Online Hashing with Bit Selection for Image Retrieval

Zhenyu Weng, Yuesheng Zhu, *Senior Member, IEEE*

***Abstract*—Online hashing methods have been intensively investigated in semantic image retrieval due to their efficiency in learning the hash functions with one pass through the streaming data. Among the online hashing methods, those based on the target codes are usually superior to others. However, the target codes in these methods are generated heuristically in advance and cannot be learned online to capture the characteristics of the data. In this paper, we propose a new online hashing method in which the target codes are constructed according to the data characteristics and are used to learn the hash functions online. By designing a metric to select the effective bits online for constructing the target codes, the learned hash functions are resistant to the bit-flipping error. At the same time, the correlation between the hash functions is also considered in the designed metric. Hence, the hash functions have low redundancy. Extensive experiments show that our method can achieve comparable or better performance than other online hashing methods on both the static database and the dynamic database.**

***Index Terms*—online hashing, approximate nearest neighbor search, bit selection, target codes, image retrieval**

I. Introduction

DUE to the ever-growing amount of multimedia, it is a substantial challenge to retrieve the relevant content from a large multimedia database by performing the nearest neighbor search. As the multimedia data, such as videos and images, are usually represented by high-dimensional feature descriptors [1], [2], conventional nearest neighbor search methods such as tree-based methods [3] cannot deal with the high-dimensional data. Hence, much attention has been paid lately to the hashing methods [4]. By using the hash functions to map the high-dimensional data to the binary codes and calculating the Hamming distance between binary codes, hashing methods can perform the approximate nearest neighbor search efficiently with a small storage space for storing the binary codes. Hashing methods have been used in many multimedia applications, such as image retrieval [5], [6], [7], [8], [9], [10], [11], video retrieval [12], [13], [14], object tracking [15], person re-identification [16], feature matching [17], [18], cross-model retrieval [19], [20] and image semantic indexing [21].

By learning the hash functions according to the data, the data-dependent hashing methods have achieved good search results in image retrieval [22], [23], [24], [25], [26]. However, most of them are batch-learning methods and are confronted with challenges in the big data applications since new images are generated every minute in practice. When new images are added into the database, the new images entail new label classes, resulting in changes of the data distribution. The hash functions have to be re-learned by accumulating all the data whenever new data arrives, which is time consuming. In addition, it is prohibitively expensive to learn the hash functions by loading the large-scale database into memory at once due to the limited memory space.

To address the above challenges, online hashing methods have been developed to learn the hash functions from the growing data in an online fashion. Online hashing methods consist of two parts, hash function update [27], [28] and binary code update [29], [30]. The former part focuses on learning the effective hash functions from the data to generate compact binary codes. The later part focuses on designing a metric to decide when to update all (or part of) the binary codes since it is time consuming and unnecessary to update the binary codes whenever the hash functions are updated. These two parts are orthogonal and are employed together in practice. In this paper, we focus on the former part and propose a new online hashing method to learn the hash functions online from the data.

According to whether the label information is used in learning the hash functions, current online hashing methods can be categorized as online unsupervised hashing methods and online semi-supervised /supervised hashing methods. Online sketching hashing (OSH) [31] maintains a data sketch to preserve the distribution of the growing data and learns the hash functions from the data sketch. It is an online unsupervised hashing method. Considering the case that some data is labeled and the other data is unlabeled, online semi-supervised hashing methods [32], [33] learn the hash functions to preserve the supervised information of the labeled data and the distribution of the unlabeled data. In the case that all the data is labeled, online kernel hashing (OKH) [34] and adaptive hashing (AdaptH) [28] take the pairwise label information as the input and update the hash functions from the data. balanced similarity for online discrete hashing (BSODH) [35] uses an asymmetric graph regularization to preserve the similarity between the streaming data and the existing dataset, and then updates the hash function according to the graph. Different from the above online hashing methods that rely on the similarity preserving constraint, online hashing methods based on the target codes encode the label information into the target codes and learn the hash functions according to the target codes. By generating error correcting output codes (ECOC) as target codes, online supervised hashing (OSupH) [27] learns the hash functions to fit the target codes. To reduce the correlation between the hash functions, Hadamard codebook based online hashing (HCOH) [36] introduces the Hadamard matrix and regards each column of Hadamard matrix as the

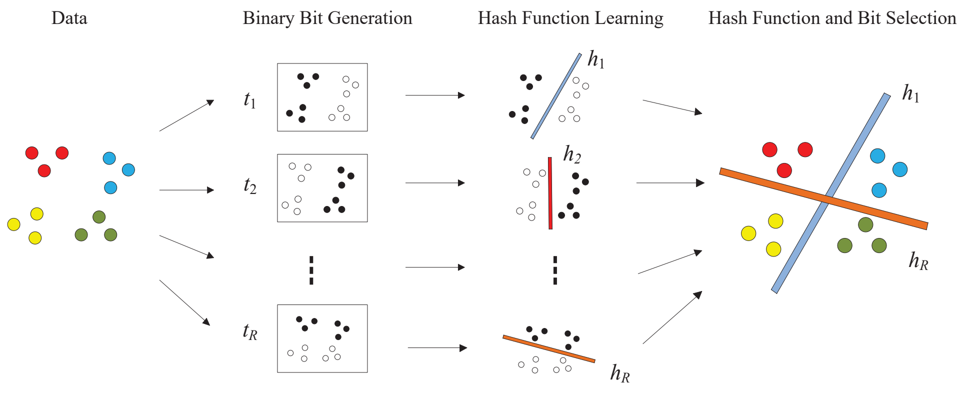


Fig. 1: The diagram of the proposed method. Each hash function h is learnt according to the corresponding bit t of the target code. And the effective hash functions are selected to generate compact binary codes with good performance.

target code for each class label. By taking the correlation between the hash functions into account, HCOH can achieve higher search accuracy than OSupH. However, the target codes in these methods only consider the correlation between hash functions but not the robustness of the hash functions to fit the target codes. To solve this problem, in this paper, we propose an online supervised hashing method to construct the target codes online that can take the correlation and the robustness of hash functions into account and learn the hash functions according to the target codes. Different from OSupH and HCOH that generate the target codes heuristically and learn the hash functions collectively according to the target codes, in our method, each hash function is treated as an independent case and learned in a passive-aggressive (PA) way [37] corresponding to one bit of the target codes. As the hash functions are learned independently, the target codes are constructed online by selecting the effective bits according to a designed metric, and the hash functions are learned according to the constructed target codes. The diagram of our method is shown in Fig. 1, which is composed of three stages, binary bit generation, hash function learning, hash function and bit selection. As our method constructs the target codes by selecting the effective binary bits, our method is called online selection hashing (OSelH). The main contributions of this paper are as follows:

a. Different from OSupH and HCOH in which the target codes are generated heuristically and the hash functions are learned collectively according to the generated codes, in our method, the hash functions are learned independently. Hence, the effective target codes can be constructed by selecting the effective bits online to reduce the correlation between hash functions and improve the robustness of hash functions, while the target codes of OSupH and HCOH are generated heuristically only in advance and cannot be learned online. To our knowledge, the existing online hashing methods based on target codes do not support constructing target codes online according to the data.

b. A metric is proposed to select the effective binary bits to construct the target codes. The metric consists of two selection criteria. One selection criterion is designed to select the bits that can make the hash functions resistant to the bit-flipping error, and the other one is designed to select the bits that can result in the low correlation between the hash functions.

c. We compare our method with other online hashing methods in both a static environment and a dynamic environment, while in prior work online hashing methods are compared in either the static environment or the dynamic environment. Extensive experiments show that our method achieves better search accuracy than other online hashing methods.

The rest of this paper is organized as follows: The related work is surveyed in Sec. II. The proposed method is elaborated in Sec. III. The experimental results and analysis are provided in Sec.IV. Finally, we conclude this paper in Sec.V.

II. Related Work

A. Batch-based hashing

Locality sensitive hashing (LSH) [38] is one of the representative hashing methods, which generates the hash functions by random projection. However, LSH is a data-independent hashing method and usually needs long binary codes to achieve good search accuracy. Compared to data-independent hashing methods, data-dependent hashing methods can generate compact binary codes by learning the hash functions from the data. The data-dependent hashing methods can be categorized as unsupervised hashing methods [39], [40], [41], [42], [43] and (semi-)supervised hashing methods [44], [45], [46], [47], [48]. The representative unsupervised hashing methods include iterative quantization (ITQ) [49], spherical hashing (SpH) [50], inductive manifold hashing (IMH) [51], jointly sparse hashing (JSH) [52], and adaptive binary quantization (ABQ) [53]. These methods mainly learn the hash functions according to the data distribution. By utilizing the supervised information, supervised hashing methods can preserve the semantic similarity in the original space and achieve better search accuracy than the unsupervised hashing methods. By taking the pointwise label information as the input, supervised discrete hashing (SDH) [46] learns the hash functions to generate the optimal binary hash codes for linear classification. By taking the pairwise label information, kernel supervised hashing (KSH) [54] and fast hashing (FastH) [55] generate the binary codes that can preserve the pairwise relationship.

Benefiting from the deep learning techniques [2], image retrieval methods based on deep learning techniques [56], [57], [58], [59], [60], [61] have recently achieved some promising results. Considering about the efficiency of the querying process, deep hashing methods [62], [63], [64], [65], [66] are proposed to learn the hash functions by using deep learning techniques and to accelerate the querying process for image retrieval by mapping image data to compact binary codes. To deal with the memory limit, deep hashing methods access the dataset multiple times and use a mini-batch of data to learn the hash functions each time.

