Adaptive Binary Quantization for Fast Nearest Neighbor Search

Zhujin Li[1] and **Xianglong Liu**[*2] and **Junjie Wu**[3] and **Hao Su**[4]

Abstract. Hashing has been proved an attractive technique for fast nearest neighbor search over big data. Compared to the projection based hashing methods, prototype based ones own stronger capability of generating discriminative binary codes for the data with complex inherent structure. However, our observation indicates that they still suffer from the insufficient coding that usually utilizes the complete binary codes in a hypercube. To address this problem, we propose an adaptive binary quantization method that learns a discriminative hash function with prototypes correspondingly associated with small unique binary codes. Our alternating optimization adaptively discovers the prototype set and the code set of a varying size in an efficient way, which together robustly approximate the data relations. Our method can be naturally generalized to the product space for long hash codes. We believe that our idea serves as a very helpful insight to hashing research. The extensive experiments on four large-scale (up to 80 million) datasets demonstrate that our method significantly outperforms state-of-the-art hashing methods, with up to 58.84% performance gains relatively.

1 Introduction

In the past decade, hashing technique has been widely studied for fast nearest neighbor search, owing to its successful applications in many areas like large-scale visual search [4, 23, 33, 25, 30], machine learning [9, 19, 26, 21], recommendation system [22], etc. As the most essential concept in hashing based nearest neighbor search, Locality-Sensitive Hashing (LSH) was first introduced in [8], which guarantees that the nearest neighbors share the similar binary codes, and thus enables fast search with compressed storage over gigantic databases.

The pioneer LSH research adopted a random projection paradigm for the metrics like l_p-norm ($p \in (0, 2]$) [1]. Due to its simple form and efficient computation, *projection based hashing* has become the most widely accepted hashing paradigm, where the data point is first projected along certain direction and further quantized to a binary value. Since the randomly generated projection vectors are independent from the data, they usually suffer from the heavy redundancy and the lack of discriminative power for nearest neighbors. To leverage the information contained in the data, recent studies attempted to learn the projection based hash function, which have shown the success in the generation of discriminative hash codes [31, 3, 13, 12, 27, 16, 33, 7, 22, 17, 34, 32, 11].

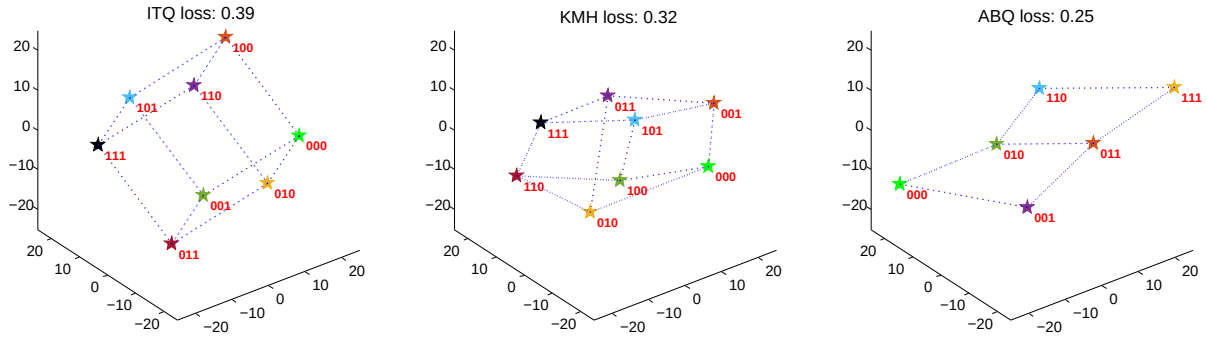
[1] State Key Lab of Software Development Environment, Beihang University, China, email: lizhujin@outlook.com
[2] State Key Lab of Software Development Environment, Beihang University, China, email: xlliu@nlsde.buaa.edu.cn, *corresponding author
[3] Beihang University, China, email: wujj@buaa.edu.cn
[4] Stanford University, USA, email: haosu@stanford.edu

Despite the progress in the projection based hashing research, state-of-the-art methods still cannot well approximate the nearest neighbor relations using their binary codes. This is mainly because that the linear form is somewhat beyond the strong capability of capturing the data characteristics with complex inherent structures [5, 24]. Although the nonlinear mapping techniques, like kernel which uplifts the data into an informative space, have been widely used to alleviate the problem [28, 15, 14, 18], they are a little time-consuming on the one hand, and still hard to exploit the underlying data structures using the binary quantization on the other hand.

In the literature, clustering has been proved a powerful quantization method to well model the complex relationships among data using a number of prototypes. This inspires the recent hashing studies that attempt to exploit the clustering structure among data in the binary quantization. Typical methods include spherical hashing (SPH) [6] and K-means hashing (KMH) [5], both of which explicitly pursued a number of prototypes to approximate the data relations, and adopted different coding schemes to quantize the data samples based on these prototypes. Different from projection based hashing where each hash function is parameterized by the projection vectors, these *prototype based hashing* methods define the hash functions based on the discovered prototypes, and promisingly increase the search performance with much less quantization loss.

Existing prototype based hashing methods like KMH make use of the complete binary code set, which geometrically forms a hypercube with a fixed dimension and structure among the codes (or the vertices). In practice, the real-world data usually distribute with a complex structure, which can be hardly characterized by such a hypercube. A demonstrative case on a subset of SIFT-1M dataset is presented in Figure 1, where KMH as the typical prototype based hashing outperforms the state-of-the-art projection based ITQ (see Figure 1 (a) and (b)), however using 3-bit codes the Hamming distances among the prototypes (marked by stars with different colors) cannot well approximate the original ones (the hypercube is skewed in Figure 1(b)).

In fact, a better coding solution only relying on a small subset of binary codes (instead of the complete set) can largely educe the quantization loss (see Figure 1 (c)). This is because the incomplete coding can match with the data distribution in a more reasonable way, and thus better approximate neighbor relations. Motivated by this observation, this paper proposes an adaptive binary quantization (ABQ) method that can pursue a discriminative hash function with varying number of prototypes, each of which is associated with a unique and compact binary code. The prototype set and codes are jointly discovered to respectively characterize the data distribution in original space and align the code space to the prototype distribution. Therefore, the learnt prototype based hash function can promise



(a) ITQ (b) KMH (c) ABQ (ours)

Figure 1. The geometric view of the binary quantization using different methods on a subset of SIFT-1M (projected into 3-dimensional space using PCA like ITQ method). The quantization loss is computed according to (3).

discriminative binary codes that can largely approximate the neighbor structures. We further apply product quantization to generalizing our method for long hash codes. Experimental results over four large-scale datasets demonstrate that the proposed method significantly outperforms state-of-the-art hashing methods.

2 Prototype Based Binary Quantization

Along the direction of prototype based hashing, this section will present our proposed adaptive binary quantization method (denoted as ABQ hereafter) in details.

