Computer Vision and Image Understanding 125 (2014) 214–222

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

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

Quadra-embedding: Binary code embedding with low quantization error ☆

Youngwoon Lee [a,b], Jae-Pil Heo [b], Sung-Eui Yoon [b,*]

[a] ETRI, 218 Gajeong-ro, Yuseong-gu, Daejeon, Republic of Korea
[b] KAIST, 291 Daehak-ro(373-1 Guseong-dong), Yuseong-gu, Daejeon, Republic of Korea

ARTICLE INFO

Article history:
Received 4 April 2013
Accepted 18 April 2014
Available online 28 April 2014

Keywords:
Large-scale image retrieval
Binary code embedding
Hashing
Quantization

ABSTRACT

Thanks to compact data representations and fast similarity computation, many binary code embedding techniques have been proposed for large-scale similarity search used in many computer vision applications including image retrieval. Most prior techniques have centered around optimizing a set of projections for accurate embedding. In spite of active research efforts, existing solutions suffer from diminishing marginal efficiency and high quantization errors as more code bits are used.

To reduce both quantization error and diminishing efficiency we propose a novel binary code embedding scheme, *Quadra-Embedding*, that assigns two bits for each projection to define four quantization regions, and a binary code distance function tailored to our method. Our method is directly applicable to most binary code embedding methods. Our scheme combined with four state-of-the-art embedding methods has been evaluated and achieves meaningful accuracy improvement in most experimental configurations.

1. Introduction

Scalable and efficient similarity search plays a key role in many large-scale computer vision applications dealing with high-dimensional data space. One example is the web-scale image retrieval. Common image descriptors, e.g., Bag-of-visual-Words or GIST, used for image retrieval have hundreds or thousands of dimensionality, and there are billions of images available on the web.

Traditional solutions adopting hierarchical structures (e.g., kd-trees) [1,2] do not provide sufficient scalability in terms of both computational time and storage costs for high-dimensional, large-scale data sets. Recently, embedding high-dimensional data to short binary codes has been recognized as one of the most promising approaches to address both high-dimensionality and scalability issues of data, since it can provide a compact representation for data and efficient similarity search. Consequently, a lot of binary code embedding algorithms have been studied lately [3–12]. The core goal of binary code embedding methods is to preserve similarities among original high-dimensional data in the corresponding binary codes, i.e. neighbor data points in the original high-dimensional space should be encoded to similar binary codes.

Most binary embedding methods compute a binary code of each data element using multiple projections, which can be categorized into two groups: data-independent and data-dependent methods. Data-independent methods construct the projections based on vectors randomly drawn from some specific distributions. The well-known work in this category is locality-sensitive hashing (LSH) [13]. LSH is extended to various similarity metrics [4,6,7]. However, recent researches give more attentions to data-dependent methods for designing more data-friendly projections in order to achieve the higher accuracy. Notable examples include spectral hashing [5], sequential projection [8], joint optimization [9], and iterative quantization [10].

Despite the intensive research efforts to obtain effective projections, there are still remaining issues; (1) diminishing returns as the number of projections increases, and (2) quantization errors in high-density regions. The main cause of the diminishing return is the growing difficulty of defining both independent and informative projections, as the number of projections increases. In regard to the quantization error, quantization boundaries are usually within dense regions, causing that neighbor data points are assigned to different binary codes [14]. In this paper we aim to resolve these two issues. More precisely, our contributions are:

- For each projection, we assign two bits to define four quantization regions as shown in Fig. 1a instead of the conventional binary regions based on a single bit (Section 3.2). In addition

☆ This paper has been recommended for acceptance by L.S. Davis.

* Corresponding author. Fax: +82 42 350 3510.
E-mail addresses: lywoon89@gmail.com (Y. Lee), jaepilheo@gmail.com (J.-P. Heo), sungeui@gmail.com (S.-E. Yoon).

http://dx.doi.org/10.1016/j.cviu.2014.04.007
1077-3142/

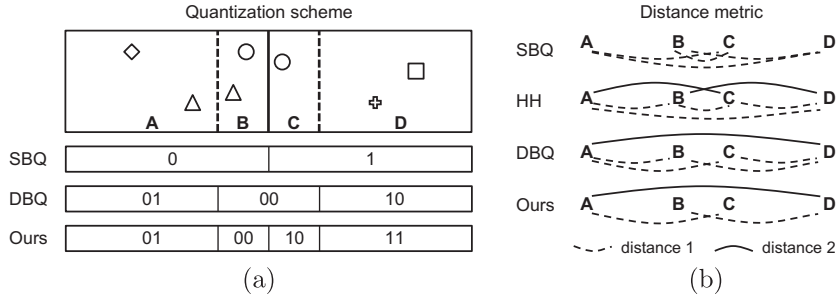

Fig. 1. (a) Different quantization schemes for a data set shown in the top row given a projection. The solid vertical line is a hyperplane associated with the projection, while two dashed lines are additional partitioning planes, i.e. offset surfaces used for defining the buffer area. *SBQ* and *DBQ* indicate the Single-Bit Quantization and Double-Bit Quantization [14], respectively. (b) Different distance metrics used in different encoding schemes. Any pair that does not have distance of 1 or 2 has zero distance. *HH* is Hierarchical Hashing [11], which uses the same encoding scheme with *Ours*.

we propose a novel distance measure between two binary codes tailored to our binary code embedding scheme (Section 3.3).
- We formulate an optimization problem to decide partitioning boundaries of four regions suitable for our distance metric by minimizing the quantization error (Section 3.4).

According to our best knowledge, only two existing approaches, Hierarchical Hashing (HH) [11] and Double-Bit Quantization (DBQ) [14] aimed for similar goals that our method strives for. HH allowed each projection to have four quantization states, but used the common Hamming distance that does not exploit all the benefits of having four states. DBQ quantized projection values into three different states with two bits and used the Hamming distance (see Fig. 1). In contrast, our method fully utilizes four states that two bits can encode, and adopts a specialized distance metric that further lowers down the quantization error caused by having four regions. To demonstrate benefits of our method, we have tested our method in three well-known image retrieval benchmarks in the context of two different types of nearest neighbor search, k-NN and ε-NN, in Section 4. Our method achieves significant improvements in accuracy over other prior encoding schemes across different hashing methods including LSH [4], spectral hashing [5], shift-invariant kernel-based LSH [7], and iterative quantization [10].

This paper is an extended version of our earlier work initially presented at ACCV 2012 [15]. Additional materials for the extension include a simple thresholding strategy (Section 3.4) and detail descriptions on various components of our earlier work [15] with empirical results (Section 4.6 and Section 4.7).

