method improves the performances of existing state-of-the-art methods largely.
- A new unsupervised hashing method is developed to instantiate the framework, which simultaneously preserve the pairwise distance-preserving objective and the point-wise constraint. The experiments prove that our unsupervised method outperforms the state-of-the-art unsupervised hashing methods.

The rest of this paper is organized as follow. We first review the related work with our approach in Section 2. Then in Section 3, we present the proposed hashing framework, and its two instantiations, *i.e.*, pseudo-supervised method and unsupervised hashing method. After that, the experiments are shown in Section 4. Finally we make a conclusion in Section 5.

2. RELATED WORK

In this section, we review the related work from two aspects: unsupervised binary hashing methods, and the use of deep learning techniques in hashing.

Unsupervised hashing methods learn hash functions with training data points without any label or class information. The fundamental of unsupervised hashing methods is distance preservation across the original space and the Hamming space. K-means Hashing (KMH) [6] simultaneously performs k-means clustering and learns the binary indices of the quantized cells. The Euclidean distance between data points is implicitly approximated by the Hamming distance between the binary indices of the corresponding cells. Spectral Hashing (SH) [23] formulates the hash function learning problem as a particular form of graph partition to seek a binary code with balanced and uncorrelated bits. Minimal Loss Hashing (MLH) [18] adopts a pairwise hinge-like loss function and minimizes its upper bound to learn binary codes. Order Preserving Hashing (OPH) [21] learns hash functions by maximizing the alignment between the similarity orders computed from the original space and the ones in the Hamming space. Topology Preserving Hashing (TPH) [25] aims at preserving the neighbor rankings of data points in Hamming space. Binary Reconstructive Embedding (BRE) [11] utilizes pairwise relations between samples and minimizes the squared error between the original normalized Euclidean distance and the normalized Hamming distance. In our approach, we generalize the distance preserving constraint with a linear transformation and optimize it in a different way.

Nowadays, deep learning becomes more and more popular. Many supervised hashing methods are involved with deep learning techniques. [24] proposes a supervised hashing method based on convolutional neural networks, which automatically learns a good image representation tailored to hashing as well as a set of hash functions. [13] proposes a deep architecture for supervised hashing, in which images

are mapped into binary codes via carefully designed deep neural networks. The model in [13] considers a triplet ranking loss denoted by the image labels. [28] proposes a deep semantic ranking based method for hashing function learning to handle multi-level semantic similarity between multi-label images. [27] proposes a supervised learning framework to generate compact and bit-scalable hashing code directly from raw images. The model in [27] maximizes the margin between the matched pairs and the mismatched pairs in the Hamming space.

On the other hand, most existing unsupervised hashing methods based on deep learning techniques learn to generate binary codes by exploring the data distribution or the intrinsic properties of expected binary codes. To our best knowledge, there is no work explicitly exploring distance preserving constraint with neural networks for unsupervised binary hashing, which is probably due to the lack of labels or class information. Semantic hashing [19] is a pioneering work using deep learning techniques for hashing. They apply the stacked Restricted Boltzmann Machine (RBM) to learn compact binary codes for visual search. [3] also uses a point-wise objective function. It focuses on the binary auto-encoder model, which seeks to reconstruct a data point from binary code produced by the hash functions. Compared with these unsupervised methods, our pseudo-supervised method uses dual neural networks to realize linear distance projection relationship and simultaneously keep original hashing functions learnt by existing hashing methods, which involves pairwise constraint and adheres to the distance-preserving goal of hashing.

3. OUR APPROACH

In this section, we first give some notations to facilitate the following discussion in Section 3.1. Then, we discuss our linear distance preserving objective and the corresponding general hashing framework in Section 3.2 and 3.3. After that, we discuss two instantiations of our hashing framework in Section 3.4 and 3.5, respectively.

3.1 Notations

Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times D}$, each row of $\mathbf{X}$ is a high dimensional data vector $\mathbf{x}_i \in \mathbb{R}^D$. The Euclidean distance between the original data vector $\mathbf{x}_i$ and $\mathbf{x}_j$ is computed by $L2$-norm, $d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$, and we denote it as $d_{i,j}$ for conciseness.

Using binary hashing, each original high dimensional data vector $\mathbf{x}_i$ is mapped to a low dimensional binary vector $\mathbf{b}_i \in \mathbb{H}^L$ (usually, $L << D$). The resulted binary vectors significantly reduce the memory cost and the distance in Hamming space (*i.e.*, Hamming distance) can be efficiently computed by modern CPUs. The Hamming distance $h(\mathbf{b}_i, \mathbf{b}_j)$ between the corresponding binary vector $\mathbf{b}_i$ and $\mathbf{b}_j$ is denoted as $h_{i,j}$ for conciseness.

The usual paradigm of binary hashing is to first map each data point into a low dimensional space, then quantize the mapped data point to a binary vector. Usually the mapping function is supposed to be a linear form. Let $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_L] \in \mathbb{R}^{D \times L}$ be the learned projection matrix, then the mapping of $\mathbf{X}$ can be obtained as $\mathbf{U} = \mathbf{XW}$, which is further binarized to obtain the binary codes as follows:

$$\mathbf{B} = (sgn(\mathbf{U}) + 1)/2 = (sgn(\mathbf{XW}) + 1)/2, \quad (1)$$

where $sgn(\cdot)$ is an element-wise sign function.

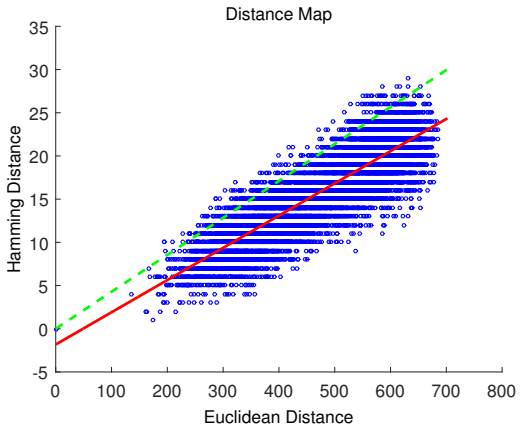

Figure 2: Distance map between Hamming distance and Euclidean distance of pairs. The Hamming distance of a pair of data points is computed from the corresponding binary codes generated by ITQ[5]. Each blue point denotes the Euclidean distance and the corresponding Hamming distance of a pair of data points. The green dash line is adopted in BRE [11]. The red line is generated by our method, which fits the distribution of the distance map.

