Pattern Recognition 75 (2018) 175–187

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Quantization-based hashing: a general framework for scalable image and video retrieval

Jingkuan Song[a], Lianli Gao[a,*], Li Liu[b], Xiaofeng Zhu[c], Nicu Sebe[d]

[a] *Department of Computer Science, University of Electronic Science and Technology of China, 611731, China*
[b] *University of East Anglia, UK*
[c] *Guangxi Key Lab of Multi-source Information Mining and Security, Guangxi Normal University, Guangxi, 541004, China*
[d] *Department of Information Engineering and Computer Science, University of Trento, Trento, 38100, Italy*

ARTICLE INFO

Article history:
Received 24 November 2016
Revised 18 January 2017
Accepted 21 March 2017
Available online 30 March 2017

Keywords:
Hashing
Pseudo labels
Multimedia retrieval

ABSTRACT

Nowadays, due to the exponential growth of user generated images and videos, there is an increasing interest in learning-based hashing methods. In computer vision, the hash functions are learned in such a way that the hash codes can preserve essential properties of the original space (or label information). Then the Hamming distance of the hash codes can approximate the data similarity. On the other hand, vector quantization methods quantize the data into different clusters based on the criteria of minimal quantization error, and then perform the search using look-up tables. While hashing methods using Hamming distance can achieve faster search speed, their accuracy is often outperformed by quantization methods with the same code length, due to the low quantization error and more flexible distance lookups. To improve the effectiveness of the hashing methods, in this work, we propose Quantization-based Hashing (QBH), a general framework which incorporates the advantages of quantization error reduction methods into conventional property preserving hashing methods. The learned hash codes simultaneously preserve the properties in the original space and reduce the quantization error, and thus can achieve better performance. Furthermore, the hash functions and a quantizer can be jointly learned and iteratively updated in a unified framework, which can be readily used to generate hash codes or quantize new data points. Importantly, QBH is a generic framework that can be integrated to different property preserving hashing methods and quantization strategies, and we apply QBH to both unsupervised and supervised hashing models as showcases in this paper. Experimental results on three large-scale unlabeled datasets (i.e., SIFT1M, GIST1M, and SIFT1B), three labeled datastes (i.e., ESPGAME, IAPRTC and MIRFLICKR) and one video dataset (UQ_VIDEO) demonstrate the superior performance of our QBH over existing unsupervised and supervised hashing methods.

1. Introduction

Nearest neighbor (NN) search in large data sets has wide applications [1] in computer vision, information retrieval, pattern recognition, recommendation systems, etc. However, exact NN search is often intractable because of the large scale of databases and the curse of the high dimensionality. Instead, approximate nearest neighbor (ANN) search is more practical and can achieve orders of magnitude speed-ups compared to exact NN search with near-optimal accuracy [1].

While several effective data structures, such as randomized k-d forest [2], FLANN [3], and neighborhood graph search [4] were proposed, more recently, important research efforts have been devoted to learning-based hashing methods [1,5–22], due to their compact binary representation and efficient Hamming distance. Such approaches map data points to compact binary codes through a hash function, which can be generally expressed as:

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) \in \{0, 1\}^L \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^{M\times 1}$, $\mathbf{h}(.)$ is the hash function, $\mathbf{y}$ is a binary vector with code length L, and M is the dimensionality of $\mathbf{x}$. Hashing methods can be categorized as unsupervised and supervised. The unsupervised learning of the hash functions is usually based on the criterion of preserving important properties of the training data points. Typical approaches preserve the consistency property (i.e., the similarity of binary codes should be consistent with that of the original data points) [5,7,23], the similarity alignment property (i.e., the Hamming distance of the binary code should approximate the Eu-

[*] Corresponding author.
E-mail addresses: lianli.gao@gmail.com, lianli.gao@uestc.edu.cn (L. Gao).

http://dx.doi.org/10.1016/j.patcog.2017.03.021

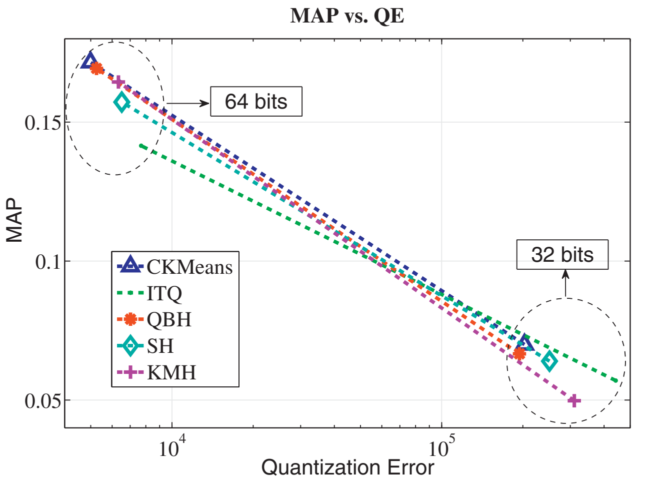


Fig. 1. Mean Average Precision (MAP) *vs.* quantization error for different methods on SIFT1M dataset.

clidean distance of the original data points) [24], the order preserving property (i.e., the order of a reference data item computed from the original space and the Hamming space should be aligned) [25], etc. A fundamental limitation of property-preserving hashing methods is that different methods try to preserve varied properties according to the applications, and thus the performance may degrade when a specifically designed hashing method is applied to another application. Supervised hashing is designed to preserve some label-based similarity [10,26,26,27]. The performance of supervised hashing methods is usually significantly superior to unsupervised methods. However, in practice the supervised information is scarce, especially for large scale datasets.

On the other hand, the quantization-based methods aim at minimizing the quantization error, and have been shown to achieve superior accuracy [9,28] over hashing methods, with sacrifice of efficiency. These methods usually cannot be decomposed into multiple subfunctions as in Eq. (1). Hashing methods use the Hamming distance to retrieve the ANNs, and quantization methods adopt the lookup tables to approximate the Euclidean distance. While hashing methods enjoy substantial speedups over quantization methods for ANN search due to the fact that the Hamming distance requires only XOR and bit-count operation, quantization methods gain high accuracy due to the low quantization error and flexible table lookups.

While hashing and quantization methods have their distinct strengths and weaknesses, it is worth exploiting their connections. If we take the data points with the same hash code as a cluster, we can calculate the quantization error of each hashing method. In Fig. 1, we illustrate the mean average precision vs. quantization error of different methods, i.e., CKMeans [9], ITQ [29], QBH, SH [5] and KMH [8] on SIFT1M dataset [6]. We can observe that lower quantization error can almost guarantee a good performance. This indicates that reducing the quantization error for conventional hashing methods might improve their performance. ITQ [29] is a pioneer in combining quantization and hashing. It firstly uses PCA-based hashing to reduce the dimension, after which, the subspace is rotated to reduce the quantization error. However, it restricts the quantization part to be hypercubic quantization [1], in which a data point is approximated to its hash code directly. Some followers of ITQ have the same limitation [30]. KMH [8] takes a step further by adopting a more general quantization strategy, in which a data point is approximated by a cluster center, and its hash code is the index of its cluster center. However, KMH does not exploit the applying of quantization models to a general property preserving hashing model.

Inspired by this, in this paper, we propose Quantization-based Hashing (QBH), a general framework which incorporates quantization error into the conventional property preserving hashing models to improve the effectiveness of the hash codes. It is worth highlighting the following contributions:

- As far as we know, we are the first to propose a general framework to incorporate the quantization-based methods into the conventional similarity-preserving hashing, in order to improve the effectiveness of hashing methods. In theory, any quantization method can be adopted to reduce the quantization error of any similarity-preserving hashing methods to improve their performance.
- This framework can be applied to both unsupervised and supervised hashing. We experimentally obtained the best performance compared to state-of-the-art supervised and unsupervised hashing methods on six popular datasets.
- We successfully show it to work on a huge dataset SIFT1B (1 billion data points) by utilizing the graph approximation and out-of-sample extension.

The remainder of this paper is organized as follows. Section 2 discusses the related work, followed by an overview of QBH in Section 3. The details of QBH for unsupervised hashing models are introduced in Section 4. QBH for supervised hashing models is given in Section 5 and we further discuss the relationships of our QBH with existing works in Section 6. Section 7 illustrates the experimental results and we draw conclusions in Section 8.

2. Related work

In this section, we discuss the related work. A summary of representative hashing and quantization algorithms with respect to similarity preserving functions, quantization error preserving, hash function, supervision and similarity in the coding space is shown in Table 1. In this table, the superscript ‘o’ indicates the original space, ‘h’ means the Hamming space, ‘q’ means the quantized code, ‘d’ is distance, ‘s’ is the similarity and ‘**c**’ means the cluster center. For example, d_{ij}^{o} means the distance between any pair of items ($\mathbf{x}_i$, $\mathbf{x}_j$), which can be calculated using Euclidean distance or other distances; s_{ij}^{o} means the similarity between any pair of items ($\mathbf{x}_i$, $\mathbf{x}_j$), which is often calculated by using d_{ij}^{o}; d_{ij}^{q} means the distance between the quantized codes of any pair of items ($\mathbf{x}_i$, $\mathbf{x}_j$).

2.1. Hashing

As mentioned in Eq. (1), learning-based hashing learns a hash function, $\mathbf{y} = \mathbf{h}(\mathbf{x}) \in \{0, 1\}^L$, mapping an input item $\mathbf{x}$ to a compact code $\mathbf{y}$. Depending on the similarity preserving criteria, we categorize the existing hashing methods into various groups:

1) Similarity-distance product minimization: requires that the distance in the coding space is smaller if the similarity in the original space is larger. Some popular methods include SH [5], MLH [25], LDA Hashing [34]. In SH, the similarity matrix is constructed as $s_{ij}^{o} = e^{-\|\mathbf{x}_i-\mathbf{x}_j\|_2^2/2\sigma^2}$, but in MLH and LDA Hashing methods, the similarity matrix is built in a supervised way, i.e., s_{ij}^{o} is 1 if $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same semantic class, 0 (or -1) otherwise.
2) Similarity-similarity product maximization: requires that the similarity in the coding space is larger if the similarity in the original space is larger. Some representative works include SSH [35], and GCC [27]. In SSH, the similarity s_{ij}^{o} in the input space

Table 1
A summary of representative hashing and quantization algorithms with respect to similarity preserving functions, quantization error reduction, hash function, supervision and similarity in the coding space. HD = Hamming Distance, LT = Lookup Table.