2.1 Hash Function with Prototypes

Supposing we have a set of n training samples, we denote $\mathbf{x}_i \in \mathbb{R}^{d\times 1}$, $i = 1 \dots n$ to be the feature vector of i-th sample, where d is the feature dimension. Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ be the data matrix. Our basic idea is to learn an informative binary hash function that encodes the training data $\mathbf{X}$ into b-length hash codes $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n] \in \{-1, 1\}^{b\times n}$, which show sensitivity to neighbor structures of the data.

The literature has proved that as representative samples among the large-scale database, the prototypes show robustness to the more general metric structure for the data in high dimensional space. Subsequently, in order to capture the neighbor structure using the prototype based hashing, the hash codes should preserve the relations among the prototypes. To meet this goal, one simple, yet powerful way is to assign each prototype a unique binary code in certain order.

In particular, a set of prototypes $\mathcal{P} = \{\mathbf{p}_k | \mathbf{p}_k \in \mathbb{R}^d\}$ are learnt from the training data, and each prototype is associated with a b bit binary code $\mathbf{c}_k \in \{-1, +1\}^b$, forming a binary codebook $\mathcal{C}$. Then for any data point $\mathbf{x}$, it can be represented by its nearest prototype $\mathbf{p}_{i^*(\mathbf{x})}$ according to the specific distance function $d_o(\cdot, \cdot)$, where

$$i^*(\mathbf{x}) = \arg\min_k d_o(\mathbf{x}, \mathbf{p}_k), \tag{1}$$

and encoded by the code $\mathbf{c}_{i^*(\mathbf{x})}$ associated with $\mathbf{p}_{i^*(\mathbf{x})}$.

Subsequently, we can define the hash function $h(\mathbf{x})$ as

$$h(\mathbf{x}) = \mathbf{c}_{i^*(\mathbf{x})}. \tag{2}$$

Most of existing hashing methods attempt to pursue a series of hash functions, each of which generates a hash bit, forming a long hash code. Therefore, these methods have to append additional constraints to reduce the redundancy among these individual bits, which usually degenerates the performance with unreasonable assumptions. Our prototype based hashing like the most related work k-means hashing [5] can exploit the complex data structure and jointly generate a number of hash bits at the same time.

Ideally, the small set of representative prototypes can reduce the computation and introduce sparsity without using the full dataset in binary quantization step. Meanwhile, they can capture the discriminative essence of the dataset with the sensitivity to metric structure and the robustness to overfitting. Therefore, choosing the positioning of the prototypes wisely can lead to a drastically reduced effort while maintaining the discriminative power of the original dataset.

2.2 Space Alignment

The binary codes encoding the data are constrained in the vertices of a hypercube with constant affinities between them. However, in practice it rarely happens that the data geometrically distribute in such a perfect structure. Therefore, an optimal binary coding strategy is highly required to jointly find the discriminative prototypes and their associated binary codes, which respectively characterize the inherent data relations, and maintain the affinities between samples in Hamming space.

Intuitively, the prototype based hash function h should approximate the relations between any two samples $\mathbf{x}_i$ and $\mathbf{x}_j$ using their binary codes. A straightforward way is to concentrate on the distance consistence so that codes in Hamming space will be aligned with the original data distribution. Formally, we introduce the quantization loss to measure the space alignment:

$$\mathcal{Q}(\mathbf{Y}, \mathbf{X}) = \frac{1}{n^2} \sum_{i,j=1}^{n} \|\lambda d_o(\mathbf{x}_i, \mathbf{x}_j) - d_h(\mathbf{y}_i, \mathbf{y}_j)\|^2 \tag{3}$$

where $d_h(\mathbf{y}_i, \mathbf{y}_j) = \frac{1}{2}\|\mathbf{y}_i - \mathbf{y}_j\|$ is the square root of the Hamming distance between $\mathbf{y}_i = h(\mathbf{x}_i)$ and $\mathbf{y}_j = h(\mathbf{x}_j)$, and λ is a constant scale parameter for the space alignment.

The above loss function involves n^2 sample pairs, which prevent the efficient learning over a large training set. As we mentioned above, the prototypes, as promising representatives of the whole data, have been proved to be able to substantially reduce the computation in many applications. Therefore, for any $\mathbf{x}_i$ the distance from

another sample $\mathbf{x}_j$ can be approximated as follows:

$$d_o(\mathbf{x}_i, \mathbf{x}_j) \approx d_o(\mathbf{x}_i, \mathbf{p}_{i^*(\mathbf{x}_j)}). \tag{4}$$

Motivated by the fact that the hash code of each sample $\mathbf{x}_i$ is actually equivalent to that of its nearest prototypes, namely, $\mathbf{y}_i = \mathbf{c}_{i^*(\mathbf{x}_i)}$, the above loss function can be rewritten in a more simple and efficient way with respect to the prototypes $\mathcal{P}$ and their binary codes $\mathcal{C}$

$$\mathcal{Q}(\mathcal{P}, \mathcal{C}, i^*(\mathbf{X})) = \sum_{i=1}^{n} \sum_{k=1}^{|\mathcal{P}|} \frac{w_k}{n^2} \|\lambda d_o(\mathbf{x}_i, \mathbf{p}_k) - d_h(\mathbf{c}_{i^*(\mathbf{x}_i)}, \mathbf{c}_k)\|^2,$$

where w_k is the number of samples represented by $\mathbf{p}_k$.

Note that the above approximation actually corresponds to the widely-used asymmetric distance, where the database samples are substituted by their prototypes. The literature has shown that such asymmetric approximation usually owns great power to alleviate the quantization loss. Minimizing the above loss leads to a set of prototypes that well capture the intrinsic neighbor structure among the data, and thus a discriminative coding solution that consistently preserves the original relations in Hamming space. Besides, the above loss actually enforces that the close samples in the original space can be clustered in the same group represented by one prototype, and meanwhile their hash codes also maintain the distribution in Hamming space, which together align the neighbor structures between the two spaces.

Therefore, we can formulate the hashing problem in terms of the space alignment as follows:

$$\begin{aligned} \min_{\mathcal{P}, \mathcal{C}, i^*(\mathbf{X})} \quad & \mathcal{Q}(\mathcal{P}, \mathcal{C}, i^*(\mathbf{X})) \\ \text{s.t.} \quad & \mathbf{c}_k \in \{-1, 1\}^b; \quad \mathbf{c}_k^T \mathbf{c}_l \neq b,\ l \neq k. \end{aligned} \tag{5}$$

Here, the constraints on the binary codebook $\mathcal{C}$ will guarantee that each prototype will be assigned a unique binary code.

