Improved Search in Hamming Space using Deep Multi-Index Hashing

Hanjiang Lai, Yan Pan, Si Liu, Zhenbin Weng, Jian Yin .

***Abstract*—Similarity-preserving hashing is a widely-used method for nearest neighbor search in large-scale image retrieval tasks. Considerable research has been conducted on deep-network-based hashing approaches to improve the performance. However, the binary codes generated from deep networks may be not uniformly distributed over the Hamming space, which will greatly increase the retrieval time. To this end, we propose a deep-network-based multi-index hashing for retrieval efficiency. We firstly introduce the multi-index hashing (MIH) mechanism into the proposed deep architecture, which divides the binary codes into multiple substrings. Each substring corresponds to one hash table. Then, we add the two balanced constraints to obtain more uniformly distributed binary codes: 1) balanced substrings, where the Hamming distances of each substring are equal for any two binary codes, and 2) balanced hash buckets, where the sizes of each bucket are equal. Extensive evaluations on several benchmark image retrieval datasets show that the learned balanced binary codes bring dramatic speedups and achieve comparable performance over the existing baselines.**

***Index Terms*—Image Retrieval, Deep Multi-Index Hashing, Deep-Learning, Nearest Neighbor Search.**

I. Introduction

Nearest neighbor (NN) search has attracted increasing interest due to the ever-growing large-scale data on the web. Recently, similarity-preserving hashing methods that encode images into binary codes have been widely studied. Learning good hash functions for image retrieval should satisfy the two principles: 1) *accuracy* that aims to obtain good performance measured in terms of Mean Average Precision (MAP) or other metrics, and 2) *efficiency* that aims to reduce the time taken for searching in the learned binary space. In this paper, we focus on developing a deep-network-based hashing approach for retrieval efficiency.

Many algorithms have been proposed for improving the performance [1], [2], [3], [4], [5], which can be divided into three categories: unsupervised methods [6], [1], semi-supervised methods [5] and supervised methods [2], [4]. Among these methods, one of the leading approaches is the deep-networks-based hashing, which the success mainly comes from the powerful image representations learned from the deep network architectures. These methods can simultaneously learn the image representations as well as the binary hash codes, and achieve high performance. For instance, Lin et al. [7] proposed a method that learns hash codes and image representations in a point-wised manner. Li et al. [8] proposed a deep pairwise-supervised hashing (DPSH) to perform simultaneous hash code learning and feature learning. Zhao et al. [9] presented a deep semantic ranking based method for learning hash functions that preserve multilevel semantic similarity for multi-label images. Further, Zhuang [10] proposed a fast deep network for triplet supervised hashing. Liu et al. [11] proposed deep sketch hashing (DSH) for free-hand sketch-based image retrieval, etc.

The high performances have been achieved by the deep-network-based methods. It is still time-consuming to use the linear scan to find the neighbors, especially in front of the large-scale dataset (e.g., millions or billions of images). To reduce the retrieval time, several works have been proposed for efficient searching in Hamming space. One popular approach is to use binary codes as the indices into a hash table [12]. To tackle r-neighbor search, it examines all hash buckets whose indices are within r bits of a query. The main problem is that the number of distinct hash buckets to examine grows very rapidly. Thus, this method only practical for small r or short codes. Further, Norouzi et al. [3] proposed multi-index hashing (MIH) method for large r and long codes, which divides the binary codes into smaller *substrings* and build multiple hash tables. They proved that the query time of MIH is sub-linear in the size of dataset when the binary codes are uniformly distributed.

However, most of existing deep-network-based hashing methods aim at improving the performance, while completely ignoring the retrieval efficiency, that is the learned binary codes from deep networks are always not uniformly distributed over the Hamming space. In this paper, we propose a deep architecture by incorporating the MIH mechanism into the network. We consider both the accuracy and efficiency in the proposed deep network. Similar to the deep-network-based approaches, we adopt deep neural network to learn the powerful binary hash codes and maintain the good performance. Further, we propose to learn more uniformly distributed binary codes. One of the most attractive advantages of our framework is that it can reduce the search cost while preserving the performance.