Approach		Similarity Preserving	Quantization Error	Hash Function	Supervision	Distance
Hashing	SH [5]	$s^o_{ij} d^h_{ij}$	-	Linear	Unsupervised	HD
	MLH [25]	$s^o_{ij} d^h_{ij}$	-	Linear	Supervised	HD
	SSH [6]	$s^o_{ij} s^h_{ij}$	-	Linear	Supervised	HD
	BRE [24]	$(d^o_{ij} - d^h_{ij})^2$	-	Kernel	Unsupervised	HD
	SHK [31]	$(s^o_{ij} - s^h_{ij})^2$	-	Linear	Supervised	HD
	OPH [32]	rank order	-	Linear	Unsupervised	HD
	TLH [33]	triplet loss	-	Linear + NN	Unsupervised	HD
	KMH [8]	$(d^q_{ij} - d^h_{ij})^2$	$(\mathbf{x} - \mathbf{c}_{i(\mathbf{x})})^2$	Linear	Unsupervised	HD
Quantization	ITQ [29]	$\mathbf{W} var(\mathbf{X}) \mathbf{W}^T$	$\mathbf{Wx} - sgn(\mathbf{Wx})$	Linear	Unsupervised	HD
	K-means	-	$\mathbf{x} - \mathbf{c}_{i(\mathbf{x})}$	Linear	Unsupervised	LT
	PQ [28]	-	$\mathbf{x} - \mathbf{c}_{i(\mathbf{x})}$	Nearest vector	Unsupervised	LT

is 1 if the pair of items $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to a same class or are nearby points, and â;;1 otherwise. The similarity in the Hamming space is defined as $s^o_{ij} = y_i^T y_j$. GCC utilizes the same way as SSH to define s^o_{ij}, and learns a hash function simultaneously using SVM.

3) Distance-distance difference minimization: requires that the difference between the distances of a pair of data points in the original space and Hamming space is as small as possible. BRE [24] belongs to this group. In BRE, the Euclidean distance is used in both the input and coding spaces. The objective function is formulated as $\min \sum_{(i,j)\in\varepsilon} \left(\frac{1}{2} \left\| x_i - x_j \right\|_2^2 - \frac{1}{M} \left\| y_i - y_j \right\|_2^2 \right)$. Here ε is a set of points to be considered, and M is a scalar.
4) Similarity-similarity difference minimization: Similarly, this type of methods want that the difference between the similarities of a pair of data points in the original space and Hamming space is as small as possible. KSH [31], FastH [10] and [36] are some representative approaches in this group. KSH aims to minimize an objective function, $\min \sum_{(i,j)\in\varepsilon} \left(s^o_{ij} - \frac{1}{M} y_i^T y_j \right)^2$, where $s^o_{ij} = 1$ if (i, j) is similar, and $s^o_{ij} = 1$ if dissimilar. In [10] and [36], the authors use the same objective function as KSH to learn the hash codes, but provide different solutions.
5) Ranking order preserving: aims to learn hash functions through aligning the orders computed from the original space and the ones in the coding space. In OPH [32], given a query, the data points are first divided into M categories, using the distance in the Hamming space and the original space. The objective function maximizes the alignment between each pair of two categories in different spaces.
6) Triplet-based similarity preserving: formulates the hashing problem by maximizing the similarity order agreement defined over triplets of items, (x, x^+, x^-), where the pair (x, x^-) is less similar than the pair (x, x^+). In [33], the triplet loss is defined as: $\ell_{triplet}(y, y^+, y^-) = \max(1 - ||y - y^-||_1 + ||y - y^+||_1, 0)$.

Some recent work also adopt deep learning to train the hashing functions [36–40], and have achieved great improvement in terms of accuracy. The criteria for learning good hash code is similar to traditional hashing methods, but the hash functions are strengthened by the deep learning techniques. There are also some others utilizing multiple tables [41] to obtain higher recall.

After hashing code learning, the hash codes are learned for the training data. To deal with the novel data point, the hashing functions have to be learned. Some methods [7,27,42] learn the hash function during the learning of hash codes. While some others [5,10,36,43] propose out-of-sample extension. Some works also utilize different retrieval schemes, e.g., asymmetric hamming distance [44].

A fundamental limitation of property-preserving hashing methods is that different methods try to preserve varied properties according to the applications, and thus the performance may degrade when a specifically designed hashing method is applied to another application. Furthermore, despite of the extraordinary improvement of the hashing methods, their accuracy is still greatly outperformed by the quantization methods with the same code length, due to the low quantization error and more flexible distance lookups.

2.2. Quantization methods

In the quantization-based encoding methods, different constraints on the codeword lead to different approaches, i.e. K-Means, Product Quantization (PQ) [28] and Cartesian K-Means (CKMeans) [9].

2.2.1. *K-Means*

Given N P-dimensional points $\mathcal{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\} \subset \mathbb{R}^P$, the K-means algorithm partitions the database into K clusters, each of which associates one codeword $\mathbf{d}_i \in \mathbb{R}^P$. Let $\mathbf{D} = [\mathbf{d}_1, \cdots, \mathbf{d}_K] \subset \mathbb{R}^P$ be the corresponding codebook. Then the codebook is learned by minimizing the within-cluster distortion, i.e.

$$\begin{aligned} \min \quad & \sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{D}\mathbf{b}_i\|_2^2 \\ \text{s.t.} \quad & \mathbf{b}_i \in \{0,1\}^K \\ & \|\mathbf{b}_i\|_1 = 1 \quad i \in \{1, \cdots, N\}. \end{aligned}$$

where $\mathbf{b}_i$ is a 1-of-K encoding vector (K dimensions with one 1 and $K-1$ 0s.) to indicate which codeword is used to quantize $\mathbf{x}_i$, and $\|\cdot\|_1$ is the l_1 norm.

The problem can be solved by iteratively alternating optimization with respect to $\mathbf{D}$ and $\{\mathbf{b}_i\}_{i=1}^N$.

2.2.2. *Product quantization*

One issue of K-Means is that the size of the codebook is quite limited due to the storage and computational cost. To address this problem, PQ [28] splits each $\mathbf{x}_i$ into M disjoint subvectors. Assume the m-th subvector contains S_m dimensions and then $\sum_{m=1}^M S_m = P$. Without loss of generality, S_m is set to $S \triangleq P/M$ and P is assumed to be divisible by M. On the m-th subvector, K-means is performed to obtain K *sub codewords*. By this method, it generates K^M clusters with only $O(KP)$ storage, while K-means requires $O(K^M P)$ storage with the same number of clusters. Meanwhile, the computing

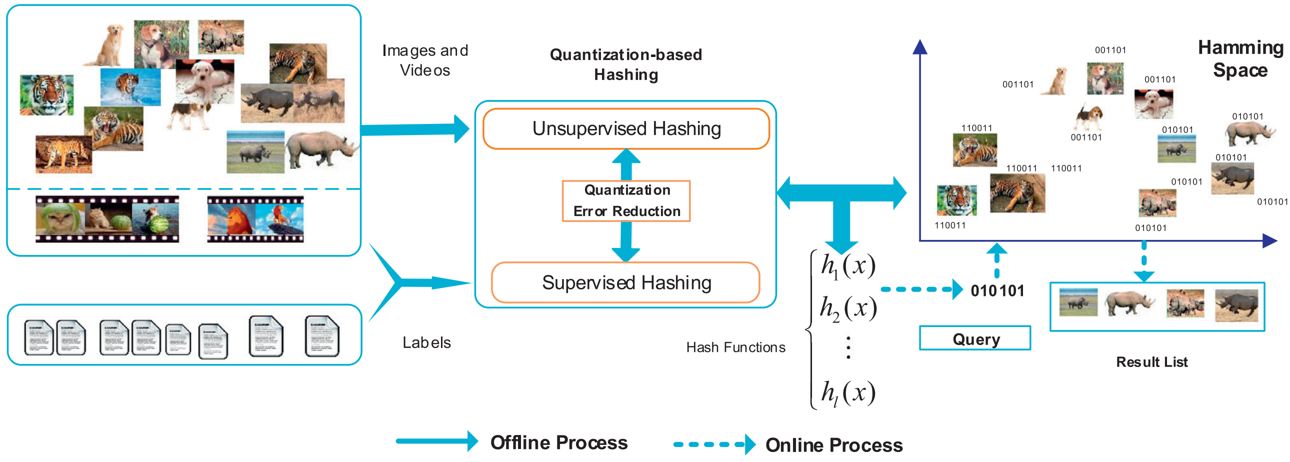


Fig. 2. Quantization-based hashing (QBH) for image and video retrieval.

complexity is reduced from $O(K^MP)$ to $O(KP)$ to encode one data point.

Let $\mathbf{D}^m \in \mathbb{R}^{S\times K}$ be the matrix of the m-th sub codebook and each column is a S-dimensional sub codeword. PQ can be taken as optimizing the following problem with respect to $\{\mathbf{D}^m\}_{m=1}^M$ and $\{\mathbf{b}_i^m\}_{i=1,m=1}^{N,M}$.

$$\begin{aligned} \min \quad & f_{\text{pq},M,K} = \sum_{i=1}^{N} \left\| \mathbf{x}_i - \begin{bmatrix} \mathbf{D}^1\mathbf{b}_i^1 \\ \vdots \\ \mathbf{D}^M\mathbf{b}_i^M \end{bmatrix} \right\|_2^2 \qquad . \\ \text{s.t.} \quad & \mathbf{b}_i^m \in \{0,1\}^K \\ & \|\mathbf{b}_i^m\|_1 = 1 \quad i \in \{1,\cdots,N\}, m \in \{1,\cdots,M\} \end{aligned} \tag{2}$$

where $\mathbf{b}_i^m$ is also the 1-of-K encoding vector on the m-th subvector and the index of 1 indicates which sub codeword is used to encode $\mathbf{x}_i$.

2.2.3. Cartesian K-Means

CKMeans [9] optimally rotates the original space and formulates the problem as

$$\begin{aligned} \min \quad & f_{\text{ck},M,K} = \sum_{i=1}^{N} \left\| \mathbf{x}_i - \mathbf{R} \begin{bmatrix} \mathbf{D}^1\mathbf{b}_i^1 \\ \vdots \\ \mathbf{D}^M\mathbf{b}_i^M \end{bmatrix} \right\|_2^2 \qquad . \\ \text{s.t.} \quad & \mathbf{R}^T\mathbf{R} = \mathbf{I} \\ & \mathbf{b}_i^m \in \{0,1\}^K \\ & \|\mathbf{b}_i^m\|_1 = 1 \quad i \in \{1,\cdots,N\}, m \in \{1,\cdots,M\} \end{aligned} \tag{3}$$

The rotation matrix $\mathbf{R}$ is optimally learned by minimizing the distortion.

If $\mathbf{R}$ is constrained to be the identity matrix $\mathbf{I}$, it will be reduced to Eq. (2). Thus, we can assert that under the optimal solutions, we have $f^*_{\text{ck},M,K} \le f^*_{\text{pq},M,K}$, where the asterisk superscript indicates the objective function with the optimal parameters. The quantization error can be further reduced by adopting an elaborate design of encoding scheme, e.g., Optimized Cartesian K-means [45], Tree Quantization [46] and Composite Quantization [47,48,48].