3.2 Linear Distance Preserving Objective

The main methodology of unsupervised hashing methods is distance-preserving across the original Euclidean space and Hamming space, *i.e.*, $h_{i,j} \propto d_{i,j}$. To utilize the distance-preserving constraint in our framework, we propose a new distance preserving objective, in which we keep the linear projection relationship of the two distances between pairwise data points. The objective can be achieved by minimizing the following function:

$$\Phi(\mathbf{W}, a, b) = \|\mathbf{H} - a\mathbf{E} - b\|_F^2, \quad (2)$$

where $\|\cdot\|_F$ denotes Frobenius norm, a and b[1] are the parameters of linear distance transformation, which define the red line in Fig. 2 for intuitive, and implicitly impact the selection of the projection matrix $\mathbf{W}$. $\mathbf{E}$ and $\mathbf{H} \in \mathbb{R}^{n \times p}$, where p denotes the number of pairs for each data point. The element $\mathrm{E}(i,j)$ in $\mathbf{E}$ denotes the $L2$ distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ in the Euclidean space, *i.e.*, $d_{i,j}$, while the element $\mathrm{H}(i,j)$ in $\mathbf{H}$ denotes the Hamming distance between the corresponding binary code $\mathbf{b}_i$ and $\mathbf{b}_j$ in the Hamming space, *i.e.*, $h_{i,j}$. To facilitate the solution, the Hamming distance between the binary code $\mathbf{b}_i$ and $\mathbf{b}_j$ can be described as

$$\mathrm{H}(i,j) = L - \mathbf{b}_i^T\mathbf{b}_j - (\mathbf{1}_{L\times 1} - \mathbf{b}_i)^{\mathrm{T}}(\mathbf{1}_{L\times 1} - \mathbf{b}_j). \quad (3)$$

We give a visual illustration for this distance preserving goal in Fig. 2. We randomly select 10,000 pairs of data points from ANN_SIFT1M dataset [8] and compute their Euclidean distances. Then we generate their corresponding binary codes in Hamming space by ITQ [5] with code length equal to 32 bits, and compute the Hamming distance between the binary codes of these pairs. Each pair of data

[1] a, b are scalars, the multiplication between a and $\mathbf{E}$ is element-wise multiplication between a and each element of $\mathbf{E}$, and the subtraction to b is also element-wise, we omit the $\mathbf{1}$ matrix for conciseness.

points is painted by a blue marker in Fig. 2. The red linear line is generated by Least Square method applied on those distance pairs. All those data pairs distribute in a long strip area, which locates around the red line. We can intuitively find that if the strip area shrinks towards the red linear line, *i.e.*, becomes thinner, the distance preserving performance of the hash functions will be better. The shrink manipulation makes the pairs with the same Euclidean distance have more similar Hamming distances, which implicitly realizes the distance-preserving objective.

Similar to our approach, BRE [11] imposes distance preserving constraint by minimizing the deviation between the normalized Hamming distance and the normalized Euclidean distance. It can be regarded as a special case of our method, when $a = L/e_{max}, b = 0$ (e_{max} is the maximum of the Euclidean distance). The BRE method chooses the diagonal of the coordinate area as the supervisor of distance map, as the green dash line in Fig. 2. We can intuitively find the diagonal line is not essentially the best choice for the real data distribution, which makes the strip area shrink slow and degrades the performance.

3.3 Linear Distance Preserving Framework

In our framework, we randomly select N_p pairs of data, and save their indices into $\mathbf{P} = \left[\mathbf{p}_1^T, \mathbf{p}_2^T, \ldots, \mathbf{p}_{N_p}^T\right]^T \in \mathbb{R}^{N_p \times 2}$, where each row $\mathbf{p}_i \in \mathbb{R}^{1\times 2}$ denotes the indices of a pair of data points. Then we can compute the $L2$ distance $\mathbf{d}$ for these pairs. Noted that the $\mathbf{d} \in \mathbb{R}^{N_p \times 1}$ here is a vector, which can be seen as the vector form of the matrix $\mathbf{E}$ in Eq. (2). To make our description clear, we rewrite two vector forms of the indices $\mathbf{P}$ as $\tilde{\mathbf{P}} = \big[\mathbf{p}_1(1), \mathbf{p}_2(1), \ldots, \mathbf{p}_{N_p}(1), \mathbf{p}_1(2), \mathbf{p}_2(2), \ldots, \mathbf{p}_{N_p}(2)\big]^T$ and $\hat{\mathbf{P}} = \big[\mathbf{p}_1(2), \mathbf{p}_2(2), \ldots, \mathbf{p}_{N_p}(2), \mathbf{p}_1(1), \mathbf{p}_2(1), \ldots, \mathbf{p}_{N_p}(1)\big]^T$. We use these indices to reorganize our training data matrix $\mathbf{X}$ as $\tilde{\mathbf{X}} = \mathbf{X}(\tilde{\mathbf{P}}) \in \mathbb{R}^{2N_p \times D}$, in which each row $\tilde{\mathbf{x}}_i$ of $\tilde{\mathbf{X}}$ is $\mathbf{x}_{\tilde{\mathbf{P}}_i}$. Similarly, $\hat{\mathbf{X}} = \mathbf{X}(\hat{\mathbf{P}})$, in which each row $\hat{\mathbf{x}}_i$ of $\hat{\mathbf{X}}$ is $\mathbf{x}_{\hat{\mathbf{P}}_i}$. In our optimization, we replace the sign function with sigmoid function, *i.e.*, the projected data $\mathbf{U} = sigmoid(\mathbf{XW}) \in \mathbb{R}^{n\times L}$, and the binary codes $\mathbf{B} = round(\mathbf{U})$. We reorganize $\mathbf{U}$ as $\tilde{\mathbf{U}} = \mathbf{U}(\tilde{\mathbf{P}}) \in \mathbb{R}^{2N_p \times L}$, and $\hat{\mathbf{U}} = \mathbf{U}(\hat{\mathbf{P}})$. And we can compute Hamming distance $\mathbf{h}$ for all the pairs as Eq. (3).

Our linear distance preserving framework simultaneously considers the pairwise linear distance preserving objective, point-wise constraint, and orthogonal projection constraint. The whole optimization function of the proposed framework can be devised as follows:

$$\min_{\mathbf{W},a,b} \frac{\lambda}{N_p}\|\mathbf{h} - a\mathbf{d} - b\|_2^2 + \frac{\alpha}{N_p}\left\|\tilde{\mathbf{U}} - \mathbf{C}\right\|_F^2 + \beta\|\mathbf{W}^T\mathbf{W} - \mathbf{I}\|_F^2, \quad (4)$$

where the first term focuses on linear distance preserving, which can be seen as the vector form of Eq. (2), and the third term imposes the orthogonality constraint between projection vectors to achieve the bit independence. The second term is a point-wise constraint, and $\mathbf{C}$ can be interpreted in different ways, which leads to different instantiations. When $\mathbf{C}$ is obtained by a certain existing binary hashing method, we can obtain a pseudo-supervised hashing scheme, which can be realized with dual neural networks, as discussed in the following Section 3.4. On the other hand, we can also regard $\mathbf{C}$ as a constant matrix and derive a new unsupervised hashing scheme, which will be discussed in Section 3.5.