It should be pointed out that here the number of prototypes or the size of the codebook isn't fixed beforehand, which is quite different from prior hashing research like [6, 5] where all possible binary codes (*i.e.*, 2^b using b bits) are assumed to be used in the binary quantization. Indeed, we adaptively decide the number in our optimization according to the data metric structure. To some extent, this strategy will avoid the rigorous and difficult alignment between the prototypes and the hypercube binary codes, and thus faithfully helps discover more consistent and discriminative prototypes and the corresponding codebook.

By solving the above problem, the prototype set $\mathcal{P}$ can be obtained that captures the overall data distribution, which can also be reflected by the codebook $\mathcal{C}$. Each prototype will be associated with a distinct binary code, which together serve as a hash function that encodes those points belonging to the prototype using the corresponding binary code. For a novel sample $\mathbf{x}$, its hash bits can be computed fast by first determining its nearest prototype according to (1), and then assigning the binary code according to (2).

3 Alternating Optimization

To solve the above problem with respect to a small b, we present an alternating optimization solution, which pursues the near-optimal prototypes and adaptively determine the corresponding binary codes in an efficient way. For the efficiency, usually we choose a small b (*e.g.*, $b \leq 8$), and later we will discuss how to obtain a much longer hash code.

3.1 Adaptive Coding

Supposing we have the prototypes and the assignment index for each sample (see the initialization in Section 3.4), the problem turns to the discriminative binary coding that consistently keeps the distribution information of the samples in the original space. Although k-means hashing [5] as the most related work can capture the cluster structure and find an encouraging binary coding solution, its discriminative power is still limited due to the concentration on the full hypercube structure, which is beyond the true data distribution in practice.

Quite different from the previous research, we adopt an adaptive coding that directly finds the binary codes most consistent with the prototypes. Given the prototypes $\mathcal{P}$ and the assignment of each sample, we will sequentially find a locally optimal binary code for each prototype in a greedy way. Specifically, supposing the prototypes $\mathbf{p}_1, \ldots, \mathbf{p}_l$ ($1 \leq l \leq |\mathcal{P}|$) have been respectively assigned the binary codes $\mathbf{c}_1, \ldots, \mathbf{c}_l$, we next select the optimal code $\mathbf{c}_k$ for prototype $\mathbf{p}_k$ from the set $\tilde{\mathcal{C}} = \{-1, 1\}^b - \{\mathbf{c}_1, \ldots, \mathbf{c}_l\}$ of remaining hash codes. Then for $\mathbf{c}_k$, the objective function in (5) turns to

$$\begin{aligned} \min_{\mathbf{c}_k \in \tilde{\mathcal{C}}} & \sum_{i^*(\mathbf{x}_i)=k} \sum_{k' \neq k} w_{k'} \|\lambda d_o(\mathbf{x}_i, \mathbf{p}_{k'}) - d_h(\mathbf{c}_k, \mathbf{c}_{k'})\|^2 \\ & + \sum_{i^*(\mathbf{x}_i) \neq k} w_k \|\lambda d_o(\mathbf{x}_i, \mathbf{p}_k) - d_h(\mathbf{c}_{i^*(\mathbf{x})}, \mathbf{c}_k)\|^2. \end{aligned} \tag{6}$$

Since the code space is quite limited ($|\tilde{\mathcal{C}}| \leq 2^b$), the above optimal code pursuit can be completed efficiently using exhaustive search over $\tilde{\mathcal{C}}$.

As to the first step, we can simply choose any binary code as the optimal $\mathbf{c}_1$. This is because the code space is highly symmetric with a hypercube structure. After repeating $|\mathcal{P}|$ steps, we can assign each prototype a unique hash code with the minimal coding loss, and meanwhile greedily keep the original data distribution information.

3.2 Prototype Update

While the binary codebook $\mathcal{C}$ is discovered, the prototypes $\mathcal{P}$ should be further calibrated to simultaneously capture the data distribution and align it to the geometric structure in the code space. Therefore, the above problem turns to:

$$\min_{\mathcal{P}} \sum_{i=1}^{n} \sum_{k=1}^{|\mathcal{C}|} w_k \|\lambda d_o(\mathbf{x}_i, \mathbf{p}_k) - d_h(\mathbf{c}_{i^*(\mathbf{x}_i)}, \mathbf{c}_k)\|^2. \tag{7}$$

To pursue a set of prototypes that well represent the data, we adopt a two-step optimization, since the prototype discovery involves the assignment variable $i^*(\mathbf{x}_i)$. We first determine which prototype the samples belong to, and then update the position of each prototype based on the assignment.

Deriving from (7), the prototype that yields the least loss for each sample $\mathbf{x}_i$ can be found using a simple search:

$$\min_{k' \leq |\mathcal{C}|} \sum_{k=1}^{|\mathcal{C}|} w_k \|\lambda d_o(\mathbf{x}_i, \mathbf{p}_k) - d_h(\mathbf{c}_{k'}, \mathbf{c}_k)\|^2. \tag{8}$$

With the assignment of each sample, we approximately recalculate the position of each prototype:

$$\mathbf{p}_k = \frac{1}{w_k} \sum_{i^*(\mathbf{x}_i)=k} \mathbf{x}_i, \quad 1 \leq k \leq |\mathcal{C}|. \tag{9}$$

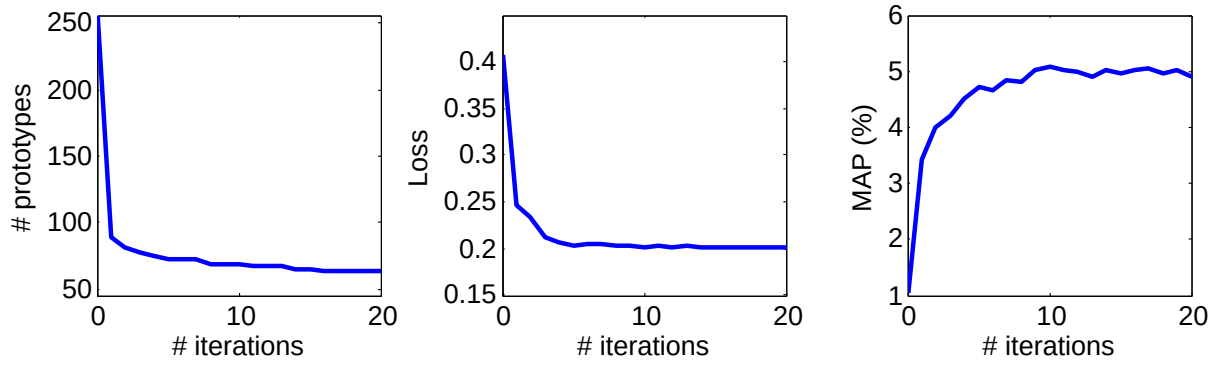


Figure 2. Demonstration of the adaptive binary quantization in one subspace ($b = 8$) on GIST-1M using 32 bits.

Algorithm 1 Adaptive Binary Quantization.