2. Related work

2.1. Image descriptors

To compactly and robustly represent images, many image descriptors have been developed [16]. Some well-known examples include Bag-of-visual-Words (BoW) [17] and GIST [18]. Finding similar images is then reduced to nearest neighbor search among image descriptors. Since these image descriptors are defined as high-dimensional vectors (e.g., larger than a few hundreds), performing nearest neighbor search for such high-dimensional image descriptors in an efficient and effective manner is very challenging [13]. Conventional approaches such as using hierarchical data structures (e.g., kd-trees) [19,1,2,20] based on recursive space partitioning have been known to be inefficient for such high-dimensional data in terms of search time and storage costs.

2.2. Binary code embedding methods

To overcome the difficulty of handling large-scale and high-dimensional data sets, a significant amount of researches has been put on embedding high-dimensional data into compact binary codes preserving similarity among original data.

One of the most popular embedding methods is Locality-Sensitive Hashing (LSH) [13], which uses random projections drawn from a specific distribution. Many extensions of LSH have been proposed such as LSH with p-stable distributions [4], min-hash [21], learned metrics [22], kernelized LSH [6], shift-invariant kernel-based LSH [7], etc.

Data-independent methods [4,7] based on the random projections do not exploit data distributions. Departing from data-independent techniques, data-dependent approaches have been more widely studied recently, because they can achieve higher accuracy given a fixed code length over data-independent techniques. Spectral hashing [5] derived projection directions based on spectral graph partitioning. He et al. [9] introduced a hashing technique that jointly optimizes both search accuracy and time. Heo et al. [12] proposed spherical hashing that partitions data using hyperspheres instead of commonly adopted hyperplanes. These data-dependent techniques share similar optimization goals such as balanced partitioning for each hashing function and independence between hashing functions.

In addition to considering aforementioned optimization goals there are some efforts to reduce the quantization error of a binary hashing function. Kulis and Darrell [23] and Norouzi and Fleet [24] minimized an empirical loss function on the Euclidean distances among data points and the Hamming distances among binary codes that share the common goal, minimizing errors made by quantization. Wang et al. [8] sequentially constructed each hyperplane to reduce the quantization error. Gong and Lazebnik [10] computed hyperplanes based on principal component analysis and then rotated them to minimize the quantization error iteratively. Joly and Buisson [25] used hyperplanes computed by support vector machine that produces maximum margins between data points and hyperplanes.

2.3. Encoding schemes with low quantization error

All the prior binary code embedding techniques mentioned in the last section aim to optimize projections for achieving the similarity-preserving property. On the other hand, our approach targets for reducing the quantization error and thus maximizing the discriminative power of a given set of projections.

There have been only a few research efforts on designing encoding schemes that can reduce the quantization error. Liu et al. [11] interpreted an approximate nearest neighbor structure with an anchor graph and then applied the graph Laplacian technique to the graph, in order to construct discriminative projections. In order to attain higher accuracy, Liu et al. [11] proposed Hierarchical Hashing (HH) that uses an additional hash bit for each projection that minimizes the cut value of the graph Laplaician. Although

HH achieved significant improvement for Anchor Graph Hashing (AGH) [11], it is not straightforward to apply its encoding scheme to other hashing methods. On the other hand, our technique is orthogonal and can be used together with most prior embedding methods mentioned in the last section.

Kong and Li [14] proposed a double-bit quantization, a similar strategy to our method for reducing the quantization error. This technique quantizes projection values into three states with two bits to inform whether a point is near a partitioning hyperplane or not. This technique uses the common Hamming distance for their encoding scheme. It was applied to prior binary code embedding methods such as iterative quantization [10] and achieved meaningful improvement. Unlike this method, our approach utilizes four states, the maximum number of states that two bits can represent, and adopts a novel distance function that can further reduce the quantization error caused by additional hashing boundaries. We will demonstrate that our method shows noticeable improvements over this technique in a wide variety of settings.

3. Our approach

In this section we elaborate the motivation of our approach, followed by explaining components of our method.

3.1. Motivation

Most binary code embedding techniques do not differentiate nor encode whether data points are located closely or far away, when they have the same binary code. This phenomenon becomes more pronounced when we encode data points with short binary code sizes. As increasing their code sizes one can achieve higher accuracy. Nonetheless, the benefit realized by having more projections, resulting in allocating more bits, diminishes quickly in most prior binary code embedding techniques, as demonstrated in Fig. 2. This is mainly because it is difficult to decorrelate hashing functions and to find useful projections for hashing [11,14].

Before presenting our method, we first illustrate the quantization error of projection-based binary code embedding methods in Fig. 1. At a high level, the quantization error is caused by two opposing cases: nearby points with a high Hamming distance, and faraway points with a low Hamming distance. We name these two as *inter-quantization* and *intra-quantization* errors, respectively. As an example of the inter-quantization error, two nearby circular points in the top row of Fig. 1a are partitioned by a hyperplane shown in the solid line and thus have different binary codes. Therefore, the Hamming distance between them is not zero, even though these two circular points are closely located. On the other hand, the right circular point is far from the rectangular point,

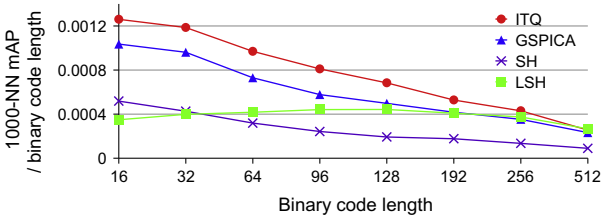

Fig. 2. This figure shows the diminishing return of having more projections for hashing. *Y*-axis is the mAP (mean Average Precision) of nearest neighbor search divided by the number of projections. In the case of data-dependent methods (ITQ [10], GSPICA [9], and spectral hashing (SH) [5]), mAP values per each hash bit consistently decrease as the total number of projections increases. LSH, a data-independent technique, however, shows the diminishing return after 128 bit code lengths.

but they are partitioned together in the right side of the hyperplane. This is an example of the intra-quantization error, since they have the same binary code and thus zero Hamming distance even though they are located faraway.

Although both cases generating the quantization error are important, we focus on minimizing the inter-quantization error caused by nearby points with high Hamming distances. This decision is based on two reasons: first, most hashing techniques compute a short list of candidate nearest neighbor points by identifying data points that have zero or low Hamming distance. Once two nearby points have a high Hamming distance, these nearby points may not be included in the short list, unless computing an excessively long candidate list. Second, we can effectively control intra-quantization error by culling out faraway points that have low Hamming distances from the query point by adopting more expensive distance computation or re-ranking methods on the short list.

3.2. Binary code embedding function

In order to reduce the inter-quantization error, we introduce a buffer area along a hyperplane. The buffer area is defined by two offset surfaces that are constructed by offsetting the hyperplane (or hypersphere) with offset distance r, called *margin*. Two dashed lines shown in the top row of Fig. 1a are two offset surfaces defining the buffer area given a hyperplane denoted by the solid line.