However, the above data-dependent hashing methods are batch-based methods and cannot deal with the new data. When new data arrives, they have to re-learn the hash functions by using all the data, which is time consuming. Online hashing is a separate and distinct task within the hashing compared with deep hashing [67], which focuses on updating the hash functions online with new data. In this paper, we propose a new online hashing method to learn the hash functions efficiently with new data.

B. Zero-shot hashing

Zero-shot hashing methods [68], [69], [70] attempt to address the problem of new image classes that are not seen in the training phase due to the insufficient training data. As some image classes are not covered in the training phase, the zero-shot hashing methods describe the unseen images by transferring the supervised knowledge from the seen image classes to the unseen image classes. There are two ways for the zero-shot hashing methods to transfer the supervised knowledge. In the first way [68], [69], they use the class-attributes as the mediator and construct a shared attribute space between the seen classes and the unseen classes. In the second way [70], they use the word embedding technique to represent the class concepts, and transfer the knowledge among these classes according to the correlation between their concepts. In either way, the seen classes and the unseen classes should be highly related, and considerable auxiliary information is needed, which limits their applications. Different from zero-shot hashing methods, our method can learn the new images along with new label classes online without the auxiliary information and the limitation that the new image classes should be related to the existing image classes.

III. Online Selection Hashing

A. Hash function learning

Hashing methods aim to learn the hash functions $H(\mathbf{x}) = \{h_1(\mathbf{x}), \cdots, h_r(\mathbf{x})\}$ to map a given data vector $\mathbf{x} \in \mathbb{R}^D$ to an r-bit binary code $\mathbf{b} \in \{-1,1\}^r$, where D is the dimensionality of the data and r is the length of the binary code. Each hash function corresponds to one bit of the generated binary code. Following [27], the k^{th} hash function is defined as

$$h_k(\mathbf{x}) = \mathrm{sgn}(\mathbf{w}_k^T \mathbf{x}) \tag{1}$$

where $\mathbf{w}_k$ is a projection vector, and $\mathrm{sgn}(\cdot)$ is an elementwise function defined as

$$\mathrm{sgn}(a) = \begin{cases} -1 & a \leq 0 \\ 1 & a > 0 \end{cases} \tag{2}$$

We assume a stochastic environment in which data tuple $(\mathbf{x}, y)$ arrives sequentially, where y is the pointwise label information. In our method, y is assigned an initial code $\mathbf{t} \in \{-1,1\}^R$ at first where $R \geq r$. Then, the effective bits of the initial code are selected online to construct the target code. Different from other online hashing methods [36], [27] that learn the hash functions collectively by generating a projection matrix, our method treats each hash function as an independent case and learns each hash function according to one bit of $\mathbf{t}$. As each bit is 1 or -1, learning each hash function can be treated as a binary classification problem which can be well addressed by off-the-shelf online binary classification methods [71], [72], [37]. Following [37], we want to learn a hash function to minimize the loss $l(t_k, h_k(\mathbf{x}))$, where t_k is the k^{th} bit of the target code $\mathbf{t}$. As shown in Fig. 2, once an initial code $\mathbf{t}$ is assigned to the label, each hash function $h_k(\cdot)$ can be learned independently according to its corresponding bit of the codes.

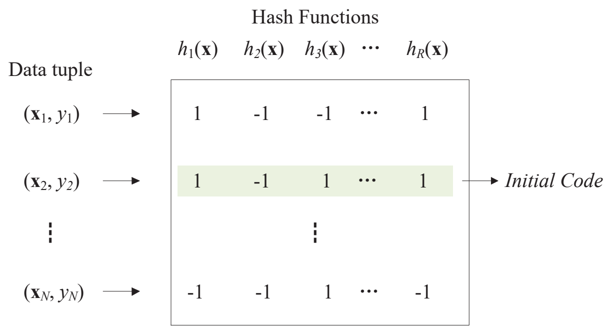


Fig. 2: Demonstration of the hash function learning.

For notational simplicity, we use $\mathbf{w}$ to denote $\mathbf{w}_k$ in the k^{th} hash function as each hash function is learned independently, and t_i to denote the corresponding bit of the target code $\mathbf{t}$ for the label y_i in the i^{th} data tuple $(\mathbf{x}_i, y_i)$. Here, the loss function for the i^{th} data tuple $(\mathbf{x}_i, y_i)$ is defined as a hinge loss function, which is $l(t_i, h(\mathbf{x}_i)) = \max(0, 1 - t_i \mathrm{sgn}(\mathbf{w}^T \mathbf{x}_i))$.

As $\mathrm{sgn}(\cdot)$ is not differentiable, resulting in the difficulty of optimizing the loss function, analogous to [44], we reformulate the loss function by replacing the sign of projection with its signed magnitude, which is

$$l'(t_i, h(\mathbf{x}_i)) = \begin{cases} 0 & t_i(\mathbf{w}^T \mathbf{x}_i) \geq 1 \\ 1 - t_i(\mathbf{w}^T \mathbf{x}_i) & otherwise \end{cases} \tag{3}$$

For brevity, we use l'_i to denote the loss $l'(t_i, h(\mathbf{x}_i))$ for the i^{th} data tuple $(\mathbf{x}_i, y_i)$. Hence, the global loss that accumulates the hinge losses over all the data tuples is:

$$L = \sum_i l'_i \tag{4}$$

To minimize the global loss, we apply the PA algorithm [37] iteratively over the data tuples to optimize $\mathbf{w}$. The

PA algorithm has been proven as a very successful and popular online learning technique for solving many real world applications [73]. First, $\mathbf{w}$ is initialized with a random value. Then, with each training tuple $(\mathbf{x}_i, y_i)$, we solve the following convex problem with soft margin:

$$\begin{aligned} &\mathbf{w}_i = \arg\min_{\mathbf{w}} \tfrac{1}{2}||\mathbf{w} - \mathbf{w}_{i-1}||^2 + C\xi \\ &s.t. \quad l'_i \leq \xi \; and \; \xi \geq 0 \end{aligned} \tag{5}$$

where $||\cdot||$ is L2-norm and C is the aggressiveness parameter that controls the trade-off between maintaining $\mathbf{w}$ close to the previous parameter $\mathbf{w}_{i-1}$ and minimizing the current loss l'_i.

When $l'_i = 0$, there is no need to change $\mathbf{w}$, so $\mathbf{w}_i = \mathbf{w}_{i-1}$. Otherwise, we use Lagrangian

$$\begin{aligned} \mathcal{L}(\mathbf{w}, \tau, \xi, \lambda) = \tfrac{1}{2}||\mathbf{w} - \mathbf{w}_{i-1}||^2 + C\xi + \\ \tau(1 - \xi - t_i\mathbf{w}^T\mathbf{x}_i) - \lambda\xi \end{aligned} \tag{6}$$

where τ and λ are Lagrange multipliers.

Let $\frac{\partial\mathcal{L}(\mathbf{w},\tau,\xi,\lambda)}{\partial\mathbf{w}} = 0$; we have

$$\frac{\partial\mathcal{L}(\mathbf{w}, \tau, \xi, \lambda)}{\partial\mathbf{w}} = \mathbf{w} - \mathbf{w}_{i-1} - \tau t_i\mathbf{x}_i = 0 \tag{7}$$

Thus, the new $\mathbf{w}$ is

$$\mathbf{w} = \mathbf{w}_{i-1} + \tau t_i\mathbf{x}_i \tag{8}$$

Let $\frac{\partial\mathcal{L}(\mathbf{w},\tau,\xi,\lambda)}{\partial\xi} = 0$; we have

$$\frac{\partial\mathcal{L}(M, \tau, \xi, \lambda)}{\partial\xi} = C - \tau - \lambda = 0 \tag{9}$$

Plugging Eqn. (8) and Eqn. (9) into Eqn. (6), we have

$$\mathcal{L}(\tau) = \frac{1}{2}\tau^2 t_i^2||\mathbf{x}_i||^2 + \tau(1 - t_i(\mathbf{w}_{i-1} + \tau t_i\mathbf{x}_i)^T\mathbf{x}_i) \tag{10}$$

Let $\frac{\partial\mathcal{L}(\tau)}{\partial\tau} = 0$, then

$$\frac{\partial\mathcal{L}(\tau)}{\partial\tau} = -\tau t_i^2||\mathbf{x}_i||^2 + l'_i = 0 \tag{11}$$

Therefore, we obtain

$$\tau = \frac{l'_i}{t_i^2||\mathbf{x}_i||^2} \tag{12}$$

As $t_i \in \{-1, 1\}$, t_i^2 is always 1. Hence, Eqn. (12) is equal to

$$\tau = \frac{l'_i}{||\mathbf{x}_i||^2} \tag{13}$$

According to Eqn. (9), since $\lambda \geq 0$, $\tau \leq C$. Therefore, according to Eqn. (8) and Eqn. (13), the optimal $\mathbf{w_i}$ is

$$\mathbf{w}_i = \mathbf{w}_{i-1} + \tau t_i\mathbf{x}_i \tag{14}$$

where

$$\tau = \min\{C, \frac{l'_i}{||\mathbf{x}_i||^2}\} \tag{15}$$

Let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)$ be a sequence of examples where $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \{-1, 1\}$, and $||\mathbf{x}_i|| \leq R$ for all i. Assume $\mathbf{w}^*$ is the best projection vector obtained in hindsight and l_i^* is the loss suffered by $\mathbf{w}^*$. As each hash function is learned by the PA algorithm, according to Theorem 4 in [37], the prediction mistakes of the hash function are bounded by

$$\max\{R^2, 1/C\}(||\mathbf{w}^*||^2 + 2C\sum_{i=1}^{T} l_i^*) \tag{16}$$

The detailed proof of the bound of the PA algorithm can be seen in [37].