Input: Training data $\mathbf{X}$, and the binary code length b.
Output: Hash function h, the prototype set $\mathcal{P}$ and the corresponding binary code set $\mathcal{C}$.

```
Initialize the assignment index i*(X) and the prototype set P using k-means.
Initialize the scale parameter λ according to (11).
repeat
    for l = 1, ..., |P| do
        Find the local optimal code c_l for p_l by solving (6);
    end for
    Update the prototype set P according to (8) and (9);
    Update the distribution i*(X) according to (10);
until convergence
```

In this step the number of the prototypes varies, *i.e.*, $\mathcal{P}$ is shrunk, where the uninformative prototypes are eliminated. This is the most different part from the previous research. Subsequently, the prototype set can gradually adapt the binary codes to the data distribution in the alternating optimization.

Figure 2 demonstrates how the performance benefits from the adaptive prototype set, where as the number of prototypes decreases, only a subset of the binary codes in the hypercube are utilized to maximally capture the neighbor relations among data (also see Figure 1(c)), significantly reducing the quantization loss and meanwhile improving the precision of nearest neighbor search.

3.3 Distribution Update

After the prototype set $\mathcal{P}$ is updated, the binary codebook size in the next alternating round will be also determined. Moreover, the data distribution with respect to $\mathcal{P}$, characterized by the variable $i^*(\mathbf{X})$, will change slightly. Since the binary coding should maximally preserve the data distribution, we further append an assignment updating step to capture the distribution variation. This can be easily done by employing a similar step in k-means:

$$i^*(\mathbf{x}_i) = \arg\min_{k \le |\mathcal{P}|} d_o(\mathbf{x}_i, \mathbf{p}_k). \tag{10}$$

This is consistent with the hash function definition in (1), guaranteeing that the hash function can discriminatively preserve the intrinsic data relations based on the prototypes.

3.4 Algorithm Details

Algorithm 1 lists the main steps of our adaptive binary quantization, where some algorithm details are discussed as follows.

3.4.1 *Initialization*

To start the alternating optimization, we should first initialize the indices $i^*(\mathbf{X})$ and the prototype set $\mathcal{P}$. In practice this can be completed by first fixing the size of $\mathcal{P}$ to 2^b, and then performing the classical k-means algorithm on the training data $\mathbf{X}$, where the cluster centers are treated as the prototypes $\mathcal{P}$, and each sample is assigned to its nearest prototype.

Here we simply adopt k-means clustering to initialize the prototypes. Although the quality of the prototypes depends on the clustering algorithm or seed selection in the k-means initialization phase, we found that they do not affect the overall performance much. This is mainly because the positions and the quantity of the prototypes will be refined gradually in the iterative optimization to align the data distribution to the code space, and even with a coarse initialization, one can still obtain the identical informative prototypes in a number of iterations. Besides, since it has minor effects on the performance according to our empirical results, we randomly select the order in which the prototypes are processed in the adaptive coding step, *i.e.*, Equation (6).

As to the scale parameter λ in Equation (3), it is intuitively adopted to make the distances comparable between the original and Hamming space. Since we found it usually insensitive to the binary coding process, we simply set it to a constant based on the initialization using k-means, assuming that all 2^b prototypes are assigned different binary codes:

$$\lambda = \frac{\frac{1}{2^b} \sum_{\mathbf{c}_k, \mathbf{c}_l \in \{-1,1\}^b} d_h(\mathbf{c}_k, \mathbf{c}_l)}{\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{2^b} d_o(\mathbf{x}_i, \mathbf{p}_k)}. \tag{11}$$

3.4.2 *Product Quantization*

For a desired level of performance, usually a long hash code is required in many practical applications. However, for the representation power and the computational efficiency, the prototype number usually ranges from tens to hundreds at most, which makes the above algorithm only generate small codes with $b \le 8$. Fortunately, our problem can be naturally generalized to product space for longer hash codes, following the idea of product quantization (PQ) [10, 5]. In order to generate a sufficient long code of $b^* \gg b$ length, the PQ

method divide the original space into $M = b^*/b$ subspaces, in which a small code of $b = b^*/M$ length is respectively associated with each sample and concatenated as a long one in a Cartesian product manner.

Specifically, a vector $\mathbf{x}$ is represented as M sub-vectors in the way $\mathbf{x} = [\hat{\mathbf{x}}^{(1)}, \hat{\mathbf{x}}^{(2)}, \ldots, \hat{\mathbf{x}}^{(M)}]^T$, where $\hat{\mathbf{x}}^{(m)} \in \mathbb{R}^{d\times 1}$ is the m-th sub-vector of $\mathbf{x}$, and its hash code $\hat{\mathbf{y}}^{(m)} \in \{-1, 1\}^{b\times 1}$ can be generated using the proposed adaptive quantization based on the sub-prototypes $\hat{\mathbf{p}}^{(m)} \in \mathbb{R}^{d\times 1}$ and the sub-codebook $\hat{\mathbf{c}}^{(m)} \in \{-1, 1\}^{b\times 1}$. The hash code $\mathbf{y}$ for vector $\mathbf{x}$ is the concatenation of the sub-codes of its sub-vectors: $\mathbf{y} = [\hat{\mathbf{y}}^{(1)}, \hat{\mathbf{y}}^{(2)}, \ldots, \hat{\mathbf{y}}^{(M)}]$.

Recall that Equation (4) corresponds to the asymmetric distance computation (ADC) in PQ. If the original distance d_o is defined as Euclidean distance, PQ can approximate the distance between two vectors using codewords (prototypes):

$$\begin{aligned} d_o(\mathbf{x}_i, \mathbf{x}_j) &\approx d_o(\mathbf{x}_i, \mathbf{p}_{i^*(\mathbf{x}_j)}) \\ &= \sqrt{\sum_{m=1}^{M} d_o(\hat{\mathbf{x}}_i^{(m)}, \hat{\mathbf{p}}^{(m)}_{i^*(\hat{\mathbf{x}}_j^{(m)})})^2} \end{aligned} \tag{12}$$

In each subspace, the learnt codes can approximate the original distance d_o well using the Hamming based distance d_h, *i.e.*, $\lambda d_o(\hat{\mathbf{x}}_i^{(m)}, \hat{\mathbf{p}}_k^{(m)}) \approx d_h(\hat{\mathbf{y}}_i^{(m)}, \hat{\mathbf{c}}_k^{(m)})$. Then, with the definition of the distance d_h, we have:

$$\begin{aligned} \lambda d_o(\mathbf{x}_i, \mathbf{p}_k) &\approx \sqrt{\sum_{m=1}^{M} \frac{1}{4}\|\hat{\mathbf{y}}_i^{(m)} - \hat{\mathbf{c}}_k^{(m)}\|^2} \\ &= \frac{1}{2}\|\mathbf{y}_i - \mathbf{c}_k\| = d_h(\mathbf{c}_{i^*(\mathbf{x}_i)}, \mathbf{c}_k) \end{aligned} \tag{13}$$

Putting (12) and (13) together, it is easy to show that the original distance between any two samples can be approximated by the Hamming based distance between their hash codes in the Cartesian space:

$$\lambda d_o(\mathbf{x}_i, \mathbf{x}_j) \approx d_h(\mathbf{c}_{i^*(\mathbf{x}_i)}, \mathbf{c}_{i^*(\mathbf{x}_j)}) \tag{14}$$

Note that the above approximation requires that the scale parameter λ remains the same across all subspaces, which holds roughly in practice over many datasets. Therefore, we set it to the average of the values computed according to Equation (11) in all subspaces.