3.4 Linear Distance Preserving Pseudo Supervised Hashing

Since the existing unsupervised hashing methods have fully explore the data distribution or the intrinsic properties of expected binary codes, which helps learn good hashing function. We instantiate the proposed framework by a pseudo-supervised hashing method, which can boost the performance of existing unsupervised hashing methods by further imposing the pairwise distance preserving objective. In this implementation, we first select an existing unsupervised hashing method to generate binary codes $\mathbf{C}$ as pseudo-supervised information. The supervised binary codes $\mathbf{C}$ are also reorganized as $\tilde{\mathbf{C}} = \mathbf{C}(\tilde{\mathbf{P}}) \in \mathbb{H}^{2N_p \times L}$. This method preserves the linear distance transformation relationships, and simultaneously keeps the hashing functions learnt by original unsupervised hashing methods. It merges the pseudo-supervised information into our framework as follows:

$$\min_{\mathbf{W},a,b} \frac{\lambda}{N_p}\|\mathbf{h} - a\mathbf{d} - b\|_2^2 + \frac{\alpha}{N_p}\left\|\tilde{\mathbf{U}} - \tilde{\mathbf{C}}\right\|_F^2 + \beta\|\mathbf{W}^T\mathbf{W} - \mathbf{I}\|_F^2, \quad (5)$$

where the second part of this objective aims to preserve the fidelity of the involved unsupervised method. This part can make the binarized output of our method similar with the output of the original unsupervised hashing method, which preserves the projections of the referred unsupervised hashing functions. Here to solve the whole objective function with pairwise distance-preserving constraint, we propose a new learning algorithm based on dual neural networks. We use neural networks here with the motivation of the fact that neural networks can imitate the transformations of any complexity. The two neural networks share the same structure, parameters, and weight matrices, but are fed with different input samples, as shown in Fig. 1.

It is difficult to simultaneously optimize the Eq. (5) with respect to $\mathbf{W}$, a, and b. To address this difficulty, we propose an alternative scheme. First, we fix $\mathbf{W}$, and optimize the objective function with respect to a and b. Then we fix a and b, and optimize the objective function with respect to $\mathbf{W}$. The above two steps are repeated until we achieve a convergence. In the following, we discuss the learning process in details.

3.4.1 a, b-step

With $\mathbf{W}$ fixed, the optimization on the objective function in Eq. (5) becomes

$$\min_{a,b} \|\mathbf{h} - a\mathbf{d} - b\|_2^2. \quad (6)$$

This is a Linear Regression problem, which can be directly solved by Least Square method.

3.4.2 $\mathbf{W}$-step

With a and b fixed, the optimization in Eq. (5) degenerates to

$$\min_{\mathbf{W}} \frac{\lambda}{N_p}\|\mathbf{h} - a\mathbf{d} - b\|_2^2 + \frac{\alpha}{N_p}\left\|\tilde{\mathbf{U}} - \tilde{\mathbf{C}}\right\|_F^2 + \beta\|\mathbf{W}^T\mathbf{W} - \mathbf{I}\|_F^2. \quad (7)$$

In our optimization, we use the Back Propagation to learn the weight matrices of neural networks. We only need to compute the gradient of the objective function with respect to the weight matrix of output layer, which is denoted by $\mathbf{W}$ for simplicity.

To compute the gradient of the output layer, we expand the Eq. (7) as follows:

$$J(\mathbf{W}) = \frac{1}{N_p}\sum_i \Big\{\lambda\left(h_i - ad_i - b\right)^2 + \alpha\|\mathbf{u}_{\mathbf{p}_i(1)} - \mathbf{c}_{\mathbf{p}_i(1)}\|_2^2 + \alpha\|\mathbf{u}_{\mathbf{p}_i(2)} - \mathbf{c}_{\mathbf{p}_i(2)}\|_2^2\Big\} + \beta\|\mathbf{W}^T\mathbf{W} - \mathbf{I}\|_F^2. \quad (8)$$

According to the above objective function and the chain rule in computing derivatives, we can compute the gradient of the objective function as follows:

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{2}{N_p}\sum_i \Big\{\lambda\left(h_i - ad_i - b\right)\Big(\frac{\partial h_i}{\partial \mathbf{b}_{\mathbf{p}_i(1)}}\cdot\frac{\partial \mathbf{b}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}} + \frac{\partial h_i}{\partial \mathbf{b}_{\mathbf{p}_i(2)}}\cdot\frac{\partial \mathbf{b}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}\Big) + \alpha\big(\mathbf{u}_{\mathbf{p}_i(1)} - \mathbf{c}_{\mathbf{p}_i(1)}\big)\frac{\partial \mathbf{u}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}} + \alpha\big(\mathbf{u}_{\mathbf{p}_i(2)} - \mathbf{c}_{\mathbf{p}_i(2)}\big)\frac{\partial \mathbf{u}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}\Big\} + 2\beta\mathbf{W}\Big(\mathbf{W}^T\mathbf{W} - \mathbf{I}\Big), \quad (9)$$

where

$$\frac{\partial h_i}{\partial \mathbf{b}_{\mathbf{p}_i(1)}} = (\mathbf{1} - 2\mathbf{b}_{\mathbf{p}_i(2)}), \frac{\partial h_i}{\partial \mathbf{b}_{\mathbf{p}_i(2)}} = (\mathbf{1} - 2\mathbf{b}_{\mathbf{p}_i(1)}). \quad (10)$$

For the $\frac{\partial \mathbf{b}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}}$ and $\frac{\partial \mathbf{b}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}$ in Eq. (9), we can not compute them directly because $\mathbf{b}_{\mathbf{p}_i(1)}$ and $\mathbf{b}_{\mathbf{p}_i(2)}$ are not continuous. We use the sigmoid function instead of the binarized function. In other words, we use the $\frac{\partial \mathbf{u}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}}$ instead of $\frac{\partial \mathbf{b}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}}$, and $\frac{\partial \mathbf{u}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}$ instead of $\frac{\partial \mathbf{b}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}$. The terms $\frac{\partial \mathbf{u}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}}$ and $\frac{\partial \mathbf{u}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}$ are computed as the classical neural networks do. Compared with classical neural networks, we can find the dual neural networks have different coefficients for each $\frac{\partial \mathbf{u}}{\partial \mathbf{W}}$.