While quantization methods gain high accuracy due to the low quantization error and flexible table lookups, their ANN search speed is outperformed by hashing methods due to the fact that the Hamming distance requires only XOR and bit-count operation. Therefore, it is necessary to design an algorithm that can take the benefits of both, i.e., lower quantization error and fast search speed. Motivated by this, we propose quantization-based hashing.

3. Quantization-based hashing

In this work, we propose a general framework of image and video retrieval (depicted in Fig. 2). This framework consists of two phases. In the first phase which is offline, we use the proposed QBH algorithm to learn the hash codes of the training data. To deal with novel images/videos, we utilize out-of-sample-extension to obtain the hash functions. Using the derived hash functions, each test data point can be represented by the generated hash codes in the Hamming space. In the second phase which is online, the query image/video is also represented by hash codes generated from the hash functions. Retrieval can be efficiently achieved where only efficient XOR operation on the hash codes is performed to compute the similarity between two data points.

The key research issue is how to train the hash functions which affect both accuracy and efficiency. We propose quantization-based hashing, a general framework which can be integrated to both supervised and unsupervised hashing models. The motivation for QBH is that, by taking each bucket with the same hash codes as a cluster, hashing methods can be considered as a kind of quantization method. Inspired by Fig. (1) and ITQ, by reducing the quantization error, the performance of the hash codes can potentially be improved. In this way, the determination of the binary code of a data point should not only be affected by the similarity-preserving constraint, but also by the quantization error. By putting the similarity preserving error E_{simp} and the quantization error E_{quan} together, we arrive at the following objective function for quantization-based hashing:

$$E = E_{simp} + E_{quan}, \tag{4}$$

We first introduce our QBH and show how QBH is applied to unsupervised hashing models in Section 4. Then we introduce how to integrate QBH to supervised models in Section 5. In Table 2 we introduce the notations which will be used in this paper.

4. QBH For unsupervised hashing models

In this section, we introduce our unsupervised QBH. We first propose a simple solution in Section 4.1 which will be further improved in Section 4.2.

4.1. A Basic QBH

Suppose there are two data points $\mathbf{x}_i$ and $\mathbf{x}_j$. Their hash codes are $\mathbf{y}_i$ and $\mathbf{y}_j$, respectively. For property preserving hashing methods, some important properties of the data in the original space

Table 2
Notations[1]

Notation	Description
N, M	the number of data points and dimensions
$\mathbf{X}$	the $M \times N$ matrix for the dataset
$\mathbf{Y}$	the $L \times N$ matrix for the hash codes of $\mathbf{X}$
$\mathbf{x}_i$, $\mathbf{y}_i$	a data point and its binary hash code
$\mathbf{C}$	the $M \times K$ matrix for cluster centers
$\mathbf{A}$	the affinity matrix
L, K	length of hash code and number of clusters
ϵ	the number of nearest neighbors
α	the regularization parameters

[1] Generally, we use the uppercase unbolded symbol as a constant, the lowercase unbolded as the index, the uppercase bold as the matrix and the lowercase bold as the vector.

should be preserved (as shown in Table 1), e.g, the similarity-distance product minimization property [5,25], the similarity-similarity product maximization property [27,35], the order property [32] and the triangle order property [25]. Based on these properties to be preserved, different objective functions are given. Without loss of generality, we first use the similarity-distance product minimization property in this work as a showcase, and then we integrate our QBH to other property preserving hashing in Section 5.

To exploit the similarity property, we first construct an affinity matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, with each entry $\mathbf{a}_{ij}$ representing the similarity between the data points $\mathbf{x}_i$ and $\mathbf{x}_j$, which is usually defined as follows:

$$\mathbf{a}_{ij} = \begin{cases} e^{-\|\mathbf{x}_i-\mathbf{x}_j\|_2^2/2\sigma^2}, \text{ if } \mathbf{x}_i \in \mathcal{N}_K(\mathbf{x}_j) \text{ or } \mathbf{x}_j \in \mathcal{N}_K(\mathbf{x}_i) \\ 0, \text{otherwise.} \end{cases} \tag{5}$$

where $1 \le (i, j) \le n$, $\mathcal{N}_K(\cdot)$ is the K-nearest-neighbor set, and σ is a parameter. Then the similarity preserving error can be presented as:

$$E_{simp} = \sum_{ij} \mathbf{a}_{ij} \left\|\mathbf{y}_i - \mathbf{y}_j\right\|_H, \; s.t. \; \mathbf{Y}^T\mathbf{Y} = \mathbf{I}, \; \mathbf{y}_i \in \{0, 1\}^L. \tag{6}$$

where $\mathbf{y}_i$ is the hash code for the data point $\mathbf{x}_i$, and L is the hash code length. The $\mathbf{Y}$ for property preserving hashing can be obtained by minimizing (6).

By taking each bucket with the same hash codes as a cluster, hashing methods can be considered as a kind of quantization method. Suppose that all the data point with hash code y_i form the cluster with the index k_i. Then this cluster center $\mathbf{c}_{k_i}$ can be represented as:

$$\mathbf{c}_{k_i} = \frac{1}{size(\prod(k_i))} \sum_{\mathbf{x}_i \in \prod(k_i)} \mathbf{x}_i \tag{7}$$

where $\prod(k_i)$ indicates the set of data points in cluster k_i. The connection between $\mathbf{y}_i$ and k_i is:

$$\begin{aligned} \mathbf{y}_i &= b(k_i) \\ k_i &= r(\mathbf{y}_i). \end{aligned} \tag{8}$$

where $r(.)$ converts a binary code y_i into an integer k_i and $b(.)$ converts an integer k_i to a binary code $\mathbf{y}_i$. Then the quantization error of x_i is:

$$E_{quan} = \left\|\mathbf{x}_i - \mathbf{c}_{k_i}\right\|_2^2. \tag{9}$$

We want to generate hash codes to preserve the property in (6) and also use the supervision information in (9). However, these requirements may be conflicting for some data points. For example, a hash code $\mathbf{y}_i$ for a data point $\mathbf{x}_i$ might result in a minimal property preserving error in (6) but a larger quantization error in (9), and vice versa. In this way, the determination of the binary code of a data point should not only be affected by the property-preserving constraint, but also by the quantization error. By putting them together, we arrive at the following objective function for QBH:

$$\begin{aligned} &\min_{\mathbf{C},\mathbf{Y}} \sum_{ij} \mathbf{a}_{ij} \left\|\mathbf{y}_i - \mathbf{y}_j\right\|_H + \alpha \sum_i \left\|\mathbf{x}_i - \mathbf{c}_{r(\mathbf{y}_i)}\right\|_2^2 \\ &s.t. \;\; \mathbf{y}_i \in \{0, 1\}^L. \end{aligned} \tag{10}$$

where $\mathbf{C} \in \mathbb{R}^{M\times K}$ are the learned cluster centers, M is the dimensionality of the data point, $K = 2^L$ is the number of cluster centers, $\mathbf{c}_{r(y_i)}$ is the $r(y_i)$-th column of $\mathbf{C}$, and α is a parameter to balance the two components.

4.1.1. Solution

The solution is illustrated in Algorithm (1). We first initialize $\mathbf{C}$ with K random selected data points, and then update $\mathbf{Y}$ and $\mathbf{C}$ iteratively until convergence.

Given $\mathbf{C}$, the update of each $\mathbf{y}_i$ not only depends on the quantization error (i.e., the distance of $\mathbf{x}_i$ to each cluster centers), but also on the property preserving error (i.e., the weighted distance of $\mathbf{y}_i$ to the other $\mathbf{y}_j$). The determination of $\mathbf{y}_i$ is a balance between these two errors, and it can be solved as:

$$\mathbf{y}_i^* = \arg\min_{\mathbf{y}_i} \sum_j \mathbf{a}_{ij} \left\|\mathbf{y}_i - \mathbf{y}_j\right\|_H + \alpha \left\|\mathbf{x}_i - \mathbf{c}_{r(\mathbf{y}_i)}\right\|_2^2. \tag{11}$$

On the other hand, the update of $\mathbf{C}$ given $\mathbf{Y}$ is the same as in the classical K-means, and each $\mathbf{c}$ is the mean of the $\mathbf{x}_i$ with the same $\mathbf{y}_i$, as in (7).

One limitation of this method is that when N is large, a relative long code length (e.g., $L = 32$ or 64) may be required to gain a satisfactory performance. In this case, at least $(2^{L-1} + 1)$ cluster centers are required. Then, Eq. (10) will be intractable either because of the huge memory cost for storing the cluster centers, or due to the high time cost for computing each $\mathbf{y}_i^*$.

4.2. A Generalized QBH

In the previous section, we proposed a basic method to integrate the advantages of quantization-based methods into hashing methods, yet the solution is both time- and memory-consuming. In this section, we generalize the basic method by modifying the construction of cluster centers in (10) and propose a solution that is fast and memory efficient.

Instead of restricting taking each column of $\mathbf{C}$ as the mean of data points in each bucket as in (7), we put no constraint on $\mathbf{C}$ and we take different combinations of $\mathbf{C}$'s columns as cluster centers. Then we have the following objective function:

$$\begin{aligned} &\min_{\mathbf{C},\mathbf{Y}} \sum_{ij} 2\mathbf{a}_{ij} \left\|\mathbf{y}_i - \mathbf{y}_j\right\|_H + \alpha \sum_i \|\mathbf{x}_i - \mathbf{C}\mathbf{y}_i\|_2^2 \\ &s.t. \;\; \mathbf{y}_i \in \{-1, 1\}^L. \end{aligned} \tag{12}$$

In this way, only L cluster centers are needed to generate 2^L binary codes. Without loss of generality, we assume the bits belong

Algorithm 1 Solution for the basic quantization-based hashing.

```
Input: Initialized $\mathbf{C}$, $\alpha$;
Output: $\mathbf{Y}$, $\mathbf{C}$;
repeat
    Fix $\mathbf{C}$, update $\mathbf{Y}$ according to Eqn. (11);
    Fix $\mathbf{Y}$, update $\mathbf{C}$ according to classical K-means, each $\mathbf{c}_k$ is the
    mean of the $\mathbf{x}_i$ with the same $\mathbf{y}_i$;
until convergence or max iteration is reached.
return $\mathbf{Y}$, $\mathbf{C}$;
```

to $\{-1, 1\}$, i.e., $\mathbf{y}_i \in \mathcal{H}_L \equiv \{-1, 1\}^L$. Then $2\left\|\mathbf{y}_i, \mathbf{y}_j\right\|_H = L - \mathbf{y}_i^T\mathbf{y}_j$. The objective function is equivalent to:

$$\min_{\mathbf{C},\mathbf{Y}\in\{-1,1\}^{L\times N}} \alpha\|\mathbf{X}-\mathbf{CY}\|_F^2 - \sum_{ij}\mathbf{a}_{ij}\mathbf{y}_i^T\mathbf{y}_j. \tag{13}$$

This problem is still intractable, though the size of **C** is largely reduced from $M \times 2^L$ to $M \times L$, because the discrete optimization is not submodular [9]. Given **C** and a small value of L, one can generate all possible $\mathbf{y}_i$ to find the optimal solution, but this would be impractical for large values of L.