Two offset surfaces with the given hyperplane define four disjoint regions. In order to encode where a data point x is located among these four regions, our Quadra-Embedding method allocates two binary bits, $h_1(x)$ and $h_2(x)$, per one projection $f(x)$. In the case of using a simple hyperplane for the projection $f(x)$ is defined by $w^T x$, where w is a normal of the hyperplane; projections based on kernel techniques [6,7,9] and hyperspheres [12] are defined similarly. Specifically, the first and second hash bits are defined as follows:

$$h_1(x) = sign(f(x)) \quad \text{and} \quad h_2(x) = sign(|f(x)| - r). \tag{1}$$

Instead of defining the buffer area with the margin r we can define four regions in a more general manner with three thresholds $\{t_1, t_2, t_3\}$ indicating positions of the left offset surface, the hyperplane, and the right offset surface respectively, as follows:

$$h_1(x) = sign(f(x) - t_2) \quad \text{and} \quad h_2(x) = \begin{cases} 0 & \text{if } t_1 \leqslant f(x) \leqslant t_3 \\ 1 & \text{otherwise} \end{cases}. \tag{2}$$

Intuitively, the first hash bit $h_1(x)$ indicates which side of the hyperplane contains the data point in the same manner of hash bits used in most prior hashing techniques. On the other hand, the second hash bit $h_2(x)$ indicates whether the data point is inside the buffer area. For a given data point $x \in \mathbb{R}^d$, the binary code $X \in \{0,1\}^m$ of a length m is defined by concatenating two vectors of the first and second hashing functions, $H_1(x)$ and $H_2(x)$, each of which is based on $m/2$ projections as follows:

$$X = (H_1(x), H_2(x)) = (h_1^1(x), \cdots, h_1^{m/2}(x), h_2^1(x), \cdots, h_2^{m/2}(x)). \tag{3}$$

3.3. Distance function for quadra-embedding

In order to maximize the benefit of our quantization scheme using two hash bits, we propose a Quadra-Embedding Distance (QED) function tailored to our method. The QED between two binary codes, $X = (H_1(x), H_2(x)) = (X_1, X_2)$ and $Y = (H_1(y), H_2(y)) = (Y_1, Y_2)$, is defined as follows:

$$d_{QED}(X,Y) = 2|(X_1 \oplus Y_1) \wedge (X_2 \wedge Y_2)| + |(X_1 \oplus Y_1) \wedge (X_2 \oplus Y_2)|. \tag{4}$$

Intuitively QED between X and Y measures how many regions we should cross to reach a region of the data binary code X (or Y) from the region of the query binary code Y (or X). Therefore QED imposes the zero distance on neighboring regions. It results in low inter-quantization errors, since data points are not discriminated by a single boundary unlike most prior methods.

QED among four disjoint regions is shown in Fig. 1b. For example, given the right circular point (having the binary code of 10) in the top row of Fig. 1a, the QED against itself (10), the left circular point (00), the rectangular point (11), and the diamond point (01) are 0, 0, 0, and 1, respectively. Computing our distance function has only a minor overhead compared to the Hamming distance as discussed in Section 4.5.

3.4. Threshold optimization process

Given our encoding scheme and distance function, it is critical to optimize the margin of the buffer area or more generally three positional values $\{t_1, t_2, t_3\}$ of two offset surfaces and the hyperplane. As we have larger buffer area, the inter-quantization error given its projection boundary decreases. On the other hand, a large buffer area makes a hash bit underutilized in terms of discriminative power. Our goal of optimizing thresholds is minimizing quantization errors along boundaries while keeping sufficient discriminative power.

For t_2 we can simply use the original quantization boundary, t_2^o, of a projection computed by any hashing methods. However, we have found that we can achieve higher accuracy by mildly optimizing the value of t_2, which is set to be not too far away from t_2^o. This is mainly because the original hashing method may not consider the inter-quantization error well to lower down the quantization error, while constructing projections. Nonetheless, adjusting the original quantization boundary has a potential risk to destruct benefits of the original hashing method. As a result, we adjust the value of t_2 such that it does not deviate much from t_2^o, while aggressively attempting to optimize the locations (t_1 and t_3) of two offset surfaces.

Simple threshold learning: One possible way to set thresholds is partitioning projected data points in a balanced manner. Let $T \subset \mathbb{R}^d$ be a training set containing n points and P be a set of projected data points, i.e. $P = \{p | p = f(x), x \in T\}$. Then balanced thresholds t_1, t_2, and t_3 are $\frac{1}{4}n$th, $\frac{2}{4}n$th, and $\frac{3}{4}n$th largest values in P respectively, leading that four regions contain almost the same number of data points. This approach has a small computational overhead, $O(n \log n)$ time complexity, and provides an effective set of thresholds for the case that the projected values are distributed according to the normal distribution. On the other hand, we encounter highly biased thresholds, when data is not normally distributed (e.g., spectral hashing [5]).

One alternative way to set thresholds is utilizing ε. Since ε-NN does not consider points farther than the distance ε, we can perfectly control inter-quantization errors with QED if we set thresholds t_1 and t_3 to $t_2 - \varepsilon$ and $t_2 + \varepsilon$. However, the high dimensionality of the data points makes this approach negated, since most points reside within the distance ε to the projection boundaries.

Threshold optimization: To make our method effective and robust, we propose an optimization of thresholds. Our optimization adjusts thresholds such that projected data points in each region get away from thresholds, in order to reduce the quantization error. Let four regions divided by three thresholds be P_1, P_2, P_3, and P_4, i.e. $P_1 = \{p \in P | p \leqslant t_1\}, P_2 = \{p \in P | t_1 < p \leqslant t_2\}, P_3 = \{p \in P | t_2 < p < t_3\}$, and $P_4 = \{p \in P | t_3 \leqslant p\}$. We also define μ_1, μ_2, μ_3, and μ_4 to be the mean values of four subsets P_1, P_2, P_3 and P_4 respectively.

The problem of finding a good set of thresholds can be formulated as:

Table 1
This table shows the results of our method with the balanced thresholds (*-B*) and the optimized thresholds (*-O*). Tested binary hashing methods are *ITQ* [10], *LSH* [4], *SKLSH* [7], and *SH* [5]. Although our method with the balanced thresholds performs better under various settings, our method with the optimized thresholds works robustly for *SKLSH* [7] and *SH* [5] that do not have normally distributed projections. The best mAP under the same setting is shown in the bold face.