In the experiments, we use the original unsupervised hashing method to pretrain the neural networks. The objective function used in the pretraining is the same as the second term in Eq. (5). The binary codes generated by the original unsupervised hashing method are used as supervised information in the pretraining. Since the two neural networks in the dual neural networks share the same parameters, we initialize them with the same pretrained neural networks.

Note that after each iteration, we use one neural network from the dual neural networks as the hash projection functions, and generate a new supervised matrix instead of the old $\mathbf{C}$. Since after the updating process for $\mathbf{W}$ converges, the binary codes generated by our hash functions are expected to compute new parameters of linear distance transformation. The new parameters are expected to be better than the old ones to fit the distance distribution. The learning algorithm is summarized in Algorithm 1.

3.5 Linear Distance Preserving Unsupervised Hashing

In this section, we describe a simple instantiation of our proposed linear distance preserving hashing framework, in which the quantization loss is considered as most unsupervised hashing methods do. This leads to an unsupervised hashing method, we denote it as Linear Distance Transformation Hashing (LDTH). The final objective of our LDTH method is described as follows:

$$\min_{\mathbf{W},a,b}\frac{\lambda}{N_p}\|\mathbf{h} - a\mathbf{d} - b\|_2^2 - \frac{\alpha}{N_p}\|\tilde{\mathbf{U}} - 0.5\|_F^2 + \beta\|\mathbf{W}^T\mathbf{W} - \mathbf{I}\|_F^2, \quad (11)$$

where λ, α, and β are set as constants, and N_p denotes the number of all involved training pairs. Maximizing the second term in the above function is equivalent to minimize the quantization loss[2].

Similar with the optimization scheme discussed in Section 3.4, we alternatively optimize Eq. (11) over $\mathbf{W}$ and a, b. We first fix $\mathbf{W}$ and optimize with respect to a and b. This step is the same as the solution of Eq.(6) discussed in Section 3.4.1. After that, we fix a and b, and optimize with respect to $\mathbf{W}$. The only difference between Eq. (5) and Eq. (11) is the second part of the two objective functions. The second part of Eq. (11) focuses on the quantization loss, while the second part in Eq. (5) imposes the fidelity constraints of the referred unsupervised hashing method. This step is a little different from the solution described in Section 3.4.2. In the following, we describe the learning process with respect to $\mathbf{W}$ in detail.

Algorithm 1 The pseudo-supervised boosting method

Input:
Training data matrix $\mathbf{X}$, the number of training pairs N_p, the required length L of binary code, and the pseudo-supervised binary codes $\mathbf{C}$ generated by one unsupervised hashing method.

Output:
One trained neural network $\mathbf{W}$.

1: Initialization:
Randomly select N_p training pairs $\mathbf{P}$;
Pretrain one neural network;
Initialize the dual neural networks with the pretrained neural network.
2: Generate $\tilde{\mathbf{P}}$ and $\hat{\mathbf{P}}$.
3: Compute the $L2$ distance $\mathbf{d}$ for all the pairs.
4: **Repeat:**
5: Input $\mathbf{X}$, compute the output of neural networks $\mathbf{U}$, $\mathbf{B} = round(\mathbf{U})$, $\tilde{\mathbf{C}} = \mathbf{C}(\tilde{\mathbf{P}})$.
6: Compute the Hamming distance $\mathbf{h}$ for all the pairs.
7: Compute a, b in Eq. (6) using Least Squared method.
8: **Repeat:**
9: Input $\mathbf{X}$, compute the output of neural networks $\mathbf{U}$, $\mathbf{B} = round(\mathbf{U})$, $\tilde{\mathbf{U}} = \mathbf{U}(\tilde{\mathbf{P}})$, $\hat{\mathbf{U}} = \mathbf{U}(\hat{\mathbf{P}})$.
10: Compute the Hamming distance $\mathbf{h}$ for all the pairs.
11: Compute the gradient of output layer as Eq. (9).
12: Update the weight matrices of dual neural networks by Back Propagation.
13: **until** convergence
14: Generate new pseudo-supervised binary codes $\mathbf{C}$ by one trained neural network.
15: **until** convergence

[2] 0.5 is a scalar, the subtraction to 0.5 is an element-wise manipulation, *i.e.*, each element of $\tilde{\mathbf{U}}$ subtracts 0.5. We omit the full-one matrix $\mathbf{1}$ for conciseness.

With a and b fixed, the optimization in Eq. (11) becomes

$$\min_{\mathbf{W}} \frac{\lambda}{N_p}\|\mathbf{h} - a\mathbf{d} - b\|_2^2 - \frac{\alpha}{N_p}\|\tilde{\mathbf{U}} - 0.5\|_F^2 + \beta\|\mathbf{W}^T\mathbf{W} - \mathbf{I}\|_F^2. \quad (12)$$

We use Gradient Descent algorithm to solve the above optimization problem. To compute the gradient, we first expand the above function as follows:

$$J(\mathbf{W}) = \frac{1}{N_p}\sum_i \Big\{\lambda\left(h_i - ad_i - b\right)^2 - \alpha\|\mathbf{u}_{\mathbf{p}_i(1)} - 0.5\|_2^2 - \alpha\|\mathbf{u}_{\mathbf{p}_i(2)} - 0.5\|_2^2\Big\} + \beta\|\mathbf{W}^T\mathbf{W} - \mathbf{I}\|_F^2. \quad (13)$$

According to the Eq. (13) and the chain rule in computing derivatives, we can compute the gradient of the objective function as follows

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{2}{N_p}\sum_i \Big\{\lambda\left(h_i - ad_i - b\right)\left(\frac{\partial h_i}{\partial \mathbf{b}_{\mathbf{p}_i(1)}}\cdot\frac{\partial \mathbf{b}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}} + \frac{\partial h_i}{\partial \mathbf{b}_{\mathbf{p}_i(2)}}\cdot\frac{\partial \mathbf{b}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}\right) - \alpha\big(\mathbf{u}_{\mathbf{p}_i(1)} - 0.5\big)\frac{\partial \mathbf{u}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}} - \alpha\big(\mathbf{u}_{\mathbf{p}_i(2)} - 0.5\big)\frac{\partial \mathbf{u}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}\Big\} + 2\beta\mathbf{W}\Big(\mathbf{W}^T\mathbf{W} - \mathbf{I}\Big), \quad (14)$$