B. Initial code generation

For OSupH [27], a sufficiently large set of target codes is constructed overhead. When a new label is observed, a new code from the code set is randomly selected for this label.

However, constructing a large set of target codes overhead occupies storage space. In our method, when a new label is observed, we randomly generate an R-bit ($R \geq r$) initial code to it. Then, we select r bits from the initial code to construct the target code according to the designed metric that will be introduced in the next subsection. For each bit of the initial codes, the value is generated as 1 or -1 with the same probability. We can prove that the two different labels have a low probability to share the same code.

For any two initial codes, the probability of one bit of them being the same is $p_{same} = 1/2$. Therefore, the probability of any two initial codes being the same is

$$P = p_{same}^R = \frac{1}{2^R} \tag{17}$$

Hence, according to Eqn. (17), any two different labels share the same code with a low probability of $1/2^R$.

C. Hash function and bit selection

Since the R-bit initial codes are generated and there are R hash functions each of which is learned independently according to one bit of the initial codes, we can select r effective bits to construct the target codes, and each of these effective bits has one corresponding hash function. Essentially, the bit selection process is the hash function selection process.

To find the bits for which the corresponding hash functions are resistant to the bit-flipping error, we calculate the number of the right predictions (*i.e.*, $l(t_i, h(\mathbf{x}_i)) = 0$) each hash function makes during the online learning process. Each hash function is assigned a counter c. Each time that the prediction loss for the k^{th} hash function is zero, the corresponding counter c_k is incremented by 1. Obviously, the larger the value of the counter is, the better the robustness the hash function has to resisting the bit-flipping error . Assume there are N data tuples that are received for learning the hash functions currently. The counter score is defined as

$$c'_k = c_k/N \tag{18}$$

where the range of c'_k is [0,1].

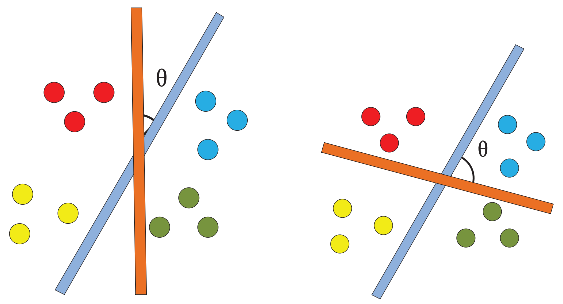
The objective is to find r hash functions that have the highest counter scores among R hash functions, which can be formulated as

$$ind = \arg\max_{|\{i\}|=r} \sum_{i\in\{i\}} c'_i \tag{19}$$

where ind is an array storing the index of r selected hash functions

Eqn. (19) is solved by taking the top r hash functions from R hash functions where the hash functions are ranked from the largest to the smallest according to their counter scores.

During the process of online learning, since each hash function is learned independently, the projection vectors in the hash functions may become progressively correlated, leading to degraded performance. In Fig. 3, each projection vector in both subfigures can divide the data into two classes well. However, in the left subfigure, the projection vectors are correlated and can just discriminate two classes, while the projection vectors in the right subfigure can discriminate four classes. As indicated in [28], the uncorrelation among the hash functions is important for avoiding redundancy in the hash functions and for attaining good performance with compact codes. Hence, we should select the hash functions with low redundancy.



(a) small nonobtuse angle θ (b) large nonobtuse angle θ

Fig. 3: Demonstration of the hash functions with different nonobtuse angles.

For the principal component analysis (PCA) method, the projection vectors are orthogonal and thus uncorrelated. Each projection vector captures its unique information. Inspired by PCA, we can use nonobtuse angle θ to measure the correlation of a pair projection vectors, as shown in Fig. 3. The larger the nonobtuse angle θ is, the higher degree of uncorrelation the pair of projection vectors has.

The similarity $sim(\cdot,\cdot)$ of any two vectors is defined as follows

$$sim(\mathbf{w}_i,\mathbf{w}_j) = |\cos\ \theta| = |\frac{\mathbf{w}_i^T\mathbf{w}_j}{||\mathbf{w}_i||^2||\mathbf{w}_j||^2}| \tag{20}$$

where the range of $sim(\cdot,\cdot)$ is [0,1].

This measure is invariant to scale, translation, rotation and orientation of the two vectors. The larger the value of the measure is, the higher the correlation the vectors have. It is also used by [74] in a regularizer imposed over the latent factors to encourage them to approach orthogonality.

By combining the similarity measure with the counter score, the objective to find the hash functions that have high counter scores and low correlation can be formulated as

$$ind = \arg\max_{|\{i\}|=r} \sum_{i\in\{i\}} c'_i \prod_{i\neq j, j\in\{i\}} 1 - sim(\mathbf{w}_i,\mathbf{w}_j) \tag{21}$$

Eqn. (21) is a combinatorial optimization problem and thus the space of possible solutions is typically too large to search exhaustively. Alternatively, we can solve Eqn. (21) with an approximate solution. We propose a greedy algorithm to find r hash functions iteratively by making Eqn. (21) maximum for the selected hash functions at each iteration.

At first, the hash functions are ranked according to their counter scores, and the largest one is selected and stored as $ind(1)$, which is

$$ind(1) = \arg\max_i c'_i \tag{22}$$

Assume we have selected $t-1$ hash functions. To make Eqn. (21) maximum for the currently selected hash function, the search for the t^{th} selected hash function can be formulated as

$$ind(t) = \arg\max_i\ (c_{i'} + \sum_{j\in ind(1:t-1)} c_{j'})\beta_{t-1} \tag{23}$$

where $\beta_{t-1} = \prod_{j\in ind(1:t-1)} 1 - sim(\mathbf{w}_i,\mathbf{w}_j)$.

Obviously, the t^{th} selected hash function can be found by enumerating the hash functions that are not in the array ind.

Therefore, with r iterations, we can select r hash functions from the pool to generate the binary codes.

To alleviate the precision problem existing in Eqn. (23), we can replace the multiplication with the $\log(\cdot)$ function. Therefore, Eqn. (23) is rewritten as

$$\begin{aligned} ind(t) =& \arg\max_i \log(c'_i + \sum_{j\in ind(1:t-1)} c'_j + \eta) + \\ & \sum_{j\in ind(1:t-1)} \log(1 - sim(\mathbf{w}_i,\mathbf{w}_j) + \eta) \end{aligned} \tag{24}$$

where η is very small number (such as 1e-9 in this paper) to avoid the case of log(0).

D. Complexity analysis

The pseudocode of our method, online selection hashing (OSelH), is given in Algorithm 1. First, a pool of R hash functions is constructed. For each data tuple $(\mathbf{x}_i, y_i)$, if y_i is a new label, an R-bit initial code is randomly generated for this label. The label and its corresponding initial code are stored in the sets Υ and T, respectively. Otherwise, the corresponding initial code for this label is selected from the code set. In the pseudocode, the function $find(\mathrm{T},\Upsilon,y_i)$ is used to find the corresponding code $\mathbf{t}_i$ for the label y_i, which can be implemented by brute force. Then, R hash functions are updated according to their corresponding bit of the code. After R hash functions are updated, we can select r hash functions from them.

Time complexity: Following the settings in OSH [31], assume there is a stream of data chunks and we learn new hash function for every chunk. Currently, there are m chunks and N data points for m chunks. The proposed method is composed of three parts, the target codes generation, the hash function learning, and the hash function selection. Each time when receiving a new data tuple, a target code is obtained by searching from the existing code set with $T_{search} = O(p)$,

Algorithm 1 Online Selection Hashing

```
Input: data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the length of the binary code $r$, the
    number of the candidate hash functions $R$
Output: hash functions $\{h_k(\cdot)\}_{k=ind(1)}^{ind(r)}$
 1: $\Upsilon = \{\}$
 2: $\mathrm{T} = \{\}$
 3: Initializing $H = \{h_k(\cdot)\}_{k=1}^R$
 4: for $i$ = 1 to $N$ do
 5:     if $y_i \notin \Upsilon$ then
 6:         Randomly generating a new target code $\mathbf{t}$
 7:         $\Upsilon = [\Upsilon\ y_i]$
 8:         $\mathrm{T} = [\mathrm{T}\ \mathbf{t}]$
 9:     end if
10:     $\mathbf{t}_i = find(\mathrm{T}, \Upsilon, y_i)$
11:     for $j$ = 1 to $R$ do
12:         Updating the hash function $h_j(\cdot)$ according to Eqn. (14)
13:     end for
14: end for
15: for $j$ = 1 to $r$ do
16:     if $j$ == 1 then
17:         Finding the hash function that has the largest counter
            score according to Eqn. (22) and storing it into $ind(j)$
18:     else
19:         Choosing the index $k$ of the hash function from the
            pool according to Eqn. (24) and storing it into $ind(j)$
20:     end if
21: end for
```

where p is the size of the code set, or by randomly generating with $T_{generation} = O(R)$, where R is the length of the initial code. It needs $O(D)$ to update each hash function, where D is the dimensionality of the data. Hence, the time complexity of updating the hash functions in the pool is $T_{update} = O(DR)$. The time complexity of learning the hash functions is

$$\begin{aligned} T_{learning} &= N(T_{search} + T_{generation} + T_{update}) \\ &= O(Np + NDR) \end{aligned} \tag{25}$$

To select the hash functions from the pool, in each iteration, it needs $T_{score} = O(D)$ to calculate a score for a hash function. Since there are R hash functions and r iterations, the time complexity of selecting the hash functions is

$$T_{selection} = rRT_{score} = O(mrRD) \tag{26}$$

The total time complexity of our method is

$$\begin{aligned} T &= T_{learning} + T_{selection} \\ &= O(Np + NDR + mrDR) \end{aligned} \tag{27}$$

The time complexity of HCOH is $O(NrD)$. As HCOH generates many target codes in advance and takes the target code according to the label directly, there is no time cost in HCOH that deals with the target code. Although our method is slower than HCOH, HCOH needs to store a considerable amount of target codes in advance, whereas our method can create a new target code only when a new class arrives which is more suitable for the real-world applications. The time complexity of OSH is $O(NDl + mDl^2 + ml^3)$, where l is the size of the data sketch. The length of the binary code does not affect OSH excessively. Hence, our method is faster than OSH for short binary codes and slower than OSH for long binary codes. The result of the training time of different online hashing methods is provided in Sec. IV.