# bits	100-NN mAP			ε-NN mAP		
	64 bits	128 bits	256 bits	64 bits	128 bits	256 bits
ITQ-O	0.172	0.323	**0.485**	0.292	**0.440**	**0.583**
ITQ-B	**0.183**	**0.323**	0.467	**0.304**	0.437	0.563
LSH-O	0.043	0.110	**0.231**	**0.134**	**0.232**	**0.357**
LSH-B	**0.047**	**0.112**	0.230	0.133	0.229	0.350
SKLSH-O	**0.034**	**0.080**	**0.187**	**0.112**	**0.191**	**0.317**
SKLSH-B	0.031	0.072	0.161	0.096	0.173	0.282
SH-O	**0.141**	**0.250**	**0.286**	**0.217**	**0.339**	**0.390**
SH-B	0.127	0.189	0.198	0.192	0.267	0.287

$$min \sum_{p \in P_1} (p - \mu_1)^2 + \sum_{p \in P_2} (p - \mu_2)^2 + \sum_{p \in P_3} (p - \mu_3)^2 + \sum_{p \in P_4} (p - \mu_4)^2. \tag{5}$$

The above equation measures the variance of projected data points within each region. The proposed formulation (Eq. (5)) is sensitive not only to a few large projected values in P_1 and P_4, but also to the high-density points near the hyperplane. To alleviate this effect, we remove negative factors with a filtering function $\ell(\cdot) = max(\cdot, 0)$.

Our objective is then defined to decide thresholds that minimize the following penalty function:

$$J = \sum_{p \in P_1} \ell(p - \mu_1)^2 + \sum_{p \in P_2} \ell(\mu_2 - p)^2 + \sum_{p \in P_3} \ell(p - \mu_3)^2 + \sum_{p \in P_4} \ell(\mu_4 - p)^2. \tag{6}$$

The filtering function $\ell(\cdot)$ ignores negative inputs and thus serves as the following two purposes: (1) the filtering function used in the second and third terms ignores projected data points near the given hyperplane, in order to less aggressively adjust t_2, and (2) the filtering function used in the first and fourth terms ignores faraway points that do not contribute much to the change of quantization error, but affect heavily the penalty function.

Values of each term of Eq. (6) can be precomputed for all the possible pairs of two neighboring thresholds in $O(n^2)$ with dynamic programming. We then find the optimal solution by exhaustively searching all the possible pairs of t_1 and t_3. Note that as we incrementally change the value of t_1 or t_3, the optimal t_2 can be computed in a constant time by looking into only a fixed number of potential candidates. This incremental computation makes our algorithm to run in $O(n^2)$. In practice our algorithm takes approximately 1.44 s on average for each projection to compute three thresholds with 20 k training data points. For example, it took less than one minute for 64 bit code lengths.

We have conducted an evaluation on the *CIFAR* dataset [26] to verify how much our optimization improves the performance of Quadra-Embedding (see Table 1). Our method with balanced thresholds achieves high accuracy for many configurations. For the case that the projection values are highly biased (e.g., spectral hashing [5] and shift-invariant kernel LSH [7]), our method with the threshold optimization improves the accuracy, especially for the long code lengths (e.g., 14% mAP improvement at 256 bit length), and shows robust results on various settings.

4. Results and discussions

In this section we explain our experiment protocols and compare our method against the state-of-the-art techniques.

4.1. Datasets

We evaluated our method on the following three different image datasets:

- *CIFAR-60 K-512D*: A set of 512 dimensional GIST descriptors [18] from the CIFAR-10 dataset [26] sampled from Tiny Images [3].
- *GIST-1 M-960D*: A set of 960 dimensional, one million GIST descriptors that were used by Jégou et al. [27].
- *GIST-75 M-384D*: A set of 384 dimensional GIST descriptors [18] of 75 million images from Tiny Images [3].

Note that we did not normalize the data points to unit-norm vectors. The original distance structure between data points was therefore preserved.

4.2. Tested hashing methods

In order to test our method, we compared our method against the following state-of-the-art hashing methods:

- *LSH*: Locality-Sensitive Hashing [4] with the translated data to have the zero mean, as discussed in [10].
- *SH*: Spectral Hashing [5].
- *SKLSH*: Shift-invariant Kernel-based Locality-Sensitive Hashing [7] with the bandwidth parameter, which is the inverse of the mean distance [28].
- *ITQ*: ITerative Quantization [10] with the translated data to have the zero mean.

For *LSH* and *ITQ*, all data points are translated to have the zero mean, which simply deals with a bias term. Since *ITQ* and *SH* are based on principal component analysis, the number of hashing functions cannot exceed the dimensionality of the original data. We therefore measured the performance up to 256 bits.

4.3. Quantization strategies

Given the hashing methods mentioned above, we tested different bit allocation methods as follows:

- *SBQ*: Single-Bit Quantization, which is commonly used in most prior hashing methods.
- *DBQ*: Double-Bit Quantization proposed by Kong and Li [14].
- *Ours*: Our binary code embedding method, Quadra-Embedding.

We tested different quantization schemes with hashing functions computed by hashing methods listed in Section 4.2. We combined one of quantization strategies with one of hashing methods. To concisely represent them, we use a naming scheme like *LSH-SBQ*, which uses *SBQ* with *LSH*. As another example, *LSH-Ours* denotes our embedding scheme combined with *LSH*.

Note that comparisons in this paper are done with the same code length, not doubling the number of bits in *SBQ*. For instance, *ITQ-DBQ* and *ITQ-Ours* with 64 bits use 32 projections computed from *ITQ* and then compared it with *ITQ-SBQ* with 64 bits (64 projections).

We did not compare our method against hierarchical hashing [11], mainly because hierarchical hashing is not naturally applicable to different hashing methods listed in Section 4.2, and its simplified version (using four regions with *DBQ*'s thresholds) has been shown to be consistently inferior than *DBQ* [14].

4.4. Protocols

We followed the evaluation protocols that have been widely used in recent papers [12,25]. We trained hashing functions with randomly chosen 20 k training points on *CIFAR-60 K-512D* and *GIST-1 M-960D*, 100 k points on *GIST-75 M-384D*. Then we randomly selected 10 k, 1 k, and 500 queries on *CIFAR-60 K-512D*, *GIST-1 M-960D*, and *GIST-75 M-384D*, respectively.

All the experimental results on *CIFAR-60 K-512D* and *GIST-1 M-960D* are averaged over five independent training times, and results on *GIST-75 M-384D* are averaged over three independent training times because of its long computational time.

