Unsupervised Online Hashing with Multi-Bit Quantization

Zhenyu Weng[1][0000−0001−7857−8687] and Yuesheng Zhu[1][0000−0003−2524−6800]

Communication and Information Security Lab, Shenzhen Graduate School , Peking University, Shenzhen, China
{wzytumbler,zhuys}@pku.edu.cn

Abstract. Online hashing methods aim to update hash functions with newly arriving data streams, which can process large-scale data online. To this end, most existing methods update projection functions online and adopt a single-bit quantization strategy that quantizes each projected component with one bit. However, single-bit quantization results in large information loss in the quantization process and thus cannot preserve the similarity information of original data well. In this paper, we propose a novel unsupervised online hashing method with multi-bit quantization towards solving this problem, which consists of online data sketching and online quantizer learning. By maintaining a small-size data sketch to preserve the streaming data information, an orthogonal transformation is learned from the data sketch to make the components of the streaming data independent. Then, an optimal quantizer is learned to adaptively quantize each component with multiple bits by modeling the data distribution. Therefore, our method can quantize each component with multiple bits rather than one bit to better preserve the data similarity online. The experiments show that our method can achieve better search accuracy than the relevant online methods for approximate nearest neighbor search.

Keywords: online hashing · unsupervised hashing · multi-bit quantization.

1 Introduction

With the development of feature representation methods, especially deep learning methods [12, 24], images and videos are represented by high-dimensional features that can obtain high-level semantic information. Conventional nearest neighbor search methods [1, 23] are inefficient for high-dimensional features as a consequence of the curse of dimensionality. The difficulty of finding the exact nearest neighbors in the high-dimensional space leads to the emergence of Approximate Nearest Neighbor (ANN) search methods [19, 25] using a compact data representation for high-dimensional data. Hashing methods [27, 26] are one type of widely-used ANN search methods. The goal of hashing methods is to learn a binary-code representation which can preserve the similarity structure

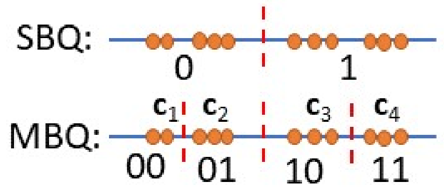


Fig. 1. A comparison example between Single-Bit Quantization (SBQ) and Multi-Bit Quantization (MBQ) for the projected component. Here MBQ adopts two bits. $\mathbf{c_i}$ is a group centroid to represent the data points in the group.

of data in the original feature space. Using a binary representation for data can not only reduce the database storage but also improve the search efficiency.

Traditional hashing methods [25, 21, 22, 3, 6] are mostly designed to learn hash functions offline from a fixed collection of training data. However, they cannot deal with the streaming data. In many real-world applications, data is available continuously in streaming fashion. When new data arrives, the data distribution changes and these methods have to accumulate all the data to re-learn hash functions, which is time-consuming. Therefore, online hashing methods [4, 8, 16, 33, 16, 31, 17, 28] have attracted much research attention recently, which can learn the hash functions online from the streaming data.

According to whether there is label information provided in the learning process, online hashing methods can be categorized into two groups, supervised online hashing methods [16, 31, 17, 28, 36, 32] and unsupervised online hashing methods [13, 33]. In this paper, we focus on unsupervised online hashing methods as it is expensive to collect a large number of labeled samples in real-world applications.

Unsupervised online hashing methods [13, 33] usually adopt a learning strategy containing two stages: projection stage and quantization stage [11]. They maintain a small-size data matrix to preserve the characters of streaming data online and learn the projection functions from the data matrix. Then, a Single-Bit Quantization (SBQ) strategy is adopted to quantize each projected component with one bit. However, the recent research [30, 5] shows that SBQ results in large information loss in the quantization process and cannot preserve the similarity structure of original data well, as shown in Fig. 1.

In this paper, we propose an unsupervised online hashing method with Multi-Bit Quantization (MBQ) to address the above problem. In our method, a data sketch is maintained from the streaming data and an orthogonal transformation is learned from the data sketch to reduce statistical dependence among the data components. By modeling the distribution of each component, a bit allocation algorithm is designed to adaptively allocate bits to each component, and an optimal quantizer is learned by independently quantizing each component. As shown in Fig. 1, by learning to quantize each component with multiple bits rather than one bit online, our method have a much bigger distance space than online hashing methods with SBQ and can better preserve the data similarity. The experiments on two widely-used datasets validate the effectiveness of our method.

2 Related Work

Since directly learning best binary codes for a given database is proven as a NP-hard problem [29], most hashing methods adopt a learning strategy containing two stages: projection stage and quantization stage [11]. In the projection stage, several projected components of real values are generated. In the quantization stage, the components generated from the projection stage are quantized into binary codes. Many hashing methods focus on the projection stage and adopt the widely-used SBQ strategy to quantize each projected component [25, 21, 22, 3]. However, the recent research [11, 10, 30, 34, 5] shows that replacing the SBQ strategy by the MBQ strategy can result in a better search accuracy. For example, the literature [11] quantizes each projected component with multiple bits and calculates Manhattan distance between the binary codes for nearest neighbor search. In addition to quantize each component with the same number of bits, some methods [30, 5] develop quantizers to adaptively quantize each projected component with a certain number of bits according to the data distribution. Although the above hashing methods with MBQ can obtain the satisfactory search performance for the static database, they cannot deal with the streaming data. When new data arrives, the data distribution changes and these MBQ methods have to accumulate all the data to re-learn the quantizer, which is time-consuming. Therefore, we propose an novel unsupervised online hashing method to learn the multi-bit quantizer online. To our best knowledge, this is the first work to adopt the MBQ strategy for online hashing.