E. Relation to the existing methods

Although there is another hash function selection method [75], it is different from our method. [75] is a batch-based method that designs the measure to select the hash functions from various batch-based hashing methods. When new data arrives, [75] has to re-learn the hash functions by accumulating all the data, which is time consuming. Instead, our method is an online hashing method, which both learns the hash functions and selects the hash functions efficiently according to new data.

Although both incremental bit learning (IBL) [33] and our method go through the hash function selection process, the significant difference is that the hash functions in IBL are learned according to the local information, whereas the hash functions in our method are learned according to the global information. In IBL, the data arrives in chunks. For each arriving chunk, new hash functions are learned by bootstrap sequential projection learning hashing (BSPLH) [76] and added into the hash functions pool. Hence, the hash functions in IBL are learned only according to the information of one chunk. In contrast, in our method, each hash function is learned according to the information of all the received data by the PA algorithm. Moreover, IBL is an online semi-supervised hashing method and our method is an online supervised hashing method.

IV. Experiments

A. Datasets

To evaluate the performance of our method, three large-scale datasets are used in the experiments.

a). Places205 dataset: Places205 [30] contains 2.5M images from 205 scene categories. For each category, we take 5,000 images as the database images and 50 images as the query images. Hence, there are 1,025,000 database images and 10,250 query images in total. In this dataset [30], each image is represented by a feature extracted from the fc7 layer of an AlexNet [2] pretrained on ImageNet and then the dimensionality is reduced to 128 by using PCA.

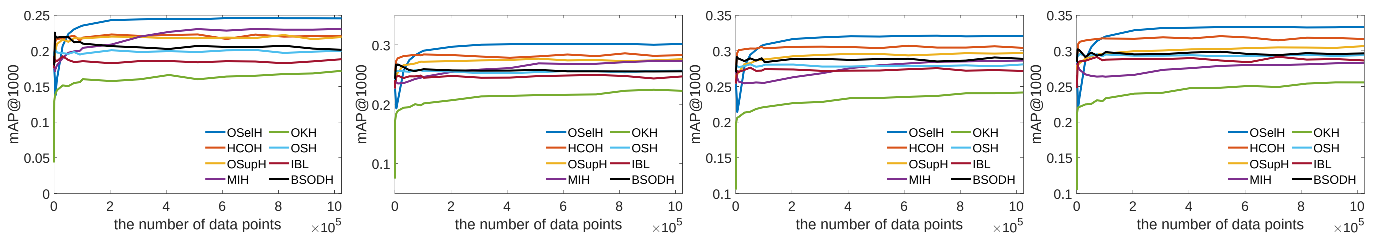
b). ImageNet dataset: ImageNet is a large-scale image database with 1.2 million images manually labeled by 2,000 classes. We randomly choose 200 classes each of which includes 1010 images. From each class, 1000 images are randomly selected as the database images and the rest as the query images. Each image is represented by a 4096-D descriptors extracted from the fc7 layer of a VGG-16 network [77] pretrained on ImageNet [78].

c). NUS-WIDE dataset: NUS-WIDE is a multilabel dataset, which includes 269,648 images. We download the images according to the given url links and remove the url links that cannot access the images. Each image is represented by

TABLE I: The mAP results (mean + variation) of different online hashing methods in the static environment. The best result is labeled with boldface, and the second best has an underline.

	Places205				ImageNet			
	32 bits	64 bits	96 bits	128 bits	32 bits	64 bits	96 bits	128 bits
OSelH	**0.245+0.002**	**0.301+0.002**	**0.320+0.002**	**0.334+0.001**	**0.481+0.004**	**0.591+0.003**	**0.630+0.003**	**0.653+0.003**
HCOH	0.221+0.003	0.282+0.002	0.304+0.002	0.316+0.001	0.479+0.005	0.588+0.004	0.628+0.002	0.652+0.002
OSupH	0.219+0.003	0.275+0.001	0.296+0.002	0.306+0.001	0.262+0.003	0.335+0.006	0.349+0.006	0.346+0.001
MIH	0.225+0.001	0.270+0.002	0.280+0.004	0.285+0.004	0.204+0.003	0.317+0.004	0.175+0.006	0.198+0.002
OKH	0.171+0.002	0.222+0.001	0.241+0.001	0.255+0.001	0.163+0.004	0.248+0.001	0.279+0.001	0.295+0.001
OSH	0.200+0.001	0.256+0.001	0.280+0.001	0.293+0.001	0.166+0.001	0.233+0.001	0.267+0.002	0.284+0.002
IBL	0.188+0.003	0.247+0.002	0.271+0.001	0.286+0.001	0.133+0.003	0.219+0.002	0.257+0.003	0.286+0.003
BSODH	0.201+0.003	0.255+0.001	0.288+0.001	0.296+0.001	0.319+0.002	0.416+0.002	0.453+0.001	0.476+0.002

	NUS-WIDE			
	32 bits	64 bits	96 bits	128 bits
OSelH	**0.725+0.001**	**0.742+0.001**	**0.745+0.001**	**0.746+0.001**
HCOH	0.658+0.001	0.698+0.002	0.717+0.001	0.732+0.001
OSupH	0.611+0.001	0.631+0.002	0.640+0.001	0.642+0.002
MIH	0.635+0.003	0.649+0.001	0.647+0.001	0.623+0.001
OKH	0.410+0.002	0.468+0.002	0.490+0.002	0.516+0.001
OSH	0.489+0.002	0.513+0.001	0.522+0.001	0.535+0.002
IBL	0.613+0.002	0.613+0.002	0.614+0.002	0.615+0.003
BSODH	0.611+0.002	0.613+0.001	0.620+0.001	0.629+0.001



(a) 32 bits (b) 64 bits (c) 96 bits (d) 128 bits

Fig. 4: The mAP results on Places205 with training data increasing in the static environment.

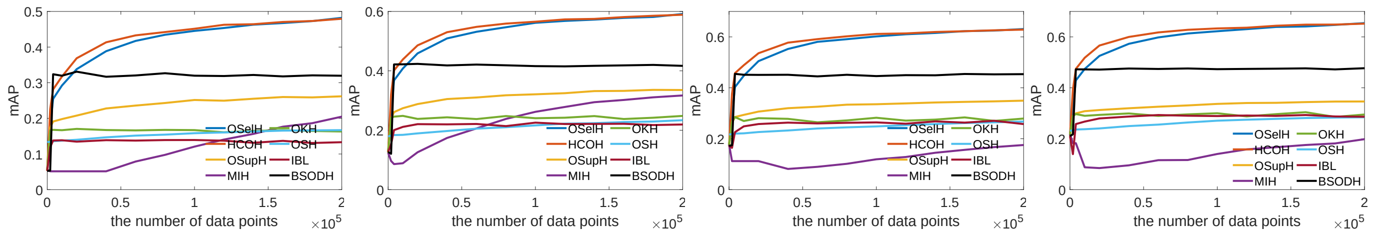
a 4096-D feature extracted from the fc7 layer of a VGG-16 network [77] pretrained on ImageNet. Following [67], we only select the 21 most frequent concepts, and this corresponds to 195,834 images. If two images share at least one common label, they are defined as a groundtruth neighbor. For each category, we randomly take 50 images as the query images and the rest as the database images.

The results are averaged by repeating the experiments 10 times. All experiments are independently run on a personal computer an I7 CPU with 24-GB memory and 64-b Windows 7 system.

B. Comparison in a statistic environment

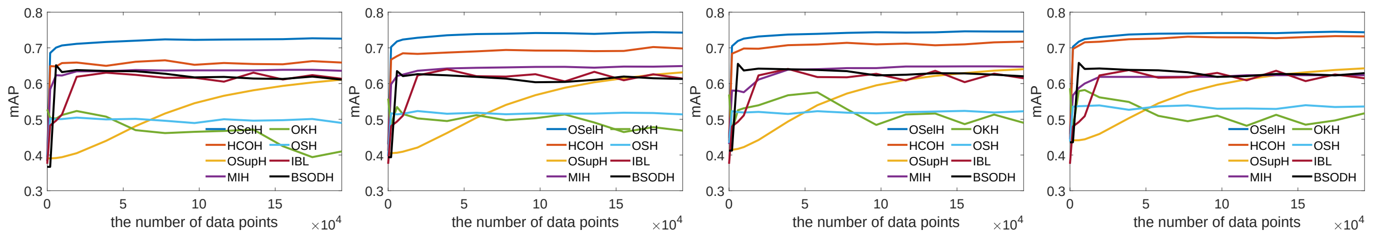
Following [36], we use the whole database of images for training and searching, and the entirety of query images as queries. The mean average precision (mAP) is used to evaluate the performance of different online hashing methods on the datasets. For Places205, the mAP result on the top 1000 retrieved images (mAP@1000) is used due to its large scale.