The performance is measured by the mean average precision (mAP), which is the average area under the recall-precision curves. The ground truths are computed based on the Euclidean distance by exhaustively testing all the data sets against query points. We adopted two major protocols for approximate nearest neighbor search, k-NN and ε-NN. The ground truths of a point x in k-NN and ε-NN are defined as follows:

$$g_{k-\mathrm{NN}}(x,y) = \begin{cases} 1 & \text{if } y \in k-NN(x) \\ 0 & \text{otherwise} \end{cases} \text{ and } g_{\varepsilon-NN}(x,y) = \begin{cases} 1 & \text{if } d(x,y) < \varepsilon \\ 0 & \text{otherwise} \end{cases}. \tag{7}$$

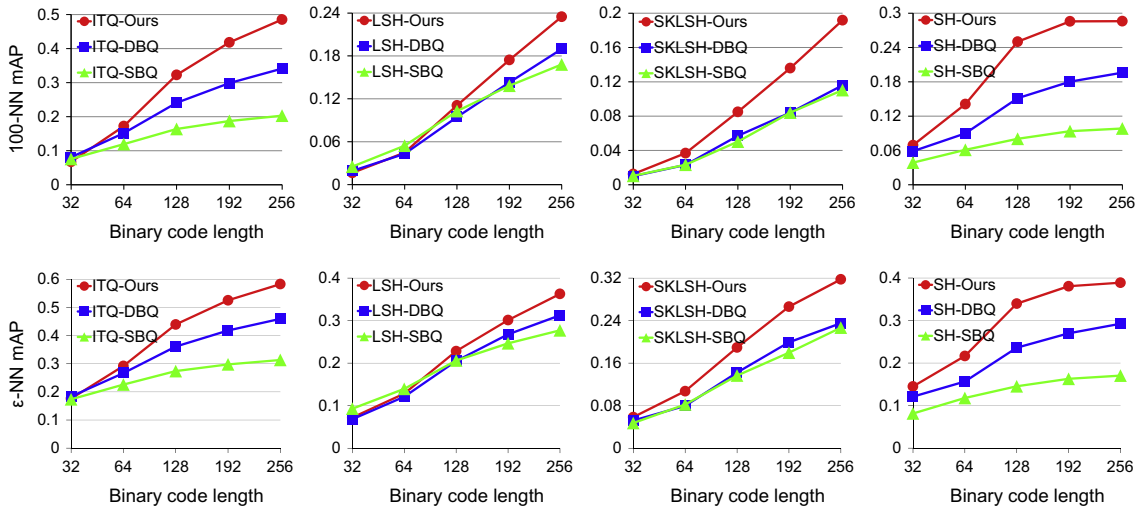

Fig. 3. Results on *CIFAR-60 K-512D* with k-NN (top) and ε-NN search (bottom). Our method improves accuracy over *SBQ* and *DBQ* with different hashing methods. Our method shows the best results when combined with *ITQ* [10].

In ε-NN, ε is determined by averaging distances of 50th nearest neighbors from query points, as used in [14]. In this setting, approximately 299 queries in *GIST-1 M-960D* and 1 731 queries in *CIFAR-60 K-512D* have no nearest neighbors within ε and thus we ignore these queries for evaluation. For *GIST-75 M-384D* we measure the performance only with k-NN, since computing the ground truths of ε-NN takes long computational time. We compared different methods including ours given the same number of bits. Specifically *DBQ* and our method use only half of the number of hashing functions that *SBQ* uses, but allocate two bits for each hashing function.

4.5. Results

Fig. 3 shows results of different methods for k-NN and ε-NN on the benchmark of *CIFAR-60 K-512D*. Especially for the benchmark we used $k = 100$ and set ε to be the average distance of the 50th nearest neighbors, respectively; we have also tried different k and ε for k-NN and ε-NN, and observed similar results. Table 2 shows mAP values at 64, 128, and 256 bits for two different types of nearest neighbor search.

In both of k-NN and ε-NN our method shows better results, more than 100% improvement in some cases, over *SBQ* and *DBQ* under different hashing methods in most cases, especially for 64 or more bits code lengths. For example, our method achieved 42%, 24%, 65%, and 46% improvement over *DBQ*, when using *ITQ*, *LSH*, *SKLSH*, and *SH* with 256 bit length for k-NN, respectively. Also, compared to *SBQ*, our method achieved 139%, 40%, 74%, and 191% improvement, when using *ITQ*, *LSH*, *SKLSH*, and *SH* for the 256 bit length, respectively.

Even at a low bit codes (e.g., 32 bits) our method showed a bit lower or modest improvement in most cases. For example, at 64 bits our method showed improvement over *DBQ* in all the cases. However, our method showed a lower accuracy over *SBQ* only in a single case when combined with *LSH*. These results indicate that at low code lengths, more projections give higher discriminative power over allocating more bits to reduce the quantization error as we did in our method. On the contrary, as we use more bits to represent high dimensional data points, our method reduces the quantization errors and thus improves the overall discriminative power over using all the bits for having different projections.

While our method showed improved results across different hashing methods, it achieved the best results with *ITQ* across all the tested code lengths and different types of NN search. Since *ITQ* constructs hashing functions to minimize the quantization error, the goal of our method aligns best with *ITQ* and thus achieved the best results.

We also tested the methods with *GIST-1 M-960D*, which is a much bigger dataset and has higher dimensionality than *CIFAR-60 K-512D*. Nonetheless in this benchmark we observed similar results achieved with *CIFAR-60 K-512D* (see Table 3 and Fig. 4).

Figs. 5 and 6 show precision-recall curves for *ITQ* with different binary code encoding schemes and different code sizes on *CIFAR-60 K-512D*. The trends among *SBQ*, *DBQ*, and our method for *LSH*, *SKLSH*, and *SH* are similar to those for *ITQ*. We do not show the

Table 2
Results on *CIFAR-60 K-512D* dataset. The mAP of the best quantization strategy for each hashing method is shown in boldface.

# bits	64			128			256		
	SBQ	DBQ	OURS	SBQ	DBQ	OURS	SBQ	DBQ	OURS
100-NN mAP									
ITQ	0.1194	0.1516	**0.1826**	0.1639	0.2408	**0.3228**	0.2028	0.3411	**0.4677**
LSH	**0.0545**	0.0431	0.0452	0.1029	0.0945	**0.1156**	0.1680	0.1897	**0.2317**
SKLSH	0.0233	0.0231	**0.0336**	0.0503	0.0570	**0.0875**	0.1104	0.1159	**0.1833**
SH	0.0612	0.0898	**0.1380**	0.0804	0.1514	**0.2194**	0.0984	0.1961	**0.2387**
ε-NN mAP									
ITQ	0.2260	0.2668	**0.2921**	0.2738	0.3610	**0.4395**	0.3134	0.4597	**0.5831**
LSH	**0.1396**	0.1210	0.1285	0.2062	0.2050	**0.2281**	0.2771	0.3125	**0.3627**
SKLSH	0.0821	0.0797	**0.1073**	0.1366	0.1424	**0.1895**	0.2267	0.2351	**0.3176**
SH	0.1178	0.1566	**0.2165**	0.1454	0.2362	**0.3397**	0.1704	0.2925	**0.3890**