In parallel to hashing methods, Product Quantization (PQ) methods [19] are another type of ANN search methods. PQ methods [9, 2, 19, 7] represent each high-dimensional feature vector by a Cartesian product of several quantized values. Recently, some online PQ methods [14, 35, 18] are also developed for streaming data. As PQ methods are related to hashing methods, we will compare our method with online unsupervised PQ methods to validate the effectiveness of our method.

3 Online Hashing with Multi-Bit Quantization

This section presents the proposed method, Online Hashing with Multi-Bit Quantization (OHMBQ), to quantize streaming data with multiple bits. The proposed OHMBQ consists of two key designs. The first is to reduce the dependence among data components by learning an orthogonal transformation from streaming data. The second is to learn an optimal quantizer by modeling the data distribution of the transformed components. A theoretical proof for learning the optimal quantizer is provided. More details are described in the following subsections.

3.1 Preliminary about multi-bit quantization

Assume there is a set of data points $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_N], \mathbf{x} \in \mathbb{R}^D$, where N is the number of data points and D is the data dimensionality. Quantization is the process of dividing a large set (or a continuous set) of data points into groups and learning the group's centroid to represent the group. Given the quantization level m, the quantizer $q : \mathbb{R}^D \to Z$ where $Z = \{0, 1, ..., m-1\}$ is characterized by the input space partition $Q(z) = \{\mathbf{x} : q(\mathbf{x}) = z\}$ and the group centroid $\mathbf{c}(z) \in \mathbb{R}^D$ for $z \in Z$. To obtain a binary representation for the data point, the index z is transformed to be a binary expression $\mathbf{t}(z) \in \{0, 1\}^b$ where b is the number of bits, and $m = 2^b$.

The quality of the quantization is measured in terms of its average distortion,

$$A = E[d(\mathbf{x}, \mathbf{c}(q(\mathbf{x})))], \tag{1}$$

where $d : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is the distortion function which takes on the metric of Euclidean distance in the nearest neighbor search.

As [3] denotes, to minimize the average distortion A, the optimal quantizer is characterized by the following properties:

$$\begin{cases} Q(z) = \{\mathbf{x} : d(\mathbf{x}, \mathbf{c}(z)) \leq d(\mathbf{x}, \mathbf{c}(z')), \forall z' \in Z\} \\ \mathbf{c}(z) = \arg\min_{\mathbf{x}'} E_{\mathbf{x}}[d(\mathbf{x}, \mathbf{x}') | \mathbf{x} \in Q(z)] \end{cases}. \tag{2}$$

It is a challenging problem to quantize the high-dimensional data into hundreds of groups. Especially, the quantization level of the quantizer $m = 2^b$ grows exponentially on the bit number b. It is impossible to collect a sufficient number of training examples to span the quantized space which comprises hundreds of bits.

To address the challenge, multi-bit quantization methods assume the data distribution $p(\mathbf{x})$ is independent in its components after projecting the data onto a new space with the projection matrix learned in the projection stage. Then, the metric is of the form $d(\mathbf{x}, \mathbf{x}') = \sum_i d_i(x_i, x'_i)$, where x_i are the components of $\mathbf{x}$. Therefore, a minimum distortion quantizer can be obtained by forming the Cartesian product of the independently quantized components. That is, let

$$\mathbf{q}(\mathbf{x}) = (q_1(x_1), q_2(x_2), ..., q_D(x_D)) = \mathbf{z}, \tag{3}$$

where $\mathbf{z} = (z_1, ..., z_D), z_i = q_i(x_i)$

Given a query $\tilde{\mathbf{x}}$, the distance between the query and the data point is calculated as

$$d(\tilde{\mathbf{x}}, \mathbf{q}(\mathbf{x})) = \sum_i d_i(\tilde{x}_i, c_i(q_i(x_i))). \tag{4}$$

According to Eqn. (4) and Fig. 1, hashing methods with MBQ obviously have a much bigger distance space than hashing methods with SBQ and thus can achieve better search accuracy.

Although multi-bit quantization methods have achieved promising search performance on the static database, they cannot be directly applied on the

Algorithm 1 Zero-Mean Sketching

```
Input: data chunk $\mathbf{D}$, sketch matrix $\mathbf{P}$, mean value $\mu$, cumulative number $n$
Output: sketch matrix $\mathbf{P}$
1: Sketch $[\mathbf{D}-\bar{\mathbf{D}}, \sqrt{\frac{nn_D}{n+n_D}}(\bar{\mathbf{D}}-\mu)]$ into $\mathbf{P}$ with FD where $\bar{\mathbf{D}}$ is the mean value of $\mathbf{D}$
   and $n_D$ is the size of the sketch $\mathbf{D}$
2: $\mu = \frac{n\mu}{n+n_D} + \frac{n_D\bar{\mathbf{D}}}{n+n_D}$
3: $n = n + n_D$
```

streaming environment where the data distribution varies when data keeps growing. Specifically, the existing techniques for reducing the dependence between data components and learning an optimal quantizer require knowing the data distribution in advance. In the following, we describe how to reduce the dependence between data components and learn an optimal quantizer online in our method.