We compare our method, OSelH, with other online hashing methods. These online hashing methods are HCOH [36], OKH [34], MIH [30], OSupH [27], OSH [31], BSODH [35], and IBL [33]. All the online hashing methods learn the hash functions with one pass through the data. OKH is based on pairwise data input and we randomly sample data pairs in sequence from the database images. IBL is an online semisupervised hashing method and we carefully implement its code following the instructions of [33]. Since OSH, BSODH, and IBL learn the hash functions with one chunk rather than a single instance at a time, the training data from Places205 is divided into 1000 chunks and 100 chunks for the other datasets. For IBL, 100 images in the chunk are selected as the labeled images and the rest as the unlabeled images. Except for IBL, all the compared hashing methods



(a) 32 bits (b) 64 bits (c) 96 bits (d) 128 bits

Fig. 5: The mAP results on ImageNet with training data increasing in the static environment.



(a) 32 bits (b) 64 bits (c) 96 bits (d) 128 bits

Fig. 6: The mAP results on NUS-WIDE with training data increasing in the static environment.

are provided by the authors. For MIH, its training time is related to the size of the sample reservoir. Specifically, a larger sample reservoir results in higher accuracy and a longer training time. As a trade-off, the size of the reservoir is set to be 200 here. For HCOH, we find that HCOH can achieve good results by setting the learning rate to 0.1 for Places205, and 0.2 for other datasets. For other compared methods, the key parameters in the methods are set as the ones recommended in the corresponding papers.

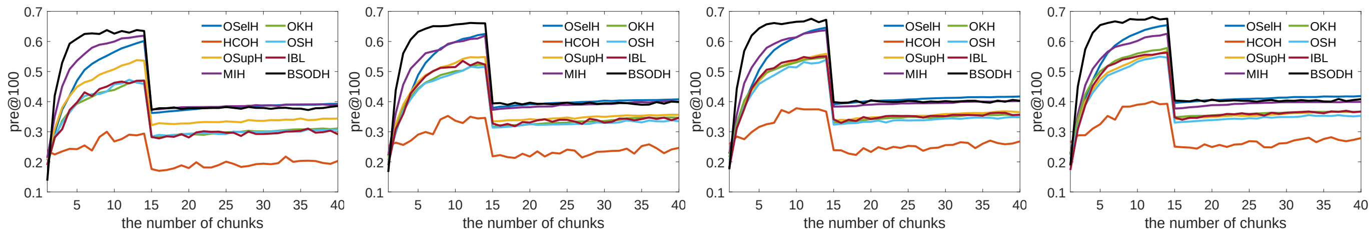
The mAP results of different online hashing methods on three datasets with the length of the binary codes varying from 32 bits to 128 bits are shown in Table I. For all the datasets, OSelH achieves the best performance among the online hashing methods. HCOH is inferior to OSelH but achieves better performance than other online hashing methods. It shows the superiority of the online hashing methods based on target codes. Moreover, as OSelH takes both the correlation and the robustness of hash functions into account, OSelH is better than HCOH.

Fig. 4, Fig. 5, and Fig. 6 show the performance of different online hashing methods with the training data increasing on Places205, ImageNet and NUS-WIDE, respectively. On Places205, by generating the target codes heuristically, HCOH and OSupH can achieve good performance from the beginning. OSelH is inferior to HCOH and OSupH at first, but the performance increase of OSelH with the increasing training data is larger than that of these two methods. With a few data points, OSelH achieves a better search accuracy than HCOH and OSupH. On ImageNet, although BSODH achieves a good search accuracy at the beginning, its performance increase with the increasing training data is not obvious. Although HCOH also achieves a good search accuracy at the beginning, its performance increase is slower than that of OSelH. Hence, as the training data grows, the search performance of OSelH is better than that of BSODH and competitive with HCOH. On NUS-WIDE, HCOH also achieves a good search accuracy at the beginning, but its performance increase is slower than that of OSelH. Hence, as the training data grows, OSelH is better than HCOH. These finds demonstrate the superiority of learning the target codes according to the data.

C. Comparison in a dynamic environment

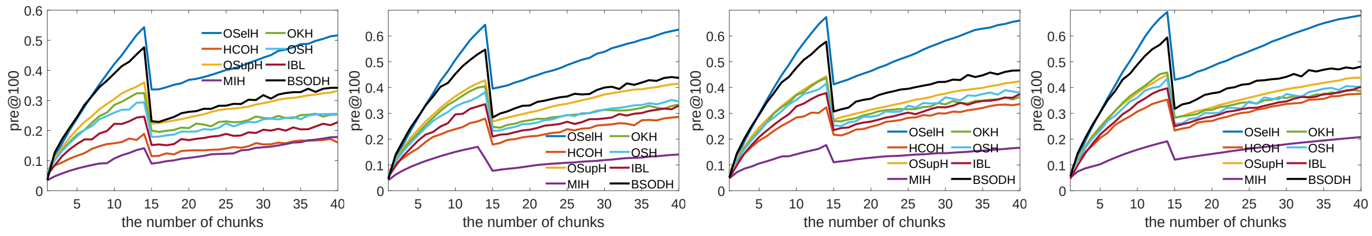
Different from the above experiments that evaluate the online hashing methods on a static database, in this subsection, we evaluate the performance of online hashing methods by constructing a dynamic database for retrieval. We simulate a growing database by adding one chunk of database images into the growing database at a time following the literature [33]. At each time, a data chunk is sampled from the database images, and the size of the chunk is set to 1000. The precision of the top 100 retrieved images (Pre@100) is used as an evaluation metric.

On Places205, 30 classes are randomly selected to construct the appearing data chunks initially, and the same classes of query images are used as the queries. After the 14^{th} iteration, the images from 20 new classes that are randomly selected are added into newly appearing data chunks, and the corresponding query images are added into the query process. Fig. 7 shows the precision results of hashing methods from 32 bits to 128 bits with the chunks arriving continuously. From the four subfigures, we can see that the performance of all the methods drops at the 15^{th} iteration since new classes of images are added into the database and it increases the difficulty of retrieving similar images. According to the results, BSODH and MIH are initially better than our method and other online hashing methods. As the data grows, especially with new classes, OSelH can achieve better search accuracy than do BSODH and MIH. These findings demonstrate the superiority of OSelH as the data grows. Although HCOH performs well on the static database, it performs poorly on the growing database.



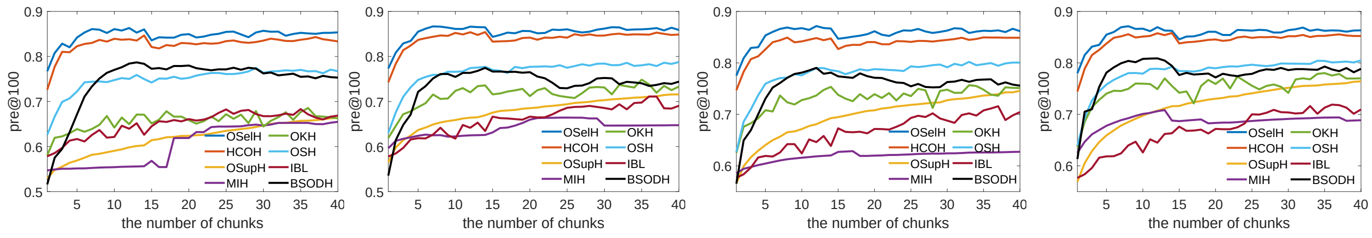
(a) 32 bits (b) 64 bits (c) 96 bits (d) 128 bits

Fig. 7: The pre@100 results on Places205 with training data increasing in the dynamic environment.



(a) 32 bits (b) 64 bits (c) 96 bits (d) 128 bits

Fig. 8: The pre@100 results on ImageNet with training data increasing in the dynamic environment.

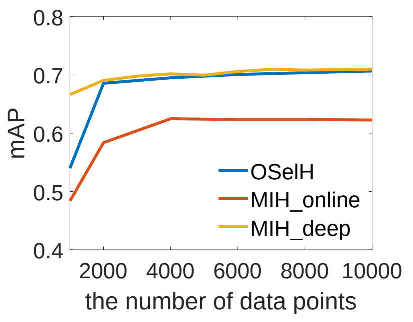


(a) 32 bits (b) 64 bits (c) 96 bits (d) 128 bits

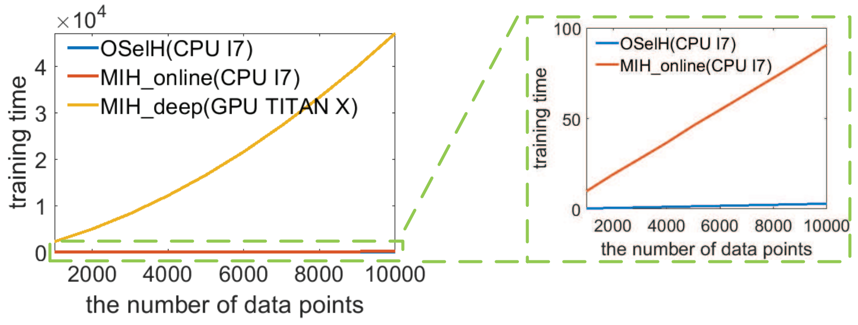
Fig. 9: The pre@100 results on NUS-WIDE with training data increasing in the dynamic environment.