More specifically, the total search cost of MIH is the cost for *neighbour candidates finding* (i.e., the number of buckets examined) plus the cost for *neighbour candidates checking* (i.e., the number of candidates checked). To reduce the search cost, we should reduce the costs for both the neighbor candidates finding and checking. First, the number of neighbor candidates is always larger than that of the true r-neighbors in MIH (please refer to Section III for more details). Thus, the minimum cost of neighbor candidates checking can

Email address: (laihanj3, panyan5, issjyin)@mail.sysu.edu.cn (H. Lai, Y. Pan, J. Yin), liusi@buaa.edu.cn (S. Liu), zbweng@gmail.com (Z. Weng)

H. Lai, Y. Pan, Z. Weng and J. Yin are with School of Data and Computer Science, Sun Yat-Sen University, China. Yan Pan is Corresponding author.

S. Liu is with Beijing Key Laboratory of Digital Media, School of Computer Science and Engineering, Beihang University, Beijing, China.

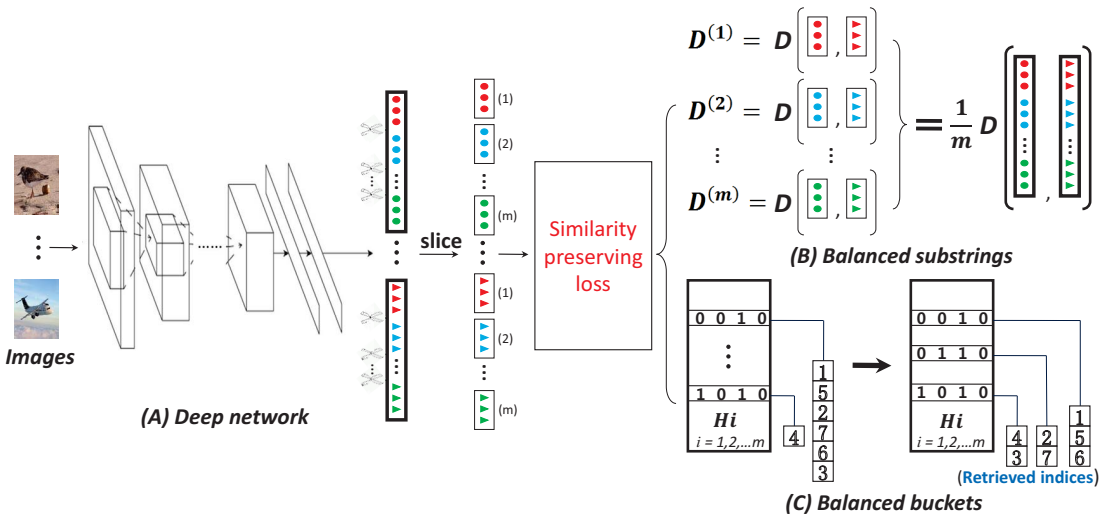


Fig. 1. An overview of the proposed deep multi-index hashing architecture. It firstly (A) encodes the images into the image representations via stacked convolutional and fully-connected layers. Then the intermediate features are divided into m slices with equal length. After that, we propose a similarity-preserving objective with two constraints for learning more uniformly distributed binary codes. The first constraint, called the balanced substrings (B), requires that there is an equal distance in each substring for any two binary codes. The second constraint, called the balanced buckets (C), lets the hash table buckets contain equal items.

be obtained when the neighbor candidates are the same as the r-neighbors. We prove that when the Hamming distances of each substring are equal for any two binary codes, which is referred as *the balanced substrings* in this paper, the number of neighbor candidates is equal to that of the r-neighbors (see Proposition 2). Thus, the balanced substrings can reduce the cost for neighbor candidates checking. Second, when the buckets are unbalanced, e.g., few buckets contain a lot of binary codes while other buckets have none/few binary codes, it will increase the cost of candidates finding or checking. When one substring of the query code locates in the bucket that contains many binary codes, it will higher the cost for candidates checking (all the binary codes in the bucket need to be checked). When the bucket contains none/few binary codes, the cost for candidates finding will be increased (the search radius need to be increased to find the specified number of neighbors). Thus, the numbers of binary codes in buckets should be equal, which is referred as *the balanced buckets*.