3.2 Online transform coding

For the static database, the assumption that the components of $\mathbf{x}$ are independent can be addressed by transform coding [3], which seeks a transformation to reduce statistical dependence among the components. This is typically done through Principal Component Analysis (PCA) where an orthogonal transformation matrix $\mathbf{U}$ is obtained by concatenating the top eigenvectors of covariance matrix $\mathbf{X}\mathbf{X}^T$. Here, $\mathbf{X}$ is zero-mean data. To learn an orthogonal transformation from the streaming data that can approximate the orthogonal transformation in PCA, we adopt Online Sketching Hashing (OSH) [13] to maintain a small-size sketch online to preserve the character of the streaming data and learn the transformation matrix from the data sketch.

Assume data comes in chunks. New data chunk $\mathbf{D}_t$ arrives at round t. $\mathbf{X}_t = [\mathbf{D}_1, ..., \mathbf{D}_t]$ denotes the data matrix accumulated from round 1 to round t, μ_t is the mean of $\mathbf{X}_t$. Since the data should be zero-mean in PCA, Online Sketching Hashing (OSH) [13] develops a zero-mean data sketching method based on Frequent Directions(FD) [15] to learn a small-size sketch $\mathbf{P} \in \mathbb{R}^{\mathbf{D}\times\mathbf{k}}$ where k is the size of the sketch such that $\mathbf{P}\mathbf{P}^T \approx (\mathbf{X}_t - \mu_t)(\mathbf{X}_t - \mu_t)^T$, which is summarized in Algorithm 1. The input to Algorithm 1 is the data chunk $\mathbf{D}$, the data sketch $\mathbf{P}$, the mean value of the cumulative data μ, and the cumulative number of the data n. More details can be found in [13]

By applying Algorithm 1 we can obtain an orthogonal transformation matrix $\mathbf{U}$ by taking the top eigenvectors from $\mathbf{P}\mathbf{P}^T$ for learning the quantizer.

3.3 Online quantizer learning

Assume the data has been transformed by the transformation matrix $\mathbf{U}$ learnt from the data sketch and the dependence among the components has been reduced. To obtain a b-bit binary code for representing the data point, we need to determine the bit number for each component, *i.e.*, the quantization level for each component, at first. Then, for each component, we need to separate the input space into groups and learn the centroid of each group.

Bit allocation is a process of determining the number of data components and determining the quantization level of each component. The number of the bits is fixed. When more bits are allocated to the component to preserve the information of the single component, the number of the selected components become smaller, which results in the information loss, and vice versa. The literature [30] determines the bit number for each component according to the data distribution of each component. In the streaming environment, since the data distribution keeps changing and accumulating all the data is time-consuming, this bit allocation algorithm cannot be used for the streaming data. The literature [3] allocates bits to each component proportionally according to the standard deviation of the component. It can be used in the streaming environment. However, this method treats each component independently. In our method, we design a new bit allocation algorithm from the perspective of the accumulative component energy.

Since b bits can be allocated to at most b components, *i.e.*, one bit for one component, b components that have the maximum information are chosen by $\mathbf{Y} = \mathbf{U}^T(\mathbf{X}_t - \mu_t)$ where $\mathbf{U} \in \mathbb{R}^{D \times b}$ is obtained by taking the top b eigenvector from $\mathbf{P}\mathbf{P}^T$. The standard deviation δ_i of the i^{th} component corresponds to the i^{th} eigenvalue. Each column $\mathbf{y} \in \mathbb{R}^b$ of $\mathbf{Y}$ has b components.

As it is time-consuming to obtain the data distribution by accumulating all the data when new data arrives, we hope to model the data distribution of each component. Fig. 2 shows the data distribution of each component after transformation on the CIFAR-10 dataset [20] for each round. From the figure, we can see that the data distribution for the component tends to be a Gaussian distribution after a few rounds. Hence, we assume each component is subject to the Gaussian distribution $N(0, \delta_i^2)$. Inspired by PCA which selects eigenvectors according to the accumulative component energy, we want to choose a small number of components L while achieving a reasonably high percentage $\alpha \in [0, 1]$ of accumulative component energy (standard deviation). The total energy of the components is $G = \sum_{i=1}^{b} \delta_i$. And the smallest L is chosen so that the cumulative energy g for L components is above the certain threshold, which is

$$g = \sum_{i=1}^{L} \delta_i \geq \alpha G. \tag{5}$$

Since each component has the identical distribution after normalizing the variance, the remaining $b - L$ bits are allocated proportionally on the remaining standard deviations of the components, which is summarized on Algorithm 2. After choosing L components and allocating one bit to each component, the remaining $b - L$ bits are allocated for $b - L$ rounds. In each round, we can find the component index i that has the largest remaining standard deviation h_i by enumeration. Then, the allocated bit number for the i^{th} component is

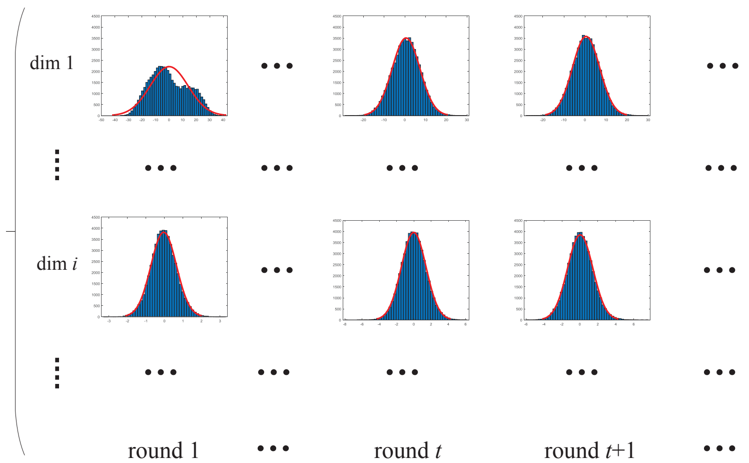


Fig. 2. The data distribution of each component after transformation on CIFAR-10 with the rounds increasing.