TABLE II: The averaged training time of learning hash functions on the data chunk (in seconds). The best result is labeled with boldface and the second best has an underline.

	Places205				ImageNet				NUS-WIDE			
	32 bits	64 bits	96 bits	128 bits	32 bits	64 bits	96 bits	128 bits	32 bits	64 bits	96 bits	128 bits
OSelH	0.03	0.07	0.09	0.11	0.99	2.19	4.38	6.17	2.39	4.77	8.08	12.32
HCOH	**0.01**	**0.03**	**0.04**	**0.06**	**0.83**	**0.88**	**1.30**	**1.71**	**1.61**	3.05	3.71	7.40
OSupH	0.53	1.47	2.96	4.44	27.01	43.00	71.49	111.74	67.39	105.99	169.94	261.09
MIH	2.44	8.13	10.62	13.75	9.11	11.13	15.48	19.95	10.19	15.90	20.07	31.79
OKH	0.14	0.15	0.16	0.17	4.01	4.11	4.15	4.18	4.16	4.18	4.21	4.21
OSH	0.09	0.13	0.15	0.17	2.45	2.46	2.50	2.61	2.52	**2.58**	**2.60**	**2.61**
IBL	0.17	0.46	0.82	1.26	217.48	433.86	647.92	865.23	232.16	465.74	703.30	930.77
BSODH	0.64	0.98	1.43	2.16	5.75	6.04	6.49	7.14	4.05	4.32	4.65	5.40



(a) mAP



(b) time (in second)

Fig. 10: The comparison with deep hashing method on NUS-WIDE.

On ImageNet, 120 classes are randomly selected to construct the appearing data chunks initially, and the same classes of query images are used as the queries. After the 14^{th} iteration, the database images from the remaining 80 classes are added into newly appearing data chunks, and the corresponding query images are added into the query process. Fig. 8 shows that on ImageNet, OSelH is also better than the other online hashing methods, especially as the data grows.

On NUS-WIDE, 15 classes are randomly selected to construct the appearing data chunks initially, and the same classes of query images are used as the queries. After the 14^{th} iteration, the database images from the remaining 6 classes are added into newly appearing data chunks, and the corresponding query images are added into the query process. Fig. 9 shows that on NUW-WIDE, OSelH and HCOH are obviously better than other online hashing methods, and OSelH can achieve the highest search accuracy.

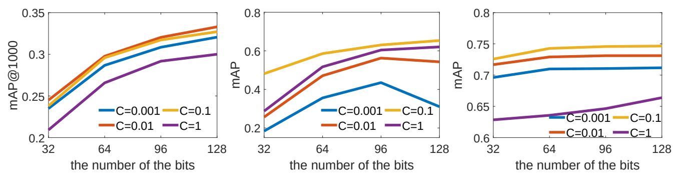
Table II shows the averaged time of different online hashing methods for learning the hash functions on the data chunk of 1000 data points. From the table, we can see that on Places205, OSelH is only slower than HCOH since there is a time cost for learning the target codes in OSelH. On ImageNet, HCOH is the fastest method. OSelH is the second fastest method for 32 bits and 64 bits, while OSH is the second fastest method for 96 bits and 128 bits. As described in Sec. III.D, the length of the binary code has a smaller effect on the training time of OSH than on that of OSelH. On NUS-WIDE, OSelH and HCOH are faster than OSH for 32 bits. Moreover, OSH is faster than HCOH and OSelH from 64 bits to 128 bits. NUS-WIDE is a multilabel dataset. OSH does not employ the label information in learning the hash functions, whereas HCOH is designed for a single-label dataset. To apply HCOH on a multilabel dataset, for each label class of one data point, HCOH has to take the corresponding target code and learn the hash function according to the target code. Hence, HCOH is slower than OSH, as is OSelH. Overall, according to the results in Table I and Table II, our method, OSelH, can achieve a good balance between effectiveness and efficiency.

D. Comparison with Deep Hashing Method

To see the difference between online hashing methods and deep hashing methods, we compare OSelH with the deep hashing version of mutual information hashing (MIH_deep) [67], which is an extension from the online hashing version of mutual information hashing (MIH_online) [30]. Fig. 10 shows the comparison results on NUS-WIDE for 32 bits in the static environment. According to the results in Fig. 10(a), OSelH can achieve a close mAP result to MIH_deep, although MIH_deep is better than MIH_online. According to the results in Fig. 10(b), MIH_online and OSelH are both much faster than MIH_deep although MIH_online as well as OSelH run on an I7 CPU and MIH_deep runs on a TITAN X GPU. When a new chunk of data arrives, MIH_deep needs to retrain the hash functions with the whole dataset and the training cost is the sum of the training cost utilizing the whole dataset and the previous training cost; however, MIH_online and OSelH just update the hash functions with the new chunk of data. Hence, MIH_online and OSelH have a smaller training cost than MIH_deep.

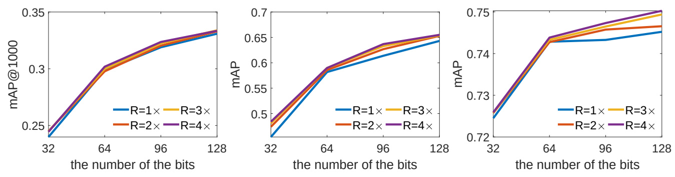
E. Evaluation of the proposed method

In the proposed method, there are two key parameters, C and R. We investigate the influence of these parameters on the static environment setting in Sec. IV.B.



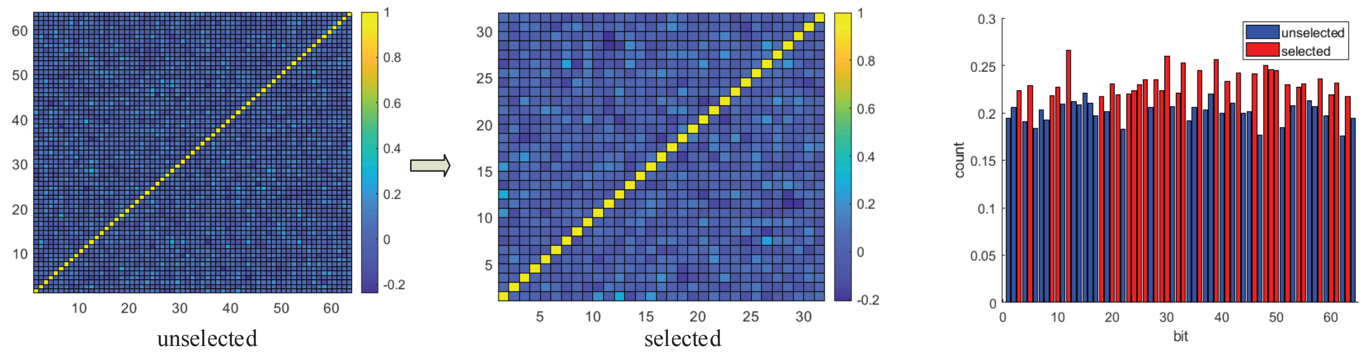
(a) Places205 (b) ImageNet (c) NUS-WIDE

Fig. 11: The mAP results of the proposed method with different C.



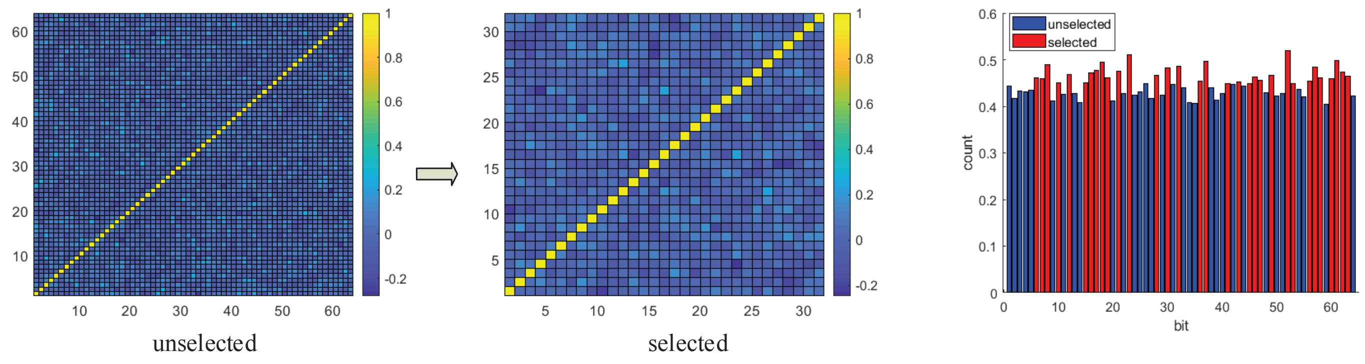
(a) Places205 (b) ImageNet (c) NUS-WIDE

Fig. 12: The mAP results of the proposed method with different R.