Base on this, we add two simple and novel constraints to reduce the total search cost, i.e., 1) the balanced substrings and 2) the balanced hash buckets. As shown in Figure. 1, our architecture consists of three main building blocks. The first block is to incorporate the MIH mechanism into our network. It learns the good image representations by the stacked convolutional and fully-connected layers followed by a slice layer which divides intermediate image features into multiple substrings, each substring corresponding to one hash table as the MIH approach. And the second and the third blocks learn the uniformly distributed binary codes, which balance the binary codes over the substrings and the hash buckets, respectively. In the second block, we balance the substrings by requiring the Hamming distances of each substring are equal for any two binary codes. According to Proposition 2, it can reduce the number of candidates. The third block is used to punish the buckets contain too many items, in which a new constraint was added to require each bucket contain at most k' items, where k' is predefined small value. After that, a similarity-preserving objective is used to capture the similarities among the images.

The main contributions of this work are three-fold.

- We propose a deep multi-index hashing, which simultaneously learns the powerful image representations and the uniformly distributed binary codes. The proposed method can maintain the high performance and reduce the retrieval time.
- We propose two simple and novel constraints to learn 1) the balanced substrings and 2) the balanced hash buckets. We show that adding these two constraints can bring dramatic speedups over the baseline without these two constraints.
- We conduct extensive evaluations on several benchmark datasets. The empirical results demonstrate the superiority of the proposed method over the state-of-the-art baseline methods.

II. Related Work

The similarity-preserving hashing methods are to encode input data into binary codes, in which the similarities among the binary codes are preserved. The representative methods include Iterative Quantization (ITQ) [1], Kernerlized LSH (KLSH) [13], Anchor Graph Hashing (AGH) [14], Spectral Hashing (SH) [15], Semi-Supervised Hashing (SSH) [5], Kernel-based Supervised Hashing (KSH) [2], Minimal Loss Hashing (MIH) [16], Binary Reconstruction Embedding (BRE) [4] and so on. The comprehensive survey can be found in [17].

Deep-network-based hashing method has emerged as one of the leading approaches. Many algorithms [18], [19], [11], [20], [10], [7], [21], [22], [9] have been proposed, including the point-wise approach, the pair-wise approach and the

ranking-based approach. The point-wise methods take a single image as input and the loss function is built on individual data. For example, Lin et al. [7] showed that the binary codes can be learned by employing a hidden layer for representing the latent concepts that dominate the class labels, thus they proposed to learn the hash codes and image representations in a point-wised manner. Yang et al. [21] proposed a loss function defined on classification error and other desirable properties of hash codes to learn the hash functions. The pair-wise methods take the image pairs as input and the loss functions are used to characterize the relationship (i.e., similar or dissimilar) between a pair of two images. Specifically, if two images are similar, then the Hamming distance between the two images should be small, otherwise, the Hamming distance should be large. Representative methods include deep pairwise-supervised hashing (DPSH) [8], deep supervised hashing (DSH) [23] and so on. The ranking-based methods cast the hashing problem as the ranking problem. Lai et al. [24] proposed a one-stage supervised hashing method which takes three images as input and preserves relative similarities of the form "image I is more similar to image I^+ than to image I^-". Zhao et al. [9] proposed a deep semantic ranking-based method for learning hash functions that preserve multi-level semantic similarity between multi-label images. Zhuang [10] proposed a fast deep network for triplet supervised hashing. The ranking-based methods can be applied in both the single-label and multi-label image retrieval. Recently, Sablayrolles et al. [25] show that the current evaluation protocols, e.g., MAP, are not satisfactory: a trivial solution that encodes the output of a classifier significantly outperforms existing supervised or semi-supervised hashing methods. Based on this, two alternative protocols are proposed for hashing: retrieval of unseen classes and transfer learning.