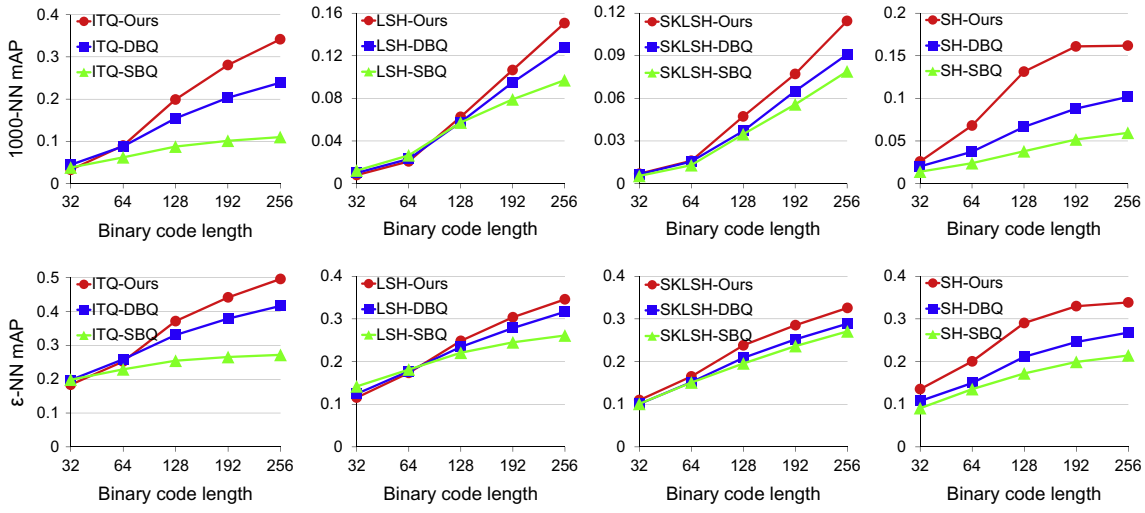

Fig. 4. Results on *GIST-1 M-960D* with k-NN (top) and ε-NN search (bottom). Our method improves accuracy over *SBQ* and *DBQ* with different hashing methods. Our method shows the best results when combined with *ITQ* [10].

Table 3
Results on *GIST-1 M-960D* dataset. The mAP of the best quantization strategy for each hashing method is shown in boldface.

# bits	64			128			256		
	SBQ	DBQ	OURS	SBQ	DBQ	OURS	SBQ	DBQ	OURS
1000-NN mAP									
ITQ	0.0620	0.0881	**0.0899**	0.0875	0.1545	**0.1989**	0.1101	0.2388	**0.3415**
LSH	**0.0264**	0.0230	0.0208	0.0573	0.0576	**0.0626**	0.0970	0.1279	**0.1506**
SKLSH	0.0129	0.0155	**0.0160**	0.0347	0.0370	**0.0473**	0.0790	0.0910	**0.1145**
SH	0.0239	0.0373	**0.0681**	0.0379	0.0665	**0.1313**	0.0595	0.1017	**0.1618**
ε-NN mAP									
ITQ	0.2289	**0.2585**	0.2520	0.2544	0.3304	**0.3713**	0.2715	0.4164	**0.4958**
LSH	**0.1812**	0.1774	0.1734	0.2206	0.2343	**0.2484**	0.2610	0.3167	**0.3458**
SKLSH	0.1508	0.1517	**0.1651**	0.1956	0.2088	**0.2380**	0.2706	0.2888	**0.3257**
SH	0.1353	0.1497	**0.2003**	0.1721	0.2112	**0.2903**	0.2143	0.2678	**0.3385**

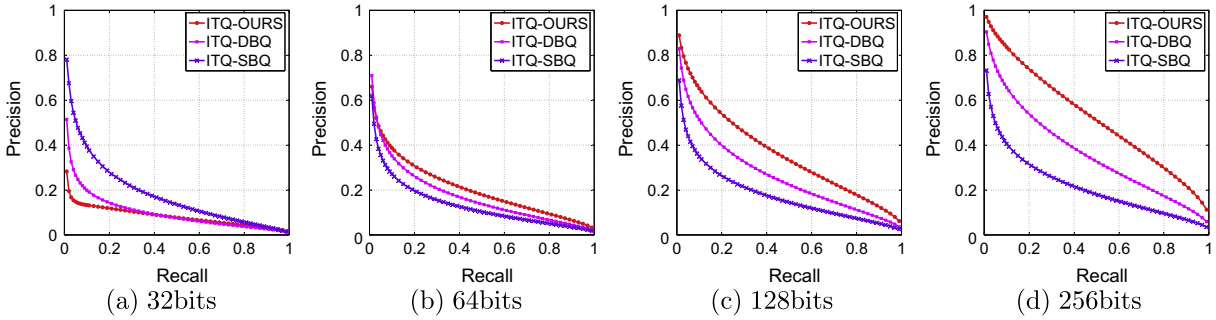

Fig. 5. Comparison between the state-of-the-art method and our method on *CIFAR-60 K-512D* dataset when $k = 100$. Refer to Fig. 3 for the mAP curves.

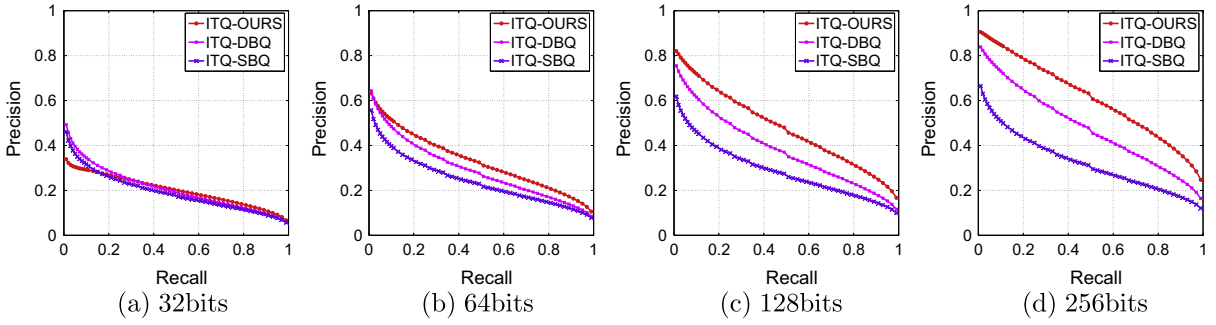

Fig. 6. Comparison between the state-of-the-art method and our method on *CIFAR-60 K-512D* dataset with ε-NN. Refer to Fig. 3 for the mAP curves.

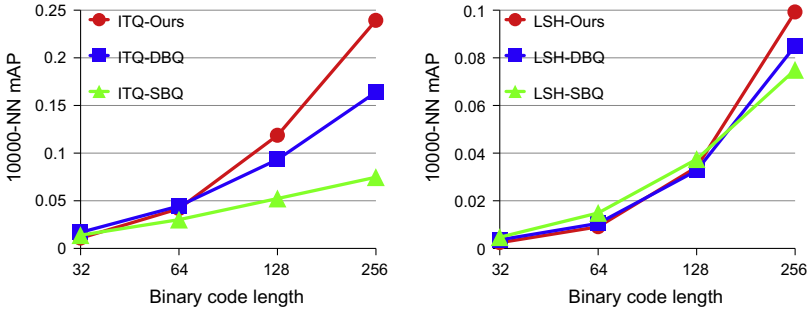

Fig. 7. Results on *GIST-75 M-384D* with *k*-NN.

curves on other binary hashings and benchmarks due to the space limitation.