where $\frac{\partial h_i}{\partial \mathbf{b}_{\mathbf{p}_i(1)}}$ and $\frac{\partial h_i}{\partial \mathbf{b}_{\mathbf{p}_i(2)}}$ share the same forms as in Eq. (10). And we also use the $\frac{\partial \mathbf{u}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}}$ instead of $\frac{\partial \mathbf{b}_{\mathbf{p}_i(1)}}{\partial \mathbf{W}}$, and $\frac{\partial \mathbf{u}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}$ instead of $\frac{\partial \mathbf{b}_{\mathbf{p}_i(2)}}{\partial \mathbf{W}}$. Then we compute the gradient of the objective function as follows:

$$\frac{\partial J}{\partial \mathbf{W}} = \tilde{\mathbf{X}}^T\left(\left\{\lambda\begin{bmatrix}\mathbf{h} - a\mathbf{d} - b\\ \mathbf{h} - a\mathbf{d} - b\end{bmatrix}\circ\left(1 - 2\hat{\mathbf{B}}\right) - \alpha\left(\tilde{\mathbf{U}} - 0.5\right)\right\}\odot\tilde{\mathbf{U}}\odot\left(1 - \tilde{\mathbf{U}}\right)\right)\frac{2}{N_p} + 2\beta\mathbf{W}\Big(\mathbf{W}^T\mathbf{W} - \mathbf{I}\Big). \quad (15)$$

where $\odot$ denotes element-wise multiplication, and $\circ$ denotes that each element of the left vector element-wisely multiplies the elements from the same row of the right matrix. After that, we can update the projection matrix as follows

$$\mathbf{W} = \mathbf{W} - \delta\frac{\partial J}{\partial \mathbf{W}}, \quad (16)$$

where δ denotes learning rate, and we empirically set $\delta = 0.8$ in all the experiments. At the beginning of our learning algorithm, we initialize $\mathbf{W}$ as a randomized orthogonal matrix. We repeat this gradient-descent algorithm until the objective function in Eq. (12) converges. The whole learning algorithm is summarized in Algorithm 2.

Although our method can not guarantee to converge to a global minimum, it always leads to a local minimum, which suffices to give a good result. Typically, we show the convergence curve of our cost function in Fig. 3.

Algorithm 2 Linear Distance Transformation Hashing

Input:
Training data matrix $\mathbf{X}$, the number of training pairs N_p, the required length L of binary code.
Output:
The projection matrix $\mathbf{W}$.
1: Initialization:
Randomly select N_p training pairs $\mathbf{P}$,
generate a randomized orthogonal matrix $\mathbf{W}$.
2: Generate $\tilde{\mathbf{P}}$ and $\hat{\mathbf{P}}$.
3: Compute the $L2$ distance $\mathbf{d}$ for all the pairs.
4: **Repeat:**
5: $\mathbf{U} = sigmoid(\mathbf{XW})$, $\mathbf{B} = round(\mathbf{U})$.
6: Compute the Hamming distance $\mathbf{h}$ for all the pairs.
7: Compute a, b in Eq. (6) using Least Squared method.
8: **Repeat:**
9: $\mathbf{U} = sigmoid(\mathbf{XW})$, $\mathbf{B} = round(\mathbf{U})$,
$\tilde{\mathbf{U}} = \mathbf{U}(\tilde{\mathbf{P}})$, $\hat{\mathbf{B}} = \mathbf{B}(\hat{\mathbf{P}})$.
10: Compute the Hamming distance $\mathbf{h}$ for all the pairs.
11: Compute the gradient of objective function in Eq. (12) as Eq. (15).
12: Update $\mathbf{W} = \mathbf{W} - \delta\frac{\partial J}{\partial \mathbf{W}}$.
13: **until** convergence
14: **until** convergence

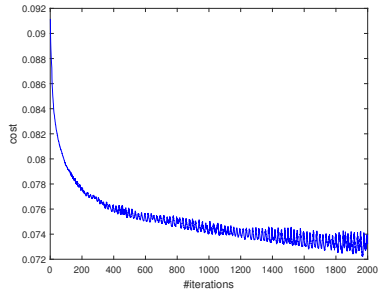

Figure 3: The convergence curve of our cost function.

4. EXPERIMENTS

4.1 Setup

We evaluate our method on two large-scale datasets: (1) ANN_SIFT1M [8]: this dataset consists of 10,000 query vectors, 1,000,000 base vectors, and 100,000 training vectors, with each vector corresponding to a 128-D SIFT feature. The groundtruth for each query corresponds to its 100 nearest neighbors ordered by increasing Euclidean distance. (2) ANN_GIST1M [8]: this dataset consists of 1,000 query vectors, 1,000,000 base vectors, and 500,000 training vectors, and each vector is a 960-D GIST feature. The groundtruth for each query contains its 100 nearest neighbors ordered by Euclidean distance in ascending order.

Since our method is unsupervised, we compare it with five representative unsupervised hashing methods, including LSH [4], BRE [11], SpH [7], ITQ [5], and MLH's unsupervised version [18]. In our experiments, we use the implementations of all these five hashing methods released by their authors with the default parameters.

In this paper, we use binary codes to perform approximate nearest neighbor search based on hash code ranking strategy, which sorts the binary codes by increasing Hamming distance. Based on this strategy, we use three evaluation metrics to measure the performance of different methods in this paper:

- Precision@K: it measures the percentage of the true neighbors in the top K retrieved results.

Table 1: Comparison on Precision(%)@500. The bold number indicates the best result under the same bit length setting. The underline text denotes the case that our pseudo-supervised boosting method does not give an improvement over the original algorithm. "Ours" denotes our pseudo-supervised boosting methods on the corresponding unsupervised hashing methods. LDTH denotes the proposed linear distance preserving unsupervised hash method.

Dataset	Code Length	Approaches					
		LSH[4]/Ours	BRE[11]/Ours	MLH[18]/Ours	SpH[7]/Ours	ITQ[5]/Ours	**LDTH**
ANN_SIFT1M	16	0.94/1.12	0.83/1.69	0.71/0.84	1.38/1.51	1.32/**2.30**	1.66
	32	2.52/2.95	2.20/4.27	2.63/2.73	3.65/3.93	3.54/**5.09**	4.12
	64	5.23/6.20	4.45/7.15	5.84/6.39	7.06/7.51	7.03/**7.71**	7.46
	128	9.30/10.21	7.70/8.62	8.64/9.12	10.63/**11.42**	10.82/10.78	11.22
ANN_GIST1M	16	0.32/0.69	0.86/0.99	0.65/0.92	0.76/0.91	1.09/**1.24**	1.23
	32	0.76/1.35	1.85/1.95	1.38/2.00	1.87/1.91	2.22/2.37	**2.45**
	64	1.61/2.79	3.06/3.05	2.74/**3.75**	3.51/3.55	3.37/3.55	3.74
	128	3.25/4.63	4.62/4.76	4.07/5.36	5.39/**5.49**	4.40/5.46	5.18

- Recall@K: it counts the percentage of true neighbors among all the ground-truth in the retrieved K samples.
- mAP (mean Average Precision): it is obtained by computing the area under the Precision-Recall curve.