Prior research has pointed out that equally splitting the space into M parts might result in ineffective hash codes, due to the unbalanced information distribution [5]. Usually independent subspaces are pursued to balance the information among the small codes of each sample. Therefore, in the space decomposition, we apply the eigenvalue allocation method to evenly distribute the variance using PCA projection without dimension reduction [2]. One can also further append an adaptive bit allocation to maximally capture the data information using different number (or code length) of hash bits [20].

3.4.3 Complexity

To learn the hash functions that can generate binary codes of b^* length, we need to compute the small codes in $M = b^*/b$ independent subspaces. For each subspace, there are maximally 2^b prototypes in d/M feature space.

At the training stage, the adaptive coding greedily finds the locally optimal code for each prototype over $\{-1, 1\}^b$ in $O(nd2^{2b})$ time. The prototype and distribution update steps require at most $O(nd2^{2b})$ time to compute the distances between training samples and prototypes. Therefore, when using t (usually $t \leq 20$) iterations in the alternating optimization, totally $O(2^{2b}ndt)$ time is spent on the training. Since the code space is quite limited for each subspace ($b \leq 8$), the term 2^{2b} can be treated as a constant. Therefore, it can be considered that the training time scales linearly with respect to the size of the training set.

When it comes to the online search, for each query point the hash function needs $O(2^b d)$ time to compute the nearest prototype and $O(1)$ time for the code assignment, which is linear to the feature dimension d as most projection based hashing methods like LSH [1] and ITQ [3]. Furthermore, our method only utilizes a small subset (*e.g.*, a quarter) of codes, which directly reduce the time consumption at the stage of hash code generation.

Compared with other lookup based methods like KMH, our method usually owns faster speed when performing online search in practice (see Table 1). The high efficiency of our method mainly benefits from the adaptive coding combined with PQ, which allows to compute and store only a small number of codewords (prototypes), while presenting a large dictionary to maximally preserve the information of data distribution in the original space.

4 Experiments

In this section we will evaluate the proposed adaptive binary quantization (ABQ) on large-scale nearest neighbor search, and compare it with several state-of-the-art hashing algorithms, including the classical projection based ones like Locality Sensitive Hashing (LSH) [1], Spectral Hashing (SH) [31], Kernelized Locality Sensitive Hashing (KLSH) [14], Anchor Graph Hashing (AGH) [18], Iterative Quantization (ITQ) [3] and Kronecker Binary Embedding (KBE) [35], and two representative prototype based ones: Spherical Hashing (SPH) [6] and K-Means Hashing (KMH) [5].

- **LSH**: LSH generates Gaussian random projection vectors and preserves the locality with high probability.
- **SH**: SH formulates the binary coding problem as a spectral embedding in the Hamming space, and generalizes the approximated solution for out-of-sample extension.
- **KLSH**: KLSH constructs randomized locality-sensitive functions with arbitrary kernel functions. We feed it the Gaussian RBF kernel $\|(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\alpha\|\mathbf{x}_i - \mathbf{x}_j\|^2)$ and 300 support samples. The kernel parameter α is tuned to an appropriate value on each dataset.
- **AGH**: AGH approximates the intrinsic structure underlying the data based on anchors, and generates hash codes based on the anchor representation.
- **ITQ**: ITQ iteratively finds the data rotation in a subspace to minimize the binary quantization error.
- **KBE**: KBE generates linear hash functions with a structured matrix, which can achieve fast hash coding over high-dimensional data. We adopt the optimized version of Kronecker projection.
- **SPH**: SPH iteratively adjusts the spherical planes to generate independent and balanced partitions, which serve as the nonlinear hash functions based on the distances to the centers. In SPH, each partition can generate a hash bit independently.
- **KMH**: KMH generates affinity affine clusters using k-means in the partitioned subspaces of the training features, and maps each cluster to a binary hash code for the out-of-sample coding.

Table 1. Hashing performance and time efficiency on SIFT-1M and GIST-1M.

		MAP			PH (32 BITS)		TIME (128 BITS)	
		32 BITS	64 BITS	128 BITS	$r = 1$	$r = 2$	TRAIN (S)	SEARCH (S)
SIFT-1M	LSH	5.43±0.30	13.00±0.82	26.04±0.68	18.89	19.70	0.03	0.02
	SH	10.70±0.58	17.84±0.37	25.30±0.59	32.20	41.93	0.25	0.25
	KLSH	7.08±0.44	15.61±0.57	29.48±0.72	23.72	23.32	0.28	0.02
	AGH	6.26±0.27	9.11±0.31	11.10±0.23	15.90	11.93	0.55	0.04
	ITQ	9.70±0.14	20.14±0.47	33.23±0.49	28.38	22.09	5.08	0.16
	SPH	8.57±0.12	18.23±0.54	31.11±0.14	26.90	30.82	8.93	0.04
	KMH	11.51±0.27	22.50±0.31	32.06±0.52	35.63	40.00	680.64	0.12
	KBE	6.43±0.31	14.73±0.61	27.65±0.57	20.62	16.97	3.28	0.02
	ABQ	**12.47**±0.26	**24.92**±0.61	**41.34**±0.56	**41.30**	**43.09**	40.37	0.06
GIST-1M	LSH	1.34±0.08	3.15±0.07	5.97±0.19	5.41	7.15	0.21	0.05
	SH	1.90±0.23	3.19±0.19	4.92±0.19	8.94	6.58	1.70	0.24
	KLSH	2.41±0.09	5.23±0.18	9.76±0.23	9.31	10.70	0.44	0.05
	AGH	2.09±0.15	3.05±0.10	3.98±0.14	5.55	4.13	0.90	0.09
	ITQ	4.43±0.06	6.93±0.10	9.49±0.15	14.08	17.8	5.87	0.17
	SPH	3.65±0.14	6.97±0.10	11.52±0.19	12.20	17.05	25.24	0.07
	KMH	3.58±0.18	5.57±0.07	6.92±0.07	14.77	17.39	2380.61	0.15
	KBE	-	-	6.58±0.22	-	-	13.66	0.06
	ABQ	**4.92**±0.06	**10.06**±0.20	**16.10**±0.17	**23.46**	**17.84**	46.10	0.10

4.1 Evaluation Protocols

To comprehensively evaluate the proposed method, we first employ two well-known large-scale datasets **SIFT-1M** (1M) and **GIST-1M** (1M) [10]. The two datasets respectively contain one million 128-D SIFT and 960-D GIST descriptors, each of which complies with a separate query subset. We respectively construct a training set of 10,000 random samples and a testing set of 1,000 random queries on both datasets. Besides, we employ another two much larger datasets **SIFT-20M** (20M) [10] and **Tiny-80M** (80M) [29], respectively consisting of 20 million 128-D SIFT and 80 million 384-D GIST features. We respectively sample 50,000 and 100,000 points as the training sets, and 3,000 random queries as the testing ones. As to the groundtruth of each query, we select the 1,000 Euclidean nearest neighbors among the database on SIFT-1M, GIST-1M and SIFT-20M, and 5,000 on Tiny-80M.