Algorithm 2 Bit Allocation

```
Input: standard deviations of components $\{\delta_i\}_{i=1}^{b}$, $\alpha$
Output: bit allocation $\{l_i\}_{i=1}^{L}$
1: Find the smallest $L$ such that $\sum_{i=1}^{L} \delta_i \geq \alpha \sum_{i=1}^{b} \delta_i$
2: Initialize $h_i \leftarrow \frac{\delta_i}{2}$ and $l_i \leftarrow 1$
3: for each round in $b - L$ rounds do
4:     $i \leftarrow \arg\max_{i'} h_{i'}$
5:     $l_i \leftarrow l_i + 1$
6:     $h_i \leftarrow h_i/2$
7: end for
```

incremented by 1 and the remaining standard deviation h_i is divided by 2 as the quantization level is doubled.

After bit allocation, we need to find the optimal quantizer for each component. [3] and [30] find the space division and the centroids by using all the data points, which is difficult in the streaming environment. As each component is subject to the Gaussian distribution, we have the following theorem to minimize the quantization error for each component.

Theorem 1. Assume X be a random variable subject to a Gaussian distribution $N(0, \delta^2)$ and F_X is the cumulative distribution function. Then, the quantizer that minimizes the quantization error is obtained by

$$\begin{cases} Q(z) = \{X : F_X^{-1}(\frac{z}{2^l}) \leq X \leq F_X^{-1}(\frac{z+1}{2^l})\} \\ c(z) = F_X^{-1}(\frac{z\times 2+1}{2^{l+1}}) \end{cases}, \tag{6}$$

Algorithm 3 Online Hashing with Multi-Bit Quantization

```
Input: data chunk $\mathbf{D}$
Output: quantizer $\mathbf{q}$
1: Maintain a data sketch according to Algorithm 1
2: Learn an transform matrix from the data sketch
3: Perform bit allocation according to Algorithm 2
4: Obtain an optimal quantizer $\mathbf{q}$ by independently minimize each component accord-
   ing to Theorem 1
```

where l is the number of bits allocated to the component, $z \in Z = \{0, 1, ..., 2^l - 1\}$, and F_X^{-1} is the inverse cumulative distribution function.

Proof. As F_X is strictly increasing on the possible values of X, F_X has an inverse function F_X^{-1} which is one-to-one function. F_X^{-1} is the inverse cumulative distribution function. Let $U = F_X(X)$. Then, for $u \in [0, 1]$,

$$\begin{aligned} P\{U \leq u\} &= P\{F_X(X) \leq u\} \\ &= P\{U \leq F_X^{-1}(u)\} \\ &= F_X(F_X^{-1}(u)) = u. \end{aligned} \tag{7}$$

Obviously, U is uniform random variable on [0,1]. For U, according to Eqn.(2), the optimal quantizer is obviously characterized by

$$\begin{cases} Q(z) = \{U : \frac{z}{m} \leq U \leq \frac{z+1}{m}\} \\ c(z) = \frac{z \times 2 + 1}{m \times 2} \end{cases}, \tag{8}$$

where $z \in Z = \{0, 1, ..., m-1\}$ and m is the quantization level.

Therefore, as F_X has an one-to-one inverse function F_X^{-1} and $U = F_X(X)$, the optimal quantizer for X is characterized by

$$\begin{cases} Q(z) = \{X : F_X^{-1}(\frac{z}{m}) \leq X \leq F_X^{-1}(\frac{z+1}{m})\} \\ c(z) = F_X^{-1}(\frac{z \times 2 + 1}{m \times 2}) \end{cases}. \tag{9}$$

3.4 Algorithm analysis

Our method, Online Hashing with Multi-Bit Quantization (OHMBQ), is summarized in Algorithm 3. OHMBQ includes four steps, data sketching, learning a transform matrix, bit allocation, and learning optimal quantizers. Specifically, assume there is a stream of data chunks, $\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_t$. For each chunk, our method learns a data sketch from the streaming data according to Algorithm 1. With the data sketch, a transform matrix is learned to reduce the dependence among streaming data components. Then, the bit allocation for each component is learned to make a good tradeoff between the information loss arising from reducing data components and the quantization error arising from quantization level per data component according to Algorithm 2. With the quantization level (*i.e.*, bit number) for each component, the optimal quantizer for each component to minimize the quantization error is learned according to Theorem 1.

Time complexity. As shown in Algorithm 3, OHMBQ includes four steps. As indicated in [13], the time complexity of data sketching is $O(Dkn_t)$, where n_t is the size of accumulated time at round t, D is the data dimensionality, and k is the size of data sketch. For each round, the time complexity of learning a transform matrix is $O(Dk^2 + k^3)$. The time complexity of bit allocation at one round is $O(bL)$ where L is the number of components and b is the number of bits. For the step of quantizing each component, the inverse cumulative distribution function is implemented by the icdf function in MATLAB, and the time complexity of quantizing L components at one round is $O(2^l L)$ where l is the maximum bit number for the component. In the experiments, the largest value of l is 5. Therefore, the cumulative time complexity of learning the quantizer at round t is $O(Dkn_t + tDk^2 + tk^3 + tbL + 2^l tL)$ where l is usually smaller or equal to 5. The time complexity is close to that of OSH [13].