One observation is that if **C** has orthogonal columns, the optimization becomes tractable. In this case,

$$\begin{aligned}&\|\mathbf{X}-\mathbf{CY}\|_F^2\\ &= \left(\|\mathbf{X}\|_F^2 + \|\mathbf{CY}\|_F^2 - 2tr\left(\mathbf{X}^T\mathbf{CY}\right)\right)\\ &= \left(\|\mathbf{X}\|_F^2 + tr\left(\mathbf{Y}^T\mathbf{C}^T\mathbf{CY}\right) - 2tr\left(\mathbf{X}^T\mathbf{CY}\right)\right).\end{aligned} \tag{14}$$

Denote $\mathbf{V} = \mathbf{C}^T\mathbf{C} \in \mathbb{R}^{L\times L}, \mathbf{E} = \mathbf{X}^T\mathbf{C} \in \mathbb{R}^{N\times L}$, and then it becomes:

$$\|\mathbf{X}-\mathbf{CY}\|_F^2 = \|\mathbf{X}\|_F^2 + L\sum_i \mathbf{v}_{ii} - 2\sum_i {\mathbf{y}_i}^T\mathbf{e}_i. \tag{15}$$

For fixed **X** and **C**, optimizing **Y** in (13) is equivalent to:

$$\min_{\mathbf{Y}\in\{-1,1\}^{L\times N}} -\sum_i {\mathbf{y}_i}^T(\alpha\mathbf{e}_i + \sum_j \mathbf{a}_{ij}\mathbf{y}_j). \tag{16}$$

The sign of each $\mathbf{y}_i$ should be consistent with the sign of $\alpha\mathbf{e}_i + \sum_j \mathbf{a}_{ij}\mathbf{y}_j$ in order to get the minimal value.

Since **C** has orthogonal columns, it can therefore be expressed in terms of rotation and scaling; i.e., $\mathbf{C} \equiv \mathbf{RD}$, where $\mathbf{R} \in \mathbb{R}^{M\times L}$ has orthogonal columns ($\mathbf{R}^T\mathbf{R} = \mathbf{I}_L$), and $\mathbf{D} \in \mathbb{R}^{L\times L}$ is diagonal and positive definite. We further introduce an offset, denoted $\mu \in \mathbb{R}^{M\times 1}$, to align **X**. **1** is a vector with all 1s. Taken together with (13), we have the objective function:

$$\begin{aligned}&\min_{\mathbf{R},\mathbf{D},\mathbf{Y},\mu} \alpha\|\mathbf{X}-\mu\mathbf{1}-\mathbf{RDY}\|_F^2 - \sum_{ij}\mathbf{a}_{ij}\mathbf{y}_i^T\mathbf{y}_j.\\ &s.t. \begin{cases}\mathbf{y}_i \in \{-1, 1\}^L\\ \mathbf{R}^T\mathbf{R} = \mathbf{I}_L\end{cases}\end{aligned} \tag{17}$$

Compared with the objective function (10), the key difference lies in that each data point $\mathbf{x}_i$ is encoded by $\mathbf{Cy}_i$ in (17) instead of by $\mathbf{c}_{r(\mathbf{y}_i)}$ in (10). That is to say, different combinations of **C**'s columns, rather than each column of **C**, are selected to form a cluster center, to quantize each $\mathbf{x}_i$.

4.2.1. Solution

We utilize coordinate descent to optimize (17). We firstly initialize $\mu = mean(\mathbf{X})$ and **Y** with spectral hashing, and then optimize **R, D, Y** and μ iteratively until convergence.

Update **R**: The objective function corresponds to the classic orthogonal procruste problem, in which one tries to find a rotation to align one point set with another. In particular, by adding $(M - L)$ rows of zeros to the bottom of **D, DY** becomes $M \times N$. Then **R** is square and orthogonal and can be estimated with SVD. In our case, the two point sets are given by the $\mathbf{X}_\mu$ and the target binary code **Y**. For a fixed **Y**, (17) is minimized as follows: first compute the SVD of the $M \times M$ matrix $\mathbf{DYX}_\mu^T$. $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = SVD(\mathbf{X}_\mu(\mathbf{DY})^T, 0)$ and then

$$\mathbf{R}^* = \mathbf{UV}^T. \tag{18}$$

where $\mathbf{X}_\mu = \mathbf{X} - \mu\mathbf{1}^T$.

Update **D**: Given **X, R, Y**, μ, the objective (17) is equivalent to:

$$\begin{aligned}&\min_{\mathbf{R}^T\mathbf{R}=\mathbf{I}_L,\mathbf{Y}\in\{-1,1\}^{L\times N}} \left\|\mathbf{R}^T\mathbf{X}_\mu - \mathbf{DY}\right\|_F^2\\ &\Rightarrow \min_{\mathbf{R}^T\mathbf{R}=\mathbf{I}_L,\mathbf{Y}\in\{-1,1\}^{L\times N}} \sum_{li}(\mathbf{b}_{li} - \mathbf{d}_l\mathbf{y}_i)^2.\end{aligned} \tag{19}$$

where $\mathbf{B} = \mathbf{R}^T\mathbf{X}_\mu \in \mathbb{R}^{L\times N}$ and $\mathbf{b}_{li}$ is the l-th row i-th column entry of **B**. Since **D** is diagonal and positive definite, $\mathbf{d}_l\mathbf{y}_i = \mathbf{d}_{ll}\mathbf{y}_{li}$. Then we have:

$$\mathbf{d}_{ll}^* = \frac{1}{N}\sum_i \mathbf{b}_{li}\mathbf{y}_{li} = \frac{1}{N}\mathbf{b}_l\mathbf{y}_l^T. \tag{20}$$

Update **Y**: Since **C** is fixed, optimizing **Y** in (13) can be reformulated as (16). Then the optimal y_i is be determined by the sign of $\alpha\mathbf{e}_i + \sum_j \mathbf{a}_{ij}\mathbf{y}_j$:

$$\mathbf{y}_i^* = \operatorname{sgn}(\alpha\mathbf{e}_i + \mathbf{a}_i\mathbf{Y}^T). \tag{21}$$

where $\mathbf{a}_i$ is the i-th row of **A**.

Update μ: Given **R, B**, and **D**, the optimal μ is given by the column average of $\mathbf{X} - \mathbf{RDY}$:

$$\mu^* \leftarrow \underset{column}{mean}(\mathbf{X} - \mathbf{RDY}). \tag{22}$$

The solution is illustrated in Algorithm (2).

Algorithm 2 Solution for the generalized quantization-based hashing.

```
Input: Initialized $\mu$, $\mathbf{Y}$;
Output: $\mathbf{Y}$, $\mathbf{R}$, $\mathbf{D}$, $\mu$;
repeat
    Update $\mathbf{R}$ according to Eqn. (18);
    Update $\mathbf{D}$ according to Eqn. (20);
    Update $\mathbf{Y}$ according to Eqn. (21);
    Update $\mu$ according to Eqn. (22);
until convergence or max iteration is reached.
return $\mathbf{Y}$, $\mathbf{R}$, $\mathbf{D}$, $\mu$;
```

Although conceptually simple, the main bottleneck in the above formulation is computation. The cost of building the underlying graph in (17) and (21) is $O(LN^2)$, which is intractable for large N [49]. The time complexity for Eqs. (18), (20), (21) and (22) are $O(L^2M + NML + 2 \times M^3)$, $O(NL)$, $O(N^2L + NML)$ and $O(ML^2 + NML)$. Usually, the M and L are relatively small, from tens to hundreds, i.e., $L \ll N$ and $M \ll N$. Therefore, the most time-consuming step is Eq. (21), which has the time complexity of $O(N^2L)$. To avoid the computational bottleneck, we propose to use anchor points to approximate the dataset. The basic idea is to directly replace the $N \times N$ affinity graph with a $N \times Z$ anchor graph, as described next.

4.2.2. Graph approximation and out-of-sample extension

An anchor graph uses a small set of Z points called anchors to approximate the data neighborhood structure [50]. Similarities of all N database points are measured with respect to these Z anchors. First, K-means clustering is performed on N data points to obtain $Z(Z \ll N)$ cluster centers $\mathbf{U} = \mathbf{u}_z,\ z = 1, \ldots, Z$ that act as anchor points.

Then, instead of building a $N \times N$ affinity matrix as in (5), we only calculate the anchor graph **A** representing the similarities between the data points **X** and the anchors **U**. In this way, the update of **Y** can be more efficient using:

$$\mathbf{y}_i = \operatorname{sgn}(\alpha\mathbf{e}_i + \mathbf{a}_i\mathbf{Y}_{anchor}^T). \tag{23}$$

where $\mathbf{a}_i$ is the i-th row of the anchor graph **A**, and $\mathbf{Y}_{anchor}$ are the hash codes for the anchor points.

Out-of-sample refers to learning an hash function that is able to encode new data points. From Algorithm 2, we can obtain **R, D**, μ, **Y** and also the hash codes $\mathbf{Y}_{anchor}$ for the anchor points. For a new data point $\mathbf{x}_i$, its hash code $\mathbf{y}_i$ can be obtained as:

$$\mathbf{y}_i = \operatorname{sgn}(\alpha\mathbf{x}_i\mathbf{RD} + \mathbf{a}_i\mathbf{Y}^T). \tag{24}$$

By using anchor points, it can be replaced by:

$$\mathbf{y}_i = \operatorname{sgn}(\alpha\mathbf{x}_i\mathbf{RD} + \mathbf{a}_i\mathbf{Y}_{anchor}^T). \tag{25}$$

where $\mathbf{w}_i$ and $\mathbf{a}_i$ are the i-th row of affinity graph $\mathbf{A}$ and anchor graph $\mathbf{A}$, respectively. A small number of the anchor points can achieve satisfactory performance, as pointed out in [50]. Thus, the encoding of new data points and the update of $\mathbf{Y}$ in the training process are very efficient due to the use of anchor points.

5. QBH For supervised hashing models

In this section, we show that QBH can also be integrated into supervised hashing models. We firstly introduce supervised QBH, and then we further show that our QBH can be used for supervised hashing models when no label information is given.