We adopt two common search schemes to evaluate the hashing performance, *i.e.*, Hamming distance ranking and hash table lookup. The former ranks all candidates based on the Hamming distances from the query, and the later treats points falling within a small Hamming radius r ($r \leq 2$) from the query code as the retrieved results. As to KMH and our ABQ with product quantization, we set $b = 4$ for SIFT features when using less than 64 bits, and $b = 8$ for all other cases. In each experiment, we run 10 times in a workstation with 2.53 GHz Xeon CPU and report the averaged performance.

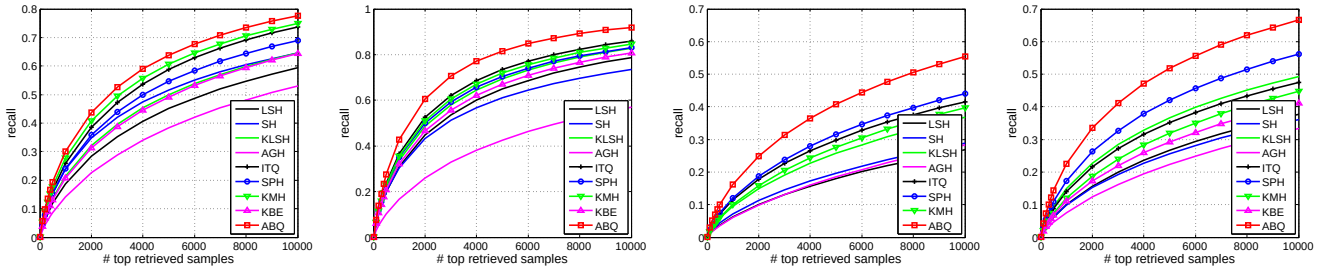
4.2 Results and Discussions

4.2.1 Euclidean Nearest Neighbor Search

We first evaluate all hashing methods in the task of Euclidean nearest neighbor search over SIFT-1M and GIST-1M. We adopt both precision and recall to comprehensively study their performance. Table 1 lists the mean average precision (MAP) using Hamming distance ranking with respect to different number of hash bits. From the table we can observe that all methods increase their MAP performance when using more hash bits from 32 to 128 bits. Moreover, methods like ITQ, SPH, KMH and our ABQ, which encode the data from the view of clustering quantization, consistently achieve much better performance than other methods like LSH, SH and AGH. This indicates that it is a promising way to discover a particular quantization strategy for binary hashing. Among all these methods, ABQ obtains the best performance, and gets significant performance gains over the best competitors, *e.g.*, using 128 bits, 24.41% over ITQ on SIFT-1M, and 39.76% over SPH on GIST-1M.

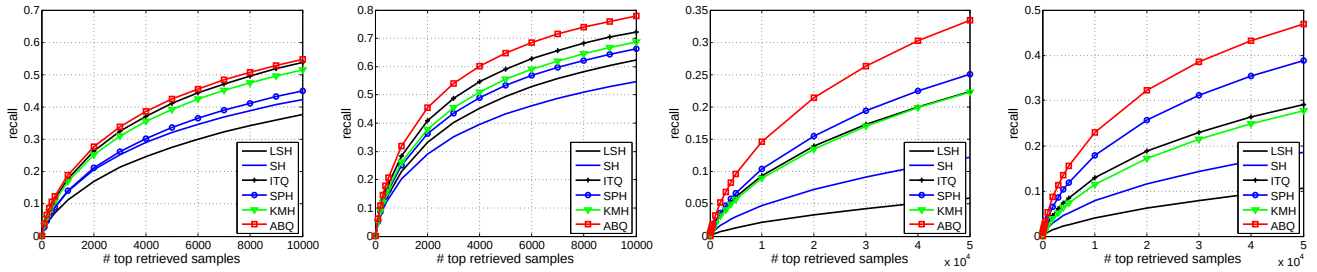
Figure 3 further plots the recall curves with respect to different number of retrieved results on both datasets, where we can get the same conclusion that ABQ performs best in all cases. The reason is mainly that compared to the baselines where the codebook is fixed, ABQ can adaptively generate the codebook of a varying size and well match the binary codes to the prototypes. For the performance of Hamming distance ranking, we compare our ABQ with the baseline methods in terms of precision, besides recall and MAP performance in the paper. Figure 3 also plots the precision curves with respect to different cutting points of the retrieved result lists on SIFT-1M and GIST-1M, where we vary the number of hash bits from 64 to 128. We can see that our ABQ performs best in all cases with significant performance superiority to other methods.

Besides Hamming distance ranking, hash table lookup is another common search strategy over the hash codes. In this case, usually a small code (*e.g.*, 32 bits for one million data) is used to avoid the memory and time consumption derived from the exponentially huge amount of indexing buckets. Table 1 further reports the precision within a small Hamming radius $r = 1$ and $r = 2$ (PH1 and PH2 for short). This is also a popular evaluation metric in practice, because with a small lookup radius, nearest neighbor search can be efficiently completed by only locating data falling in buckets with Hamming distance less than the radius from the query. Similarly, it is easy to see that the ABQ outperforms the baselines with a large margin, *e.g.*, 15.91% and 58.84% PH1 gains over KMH respectively on SIFT-1M and GIST-1M. Compared to SPH and KMH that also exploit the prototype based hash functions, the encouraging precision gains obtained by ABQ indicate that our ABQ can approximate the neighbor relations much better by encoding the data using a sub-



(a) 64 bits on SIFT-1M (b) 128 bits on SIFT-1M (c) 64 bits on GIST-1M (d) 128 bits on GIST-1M

Figure 3. Recall performance of different hashing methods on SIFT-1M and GIST-1M.



(a) 64 bits on SIFT-20M (b) 128 bits on SIFT-20M (c) 64 bits on Tiny-80M (d) 128 bits on Tiny-80M

Figure 4. Recall performance of different hashing methods on SIFT-20M and Tiny-80M.

set of binary codes in Hamming space. This intuition is also visually demonstrated in Figure 1 using a subset of SIFT-1M.