Finally we tested *LSH* and *ITQ* in *GIST-75 M-384D*, which is one of the largest image benchmarks available to computer vision community. Overall results (Fig. 7) on this benchmark are similar to those achieved in other benchmarks. In data-independent method *LSH*, our method showed comparable performance with *SBQ* and *DBQ* up to 128 bits, while outperforming them afterwards. With data-dependent method *ITQ*, our method showed better performance from 128 bits.

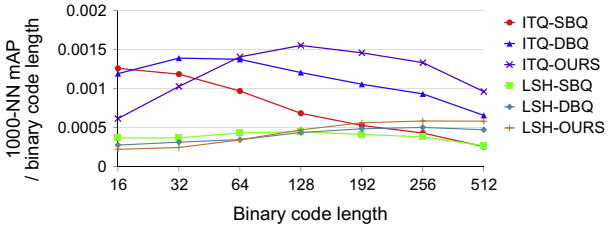

Fig. 8. This figure shows the improvement over diminishing efficiency of having more projections for hashing on *GIST-1 M-960D* dataset. *Y*-axis is the mAP (mean Average Precision) of nearest neighbor search divided by the number of projections. *DBQ* and *Ours* lessen decreasing efficiency and, furthermore, show increasing efficiency with LSH [4].

By using a small, but effective set of projections, our method slows down the diminishing marginal efficiency rate, as we use more bits for binary codes (see Figs. 2 and 8). At lower bits (e.g., 16 and 32 bits), our method shows lower accuracy due to the lack of discriminative power. However, as the number of bits increases, our method achieves significant improvement over *SBQ* and *DBQ*, and shows decreasing marginal efficiency from a far longer bit length compared to *SBQ* and *DBQ*.

4.6. Computational cost

Table 4 shows the computational cost of each component in binary code embedding schemes with 10 k training samples. All experiments are conducted on the machine with Xeon X5690 and 144 GB main memory to hold all the data in its main memory. Since our method utilizes only the half number of projections used in *SBQ*, the overall training time and cost for converting a point to a binary code is comparable to or much less than that of *SBQ*. We also measured distance computation time of our proposed distance function QED. One million distance computations of QED for 256 bit code lengths took 8.3 ms on average, while the Hamming distance took 7.4 ms. Our distance function significantly improved the overall accuracy of nearest neighbor search with comparable computational time to the Hamming distance. We believe that these computational overheads are small prices to pay for the significant performance improvements.

4.7. Discussions

Fig. 9 shows additional gains brought by different components of our method. *Ours w/ DBQ-Op.* uses our encoding scheme and distance function, but with the threshold learning of *DBQ* [14], while *Ours w/ Our-Op.* uses our optimization process mentioned in Section 3.4. By using our encoding and distance function over *DBQ* we achieved 29% improvement at 128 bit length. In addition we achieve 5% additional gain by using our optimization method.

Measuring the quantization error is not straightforward, since our proposed Quadra-Embedding scheme and Quadra-Embedding Distance (QED) are totally different from traditional approaches. For example, a measure used in *ITQ* [10] does not fit to our case, because QED between two different binary codes could be zero.

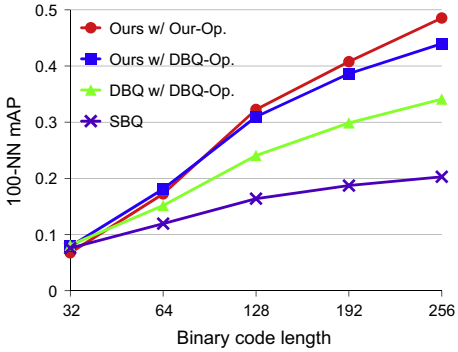

Fig. 9. This figure shows benefits of different components of our method on *CIFAR-60 K-512D* with *k*-NN protocol and projections computed by *ITQ*.

Instead of using prior measures, we measured how much our scheme reduces inter-quantization errors caused by mapping near-by points to far-away binary codes. To quantitatively measure it, we computed the average Hamming distance and QED between query points and their k-nearest neighbors (k-NNs). In case of *ITQ*/128 bits on *CIFAR-60 K-512D* dataset, the average Hamming distance to 100-NNs of *SBQ* and *DBQ* are 40.2 and 30.9 respectively, while the average QED of our scheme to 100-NNs is 7.6. This result confirms that our method encodes near points to closer binary codes better, compared to *SBQ* and *DBQ*.

Reducing the inter-quantization error, however, could degrade discriminative power. To see whether our scheme retains high discriminative power while reducing inter-quantization error, we also measured precisions within the average distances obtained in the above experiments. Precisions within these distances of *SBQ*, *DBQ*, and Ours are 0.14, 0.22 and 0.33, respectively. As a result, our method significantly reduces inter-quantization errors, while having stronger discriminative power than both *SBQ* and *DBQ*.

4.8. Comparison to product quantization

Product Quantization (PQ) [27], one of the state-of-the-art approximate nearest neighbor search methods, encodes a data point to the concatenation of cluster indices in subspaces. In [27], two different distance measures Symmetric Distance (PQ-SD) and Asymmetric Distance (PQ-AD) are proposed. PQ-SD estimates distance based on the distance between cluster centers, while PQ-AD is based on the distance between the query and the cluster center corresponding to the data point. Since distance computation of PQ requires floating point operations, not bit operations, PQ-SD and PQ-AD provide much inefficient distance computation performance compared to our method (see Table 5). With fast on-line process our method showed a higher accuracy over PQ-SD on most configurations and also showed comparable accuracy over PQ-AD on *CIFAR-60 K-512D* (see Table 6). As we use more bits for binary codes, the rate of the accuracy improvement of PQ is rather marginal compared to our method. We believe that the search accuracy difference compared to the PQ-AD is a small price to pay for the much efficient search performance. In addition, it would be possible to apply our method to PQ for efficient bit utilization and distance computation.

Table 4
Computational time of *ITQ*/256 bits on *GIST-1 M-960D*.

		SBQ	DBQ	OURS
Off-line	Training projections	40.14 s	9.47 s	9.47 s
	Training thresholds	N/A	1.56 s	41.63 s
On-line	Computing binary code	0.492 ms	0.200 ms	0.199 ms
	Computing 1 M distances	7.4 ms	7.4 ms	8.3 ms

Table 5
Computational time of *ITQ-OURS* and *PQ*/256 bits on *GIST-1 M-960D*.

		ITQ-OURS	PQ-SD	PQ-AD
Off-line	Training	51.10 s	89.271 s	89.271
On-line	Computing binary code	0.199 ms	0.201 ms	N/A
	Computing 1 M distances	8.3 ms	262.34 ms	1153.8 ms

Table 6
Comparisons with PQ on *CIFAR-60 K-512D* and *GIST-1 M-960D* datasets.