The process of encoding the database in our method is composed of projecting the data into a low-dimensional space and mapping each component value to the corresponding quantized value. Assume the size of the database is N. The time complexity of projection is $O(NDL)$ and the time complexity of mapping is $O(2^l NL)$. Therefore, the time complexity of encoding the database is $O(NDL + 2^l NL)$ where l is usually smaller or equal to 5 in the experiments.

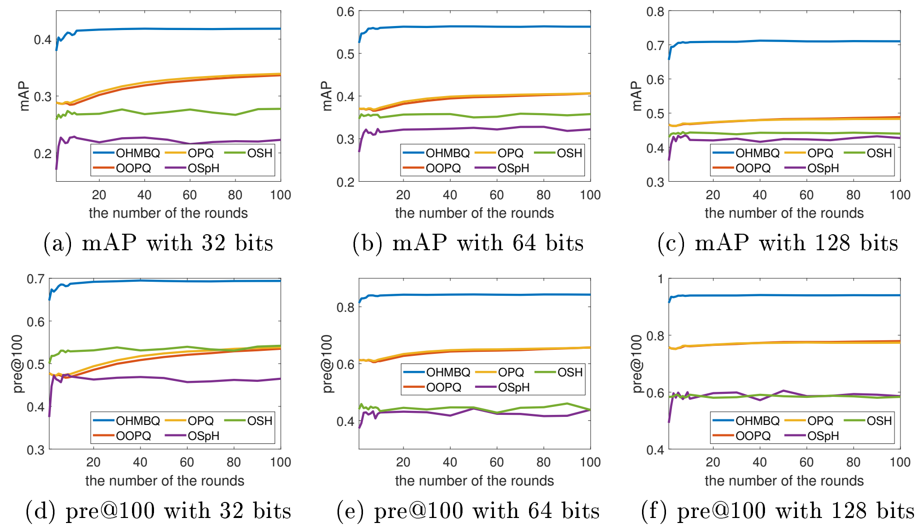
4 Experiments

4.1 Experimental setting

We adopt two widely-used datasets to evaluate the proposed method: CIFAR-10 [20] and GIST1M [9]. CIFAR-10 dataset includes 60,000 32×32 images. Each image is represented by a 4096-dimensional feature extracted from the last convolutional layer in the VGG-net [24]. In CIFAR-10, 10,000 images are randomly selected as the queries, and the rest is used for training and search. GIST1M includes 1,000,000 960-dimensional GIST features and 1,000 queries. For each query, the top K Euclidean nearest neighbors is set as the ground-truth. For both datasets, K is set to 1,000. Following [35, 33], mean Average Precision (mAP) and pre@100 are used as the measurements to evaluate the performance of ANN search. pre@100 is the precision of the top 100 retrieved data points. The experiments are repeated five times and the average results are reported.

4.2 Comparison to other online methods

We compare our method, OHMBQ, with Online Product Quantization (OPQ) [35], Online Optimized Product Quantization (OOPQ) [18], Online Sketching Hashing (OSH) [13] and Online Spherical Hashing (OSpH) [33]. OPQ and OOPQ are online unsupervised product quantization method for Approximate Nearest Neighbor (ANN) search. We carefully implement OPQ and OOPQ according to



(a) mAP with 32 bits (b) mAP with 64 bits (c) mAP with 128 bits

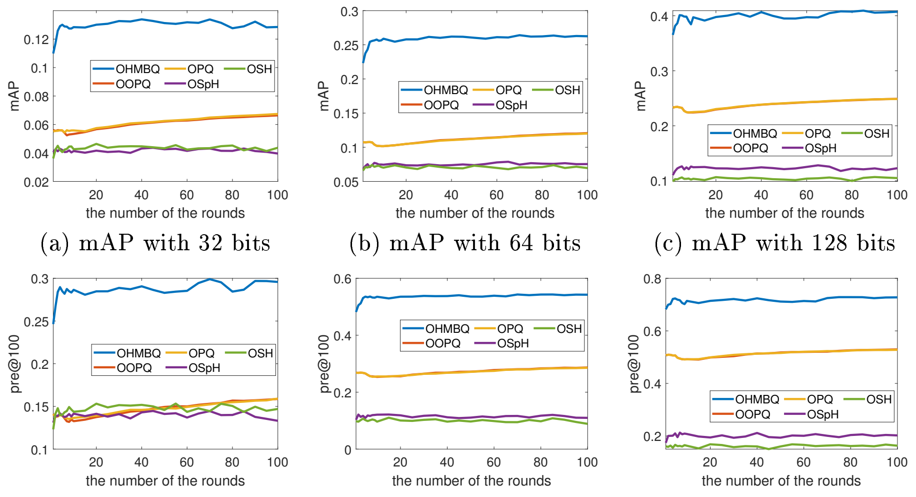
(d) pre@100 with 32 bits (e) pre@100 with 64 bits (f) pre@100 with 128 bits

Fig. 3. The search results of different methods for the early 100 chunks on CIFAR-10.

the descriptions from [35] and [18], respectively. OSH and OSpH are online unsupervised hashing methods for ANN search. The source codes of OSH and OSpH are publicly available. The experiments in [33, 35] show that Mutual Information Hashing (MIH) [4] and Online Kernel Hashing (OKH) [8], two state-of-the-art online supervised hashing methods that take the pairwise pseudo-labels as the input, are inferior to OPQ and OSpH in the unsupervised environment. Hence, we do not compare our method with these online supervised hashing methods in the unsupervised environment.