Multi-index hashing [26], [3] is proposed for the retrieval efficiency, which is an efficient method for finding all r-neighbors of a query by dividing the binary codes into multiple substrings. While the learned binary codes may not be uniformly distributed in practice, e.g., some buckets may contain too many items, which will cost much time to check many candidate codes. Several post-processing works [27], [28] have been proposed. Given already learned hash codes, these works reassign the bits to the substrings. For example, Norouzi et al. [3] proposed substring optimization that assigns the bits to substrings in a greedy fashion based on correlations between bits. Similar with that, Liu et al. [27] and Wan et al. [29] proposed data-oriented multi-index hashing, respectively. They first calculated the correlation matrix between bits and then rearranged the indices of bits to make a more uniform distribution in each hash table. Further, Ong et al. [30] relaxed the equal-size constraint in MIH and proposed multiple hash tables with variable length hash keys. Wang et al. [31] used repeat-bits in Hamming space to accelerate the searching but need more storage space. Song et al. [32] proposed a distance-computation-free search scheme for hashing.

Similar to [27], [29], [3], we also propose a method to learn more uniformly distributed binary codes. The main differences are that 1) these works fall into the category of the post-processing approaches, while our method learns binary codes and balanced procedures simultaneously during the hash learning process, and 2) these works reassigns the bits to the substrings, while our method directly reduces the search cost for neighbour candidates finding and checking (the two steps that MIH finds the neighbours).

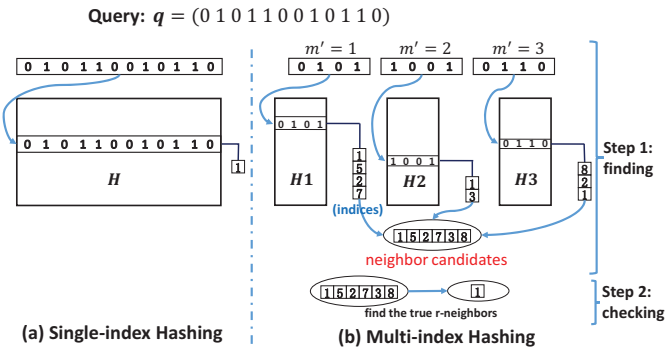


Fig. 2. An illustration of the single-index hashing and the multi-index hashing. Here, the binary codes are split into 3 hashkeys of length 4. Given a query $\mathbf{q}$, MIH finds the neighbors in two steps: neighbor candidates finding and checking. In the first step, the substrings are used to retrieve relevant example indices before a final union. In the second step, MIH finds the true neighbors from these candidates.

In this paper, we learn the image representation and the uniformly distributed hash codes together, such that the hash models and balanced procedure two tasks are learned simultaneously and help each other forward.

III. Background: Multi-Index Hashing

In this section, we briefly review MIH [3].

We first give the definition of the *r-neighbor search problem*: find all r-neighbors of a query $\mathbf{q}$, where a binary code is an *r-neighbor* of a query $\mathbf{q}$ if it differs from $\mathbf{q}$ in r bits or less. The set of the *r-neighbor* of the query $\mathbf{q}$ denotes as $\mathcal{R}^r(\mathbf{q})$.

One way to tackle r-neighbor search is to use a hash table as shown in the left side of Fig. 2. It only needs to examine the hash buckets whose indices are within r bits of a query. However, the number of hash buckets to examine is $\sum_{z=0}^{r}\binom{l}{z}$ for binary codes of l bits, which grows very rapidly. This approach is only practical for short code length and very small radii.