As mentioned in Table 1, there are also several similarities to be preserved in different hashing methods. For simplicity, we use distance-similarity product minimization as an example again. Compared with unsupervised distance-similarity product minimization hashing methods, the difference of supervised methods lies in the construction of the similarity matrix s_{ij}. Similar to MLH and LDA, when the similarity matrix is built in a supervised way, i.e., s^o_{ij} is 1 if $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same semantic class and 0 (or 1) otherwise, the hashing model is supervised. In this way, we can formulated supervised QBH the same way as (12). Next we introduce how to use QBH for supervised hashing models while no label information is given.

Existing works have demonstrated that supervised hashing methods gain superior performance over unsupervised ways. However, the supervised information is not always available, especially for some large scale datasets. To take advantage of the supervised hashing models, our proposed QBH uses the cluster centers of the training data to generate pseudo labels, based on which the hash codes can be generated in a supervised way. Without loss of generality, we utilize LDA with trace ratio criterion (similar to LDAH [34]) as a showcase for hash functions learning and demonstrate how it is incorporated into our framework. Note that some other criterion like label-similarity preserving [6,31] can also be applied to our framework.

The loss function for LDA trace ratio based hashing method is:

$$\ell(\mathbf{Y}) = \frac{tr(\mathbf{S}_b)}{tr(\mathbf{S}_w)}, \quad s.t.\ \mathbf{Y} \in \{0,1\}^{N\times L}. \tag{26}$$

where $\mathbf{S}_w$ represents within-class scatter matrix, and $\mathbf{S}_b$ is the between-class scatter matrix. They are defined as:

$$\begin{aligned} \mathbf{S}_w &= \sum_{k=1}^{C} \sum_{\mathbf{x}_i \in \Omega_k} (\mathbf{y}_i - \mathbf{c}_k)(\mathbf{y}_i - \mathbf{c}_k)^T \\ \mathbf{S}_b &= \sum_{k=1}^{C} n_k (\mathbf{c}_k - \bar{\mathbf{y}})(\mathbf{c}_k - \bar{\mathbf{y}})^T. \end{aligned} \tag{27}$$

where $\mathbf{y}_i$ is the corresponding hash code for a date point $\mathbf{x}_i$, Ω_k indicates the data points in the k-th class and $\mathbf{c}_k$ is the mean of hash codes in the k-th class. $\bar{\mathbf{y}}$ is the mean of all hash codes $\mathbf{Y}$. We also define the total scatter matrix $\mathbf{S}_t$ as:

$$\mathbf{S}_t = \mathbf{S}_b + \mathbf{S}_w = \sum_{i=1}^{N} (\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^T. \tag{28}$$

Suppose the data have been centralized, i.e., $\bar{x} = 0$. We denote the pseudo labels as F, and we define a cluster centroid matrix C to include the centroid vector of the hash codes in each class as $C = [c_1, \ldots, c_K]$. We use a linear hash function, i.e., $y_i = sgn(W^T x_i)$. This objective function is intractable and we follow [5,31] to apply the spectral relaxation trick to drop the sign functions. Then $\bar{y} = 0$. Thus S_b, S_w and S_t can be rewritten as:

$$\begin{aligned} &\mathbf{S}_w = \left(\mathbf{W}^T\mathbf{X} - \mathbf{C}\mathbf{F}^T\right)\left(\mathbf{W}^T\mathbf{X} - \mathbf{C}\mathbf{F}^T\right)^T \\ &\mathbf{S}_b = \mathbf{C}\mathbf{F}^T\mathbf{F}\mathbf{C}^T,\ \ \mathbf{S}_t = \mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}. \end{aligned} \tag{29}$$

Then the objective function becomes:

$$\ell\left(\mathbf{W}, \mathbf{W}^T\mathbf{W} = \mathbf{I}\right) = \frac{tr\left(\mathbf{C}\mathbf{F}^T\mathbf{F}\mathbf{C}^T\right)}{tr\left(\left(\mathbf{W}^T\mathbf{X} - \mathbf{C}\mathbf{F}^T\right)\left(\mathbf{W}^T\mathbf{X} - \mathbf{C}\mathbf{F}^T\right)^T\right)}. \tag{30}$$

Because $\mathbf{S}_t = \mathbf{S}_w + \mathbf{S}_b$, optimizing (26) is equivalent to optimize $\frac{tr(\mathbf{S}_b)}{tr(\mathbf{S}_t)}$. The problem is that we do not know the class labels F. Then, the final objective function for LDA trace ratio hashing with pseudo labels becomes:

$$\begin{aligned} &\ell(\mathbf{W}, \mathbf{F}, \mathbf{C}) = \frac{tr\left(\mathbf{C}\mathbf{F}^T\mathbf{F}\mathbf{C}^T\right)}{tr\left(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}\right)}. \\ &s.t. \begin{cases} \mathbf{F} \in \{0,1\}^{N\times K} \\ \|\mathbf{f}_i\|_1 = 1 \\ \mathbf{W}^T\mathbf{W} = \mathbf{I} \end{cases} \end{aligned} \tag{31}$$

where $\|\cdot\|_1$ is the l_1 norm. The second constraint $\|\mathbf{f}_i\|_1 = 1$ requires that each $\mathbf{x}_i$ belongs to a single class.

5.1. Solution

There are three unknown variables in (31), namely **W, C** and **F**. Using the class indicator matrix **F**, we can represent each cluster centroid $\mathbf{c}_k$ as:

$$\mathbf{c}_k = \frac{1}{size(\prod(k))} \sum_{\mathbf{y}_i \in \prod(k)} \mathbf{y}_i. \tag{32}$$

where $\prod(k)$ indicates the set of data points in class k. Then, **C** can be reformulated in a matrix form:

$$\mathbf{C} = \mathbf{W}^T\mathbf{X}\mathbf{F}\left(\mathbf{F}^T\mathbf{F}\right)^{-1}. \tag{33}$$

Then, (31) becomes:

$$\begin{aligned} &\ell(\mathbf{W}, \mathbf{F}) = \frac{tr\left(\mathbf{W}^T\mathbf{X}\mathbf{F}\left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{X}^T\mathbf{W}\right)}{tr\left(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}\right)}. \\ &s.t. \begin{cases} \mathbf{F} \in \{0,1\}^{N\times K} \\ \|\mathbf{f}_i\|_1 = 1 \\ \mathbf{W}^T\mathbf{W} = I \end{cases} \end{aligned} \tag{34}$$

We utilize coordinate descent to optimize (34). We firstly fix **F** and update **W**, and then fix update **W** by fixing **F**. They are updated iteratively until convergence. The solution is illustrated in Algorithm (3).

Algorithm 3 Solution for the LDA trace ratio hashing with pseudo labels.

```
Input: Initialized $\mathbf{F}$;
Output: $\mathbf{F}$, $\mathbf{W}$;
repeat
    Fix $\mathbf{F}$, update $\mathbf{W}$ according to Eqn. (36);
    repeat
        Fix $\mathbf{W}$ and $\mathbf{F}$, update $\mathbf{C}$ according to classical K-means, each $\mathbf{c}_k$ is the mean of the $\mathbf{y}_i$ within the same class;
        Fix $\mathbf{W}$ and $\mathbf{C}$, update $\mathbf{F}$;
    until convergence or max iteration is reached.
until convergence or max iteration is reached.
return $\mathbf{F}$, $\mathbf{W}$;
```

Update **W**: Given **F**, obviously solving problem (34) is to minimize the trace ratio LDA w.r.t. **W**:

$$\ell\left(\mathbf{W}, \mathbf{W}^T\mathbf{W} = \mathbf{I}\right) = \frac{tr\left(\mathbf{W}^T\mathbf{X}\mathbf{F}\left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{X}^T\mathbf{W}\right)}{tr\left(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}\right)}. \tag{35}$$

which can be directly solved with the generalized eigenvalue decomposition (GEVD) method:

$$XF\left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{X}^T\mathbf{w}_l = \lambda_l \mathbf{X}\mathbf{X}^T\mathbf{w}_l. \tag{36}$$

where λ_l is the l-th largest eigenvalue of the GEVD with the corresponding eigenvector $\mathbf{w}_l$, and $\mathbf{w}_l$ constitutes the l-th column vector of the matrix $\mathbf{W}$.

When $\mathbf{W}$ is fixed, $tr(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W})$ is irrelevant to $\mathbf{F}$. Thus, we need to maximize the following problem *w.r.t* $\mathbf{F}$:

$$\ell(\mathbf{F}) = tr\Big(\mathbf{W}^T\mathbf{X}\mathbf{F}\big(\mathbf{F}^T\mathbf{F}\big)^{-1}\mathbf{F}^T\mathbf{X}^T\mathbf{W}\Big). \\ s.t.\begin{cases}\mathbf{F}\in\{0,1\}^{N\times K}\\ \|\mathbf{f}_i\|_1 = 1\end{cases} \tag{37}$$

This is still difficult to solve due to the intractable constraint. Because $Tr(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W})$ is a constant now ($\mathbf{W}$ is fixed), maximizing between-class distance in problem (37) is equivalent to minimizing within-class distance. Problem (37) is equivalent to the following problem:

$$\ell(F) = \left\|\mathbf{W}^T\mathbf{X} - \mathbf{C}\mathbf{F}^T\right\|_F^2, s.t.\begin{cases}\mathbf{F}\in\{0,1\}^{N\times K}\\ \|\mathbf{f}_i\|_1 = 1\end{cases} \tag{38}$$

Problem (38) can be easily solved by alternating optimization, i.e., iteratively optimizing $\mathbf{C}$ when $\mathbf{F}$ is fixed and optimizing $\mathbf{F}$ when $\mathbf{C}$ is fixed.

Update $\mathbf{C}$: Each cluster center $\mathbf{c}_k$ is the mean of all data points in the class k. $\mathbf{C}$ can be updated using (32).

Update $\mathbf{F}$: Since each data point belongs to one class, $\mathbf{x}_i$ is assigned to its closest cluster center $\mathbf{c}_k$. Therefore, $\mathbf{f}_i$ is a column vector with its k-th element being 1 and others being 0.

After we get $\mathbf{W}$, the hash codes for $\mathbf{x}_i$ can be generated by $sgn(\mathbf{W}^T\mathbf{x}_i)$.

6. Relations to related work

QBH is a general framework to combine similarity preserving and quantization error reduction. There are close relationships between QBH and ITQ [29] and KMH [8]. It is beneficial to investigate these relationships.

ITQ vs. QBH.