4.2.2 Nearest Neighbor Search over Large Datasets

To investigate the performance of different hashing methods over more large-scale dataset, we adopt two of the largest datasets to-date: SIFT-20M and Tiny-80M. Table 2 reports the precision performance in terms of Hamming distance ranking and hash table lookup. Here, due to the facts that in practice users are more concerned about the top ranked results, and while computing the MAP of the full Hamming distance ranking list is quite time-consuming [18, 17], we present the average precision of top 1,000 returned samples (P@1,000) instead of MAP with respect to the varying code length (32, 64 and 128). Similar to the results in Table 1, in all cases our ABQ consistently obtains the best precision, especially on Tiny-80M dataset with remarkable superiority, *e.g.*, up to 45.87% performance gain over the best competitor SPH. As to hash table lookup, Table 2 also lists the PH1 performance using 32 bits hash table, from which we can get a similar observation that ABQ shows a better capability of capturing the neighbor structures, and thus covers much more nearest neighbors than all baselines.

Figure 4 respectively depicts the recall curves using 64 and 128 bits on SIFT-20M and Tiny-80M. Compared to all the baselines, our ABQ boosts the recall with the most significant improvement when using more hash bits, and consistently performs best in all cases. On the real-world dataset Tiny-80M, this observation can be more obvious as shown in Figure 4(c) and (d), where the recall of the top 10^4 result list increases largely from 14.61% to 22.93% with more hash bits, and meanwhile the best performance among all baselines is 17.90% achieved by SPH using 128 hash bits. This fact further demonstrates that our ABQ can faithfully boost the overall hashing performance in terms of precision and recall, using Hamming distance ranking or hash table lookup. As to the precision performance, since the groundtruth number is fixed to a constant number, a similar observation can be obtained as recall performance, which means that the proposed method can obtain the best recall and meanwhile the best precision performance on the two much larger datasets SIFT-20M and Tiny-80M.

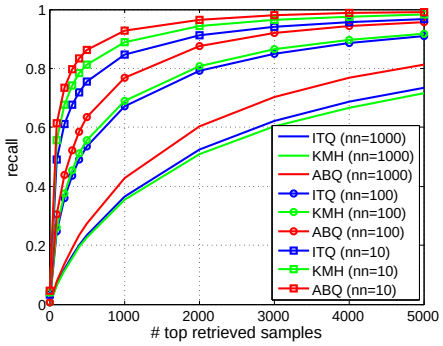
4.2.3 Groundtruth Number Effect

Prior research have pointed out that the number of groundtruth may have effects on the performance [5]. Therefore, to illustrate the robustness of our method with respect to the groundtruth number nn, we further conduct the experiments on SIFT-1M and GIST-1M by varying nn in $\{10, 100, 1000\}$. In Figure 5 we compare the recall performance of ABQ using 128 bits to those of the two state-of-the-art methods ITQ and KMH, which archived the best performance among all baselines as shown in prior experiments. In this figure, we vary the groundtruth number nn on different datasets.

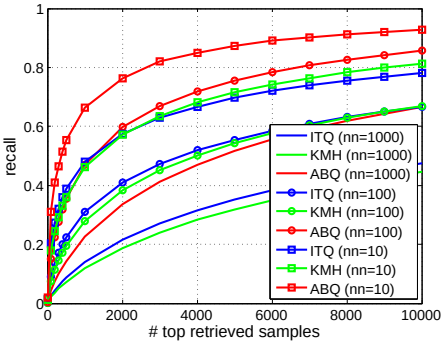
As we can see from the figure, when using more nearest neighbors as the groundtruth (from 10 to 1000), all methods decrease the recall performances. This is because that as the distance between the database point and query increases, the collision probability between them will decrease. Nevertheless, for different settings, our ABQ consistently achieves the best performance and significantly outperforms others in all cases. For instance, with $nn = 100$ on GIST-1M, ABQ can achieve much higher recall than ITQ and KMH, and even better than them with $nn = 10$. This means that our method is very robust to the task of nearest neighbor search. Besides, in all these experiments we adopt the same parameter settings, which indicates that the proposed ABQ is practical without complex parameter tuning.

Table 2. Hashing performance on SIFT-20M and Tiny-80M.

	SIFT-20M					TINY-80M				
	P@1,000			PH (32 BITS)		P@1,000			PH (32 BITS)	
	32 BITS	64 BITS	128 BITS	$r = 1$	$r = 2$	32 BITS	64 BITS	128 BITS	$r = 1$	$r = 2$
LSH	4.34	11.26	22.81	9.86	8.03	0.75	2.18	4.24	0.83	0.51
SH	8.00	13.91	20.19	22.70	17.34	2.77	5.12	9.06	3.37	1.71
ITQ	8.48	17.69	28.47	20.18	14.69	5.25	9.99	14.10	10.59	7.47
SPH	6.06	14.06	25.09	14.72	10.10	4.53	10.90	20.03	9.51	5.67
KMH	8.29	16.90	26.08	22.54	16.18	5.31	9.41	11.92	11.64	7.50
ABQ	**8.95**	**18.92**	**31.86**	**25.97**	**17.59**	**7.51**	**15.93**	**26.56**	**12.67**	**7.57**



(a) recall on SIFT-1M



(b) recall on GIST-1M

Figure 5. Recall performance of different hashing methods with respect to different number of groundtruth (10, 100, 1000) on SIFT-1M and GIST-1M.

4.2.4 Efficiency Issue

Figure 2 shows that the proposed ABQ can converge fast in less than 10 iterations. Therefore, in practice, the algorithm can achieve efficient training and support the large-scale learning. This is consistent with our complexity analysis in Section 3.4.3, *e.g.*, the training time scales linearly to the size of the training set.

Table 1 further lists the offline training time and online search time when using 128 hash bits on SIFT-1M and GIST-1M. We can see that usually the iterative binary quantization methods like ITQ, SPH, KMH and our ABQ take more training time than the others. This is mainly due to the difficulty of finding an optimal coding solution that can align the Hamming space with the original one. Among these methods, our ABQ costs much less time than KMH, while gives the best performance with a little more training time than SPH and ITQ. Moreover, at the online search stage, only a small set of prototypes (smaller than 2^b, $b \leq 8$ in each subspace) will be checked, and thus the hashing time is very close to the prior projection based methods. Namely, it can support the real-time nearest neighbor search as the existing methods do.

5 Conclusions

Inspired by our observation that in prototype based hashing there might exist a better coding solution that only utilizes a small subset of binary codes instead of the complete set, this paper proposed an adaptive binary quantization method that jointly pursues a set of prototypes in the original space and a subset of binary codes in the Hamming space. The prototypes and the codes are correspondingly associated and together define the hash function for small hash codes. Our method enjoys fast computation and the capability of generating long hash codes in product space, with discriminative power for nearest neighbor search. The significant performance gains over existing methods were obtained in our extensive experiments on several large datasets, which encourage us to further study the effective coding for binary quantization.