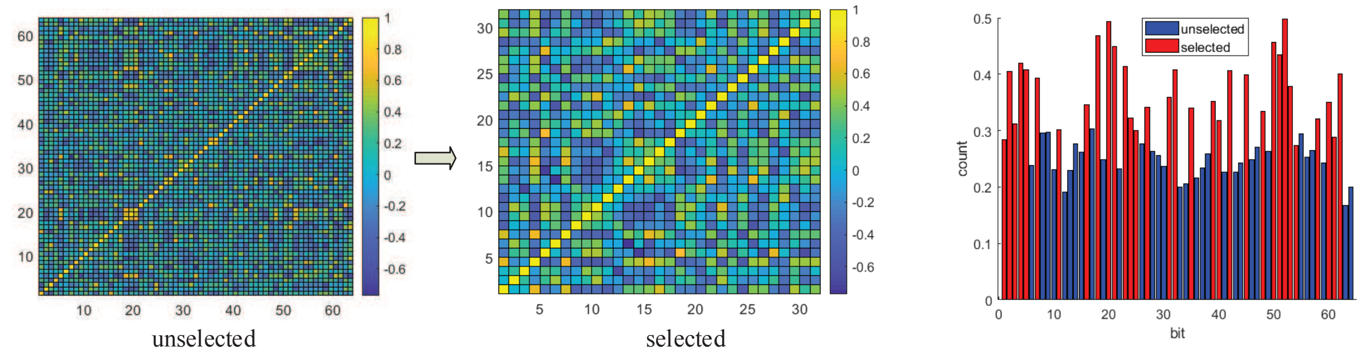
(a) correlation (b) robustness

Fig. 13: The analysis of hash functions in the proposed method on Places205.



(a) correlation (b) robustness

Fig. 14: The analysis of hash functions in the proposed method on ImageNet.



(a) correlation (b) robustness

Fig. 15: The analysis of hash functions in the proposed method on NUS-WIDE.

1). Effect of the aggressiveness parameter C: By setting the length of the initial code twice as the length of the generated binary codes, $i.e.$, $R = 2r$, we investigate the influence of the parameter C. As indicated in Eqn. (5), C is used to control the trade-off between maintaining the projection vectors close to the previous ones and minimizing the current loss. A small C means that the projection vectors should not be changed considerably. Thus, an overly small C may make the hash functions less discriminative. In contrast, a large C means that the loss of each tuple should be critically minimized. Hence, a overly large C may lead to an excessively frequent update of the projection vectors. The mAP results of the proposed method on different datasets with C varying from 0.001 to 1 are shown in Fig. 11. From the results, we can see that on all data sets, when C increases, the performance of the proposed method initially becomes increasingly better. However, when C reaches a particular threshold, further increasing C leads to performance degradation. Based on the results, C is set to 0.1 for ImageNet and NUS-WIDE and to 0.01 for Places205 to achieve the best performance.

2). Effect of the length of the initial code R: We investigate the influence of the length of the initial code R by setting C to be the suggested values above. The mAP results of the proposed method on different datasets with various R are shown in Fig. 12. The subscript from $1\times$ to $4\times$ denotes that the length of the initial code R varies from one time to four times as large as the length of the binary codes. Obviously, as R increases, there are more hash functions to select, leading to a better performance. There is a rapid increase of the accuracy from $R = 1\times$ to $R = 2\times$, and further increasing R results in a small increment of the accuracy. Although the accuracy of the proposed method is improved as the size of the pool increases, its time cost also rises. As the accuracy and the time cost are both important factors for the hashing methods, to achieve a good balance between the accuracy and the time cost, we take $R = 2\times$ for all the experiments.

3). Effect of selection: We investigate the characteristics of the proposed method regarding that it is resistant to the bit-flipping error and has low redundancy. Fig. 13, Fig. 14, and Fig. 15 show the analysis of hash functions in the proposed method for 32 bits on Places205, ImageNet, and NUS-WIDE, respectively. Fig. 13(a), Fig. 14(a), and Fig. 15(a) show the correlation between hash functions. From the figures, we can see that the proposed method can select the hash functions with a small correlation. As NUS-WIDE is a multilabel dataset while Places205 and ImageNet are a single-label dataset, selecting the hash functions with a small correlation on NUS-WIDE is more difficult than that on Places205 and ImageNet. Fig. 13(b), Fig. 14(b), and Fig. 15(b) show the ratio of the right predictions each hash function has counted during the whole learning process. From the figures, we can see that the proposed method can select the hash functions with a higher ratio, $i.e.$, better robustness to resist the bit-flipping error.

V. Conclusion

Different from other online hashing methods that generate the target codes heuristically in advance and learn the hash functions collectively according to the target codes, our proposed online supervised hashing method learns the hash functions independently. As the hash functions are learned independently, the target codes can be constructed online by selecting the effective bits, and the hash functions can be learned according to the constructed target codes. A metric is designed to select the hash functions that have high robustness to resist the bit-flipping error and have low redundancy. The experiments show that the proposed method can achieve comparable or better performance compared with other online hashing methods on both the static database and the dynamic database.

References

[1] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[3] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[4] J. Wang, T. Zhang, j. song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 40, no. 4, pp. 769–790, April 2018.

[5] L. Liu, L. Shao, F. Shen, and M. Yu, "Discretely coding semantic rank orders for supervised image hashing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5140–5149.

[6] Y. Luo, Y. Yang, F. Shen, Z. Huang, P. Zhou, and H. T. Shen, "Robust discrete code modeling for supervised hashing," *Pattern Recognition*, vol. 75, pp. 128 – 135, 2018.

[7] J. Song, L. Gao, L. Liu, X. Zhu, and N. Sebe, "Quantization-based hashing: a general framework for scalable image and video retrieval," *Pattern Recognition*, vol. 75, pp. 175 – 187, 2018.

[8] F. Shen, Y. Yang, L. Liu, W. Liu, D. Tao, and H. T. Shen, "Asymmetric binary coding for image search," *IEEE Trans. on Multimedia*, vol. 19, no. 9, pp. 2022–2032, 2017.

[9] X. Shi, F. Xing, J. Cai, Z. Zhang, Y. Xie, and L. Yang, "Kernel-based supervised discrete hashing for image retrieval," in *European Conference on Computer Vision*, 2016, pp. 419–433.

[10] M. Hu, Y. Yang, F. Shen, N. Xie, and H. T. Shen, "Hashing with angular reconstructive embeddings," *IEEE Transactions on Image Processing*, vol. 27, no. 2, pp. 545–555, Feb 2018.

[11] S. Ercoli, M. Bertini, and A. D. Bimbo, "Compact hash codes for efficient visual descriptors retrieval in large scale databases," *IEEE Trans. on Multimedia*, vol. 19, no. 11, pp. 2521–2532, 2017.

[12] Y. Hao, T. Mu, R. Hong, M. Wang, N. An, and J. Y. Goulermas, "Stochastic multiview hashing for large-scale near-duplicate video retrieval," *IEEE Trans. on Multimedia*, vol. 19, no. 1, pp. 1–14, Jan 2017.

[13] D. Zhai, X. Liu, X. Ji, D. Zhao, S. Satoh, and W. Gao, "Supervised distributed hashing for large-scale multimedia retrieval," *IEEE Trans. on Multimedia*, vol. 20, no. 3, pp. 675–686, 2018.

[14] V. E. Liong, J. Lu, Y. Tan, and J. Zhou, "Deep video hashing," *IEEE Trans. on Multimedia*, vol. 19, no. 6, pp. 1209–1219, June 2017.

[15] L. Zhang, H. Lu, D. Du, and L. Liu, "Sparse hashing tracking," *IEEE Trans. on Image Processing*, vol. 25, no. 2, pp. 840–849, Feb 2016.

[16] F. Zhu, X. Kong, L. Zheng, H. Fu, and Q. Tian, "Part-based deep hashing for large-scale person re-identification," *IEEE Trans. on Image Processing*, vol. 26, no. 10, pp. 4806–4817, Oct 2017.

[17] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua, "Ldahash: Improved matching with smaller descriptors," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 34, no. 1, pp. 66–78, 2012.

[18] J. M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y. H. Jen, E. Dunn, B. Clipp, and S. Lazebnik, "Building rome on a cloudless day," in *European Conference on Computer Vision*, 2010, pp. 368–381.

[19] M. Hu, Y. Yang, F. Shen, N. Xie, R. Hong, and H. T. Shen, "Collective reconstructive embeddings for cross-modal hashing," *IEEE Transactions on Image Processing*, pp. 1–1, 2018.

[20] D. Hu, F. Nie, and X. Li, "Deep binary reconstruction for cross-modal hashing," *IEEE Trans. on Multimedia*, vol. 21, no. 4, pp. 973–985, April 2019.

[21] Y. Yang, F. Shen, H. T. Shen, H. Li, and X. Li, "Robust discrete spectral hashing for large-scale image semantic indexing," *IEEE Transactions on Big Data*, vol. 1, no. 4, pp. 162–171, Dec 2015.

[22] J. Tang, Z. Li, and X. Zhu, "Supervised deep hashing for scalable face image retrieval," *Pattern Recognition*, vol. 75, pp. 25 – 32, 2018.

[23] C. Deng, Z. Chen, X. Liu, X. Gao, and D. Tao, "Triplet-based deep hashing network for cross-modal retrieval," *IEEE Trans. on Image Processing*, vol. 27, no. 8, pp. 3893–3903, Aug 2018.

[24] Q. Jiang, X. Cui, and W. Li, "Deep discrete supervised hashing," *IEEE Trans. on Image Processing*, vol. 27, no. 12, pp. 5996–6009, Dec 2018.

[25] Y. Cao, B. Liu, M. Long, and J. Wang, "Hashgan: Deep learning to hash with pair conditional wasserstein gan," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2018, pp. 1287–1296.