ITQ consists of two steps, i.e., similarity preserving step and quantization error reducing step. To generate L-bit hash codes, a covariance preserving method (i.e., PCA) is firstly applied on the zero-centered data $\mathbf{X}'$ to reduce the dimensionality from M to L, after which, the subspace representation is rotated by a $L \times L$ rotation matrix $\mathbf{R}$ to reduce the quantization error. The composition of PCA can be expressed as $\mathbf{W}^T\mathbf{X}'$ and $\mathbf{R}$ can be obtained by minimizing:

$$\ell_{ITQ}(\mathbf{Y},\mathbf{R}) = \left\|\mathbf{R}^T\mathbf{W}^T\mathbf{X}' - \mathbf{Y}'\right\|_F^2, s.t. \;\; \mathbf{R}^T\mathbf{R} = \mathbf{I}_{L\times L} \;\; .$$

One clear difference is that QBH combines the similarity preserving error and the quantization error into a general framework, rather than using a two step procedure as in ITQ. In this way, these two errors can be updated iteratively until convergence. Another important fact is that any property preserving error can be integrated into QBH in theory, while ITQ has the limitation that an explicit embedding function has to be learned in the first step in order to be processed by the second step. Also, ITQ restricts the quantization part to be hypercubic quantization [1], in which a data point is approximated to its hash code directly. Our QBH can incorporate more general quantization methods, in which a data point is approximated by a cluster center, and its hash code is either its cluster center or the index of its cluster center.

KMH vs. QBH.

KMH is intrinsically a quantization method. Unlike conventional quantization methods which generate cluster centers and perform the approximate search using lookup tables, KMH utilizes the Hamming distance of the cluster indices to approximate the lookup tables. The objective function for KMH is represented as:

$$\min_{\mathbf{C},k_i} \sum_{ij} (d_{ij}^q - d_{ij}^h)^2 + \alpha \sum_i \left\|\mathbf{x}_i - \mathbf{c}_{k_i}\right\|_2^2. \tag{39}$$

where k_i is the cluster index of $\mathbf{x}_i$, $d_{ij}^q = \|\mathbf{c}_{k_i} - \mathbf{c}_{k_j}\|_2^2$ is the distance of the cluster centers of $\mathbf{x}_i$ and $\mathbf{x}_j$, and d_{ij}^h is the Hamming distance of the index (k_i and k_j) of cluster centers of $\mathbf{x}_i$ and $\mathbf{x}_j$. It is clear that the first term is distance-distance difference minimization property, and the second term is quantization error. By replacing our similarity-distance product minimization property with this distance-distance difference minimization property, we can arrive at a new objective function for modified QBH. K-means hashing is a special case of our modified QBH.

One key difference is that KMH only considers how to speed up the quantization methods with hashing techniques, but ignores how to improve the general similarity-preserving hashing methods by reducing the quantization error using quantization methods. While our QBH is a general framework, which incorporates different quantization error reduction strategies into the conventional different similarity preserving hashing models to improve the performance of hash codes.

7. Experiments

We evaluate our algorithm on the task of high-dimensional approximate nearest neighbor (ANN) search. Firstly, we study the influence of the parameters in our algorithm. Then, we compare our results with state-of-the-art algorithms on three standard datasets.

7.1. Settings

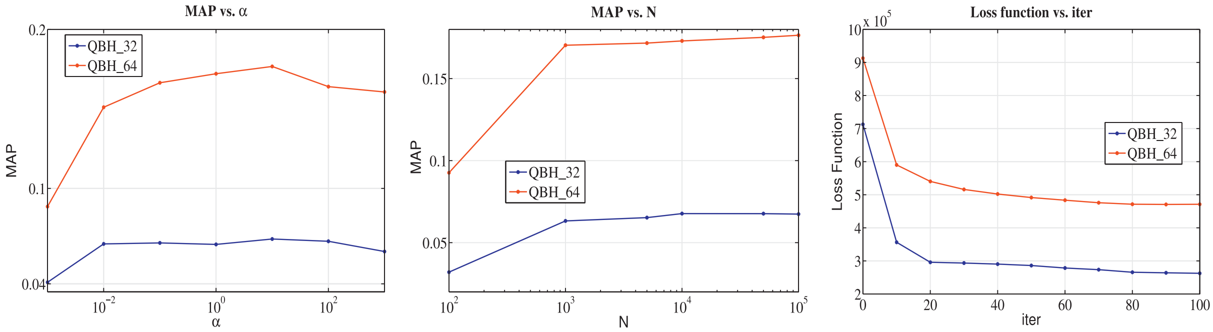
Experiments are conducted on seven widely-used high-dimensional datasets. Three of them are unlabeled datasets to evaluate the unsupervised hashing methods: SIFT1M [6], GIST1M [6], and SIFT1B [6]. Each dataset is composed of disjoint training set, query set, and base set (on which the search is performed). SIFT1M provides 10^5 training points, 10^4 query points and 10^6 database points with each point being a 128-dimensional SIFT descriptor. GIST1M provides 5×10^5 training points, 10^3 query points and 10^6 database points with each point being a 960-dimensional GIST feature. SIFT1B is composed of 10^8 training points, 10^4 query points and as large as 10^9 database points. Following [9], we use the first 10^6 training points on the SIFT1B dataset. The whole training set is used on SIFT1M and GIST1M.

The other datasets we use are image datasets with labels: ESPGAME, IAPRTC12 and MIRFLICKR. ESPGAME [51] contains 20,000 images, IAPRTC12 has 20,000 images and MIRFLICKR [10] is a collection of 25,000 images. We used the default splits for these three datasets. We use 15 different visual descriptors for these three datasets. These include one Gist descriptor, six global color histograms, and eight local bag-of-visual-words features. We concatenate them into a single feature and apply PCA to reduce the dimensionality to 2000.

We further utilize a video dataset: UQ_VIDEO [7] is a video dataset crawled from YouTube for the task of near-duplicate video detection. After filtering out the videos whose sizes are greater than 10M, the dataset contains 169,952 videos in total. 3,305,525 keyframes are further extracted from these videos. We use the provided features (LBP and HSV) and the default split for training and testing.

ANN search is conducted to evaluate our proposed approaches, and three indicators are reported.

- Recall *vs.* K: the proportion over all the queries where the true nearest neighbor falls within the top ranked K vectors by the approximate distance.



(a) Performance variance with different α (b) Performance variance with different N (c) Number of iterations

Fig. 3. Parameters study with code length 32 and 64 on SIFT1M.

- Mean Average Precision (MAP): For a single query, Average Precision (AP) is the average of the precision value obtained for the set of top-k results, and this value is then averaged over all the queries. The larger the MAP, the better the performance is.
- We further use precision-recall curve to evaluate the performance on UQ_VIDEO dataset.

We compare our QBH with other state-of-the-art hashing algorithms, such as spectral hashing (SH) [5], iterative quantization (ITQ) hashing [29] and K-means Hashing (KMH) [8]. Some other hashing methods (e.g, minimal loss hashing (MLH) [25], PCA hashing (PCAH) [6]) are not compared here because they are outperformed by these compared methods. We also compare with two supervised hashing methods, namely KSH [31] and FastH [10]. Note that our QBH can be modified to a supervised version QBH_S by modifying the affinity matrix **A** in (5) to incorporate the supervised information, as in [31]. Unsupervised QBH with model (12) and (31) have similar results. We only report QBH with model (12).

7.2. Parameters

There are several parameters, e.g., α, the size of the training datasets N and the number of iteration *iter*, affecting the performance of our algorithm. In this subsection, we study the performance variation with different parameters. Due to the space limit, we only report the results on the SIFT1M dataset. The default settings for SIFT1M are: $\alpha = 10$, $Z = 3000$, $iter = 100$ and $N = 10^5$.

We tune α from $10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}$, and the results are shown in Fig. 3(a). When α is relatively small, e.g., $\alpha = 10^{-3}$, the worst performance was achieved. That is to say, if the property preserving term is dominant, the MAP is unsatisfactory. With the increase of α, the performance is slightly improved until reaching peak at $\alpha = 10^1$. Further raising α will result in a slight drop of the performance, which means that balancing the quantization and property preserving error is better than using only one of them.

The size of training dataset N affects the training speed and model accuracy. We tune $N = 10^3, 5 \times 10^3, 10^4, 5 \times 10^4, 10^5$, and illustrate the performance changes in Fig. 3(b). There is an increasing trend with the rising of N. When N reaches 10^4, further increasing the training number will not improve the MAP significantly.

The loss function in each iteration is shown in Fig. 3(c). The loss function drops dramatically in the first 20 iterations, and then keeps stable after 30 iterations. This indicates the efficiency of our solution.

7.3. Results on unlabeled datasets

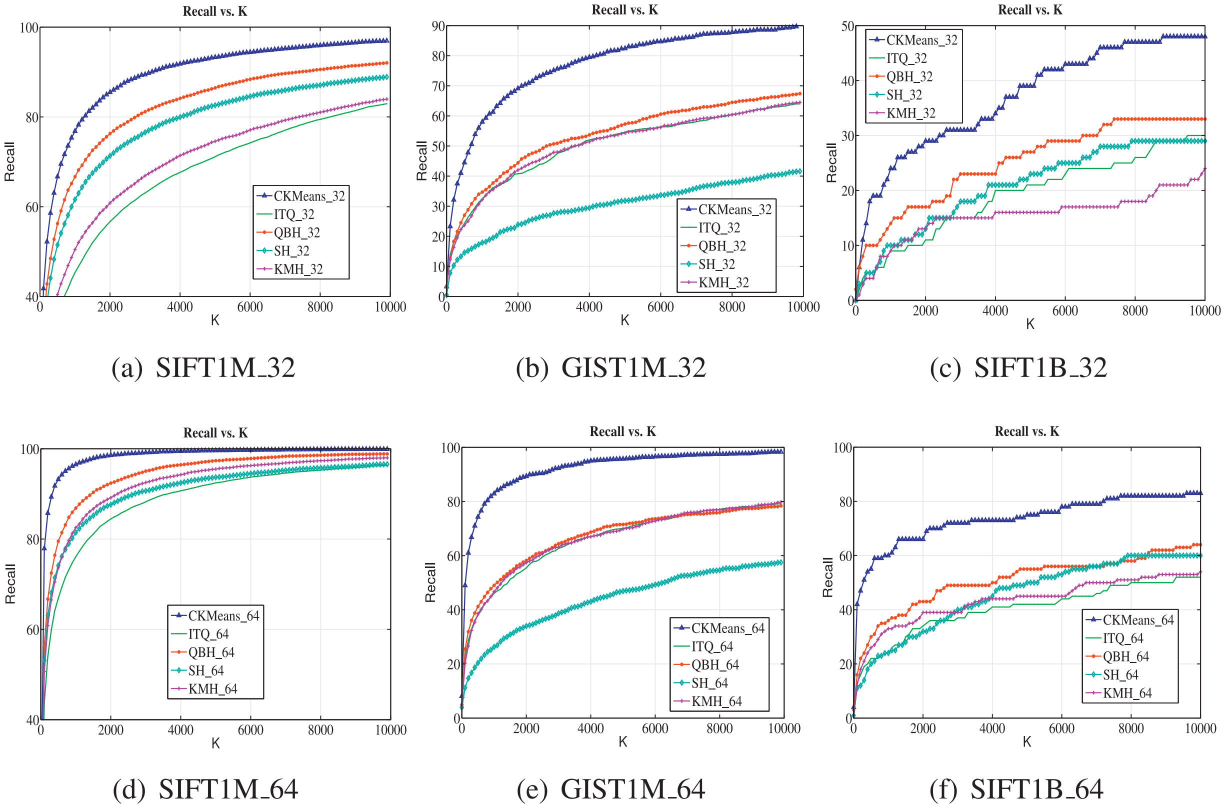
Fig. 4 shows the comparisons of different unsupervised hashing methods on the three unlabeled datasets. We have tested L=32 and L=64. We used the codes provided by the authors to compare different algorithms. For KMH, the bit number of each subspace is 8 for L=32, and is 16 for $L = 64$. For the other algorithms, the default settings are used. From these figures, we have the following observations:

- Our method consistently outperforms the other hashing methods in all datasets. In SIFT1M and SIFT1B datasets, the improvement of QBH over the counterparts is more significant, compared with that in GIST1M. Also, the improvements gap between QBH and other hashing methods is larger in the case of $L = 32$, than that of $L = 64$. On the other hand, CKMeans achieves the best performance in terms of recall compared with all the other hashing methods, and the improvements over state-of-the-art hashing methods is 5% to 20%.
- With the increase of code length, the performance of different hashing methods is improved accordingly. More specifically, the recall improvements of KMH (20%-28%) and ITQ (20%-25%) are generally more significant than SH (10%-15%) on SIFT1M and SIFT1B dataset, while the improvements on GIST1M are more consistent.
- SH performs surprisingly well on SIFT1M and SIFT1B datasets, but it is inferior on GIST1M. KMH is competitive in most settings, especially when the code length is 64 bits.

7.4. Results of supervised hashing methods on image datasets

Fig. 5 shows the comparisons of both unsupervised and supervised hashing methods on the three labeled datasets. Similarly, we tested L=32 and L=64 and we used the codes provided by the authors. For KMH, the bit number of each subspace is 8 for L=32, and is 16 for $L = 64$. For FastH, we chose 'graphCut' for binary code inferring and 'boost_tree' for hash function learning. For the other algorithms, the default settings are used. From these figures, we have the following observations:

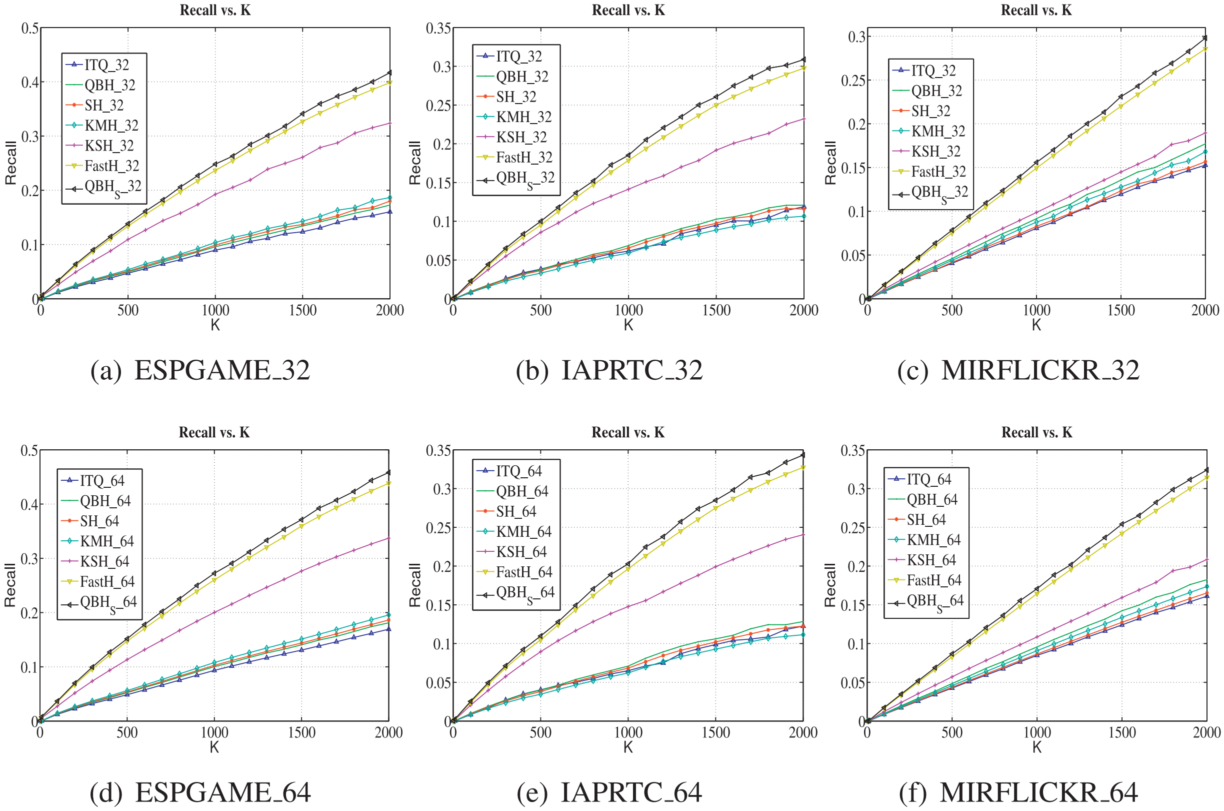
- The supervised hashing methods outperform their unsupervised counterparts significantly. More specifically, the gap between the best supervised and unsupervised hashing methods is 25%, 21% and 14% respectively, for ESPGAME, IAPRTC and MIRFLICKR datasets, with $L = 64$. This indicates that exploiting the content information only cannot preserve the label-based similarity.
- For all the unsupervised methods, similar performance is achieved. Our QBH performs the best in general, but it was



(a) SIFT1M_32 (b) GIST1M_32 (c) SIFT1B_32

(d) SIFT1M_64 (e) GIST1M_64 (f) SIFT1B_64

Fig. 4. The comparison of different unsupervised hashing methods with code length 32 and 64. SH [5], ITQ [29] and KMH [8], CKMeans [9].



(a) ESPGAME_32 (b) IAPRTC_32 (c) MIRFLICKR_32

(d) ESPGAME_64 (e) IAPRTC_64 (f) MIRFLICKR_64

Fig. 5. The comparison of different unsupervised and supervised hashing methods with code length 32 and 64. SH [5], ITQ [29] and KMH [8], KSH [31] and FastH [10].

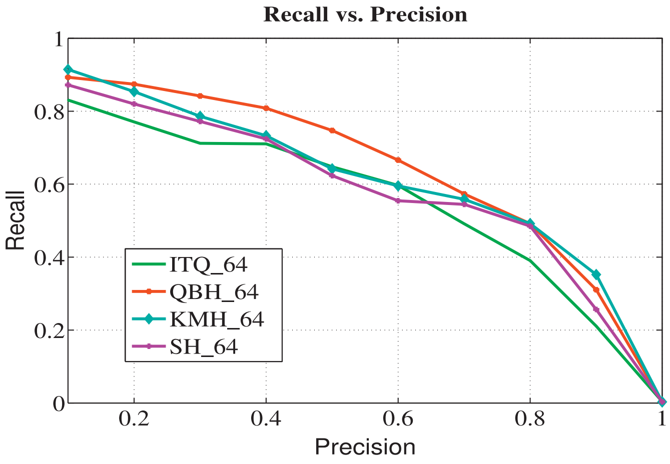


Fig. 6. Results on UQ_VIDEO dataset.

Table 3
Time cost (seconds) for 1000-NN search using QBH and SH with code length 32 and 64 on different sizes of SIFT1B dataset.

Size	QBH_32	SH_32 [5]	QBH_64	SH_64 [5]
100M	0.031	0.030	0.050	0.049
200M	0.034	0.033	0.055	0.055
300M	0.039	0.037	0.067	0.066
400M	0.042	0.040	0.076	0.075
500M	0.046	0.044	0.088	0.087
600M	0.050	0.048	0.095	0.094
700M	0.056	0.055	0.103	0.101
800M	0.062	0.060	0.116	0.114
800M	0.067	0.065	0.129	0.127
1B	0.079	0.077	0.143	0.141

outperformed by KMH in ESPGAME dataset. For all the supervised methods, our QBH_S show consistently superior performance over FastH and KSH.

- With the increase of code length, the supervised hashing methods gain great improvement. On the other hand, unsupervised methods are not very sensitive to the code length.

7.5. Results of unsupervised hashing methods on video dataset

To further test the accuracy and scalability of QBH, we test QBH on a video dataset and compare it with existing unsupervised hashing methods. More specifically, we use the same 24 queries and groundtruth as in [7] to search the dataset. The results are shown in Fig. 6. From this figure, we can observe that our QBH performs the best in general, but its recall is outperformed by KMH when the precision is smaller than 0.1 and greater than 0.8. For the other methods, ITQ shows inferior performance of recall when the precision is less than 0.4 and greater than 0.7.

7.6. Efficiency of the hash code learning and search

All the experiments were conducted on a computer with Intel Xeon(R) CPU E5-2620 @2.00 GHz 2 processors, 32 GB RAM and the 64-bit Windows 7 operating system. For the default parameters settings on SIFT1M dataset (32-bits), the training time for QBH is 1050s, while for SH, KMH and ITQ, is 315s, 2450s and 425s respectively, with the help of Matlab parallel computing.

Table 3 shows the time costs for the search of the hash codes. When the length of hash codes is fixed, different hashing methods require similar time cost to perform 1000-NN search. Therefore, we only report the time cost for QBH and SH. More specifically, we run 1000-NN search using QBH- and SH-generated hash codes of different portions, i.e., 10% - 100%, of SIFT1B dataset. We use the code provided by MIH [52]. From Table 3, it can be observed that the NN search can be efficiently performed using the hashing code. The search on 1 billion 64-bit hash codes requires only 0.14 second. Another observation is that with the increase of the dataset size and code length, longer time is required to perform NN search.

8. Conclusion and future work

In this work, we propose a Quantization-based Hashing (QBH), a general framework which brings the advantages of quantization-based methods into conventional similarity-preserving hashing methods. It is shown that QBH preserves the similarity property and has less quantization error. The framework can be applied to both unsupervised and supervised hashing. Experiments on seven real-life image and video datasets demonstrate that QBH obtains superior results over existing hashing methods in terms of accuracy while keeps the computational time at the same level. In the future, some other properties (e.g, the global similarity property [5], the order property [32], the triangle order property [25]) in the original space can be incorporated into the QBH framework and experimentally evaluated to see the performance improvement.