6 Acknowledgment

We would like to thank the referees for their comments, which helped improve this paper considerably. This work was partially supported by the National Natural Science Foundation of China (61402026, 71322104, and 71531001), the Foundation of the State Key Laboratory of Software Development Environment (SKLSDE-2016ZX-04), and National High Technology Research and Development Program of China (SS2014AA012303).

REFERENCES

[1] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni, 'Locality-sensitive hashing scheme based on p-stable distributions', in *SCG*, pp. 253–262, (2004).

[2] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun, 'Optimized product quantization', *IEEE TPAMI*, **36**(4), 744–755, (April 2014).

[3] Yunchao Gong and S. Lazebnik, 'Iterative quantization: A procrustean approach to learning binary codes', in *IEEE CVPR*, pp. 817–824, (2011).

[4] Junfeng He, Jinyuan Feng, Xianglong Liu, Tao Cheng, Tai-Hsu Lin, Hyunjin Chung, and Shih-Fu Chang, 'Mobile product search with bag of hash bits and boundary reranking', in *IEEE CVPR*, pp. 3005–3012, (2012).

[5] Kaiming He, Fang Wen, and Jian Sun, 'K-means hashing: An affinity-preserving quantization method for learning binary compact codes', in *IEEE CVPR*, pp. 2938–2945, (2013).

[6] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon, 'Spherical hashing', in *IEEE CVPR*, pp. 2957–2964, (2012).

[7] Long-Kai Huang, Qiang Yang, and Wei-Shi Zheng, 'Online hashing', in *IJCAI*, pp. 1422–1428, (2013).

[8] Piotr Indyk and Rajeev Motwani, 'Approximate nearest neighbors: towards removing the curse of dimensionality', in *ACM STOC*, (1998).

[9] Prateek Jain, Sudheendra Vijayanarasimhan, and Kristen Grauman, 'Hashing Hyperplane Queries to Near Points with Applications to Large-Scale Active Learning', in *NIPS*, 928–936, (2010).

[10] Herve Jegou, Matthijs Douze, and Cordelia Schmid, 'Product quantization for nearest neighbor search', *IEEE TPAMI*, **33**(1), 117–128, (January 2011).

[11] Qing-Yuan Jiang and Wu-Jun Li, 'Scalable graph hashing with feature transformation', in *IJCAI*, pp. 2248–2254, (2015).

[12] Zhongming Jin, Yao Hu, Yue Lin, Debing Zhang, Shiding Lin, Deng Cai, and Xuelong Li, 'Complementary projection hashing', in *IEEE ICCV*, pp. 257–264, (2013).

[13] Weihao Kong and Wu-Jun Li, 'Isotropic hashing', in *NIPS*, pp. 1–8, (2012).

[14] B. Kulis and K. Grauman, 'Kernelized locality-sensitive hashing for scalable image search', in *IEEE ICCV*, (2009).

[15] Brian Kulis and Trevor Darrell, 'Learning to hash with binary reconstructive embeddings', in *NIPS*, pp. 1–8, (2009).

[16] X. Li, G. Lin, C. Shen, A. van den Hengel, and A. Dick, 'Learning hash functions using column generation', in *ICML*, (2013).

[17] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang, 'Discrete graph hashing', in *NIPS*, (2014).

[18] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang, 'Hashing with graphs', in *ICML*, pp. 1–8, (2011).

[19] Wei Liu, Jun Wang, Yadong Mu, Sanjiv Kumar, and Shih-Fu Chang, 'Compact hyperplane hashing with bilinear functions.', in *ICML*, (2012).

[20] Xianglong Liu, Bowen Du, Cheng Deng, Ming Liu, and Bo Lang, 'Structure sensitive hashing with adaptive product quantization', *IEEE TCYB*, **PP**(99), 1–12, (2015).

[21] Xianglong Liu, Xinjie Fan, Cheng Deng, Zhujin Li, Hao Su, and Dacheng Tao, 'Multilinear hyperplane hashing', in *IEEE CVPR*, (2016).

[22] Xianglong Liu, Junfeng He, Cheng Deng, and Bo Lang, 'Collaborative hashing', in *IEEE CVPR*, (2014).

[23] Xianglong Liu, Junfeng He, Bo Lang, and Shih-Fu Chang, 'Hash bit selection: a unified solution for selection problems in hashing', in *IEEE CVPR*, (2013).

[24] Xianglong Liu, Lei Huang, Cheng Deng, Jiwen Lu, and Bo Lang, 'Multi-view complementary hash tables for nearest neighbor search', in *IEEE ICCV*, (2015).

[25] Xianglong Liu, Yadong Mu, Bo Lang, and Shih-Fu Chang, 'Mixed image-keyword query adaptive hashing over multilabel images', *ACM TOMM*, **10**(2), 22:1–22:21, (February 2014).

[26] Yadong Mu, Gang Hua, Wei Fan, and Shih-Fu Chang, 'Hash-svm: Scalable kernel machines for large-scale visual classification', in *IEEE CVPR*, (2014).

[27] Mohammad Norouzi and David J. Fleet, 'Cartesian k-means', in *IEEE CVPR*, pp. 2938–2945, (2013).

[28] Maxim Raginsky and Svetlana Lazebnik, 'Locality-sensitive binary codes from shift-invariant kernels', in *NIPS*, pp. 1–8, (2009).

[29] Antonio Torralba, Rob Fergus, and William T. Freeman, '80 million tiny images: A large data set for nonparametric object and scene recognition', *IEEE TPAMI*, **30**(11), 1958–1970, (2008).

[30] Qifan Wang, Zhiwei Zhang, and Luo Si, 'Ranking preserving hashing for fast similarity search', in *IJCAI*, pp. 3911–3917, (2015).

[31] Yair Weiss, Antonio Torralba, and Rob Fergus, 'Spectral hashing', in *NIPS*, pp. 1–8, (2008).

[32] Botong Wu, Qiang Yang, Wei-Shi Zheng, Yizhou Wang, and Jingdong Wang, 'Quantized correlation hashing for fast cross-modal search', in *IJCAI*, pp. 3946–3952, (2015).

[33] Bin Xu, Jiajun Bu, Yue Lin, Chun Chen, Xiaofei He, and Deng Cai, 'Harmonious hashing', in *IJCAI*, pp. 1820–1826, (2013).

[34] Felix Yu, Sanjiv Kumar, Yunchao Gong, and Shih-Fu Chang, 'Circulant binary embedding', in *ICML*, (2014).

[35] Xu Zhang, Felix X. Yu, Ruiqi Guo, Sanjiv Kumar, Shengjin Wang, and Shih-Fu Chang, 'Fast orthogonal projection based on kronecker product', in *IEEE ICCV*, (2015).