Following the setting in [13, 33], we divide the training data evenly into chunks of 100 data points for both datasets to simulate the streaming environment. Fig. 3 shows the search results of the online methods in the early 100 chunks from 32 bits to 128 bits on CIFAR-10. We can see that OHMBQ has achieved better search accuracy than other methods from the beginning for mAP and pre@100. For pre@100, OOPQ and OPQ are inferior to OSH at first and better than OSH later for 32bits. For mAP, OOPQ and OPQ are better than OSH from the beginning. The gap between OHMBQ and other methods is huge on CIFAR-10. Fig. 4 shows the search results of the online methods in the early 100 chunks from 32 bits to 128 bits on GIST1M. According to Fig. 3 and Fig. 4, we can see that OHMBQ can achieve a better performance since the beginning compared to other methods as OHMBQ combines the advantages of online sketching from OSH and large distance space from online product quantization methods.

Table 1 shows the search results of online methods on CIFAR-10 after all the chunks are received. According to the results in the table, OHMBQ, OOPQ, and OPQ are better than OSH and OSpH for both mAP and pre@100. Compared



(a) mAP with 32 bits (b) mAP with 64 bits (c) mAP with 128 bits

(d) pre@100 with 32 bits (e) pre@100 with 64 bits (f) pre@100 with 128 bits

Fig. 4. The search results of different methods for the early 100 chunks on GIST1M.

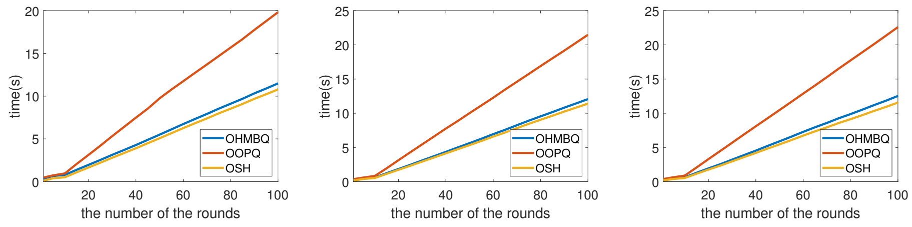
Table 1. The search results on CIFAR-10.

	mAP			pre@100		
	32 bits	64 bits	128 bits	32 bits	64 bits	128 bits
OHMBQ	**0.423**	**0.562**	**0.711**	**0.687**	**0.843**	**0.941**
OOPQ	0.361	0.425	0.498	0.574	0.683	0.784
OPQ	0.348	0.406	0.474	0.554	0.656	0.757
OSpH	0.228	0.324	0.434	0.468	0.427	0.596
OSH	0.273	0.348	0.437	0.537	0.428	0.570

Table 2. The search results on GIST1M.

	mAP			pre@100		
	32 bits	64 bits	128 bits	32 bits	64 bits	128 bits
OHMBQ	**0.129**	**0.263**	**0.403**	**0.296**	**0.542**	**0.721**
OOPQ	0.091	0.147	0.265	0.208	0.336	0.547
OPQ	0.084	0.140	0.260	0.195	0.323	0.542
OSpH	0.041	0.078	0.120	0.136	0.120	0.206
OSH	0.043	0.072	0.105	0.147	0.099	0.159

with OOPQ and OPQ, OHMBQ further improves the performance by more than 10 percent for both mAP and pre@100. Table 2 shows the search results of online methods on GIST1M after all the chunks are received. According to the results in the table, OHMBQ, OOPQ and OPQ are also better than OSH and OSpH for both mAP and pre@100. As denoted by [3], the distance between the



(a) 32 bits (b) 64 bits (c) 128 bits

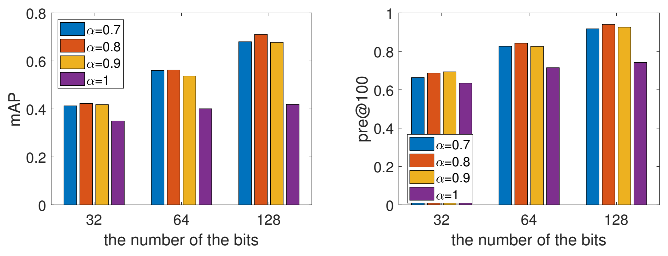
Fig. 5. The accumulated time of different methods for the early 100 chunks on GIST1M.

original data can be better estimated by using the quantized values than using the Hamming distance due to the sparseness of the code space and the limited range of Hamming distance. OHMBQ is obviously better than other methods on GIST1M which includes one million data points. Compared with OPQ and OOPQ, OHMBQ further improves the performance by more than 20 percent for both mAP and pre@100 on GIST1M. According to the results, as OHMBQ, OPQ, and OOPQ have a larger distance space than OSH and OSpH, OHMBQ, OPQ, and OOPQ achieve better search accuracy. Compared with OPQ and OOPQ, OHMBQ can achieve better search accuracy, which demonstrates the effectiveness of OHMBQ.

Apart from search accuracy, we compare OHMBQ with OOPQ and OSH in terms of the accumulated time of online learning from streaming data. According to the above results, OOPQ is the second best method in terms of search accuracy. OHMBQ and OSH both maintain the data sketch from the streaming and learn the functions from the data sketch. Fig. 5 shows the accumulated time of the online methods learning from streaming data in the early 100 chunks from 32 bits to 128 bits on GIST1M. The experiments are run on a computer with a CPU of Intel(R) Xeon(R)W-2223 at 3.60 GHz with 32 GB RAM. According to the results in the figure, we can find that although three methods have similar time costs for initialization at the beginning, OOPQ is obviously slower than OHMBQ and OSH when the chunks of data increase. OHMBQ and OSH have close time costs from 32 bits and 128 bits, which justifies the time complexity analysis from Sec 3.4.