References

[1] J. Wang, T. Zhang, J. Song, N. Sebe, H.T. Shen, A survey on learning to hash, CoRR abs/1606.00185 (2016).
[2] C. Silpa-Anan, R. Hartley, Optimised kd-trees for fast image descriptor matching, in: CVPR, 2008, pp. 1–8.
[3] M. Muja, D.G. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, in: VISAPP, 2009, pp. 331–340.
[4] J. Wang, N. Wang, Y. Jia, J. Li, G. Zeng, H. Zha, X. Hua, Trinary-projection trees for approximate nearest neighbor search, IEEE Trans. Pattern Anal. Mach. Intell. 36 (2) (2014) 388–403.
[5] Y. Weiss, A. Torralba, R. Fergus, Spectral hashing, in: NIPS, 2008, pp. 1753–1760.
[6] J. Wang, S. Kumar, S. Chang, Semi-supervised hashing for large-scale search, IEEE Trans. Pattern Anal. Mach. Intell. 34 (12) (2012) 2393–2406.
[7] J. Song, Y. Yang, Z. Huang, H.T. Shen, J. Luo, Effective multiple feature hashing for large-scale near-duplicate video retrieval, IEEE Trans. Multimedia 15 (8) (2013) 1997–2008.
[8] K. He, F. Wen, J. Sun, K-means hashing: an affinity-preserving quantization method for learning binary compact codes, in: CVPR, 2013, pp. 2938–2945.
[9] M. Norouzi, D.J. Fleet, Cartesian k-means, in: CVPR, 2013, pp. 3017–3024.
[10] G. Lin, C. Shen, Q. Shi, A. van den Hengel, D. Suter, Fast supervised hashing with decision trees for high-dimensional data, in: CVPR, 2014, pp. 1971–1978.
[11] J. Lu, V.E. Liong, X. Zhou, J. Zhou, Learning compact binary face descriptor for face recognition, IEEE Trans. Pattern Anal. Mach. Intell. 37 (10) (2015) 2041–2056.
[12] Y. Zhang, H. Lu, L. Zhang, X. Ruan, S. Sakai, Video anomaly detection based on locality sensitive hashing filters, Pattern Recognit. 59 (2016) 302–311.
[13] H. Zhao, Z. Wang, P. Liu, The ordinal relation preserving binary codes, Pattern Recognit. 48 (10) (2015) 3169–3179.
[14] R. He, Y. Cai, T. Tan, L.S. Davis, Learning predictable binary codes for face indexing, Pattern Recognit. 48 (10) (2015) 3160–3168.
[15] J. Lu, V.E. Liong, J. Zhou, Simultaneous local binary feature learning and encoding for face recognition, in: ICCV, 2015, pp. 3721–3729.
[16] J. Song, Y. Yang, X. Li, Z. Huang, Y. Yang, Robust hashing with local models for approximate similarity search, IEEE T. Cybern. 44 (7) (2014) 1225–1236.
[17] J. Song, Y. Yang, Y. Yang, Z. Huang, H.T. Shen, Inter-media hashing for large-scale retrieval from heterogeneous data sources, in: SIGMOD, 2013, pp. 785–796.
[18] X. Zhu, Z. Huang, H.T. Shen, X. Zhao, Linear cross-modal hashing for efficient multimedia search, in: ACM Multimedia, 2013, pp. 143–152.
[19] H. Zhang, M. Wang, R. Hong, T. Chua, Play and rewind: Optimizing binary representations of videos by self-supervised temporal hashing, in: ACM Multimedia, 2016a, pp. 781–790.
[20] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, T. Chua, Discrete collaborative filtering, in: SIGIR, 2016b, pp. 325–334.
[21] X. Zhu, L. Zhang, Z. Huang, A sparse embedding and least variance encoding approach to hashing, IEEE Trans. Image Process. 23 (9) (2014) 3737–3750.
[22] J. Song, L. Gao, Y. Yan, D. Zhang, N. Sebe, Supervised hashing with pseudo labels for scalable multimedia retrieval, in: ACM Multimedia, 2015, pp. 827–830.
[23] X. Liu, J. He, C. Deng, B. Lang, Collaborative hashing, in: CVPR, 2014, pp. 2147–2154.
[24] B. Kulis, T. Darrell, Learning to hash with binary reconstructive embeddings, in: NIPS, 2009, pp. 1042–1050.
[25] M. Norouzi, D.J. Fleet, Minimal loss hashing for compact binary codes, ICML, 2011.

[26] F. Shen, C. Shen, W. Liu, H.T. Shen, Supervised discrete hashing, in: CVPR, 2015, pp. 37–45.

[27] T. Ge, K. He, J. Sun, Graph cuts for supervised binary coding, in: ECCV, 2014, pp. 250–264.

[28] H. Jégou, M. Douze, C. Schmid, Product quantization for nearest neighbor search, IEEE Trans. Pattern Anal. Mach. Intell. 33 (1) (2011) 117–128.

[29] Y. Gong, S. Lazebnik, A. Gordo, F. Perronnin, Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval, IEEE Trans. Pattern Anal. Mach. Intell. 35 (12) (2013) 2916–2929.

[30] B. Xu, J. Bu, Y. Lin, C. Chen, X. He, D. Cai, Harmonious hashing, in: IJCAI, 2013, pp. 1820–1826.

[31] W. Liu, J. Wang, R. Ji, Y. Jiang, S. Chang, Supervised hashing with kernels, in: CVPR, 2012, pp. 2074–2081.

[32] J. Wang, J. Wang, N. Yu, S. Li, Order preserving hashing for approximate nearest neighbor search, in: ACM Multimedia, 2013, pp. 133–142.

[33] M. Norouzi, D.J. Fleet, R. Salakhutdinov, Hamming distance metric learning, in: NIPS, 2012, pp. 1070–1078.

[34] C. Strecha, A.M. Bronstein, M.M. Bronstein, P. Fua, Ldahash: improved matching with smaller descriptors, IEEE Trans. Pattern Anal. Mach. Intell. 34 (1) (2012) 66–78.

[35] P. Jain, B. Kulis, K. Grauman, Fast image search for learned metrics, in: CVPR, 2008, pp. 1–8.

[36] R. Xia, Y. Pan, H. Lai, C. Liu, S. Yan, Supervised hashing for image retrieval via image representation learning, in: AAAI, 2014, pp. 2156–2162.

[37] V.E. Liong, J. Lu, G. Wang, P. Moulin, J. Zhou, Deep hashing for compact binary codes learning, in: CVPR, 2015, pp. 2475–2483.

[38] H. Lai, Y. Pan, Y. Liu, S. Yan, Simultaneous feature learning and hash coding with deep neural networks, in: CVPR, 2015, pp. 3270–3278.

[39] M.Á. Carreira-Perpiñán, R. Raziperchikolaei, Hashing with binary autoencoders, in: CVPR, 2015, pp. 557–566.

[40] L. Gao, J. Song, F. Zou, D. Zhang, J. Shao, Scalable multimedia retrieval by deep learning hashing with relative similarity learning, in: ACM Multimedia, 2015, pp. 903–906.

[41] X. Liu, L. Huang, C. Deng, J. Lu, B. Lang, Multi-view complementary hash tables for nearest neighbor search, in: ICCV, 2015, pp. 1107–1115.

[42] F. Zou, Y. Chen, J. Song, K. Zhou, Y. Yang, N. Sebe, Compact image fingerprint via multiple kernel hashing, IEEE Trans. Multimedia 17 (7) (2015) 1006–1018.

[43] D. Zhang, J. Wang, D. Cai, J. Lu, Self-taught hashing for fast similarity search, in: SIGIR, 2010, pp. 18–25.

[44] Y. Lv, W.W.Y. Ng, Z. Zeng, D.S. Yeung, P.P.K. Chan, Asymmetric cyclical hashing for large scale image retrieval, IEEE Trans. Multimedia 17 (8) (2015) 1225–1235.

[45] J. Wang, J. Wang, J. Song, X. Xu, H.T. Shen, S. Li, Optimized cartesian k-means, IEEE Trans. Knowl. Data Eng. 27 (1) (2015) 180–192.

[46] A. Babenko, V.S. Lempitsky, Tree quantization for large-scale similarity search and classification, in: CVPR, 2015, pp. 4240–4248.

[47] X. Wang, T. Zhang, G. Qi, J. Tang, J. Wang, Supervised quantization for similarity search, in: CVPR, 2016, pp. 2018–2026.

[48] T. Zhang, G. Qi, J. Tang, J. Wang, Sparse composite quantization, in: CVPR, 2015, pp. 4548–4556.

[49] L. Gao, J. Song, F. Nie, Y. Yan, N. Sebe, H.T. Shen, Optimal graph leaning with partial tags and multiple features for image and video annotation, in: CVPR, 2015, pp. 4371–4379.

[50] W. Liu, J. Wang, S. Kumar, S. Chang, Hashing with graphs, in: ICML, 2011, pp. 1–8.

[51] M. Guillaumin, T. Mensink, J.J. Verbeek, C. Schmid, Tagprop: discriminative metric learning in nearest neighbor models for image auto-annotation, in: ICCV, 2009, pp. 309–316.

[52] M. Norouzi, A. Punjani, D.J. Fleet, Fast exact search in hamming space with multi-index hashing, IEEE Trans. Pattern Anal. Mach. Intell. 36 (6) (2014) 1107–1119.

Jingkuan Song is currently a Postdoctoral Research Scientist in Columbia University. He joined a University of Trento as a Research Fellow sponsored by Prof. Nicu Sebe (2014-2016). He obtained his PhD degree in Information Technology from The University of Queensland (UQ), Australia, in 2014. He received his BS degree in Computer Science from University of Electronic Science and Technology of China. His research interest includes large-scale multimedia retrieval, image/video segmentation and image/video annotation using hashing, graph learning and deep learning techniques.

Lianli Gao obtained her Ph.D degree from the School of ITEE, the University of Queensland, Australia, in 2014. Currently, she is an Associate Professor at the department of Big Media Computing Centre, University of Electronic Science and Technology of China, teaching subjects on Semantic Web and Machine Learning theory etc. Her research lines are data mining, machine learning, multimedia analysis and Semantic Web.

Nicu Sebe is currently a Professor with the University of Trento, Italy, leading the research in the areas of multimedia information retrieval and human behavior understanding. He was the General Co-Chair of the IEEE FG Conference 2008 and ACM Multimedia 2013, and the Program Chair of the International Conference on Image and Video Retrieval in 2007 and 2010, and ACM Multimedia 2007 and 2011. He will be the Program Chair of ECCV 2016 and ICCV 2017. He is a fellow of the International Association for Pattern Recognition.