4.2 Evaluation for the Pseudo-Supervised Hashing

In this section, we focus on evaluating the results of our pseudo-supervised hashing method on different hashing methods on different datasets. We use the dual neural networks to solve our boosting objective. There are two neural networks sharing the same structure and weight matrices. Considering the computational complexity issue, we adopt a neural network with three layers, including one input layer, one hidden layer, and one output layer. In terms of the activation functions, we select the tanh function for the hidden layer, and sigmoid function for the output layer. The number of units in hidden layer is simply set as the number of units in the input layer, and the number of units in the output layer is equal to the code length of objective binary codes. The parameters are set as $\alpha = 0.05$, $\beta = 5 \times 10^{-5}$, and $\lambda = 1/2L$ in all the experiments, where L denotes the code length. And for each dataset, we randomly select 20,000 training samples from the training datasets, and the total number of pairs N_p is 200,000. We alternatively optimize the a, b-step and $\mathbf{W}$-step until convergence. In the alternating update for $\mathbf{W}$, we train the neural networks for 10 epoches with the batch size equal to 500.

Fig. 4, and Fig. 5 show the recall of original unsupervised hashing methods and our corresponding pseudo-supervised boosting method on the ANN_SIFT1M dataset with the code length equal to 16 and 32 bits. Comparing with the original methods, our boosting methods consistently make considerable improvements. Fig. 6, and Fig. 7 show the recall comparisons between the five methods and our corresponding boosting methods on ANN_GIST1M dataset with the code length equal to 16 and 32 bits. Note that on each dataset we only show the results with two settings of code length, the comparisons with other code lengths are similar. We present the precision comparisons on different datasets with more settings of code lengths over all the five unsupervised hashing methods in Table 1. The mAP comparisons on different datasets are shown in Table 2. In sum, our boosting method consistently improves the performances of the five unsupervised hashing methods on different datasets under all code lengths.

4.3 Evaluation for LDTH

We report the results of our proposed LDTH method on different datasets. All the data points used in our method are pre-processed by PCA (Principal Components Analysis), which makes the pairwise dimensions in one data point uncorrelated. This helps to produce an efficient code, as justified in [5]. In all the experiments for each dataset, we randomly select 20,000 training samples from the training datasets, and randomly select two data points from the training samples as a pair. The total number of pairs N_p is 200,000. And we set α and β as 5×10^{-7}, and set λ as $1/2L$ in the experiments. We alternatively optimize the a, b-step and $\mathbf{W}$-step until convergence. In the alternating update for $\mathbf{W}$, we iteratively run the Gradient-Descent algorithm for 10 times before the update of a and b.

Fig. 8 shows the Recall@K on ANN_SIFT1M under different bit length settings. And Fig. 9 shows the Recall@K on ANN_GIST1M under different bit length settings. We evaluate the precision@500 of different methods under different code length settings in Table 1. The mAP comparison is shown in Table 2. From all the comparisons, we can see that our LDTH method outperforms the state-of-the-art methods, if not, gives comparable results.

4.4 Complexity and Time Cost

In this section, we make an analysis on the complexity of our pseudo-supervised hashing method and the proposed unsupervised LDTH method. The main time cost of our method lies in the $\mathbf{W}$-step, *i.e.*, computing the gradient of objective function with respect to $\mathbf{W}$. The computational complexity of computing gradient in LDTH is $O(N_p L^2 + L^3)$. The pseudo-supervised boosting method will be a little slower than LDTH, because it has a more complex networks structure, which gives a large constant factor to the computational complexity. Finally, we test the computational time of the proposed LDTH and boosting methods, and compare them with the baseline unsupervised hashing methods. Our PC is configured with dual-core 2.00 GHz

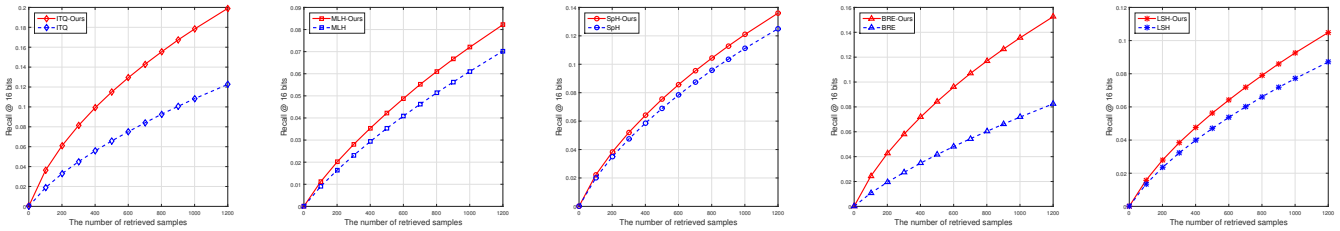

Figure 4: Recall@K comparisons on ANN_SIFT1M between the original unsupervised hashing methods and our corresponding pseudo-supervised boosting methods. Code length: 16

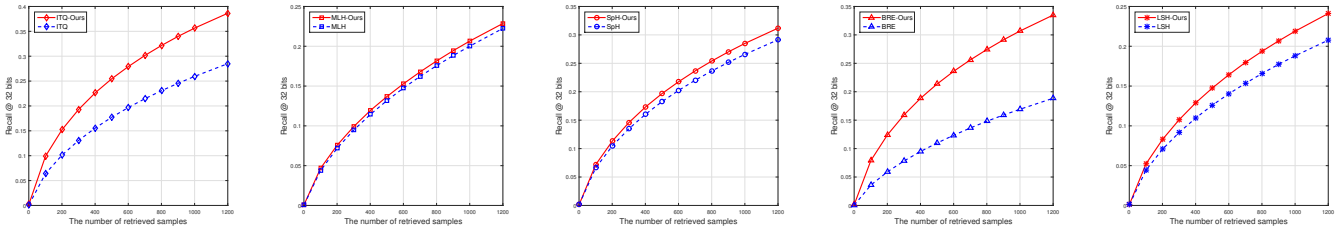

Figure 5: Recall@K comparisons on ANN_SIFT1M between the original unsupervised hashing methods and our corresponding pseudo-supervised boosting methods. Code length: 32