[26] E. Yang, C. Deng, C. Li, W. Liu, J. Li, and D. Tao, "Shared predictive cross-modal deep quantization," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5292–5303, Nov 2018.

[27] F. Cakir, S. A. Bargal, and S. Sclaroff, "Online supervised hashing," *Computer Vision and Image Understanding*, vol. 156, pp. 162–173, 2017.

[28] F. Cakir and S. Sclaroff, "Adaptive hashing for fast similarity search," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1044–1052.

[29] C. Ma, I. W. Tsang, F. Peng, and C. Liu, "Partial hash update via hamming subspace learning," *IEEE Trans. on Image Processing*, vol. 26, no. 4, pp. 1939–1951, April 2017.

[30] F. Cakir, K. He, S. A. Bargal, and S. Sclaroff, "Mihash: Online hashing with mutual information," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 437–445.

[31] C. Leng, J. Wu, J. Cheng, X. Bai, and H. Lu, "Online sketching hashing," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2503–2511.

[32] W. W. Y. Ng, X. Tian, Y. Lv, D. S. Yeung, and W. Pedrycz, "Incremental hashing for semantic image retrieval in nonstationary environments," *IEEE Trans. on Cybernetics*, vol. 47, no. 11, pp. 3814–3826, Nov 2017.

[33] W. W. Y. Ng, X. Tian, W. Pedrycz, X. Wang, and D. S. Yeung, "Incremental hash-bit learning for semantic image retrieval in nonstationary environments," *IEEE Trans. on Cybernetics*, vol. 49, no. 11, pp. 3844–3858, Nov 2019.

[34] L. K. Huang, Q. Yang, and W. S. Zheng, "Online hashing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2309–2322, 2018.

[35] M. Lin, R. Ji, H. Liu, X. Sun, Y. Wu, and Y. Wu, "Towards optimal discrete online hashing with balanced similarity," in *Association for the Advancement of Artificial Intelligence*, 2019, pp. 8722–8729.

[36] M. Lin, R. Ji, H. Liu, and Y. Wu, "Supervised online hashing via hadamard codebook learning," in *Proceedings of the ACM International Conference on Multimedia*, 2018, pp. 1635–1643.

[37] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *Journal of Machine Learning Research*, vol. 7, no. Mar, pp. 551–585, 2006.

[38] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *47th Annual IEEE Symposium on Foundations of Computer Science*, 2006, pp. 459–468.

[39] W. Liu, C. Mu, S. Kumar, and S.-F. Chang, "Discrete graph hashing," in *Advances in Neural Information Processing Systems*, 2014, pp. 3419–3427.

[40] M. A. Carreira-Perpinán and R. Raziperchikolaei, "Hashing with binary autoencoders," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 557–566.

[41] X. Li, D. Hu, and F. Nie, "Large graph hashing with spectral rotation," in *Association for the Advancement of Artificial Intelligence*, 2017.

[42] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2938–2945.

[43] F. Shen, Y. Xu, L. Liu, Y. Yang, Z. Huang, and H. T. Shen, "Unsupervised deep hashing with similarity-adaptive and discrete optimization," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 40, no. 12, pp. 3034–3044, Dec 2018.

[44] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large-scale search," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 34, no. 12, pp. 2393–2406, 2012.

[45] T. Ge, K. He, and J. Sun, "Graph cuts for supervised binary coding," in *Proceedings of the European Conference on Computer Vision*, 2014, pp. 250–264.

[46] F. Shen, C. Shen, W. Liu, and H. Tao Shen, "Supervised discrete hashing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 37–45.

[47] X. Yuan, Z. Chen, J. Lu, J. Feng, and J. Zhou, "Reconstruction-based supervised hashing," *Pattern Recognition*, vol. 79, pp. 147 – 161, 2018.

[48] C. Ma, I. W. Tsang, F. Shen, and C. Liu, "Error correcting input and output hashing," *IEEE Trans. on Cybernetics*, pp. 1–11, 2018.

[49] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 35, no. 12, pp. 2916–2929, 2013.

[50] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing: Binary code embedding with hyperspheres," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, pp. 2304–2316, 2015.

[51] F. Shen, C. Shen, Q. Shi, A. van den Hengel, Z. Tang, and H. T. Shen, "Hashing on nonlinear manifolds," *IEEE Trans. on Image Processing*, vol. 24, no. 6, pp. 1839–1851, June 2015.

[52] Z. Lai, Y. Chen, J. Wu, W. K. Wong, and F. Shen, "Jointly sparse hashing for image retrieval," *IEEE Transactions on Image Processing*, vol. 27, no. 12, pp. 6147–6158, Dec 2018.

[53] X. Liu, Z. Li, C. Deng, and D. Tao, "Distributed adaptive binary quantization for fast nearest neighbor search," *IEEE Trans. on Image Processing*, vol. 26, no. 11, pp. 5324–5336, Nov 2017.

[54] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2074–2081.

[55] G. Lin, C. Shen, Q. Shi, A. Van den Hengel, and D. Suter, "Fast supervised hashing with decision trees for high-dimensional data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1971–1978.

[56] R. R. Saritha, V. Paul, and P. G. Kumar, "Content based image retrieval using deep learning process," *Cluster Computing*, vol. 22, no. Supplement, pp. 4187–4200, 2019.

[57] X. Zhe, S. Chen, and H. Yan, "Directional statistics-based deep metric learning for image classification and retrieval," *Pattern Recognition*, vol. 93, pp. 113–123, 2019.

[58] Y. Li, Y. Zhang, X. Huang, and J. Ma, "Learning source-invariant deep hashing convolutional neural networks for cross-source remote sensing image retrieval," *IEEE Trans. on Geoscience and Remote Sensing*, vol. 56, no. 11, pp. 6521–6536, 2018.

[59] C. Bai, L. Huang, X. Pan, J. Zheng, and S. Chen, "Optimization of deep convolutional neural network for large scale image retrieval," *Neurocomputing*, vol. 303, pp. 60–67, 2018.

[60] C. Huang, H. Xu, L. Xie, J. Zhu, C. Xu, and Y. Tang, "Large-scale semantic web image retrieval using bimodal deep learning techniques," *Inf. Sci.*, vol. 430, pp. 331–348, 2018.

[61] I. Ha, H. Kim, S. Park, and H. Kim, "Image retrieval using bim and features from pretrained vgg network for indoor localization," *Building & Environment*, vol. 140, no. AUG., pp. 23–31.

[62] J. Bai, B. Ni, M. Wang, Z. Li, S. Cheng, X. Yang, C. Hu, and W. Gao, "Deep progressive hashing for image retrieval," *IEEE Trans. on Multimedia*, vol. 21, no. 12, pp. 3178–3193, 2019.

[63] J. Zhang and Y. Peng, "Query-adaptive image retrieval by deep-weighted hashing," *IEEE Trans. on Multimedia*, vol. 20, no. 9, pp. 2400–2414, Sep. 2018.

[64] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2475–2483.

[65] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2064–2072.

[66] E. Yang, C. Deng, W. Liu, X. Liu, D. Tao, and X. Gao, "Pairwise relationship guided deep hashing for cross-modal retrieval." in *Association for the Advancement of Artificial Intelligence*, 2017, pp. 1618–1625.

[67] F. Çakir, K. He, S. A. Bargal, and S. Sclaroff, "Hashing with mutual information," *IEEE Trans. on Pattern Anal. and Mach. Intell.*, vol. 41, no. 10, pp. 2424–2437, 2019.

[68] Y. Xu, Y. Yang, F. Shen, X. Xu, Y. Zhou, and H. T. Shen, "Attribute hashing for zero-shot image retrieval," in *IEEE International Conference on Multimedia and Expo*, vol. 00, July 2017, pp. 133–138.

[69] Y. Yang, Y. Luo, W. Chen, F. Shen, J. Shao, and H. T. Shen, "Zero-shot hashing via transferring supervised knowledge," in *Proceedings of the ACM International Conference on Multimedia*, 2016, pp. 1286–1295.

[70] Y. Guo, G. Ding, J. Han, and Y. Gao, "Sitnet: Discrete similarity transfer network for zero-shot hashing," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2017, pp. 1767–1773.

[71] D. Liu, P. Zhang, and Q. Zheng, "An efficient online active learning algorithm for binary classification," *Pattern Recogn. Lett.*, vol. 68, no. P1, pp. 22–26, Dec. 2015.

[72] K.-S. Goh, E. Chang, and K.-T. Cheng, "Svm binary classifier ensembles for image classification," in *Proceedings of the ACM International Conference on Information and Knowledge Management*, ser. CIKM '01, 2001, pp. 395–402.

[73] J. Lu, P. Zhao, and S. C. H. Hoi, "Online passive-aggressive active learning," *Proceedings of the 33th International Conference on Machine Learning*, vol. 103, no. 2, pp. 141–183, 2016.

[74] P. Xie, "Learning compact and effective distance metrics with diversity regularization," in *European Conference on Machine Learning*, 2015.

[75] X. Liu, J. He, and S. Chang, "Hash bit selection for nearest neighbor search," *IEEE Trans. on Image Processing*, vol. 26, no. 11, pp. 5367–5380, Nov 2017.

[76] C. Wu, J. Zhu, D. Cai, C. Chen, and J. Bu, "Semi-supervised nonlinear hashing using bootstrap sequential projection learning," *IEEE Trans. on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1380–1393, June 2013.

[77] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[78] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1–8.