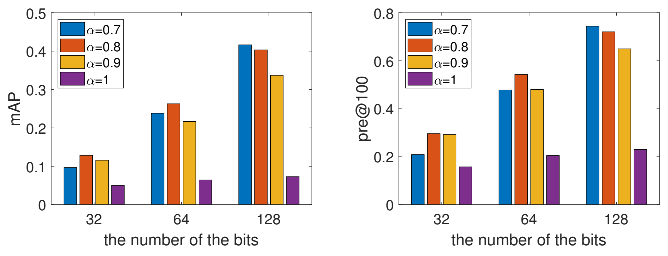
4.3 Bit allocation analysis

As described in Sec. 3.3, bit allocation is a tradeoff between dimension reduction and quantization levels. The number of bits is fixed. When more components are selected, the number of bits allocated to each component is smaller, resulting in more information loss in quantizing the component. When more bits are allocated to each component, the number of the selected components is smaller, resulting in more information loss in dimension reduction. Fig. 6 and Fig. 7 shows the search performance comparison with different α values on CIFAR-10



(a) mAP (b) pre@100

Fig. 6. The search results of OHMBQ with different α values on CIFAR-10.

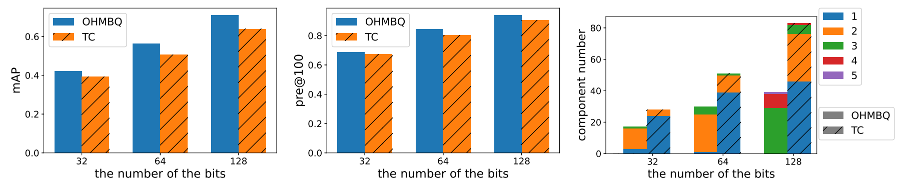


(a) mAP (b) pre@100

Fig. 7. The search results of OHMBQ with different α values on GIST1M.

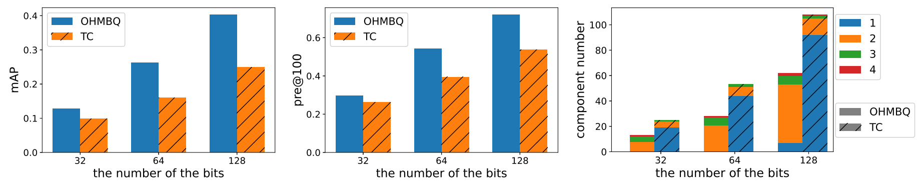
and GIST1M, respectively. The number of the selected components is increasing from $\alpha = 0.7$ to $\alpha = 1$. Meanwhile, the search accuracy increases at first and then decreases. From the results, we can see that taking $\alpha = 0.8$ can achieve good performance. For simplicity, we take $\alpha = 0.8$ for all the experiments in this paper.

Meanwhile, we compare our method with the bit allocation algorithm in Transform Coding (TC) [3], a bit allocation algorithm for the static database. Fig. 8 and Fig. 9 show the comparison between OHMBQ and TC on CIFAR-10 and GIST1M, respectively. From Fig. 8(a)(b) and Fig. 9(a)(b), we can see that the bit allocation algorithm in our method can achieve higher search accuracy than that in TC in terms of mAP and pre@100. Fig. 8(c) and Fig. 9(c) shows the distribution of various numbers of allocated bits over data components. The number in the legend denotes the number of the allocated bits. From Fig. 8(c) and Fig. 9(c), we can see that the number of the selected components in our method is smaller than that in TC, and each component can have more bits in our method. Among the bit distribution for the components, 2 bits per component takes a majority in our method while 1 bit per component takes a majority in TC.



(a) mAP (b) pre@100 (c) bit allocation

Fig. 8. The bit allocation algorithm comparison between OHMBQ and TC on CIFAR-10. (a) and (b) denote the search results. (c) denotes the distribution of various numbers of allocated bits over data components. The number in the legend of (c) denotes the number of allocated bits.



(a) mAP (b) pre@100 (c) bit allocation

Fig. 9. The bit allocation algorithm comparison between OHMBQ and TC on GIST1M. (a) and (b) denote the search results. (c) denotes the distribution of various numbers of allocated bits over data components. The number in the legend of (c) denotes the number of allocated bits.

5 Conclusions

In this paper, we propose a simple but effective online hashing method with multi-bit quantization for approximate nearest neighbor search. By learning an orthogonal transformation to make the components of the streaming data independent and modelling the statistical properties of the components, our method can learn the optimal quantizer online from the streaming data. The complexity analysis shows that the time complexity of learning the quantizer and encoding the database in our method is close to that of Online Sketching Hashing [13]. The experiments show that our method can achieve a huge search accuracy improvement compared with other online hashing methods based on SBQ as our method has a larger distance space. At the same time, compared with the online product quantization methods, our method can improve the search accuracy on the high-dimensional feature vectors by more than 10 percent in most of the cases, especially on the dataset that includes one million feature vectors.

Acknowledgements This work was supported in part by the Nature Science Foundation of China under Grant 62006007, and in part by the National Innovation 2030 Major S&T Project of China under Grant 2020AAA0104203.