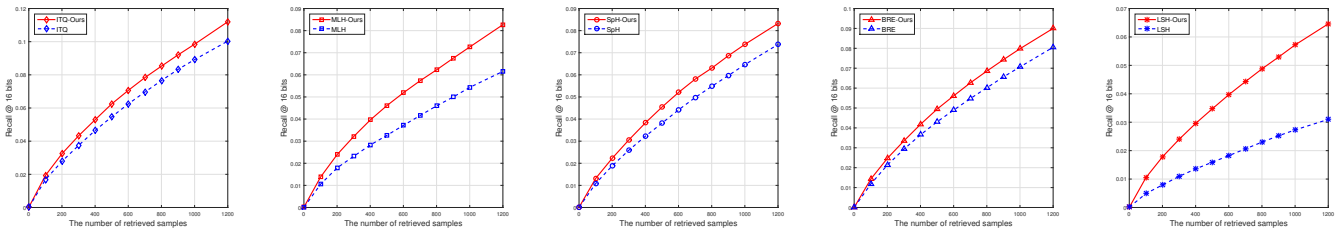

Figure 6: Recall@K comparisons on ANN_GIST1M between the original unsupervised hashing methods and our corresponding pseudo-supervised boosting methods. Code length: 16

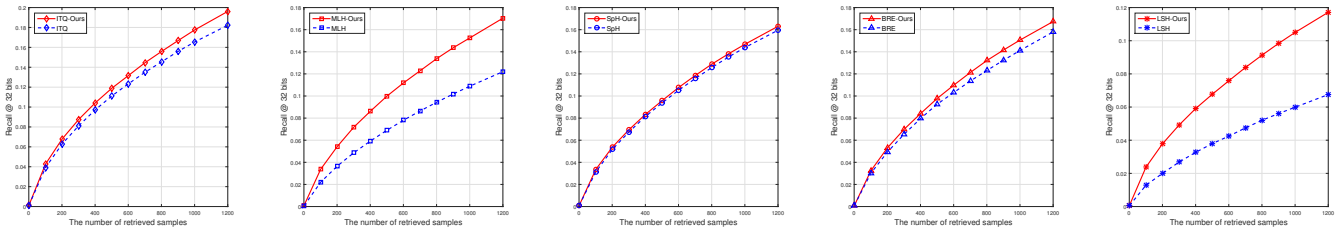

Figure 7: Recall@K comparisons on ANN_GIST1M between the original unsupervised hashing methods and our corresponding pseudo-supervised boosting methods. Code length: 32

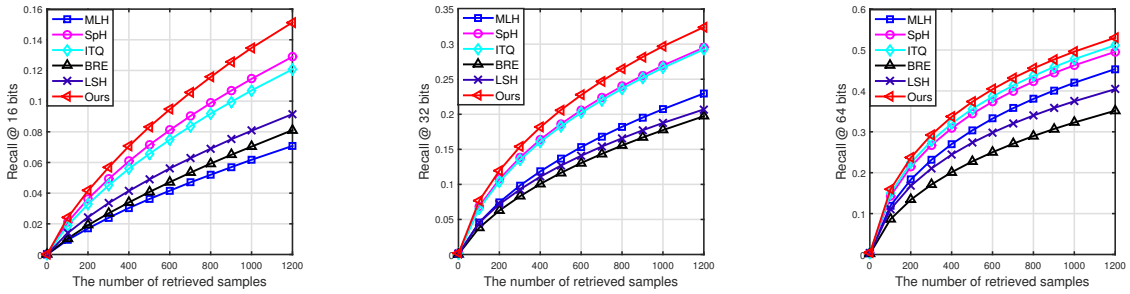

Figure 8: Recall@K comparisons between our LDTH method and the comparison methods on ANN_SIFT1M. The code length of the three figures (from left to right) is 16, 32, and 64, respectively.

Table 2: Comparison on mAP(%). The bold number indicates the best result under the same bit length setting. The underline text denotes the case that our pseudo-supervised boosting method does not give an improvement over the original algorithm. "Ours" denotes our pseudo-supervised boosting methods on the corresponding unsupervised hashing methods. LDTH denotes the proposed linear distance preserving unsupervised hash method.

Dataset	Code Length	Approaches					
		LSH[4]/Ours	BRE[11]/Ours	MLH[18]/Ours	SpH[7]/Ours	ITQ[5]/Ours	**LDTH**
ANN_SIFT1M	16	0.56/0.70	0.57/1.09	0.48/0.56	0.84/0.98	0.93/**1.62**	1.13
	32	2.12/2.55	1.75/4.18	2.17/2.33	3.37/3.75	3.31/**5.52**	4.07
	64	6.29/7.92	4.71/9.73	6.88/7.92	9.47/10.38	9.34/**11.29**	10.53
	128	15.71/18.26	11.11/13.11	12.94/14.19	19.42/**21.74**	19.91/19.68	21.40
ANN_GIST1M	16	0.18/0.38	0.43/0.52	0.38/0.53	0.41/0.49	0.64/**0.75**	0.74
	32	0.56/1.11	1.26/1.40	0.92/1.46	1.34/1.44	1.77/1.96	**2.03**
	64	1.38/2.54	2.75/2.77	2.45/3.68	3.43/3.51	3.39/3.57	**3.86**
	128	3.51/5.29	5.18/5.28	4.35/6.23	6.45/6.49	5.53/**6.56**	6.30

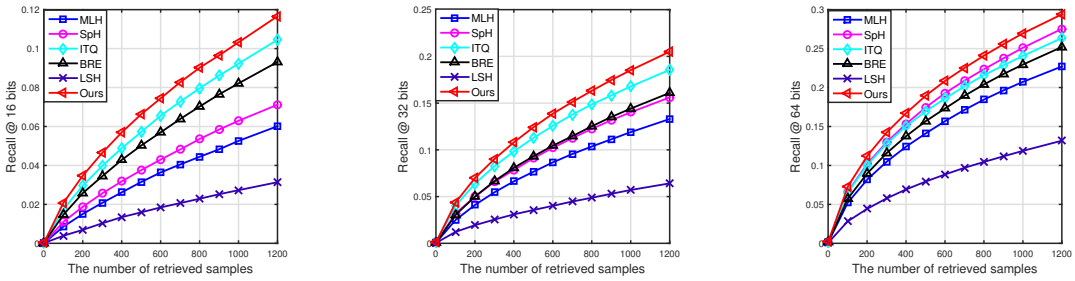

Figure 9: Recall@K comparisons between our LDTH method and the comparison methods on ANN_GIST1M. The code length of the three figures (from left to right) is 16, 32, and 64, respectively.

CPU. Table 3 shows the training and testing time of different hashing methods on ANN_GIST1M dataset with the code length equal to 32 bits. The training time of boosting method does not comprise the training time of the original hashing method. We see that training time of LDTH method is similar with the baseline hashing methods, and that of our boosting method is a little higher than our unsupervised hashing method because of the more complex network structure. The test time of our methods is comparable to those previous methods.