# bits	64	128	256
100-NN mAP on CIFAR-60 K-512D			
ITQ-OURS	0.1826	0.3228	0.4677
PQ-SD	0.1470	0.2437	0.3492
PQ-AD	0.2260	0.3527	0.4676
1000-NN mAP on GIST-1 M-960D			
ITQ-OURS	0.0899	0.1989	0.3415
PQ-SD	0.1030	0.1950	0.3292
PQ-AD	0.1520	0.2695	0.4154

5. Conclusion and future work

We have introduced a novel binary code embedding method, Quadra-Embedding technique for efficient and effective nearest neighbor search for image retrieval. In order to reduce the quantization error, our method uses two different code bits for each projection and a specialized distance function tailored to our encoding scheme. We have explained an optimization method that adjusts the margin of the buffer area for our method. We have tested our method with two different types of nearest neighbor search and a diverse set of hashing methods under three different image benchmarks. We have observed that our method achieves significant improvement over prior quantization strategies such as single-bit and double-bit quantizations [14] in most experimental configurations. These results have demonstrated the usefulness and robustness of our approach.

We have focused on lowering diminishing marginal efficiency of prior binary coding techniques. Therefore, our method cannot eliminate inborn diminishing efficiency of the original binary code embedding method, but makes the fixed-length binary codes more efficient by utilizing only a subset of highly informative hashing functions instead of using less informative hashing functions.

Many interesting future research directions lie ahead. Our method showed superior results over *SBQ* in long code lengths, especially with data-dependent methods. In case of data-dependent methods, our method showed comparable accuracy even at 32 bit code lengths and outperformed *SBQ* afterwards. However, our method showed a bit lower or comparable performance over *SBQ* with *LSH* at 32 or 64 bit code lengths, while outperforming *SBQ* afterwards. These results indicate that it is better to use a single bit for each projection for short code lengths, i.e. 32 bit code lengths. Also, our approach can be easily extended to allocate more than two bits per one hashing function. The most important question that needs to be addressed is how to determine an appropriate number of bits for each projection. It should consider variances of projected dimensions without assuming a uniform distribution, which makes all bits to carry effective information [29]. Furthermore, designing a distance metric for this encoding scheme would be very challenging and thus we may have to learn useful metric functions for maximizing the performance of this approach. Currently, we optimized our encoding scheme given a hashing method. As a next step, it would be very interesting to jointly optimize both encoding scheme and hashing functions. As demonstrated by the fact that our method achieved the best results with iterative quantization [10] that also aims to minimize the quantization error, this joint optimization can further improve the accuracy of nearest neighbor search.

Acknowledgment

We are thankful to anonymous reviewers and lab members for their valuable feedbacks. This work was supported in part by IT R&D program of MSIP/KEIT [10039165, Development of learner-participational and interactive 3D Virtual learning contents technology], NRF-2013R1A1A2058052, DAPA/ADD (UD110006MD), MEST/NRF (2013-067321), and IT R&D program of MOTIE/KEIT [10044970].

References

[1] D. Nistér, H. Stewénius, Scalable recognition with a vocabulary tree, in: CVPR, 2006.
[2] M. Muja, D.G. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, in: VISAPP, 2009, pp. 331–340.
[3] A. Torralba, R. Fergus, Y. Weiss, Small codes and large image databases for recognition, in: CVPR, 2008.
[4] M. Datar, N. Immorlica, P. Indyk, V.S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: SoCG, 2004.
[5] Y. Weiss, A. Torralba, R. Fergus, Spectral hashing, in: NIPS, 2008.
[6] B. Kulis, K. Grauman, Kernelized locality-sensitive hashing for scalable image search, in: ICCV, 2009.
[7] M. Raginsky, S. Lazebnik, Locality sensitive binary codes from shift-invariant kernels, in: NIPS, 2009.
[8] J. Wang, S. Kumar, S.-F. Chang, Sequential projection learning for hashing with compact codes, in: ICML, 2010.
[9] J. He, R. Radhakrishnan, S.-F. Chang, C. Bauer, Compact hashing with joint optimization of search accuracy and time, in: CVPR, 2011.
[10] Y. Gong, S. Lazebnik, Iterative quantization: a procrustean approach to learning binary codes, in: CVPR, 2011.
[11] W. Liu, J. Wang, S. Kumar, S.-F. Chang, Hashing with graphs, in: ICML, 2011.
[12] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, S.-E. Yoon, Spherical hashing, in: CVPR, 2012.
[13] P. Indyk, R. Motwani, Approximate nearest neighbors: toward removing the curse of dimensionality, in: STOC, 1998.
[14] W. Kong, W.-J. Li, Double-bit quantization for hashing, in: AAAI, 2012.
[15] Y. Lee, J.-P. Heo, S.-E. Yoon, Quadra-embedding: binary code embedding with low quantization error, in: ACCV, 2012.
[16] R. Datta, D. Joshi, J. Li, J.Z. Wang, Image retrieval: ideas, influences, and trends of the new age, ACM Comput. Survey 40 (2) (2008) 1–60.
[17] J. Sivic, A. Zisserman, Video google: a text retrieval approach to object matching in videos, in: ICCV, 2003.
[18] A. Oliva, A. Torralba, Modeling the shape of the scene: a holistic representation of the spatial envelope, IJCV.
[19] J.H. Friedman, J.L. Bentley, R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM TOMS 3 (3) (1977) 209–226.
[20] K. Kim, M.K. Hasan, J.-P. Heo, Y.-W. Tai, S.-E. Yoon, Probabilistic cost model for nearest neighbor search in image retrieval, Computer Vision and Image Understanding (CVIU).
[21] O. Chum, J. Philbin, A. Zisserman, Near duplicate image detection: min-hash and TF-IDF weighting, in: BMVC, 2008.
[22] P. Jain, B. Kulis, K. Grauman, Fast image search for learned metrics, in: CVPR, 2008.
[23] B. Kulis, T. Darrell, Learning to hash with binary reconstructive embeddings, in: NIPS, 2009.
[24] M. Norouzi, D.J. Fleet, Minimal loss hashing for compact binary codes, in: ICML, 2011.
[25] A. Joly, O. Buisson, Random maximum margin hashing, in: CVPR, 2011.
[26] A. Krizhevsky, Learning multiple layers of features from tiny images, Tech. rep., University of Toronto, 2009.
[27] H. Jégou, M. Douze, C. Schmid, Product quantization for nearest neighbor search, IEEE TPAMI.
[28] A. Gordo, F. Perronnin, Asymmetric distances for binary embeddings, in: CVPR, 2011.
[29] W. Kong, W.-J. Li, Isotropic hashing, in: NIPS, 2012.