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: FOCS. pp. 459–468 (2006)
2. Babenko, A., Lempitsky, V.: Additive quantization for extreme vector compression. In: CVPR. pp. 931–938 (2014)
3. Brandt, J.: Transform coding for fast approximate nearest neighbor search in high dimensions. In: CVPR. pp. 1815–1822 (2010)
4. Cakir, F., He, K., Bargal, S.A., Sclaroff, S.: Mihash: Online hashing with mutual information. In: ICCV. pp. 437–445 (2017)
5. Cao, Y., Qi, H., Gui, J., Li, K., Tang, Y.Y., Kwok, J.T.Y.: Learning to hash with dimension analysis based quantizer for image retrieval. TMM **23**, 3907–3918 (2021)
6. Chen, Y., Wang, S., Lu, J., Chen, Z., Zhang, Z., Huang, Z.: Local graph convolutional networks for cross-modal hashing. In: Proceedings of the 29th ACM International Conference on Multimedia. pp. 1921–1928 (2021)
7. Douze, M., Sablayrolles, A., Jégou, H.: Link and code: Fast indexing with graphs and compact regression codes. In: CVPR. pp. 3646–3654 (2018). https://doi.org/10.1109/CVPR.2018.00384
8. Huang, L., Yang, Q., Zheng, W.: Online hashing. TNNLS **29**(6), 2309–2322 (June 2018)
9. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. TPAMI **33**(1), 117–128 (2011)
10. Kong, W., Li, W.J.: Double-bit quantization for hashing. In: AAAI (2012)
11. Kong, W., Li, W.J., Guo, M.: Manhattan hashing for large-scale image retrieval. In: SIGIR. pp. 45–54 (2012)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NeurIPS. pp. 1097–1105 (2012)
13. Leng, C., Wu, J., Cheng, J., Bai, X., Lu, H.: Online sketching hashing. In: CVPR. pp. 2503–2511 (2015)
14. Li, P., Xie, H., Min, S., Zha, Z.J., Zhang, Y.: Online residual quantization via streaming data correlation preserving. IEEE Transactions on Multimedia **24**, 981–994 (2022). https://doi.org/10.1109/TMM.2021.3062480
15. Liberty, E.: Simple and deterministic matrix sketching. In: KDD. pp. 581–588. ACM (2013)
16. Lin, M., Ji, R., Liu, H., Sun, X., Chen, S., Tian, Q.: Hadamard matrix guided online hashing. IJCV **128**(8), 2279–2306 (2020)
17. Lin, M., Ji, R., Sun, X., Zhang, B., Huang, F., Tian, Y., Tao, D.: Fast class-wise updating for online hashing. TPAMI (2020)
18. Liu, C., Lian, D., Nie, M., Xia, H.: Online optimized product quantization. In: 2020 IEEE International Conference on Data Mining (ICDM). pp. 362–371. IEEE (2020)
19. Matsui, Y., Uchida, Y., Jégou, H., Satoh, S.: A survey of product quantization. ITE Trans. on Media Technology and Applications **6**(1), 2–10 (2018)
20. Norouzi, M., Blei, D.M.: Minimal loss hashing for compact binary codes. In: ICML. pp. 353–360 (2011)
21. Shen, F., Xu, Y., Liu, L., Yang, Y., Huang, Z., Shen, H.T.: Unsupervised deep hashing with similarity-adaptive and discrete optimization. TPAMI **40**(12), 3034–3044 (2018)
22. Shen, Y., Qin, J., Chen, J., Yu, M., Liu, L., Zhu, F., Shen, F., Shao, L.: Auto-encoding twin-bottleneck hashing. In: CVPR. pp. 2818–2827 (2020)

23. Silpa-Anan, C., Hartley, R.: Optimised kd-trees for fast image descriptor matching. In: CVPR. pp. 1–8 (2008)
24. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014)
25. Wang, J., Liu, W., Kumar, S., Chang, S.: Learning to hash for indexing big data – a survey. IEEE **104**(1), 34–57 (Jan 2016)
26. Wang, J., Zhang, T., Song, J., Sebe, N., Shen, H.T.: A survey on learning to hash. TPAMI **40**(4), 769–790 (April 2018)
27. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for large-scale search. TPAMI **34**(12), 2393–2406 (2012)
28. Wang, Y., Luo, X., Xu, X.S.: Label embedding online hashing for cross-modal retrieval. In: Proceedings of the 28th ACM International Conference on Multimedia. pp. 871–879 (2020)
29. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NeurIPS. pp. 1753–1760 (2009)
30. Weng, Z., Sun, Z., Zhu, Y.: Asymmetric hashing with multi-bit quantization for image retrieval. Neurocomputing **207**, 71–77 (2016)
31. Weng, Z., Zhu, Y.: Online hashing with efficient updating of binary codes. In: AAAI. pp. 12354–12361 (2020)
32. Weng, Z., Zhu, Y.: Online hashing with bit selection for image retrieval. IEEE Transactions on Multimedia **23**, 1868–1881 (2021)
33. Weng, Z., Zhu, Y., Lan, Y., Huang, L.K.: A fast online spherical hashing method based on data sampling for large scale image retrieval. Neurocomputing **364**, 209–218 (2019)
34. Xie, H., Mao, Z., Zhang, Y., Deng, H., Yan, C., Chen, Z.: Double-bit quantization and index hashing for nearest neighbor search. TMM **21**(5), 1248–1260 (2019). https://doi.org/10.1109/TMM.2018.2872898
35. Xu, D., Tsang, I.W., Zhang, Y.: Online product quantization. TKDE **30**(11), 2185–2198 (2018)
36. Zhan, Y.W., Wang, Y., Sun, Y., Wu, X.M., Luo, X., Xu, X.S.: Discrete online cross-modal hashing. Pattern Recognition **122**, 108262 (2022)