Table 3: Computational time of different hashing methods on ANN_GIST1M dataset. LDTH denotes the proposed linear distance-preserving unsupervised hash method. "boosting" denotes our linear distance preserving pseudo-supervised hashing method.

Method	Training(seconds)	Testing(seconds)
LSH[4]	0.002	26.584
BRE[11]	498.001	43.690
MLH[18]	255.325	30.772
SpH[7]	4.901	45.207
ITQ[5]	1.284	29.249
LDTH	56.120	26.459
boosting	106.726	28.289

5. CONCLUSIONS AND DISCUSSIONS

In this paper, we propose a general unsupervised hashing framework which simultaneously applies pairwise distance-preserving objective and point-wise constraint. Our pairwise linear distance preserving objective aims at keeping the linear projection relationships between the Euclidean distance in the original space and the Hamming distance in the binary space. This direct distance-preserving objective makes our method fully cohere with the fundamental of binary hashing. We give two instantiations of this framework. The first one is a pseudo-supervised hashing method, which aims to use the pseudo-supervised information generated by a selected hashing method to improve the performance of the selected unsupervised hashing method. It provides a general paradigm to boost existing unsupervised hashing methods. The second one is an unsupervised hashing method, which considers the quantization loss into the framework. Fully considering the pair-wise and point-wise constraints in the same framework makes our unsupervised hashing method achieve promising performance. The experiments on two large-scale datasets demonstrate that our pseudo-supervised method makes an obvious improvement for five unsupervised hashing methods, while our LDTH method outperforms the state-of-the-art unsupervised hashing methods.

In our future work, we will extend our framework with different distance-preserving constraints, such as high-order distance relationships and triplet-wise constraints. Besides, we will also explore the potential of our framework to boost

supervised hashing methods, in which deeper neural networks or convolutional neural networks may be involved.

6. ACKNOWLEDGEMENTS

This work is supported in part by the 973 Program under Contract 2015CB351803, the National Natural Science Foundation of China under Contract 61390514, 61325009, 61472378, 61429201 and 61472392, the Natural Science Foundation of Anhui Province under Contract 1508085MF109, and the Fundamental Research Funds for the Central Universities. It is also supported in part to Dr. Qi Tian by the ARO grant W911NF-15-1-0290, and Faculty Research Gift Awards by NEC Laboratories of America and Blippar.

7. REFERENCES

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

[2] L. Cao, Z. Li, Y. Mu, and S.-F. Chang. Submodular video hashing: a unified framework towards video pooling and indexing. In *ACM Multimedia*, pages 299–308, 2012.

[3] M. A. Carreira-Perpinán and R. Raziperchikolaei. Hashing with binary autoencoders. In *CVPR*, 2015.

[4] A. Gionis, P. Indyk, and R. M. others. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[5] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, pages 817–824, 2011.

[6] K. He, F. Wen, and J. Sun. K-means hashing: An affinity-preserving quantization method for learning binary compact codes. In *CVPR*, pages 2938–2945, 2013.

[7] J. Heo, Y. Lee, J. He, S. Chang, and S. Yoon. Spherical hashing. In *CVPR*, pages 2957–2964, 2012.

[8] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 33(1):117–128, 2011.

[9] T. Ji, X. Liu, C. Deng, L. Huang, and B. Lang. Query-adaptive hash code ranking for fast nearest neighbor search. In *ACM Multimedia*, pages 1005–1008, 2014.

[10] A. Joly and O. Buisson. A posteriori multi-probe locality sensitive hashing. In *ACM Multimedia*, pages 209–218, 2008.

[11] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 1042–1050, 2009.

[12] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *IEEE TPAMI*, 34(6):1092–1104, 2012.

[13] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, pages 3270–3278, 2015.

[14] R.-S. Lin, D. A. Ross, and J. Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *CVPR*, pages 848–854, 2010.

[15] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *CVPR*, 2015.

[16] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012.

[17] Z. Liu, H. Li, W. Zhou, R. Zhao, and Q. Tian. Contextual hashing for large-scale image search. *IEEE TIP*, 23(4):1606–1614, 2014.

[18] M. Norouzi and D. M. Blei. Minimal loss hashing for compact binary codes. In *ICML*, pages 353–360, 2011.

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

[20] J. Wang, S. Kumar, and S. Chang. Semi-supervised hashing for large-scale search. *IEEE TPAMI*, 34(12):2393–2406, 2012.

[21] J. Wang, J. Wang, N. Yu, and S. Li. Order preserving hashing for approximate nearest neighbor search. In *ACM Multimedia*, pages 133–142, 2013.

[22] M. Wang, W. Li, D. Liu, B. Ni, J. Shen, and S. Yan. Facilitating image search with a scalable and compact semantic mapping. *IEEE Transactions on Cybernetics*, 45(8):1561–1574, 2015.

[23] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2009.

[24] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, volume 1, page 2, 2014.

[25] L. Zhang, Y. Zhang, J. Tang, X. Gu, J. Li, and Q. Tian. Topology preserving hashing for similarity search. In *ACM Multimedia*, pages 123–132, 2013.

[26] P. Zhang, W. Zhang, W.-J. Li, and M. Guo. Supervised hashing with latent factor models. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 173–182, 2014.

[27] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE TIP*, 24(12):4766–4779, 2015.

[28] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, pages 1556–1564, 2015.

[29] W. Zhou, H. Li, R. Hong, Y. Lu, and Q. Tian. Bsift: toward data-independent codebook for large scale image search. *IEEE TIP*, 24(3):967–979, 2015.

[30] W. Zhou, Y. Lu, H. Li, Y. Song, and Q. Tian. Spatial coding for large scale partial-duplicate web image search. In *ACM Multimedia*, pages 511–520, 2010.

[31] W. Zhou, Y. Lu, H. Li, and Q. Tian. Scalar quantization for large scale image search. In *ACM Multimedia*, pages 169–178, 2012.

[32] W. Zhou, M. Yang, H. Li, X. Wang, Y. Lin, and Q. Tian. Towards codebook-free: Scalable cascaded hashing for mobile image search. *IEEE TMM*, 16(3):601–611, 2014.

[33] W. Zhou, M. Yang, X. Wang, H. Li, Y. Lin, and Q. Tian. Scalable feature matching by dual cascaded scalar quantization for image retrieval. *IEEE TPAMI*, 38(1):159–171, 2016.

[34] X. Zhu, L. Zhang, and Z. Huang. A sparse embedding and least variance encoding approach to hashing. *IEEE TIP*, 23(9):3737–3750, 2014.