To solve this problem, MIH is proposed for long code lengths. As shown in the right side of Fig. 2, the binary code $\mathbf{h}$ is partitioned into m disjoint substrings, $\mathbf{h}^{(1)}, \cdots, \mathbf{h}^{(m)}$, each substring consists of $s = l/m$ bits, where we assume l is divisible by m for convenience presentation. One hash table is built for each of the m substrings. In the substring hash table, a bucket includes all codes with the same s bits. Note that the first substring hash table only considers the first s bits of all binary codes in the database, etc.

To find the r-neighbors of a query, MIH searches all substring hash tables for entries that are within a Hamming distance of $r' = r/m$ [1]. The *neighbour candidates* denote as

[1]For ease of presentation, here we assume r is divisible by m. In practice, if $r = m \times r' + a$ with $0 < a < m$, we can set the search radii of the first $a + 1$ hash tables to be r' and the rests to be $r' - 1$.

the database codes that fall close to the query in at least one such substring. Since the hash table only considers substring of s bits, not all binary codes in neighbor candidates are true neighbors of the query, so we need to check these candidates using the entire binary codes. In summary, MIH finds the neighbors in two steps: 1) neighbor candidates finding, which finds binary codes that belong to the neighbor candidates, and 2) neighbor candidates checking, which removes the not true r-neighbors.

The total search cost of MIH is the cost for candidate finding plus the cost for candidate checking. Specially, given a query $\mathbf{q}$ with substrings $\{\mathbf{q}^{(m')}\}_{m'=1}^{m}$, MIH firstly finds the neighbour candidates. It searches the m'-th substring hash table for entries that are within a Hamming distance of $r' = r/m$. The set of candidates from the m'-th substring hash table is denoted as $\mathcal{N}_{m'}^{r'}(\mathbf{q})$. The union of all the m sets, $\mathcal{N}^{r}(\mathbf{q}) = \bigcup_{m'=1}^{m} \mathcal{N}_{m'}^{r'}(\mathbf{q})$, is the superset of the r-neighbors of $\mathbf{q}$, that is $\mathcal{R}^{r}(\mathbf{q}) \subseteq \mathcal{N}^{r}(\mathbf{q})$. Note that $\mathcal{N}^{r}(\mathbf{q})$ is the neighbour candidates. Then, MIH checks all the neighbour candidates. The false positives that are not true r-neighbors of $\mathbf{q}$ are removed by computing the full Hamming distances.

The kNN *search problem* can be formulated as the r-neighbor problem. By initializing integer $r' = 0$, we can progressive increment of the search radius r' until the specified number of neighbors is found.

IV. Deep Multi-Index Hashing

This section describes deep multi-index hashing architecture that allows us to obtain 1) the powerful binary codes and 2) the ability to quickly search inside the Hamming space.

We firstly introduce notations. There is a labeled training set $\{I_i, y_i\}_{i=1}^{n}$, where I_i is the i-th image, y_i is the class name/label of the i-th image, and the number of training samples is n. We denote $\mathbf{h}_i$ as the i-th binary codes and $\mathbf{h}_i^{(m')}$ is the m'-th substring of the i-th binary codes. Let $D(\mathbf{h}_i, \mathbf{h}_j) = ||\mathbf{h}_i - \mathbf{h}_j)||_{\mathcal{H}}$ denotes the Hamming distance between the two binary codes, and each binary code comprises l bits. The $\mathbf{Hi}$ denotes the i-th substring hash table, e.g., $\mathbf{H1}$ is the first hash table. The goal of deep multi-index hashing is to learn a deep hash model, in which the similarities among the binary codes should be preserved and also can quickly search in large-scale Hamming space.

As shown in Figure 1, the purposes of the proposed architecture are two: 1) a deep network with multiple convolutional-pooling layers to capture an efficient representation of images, and followed by a slice layer to partition the feature into m disjoint substrings, and 2) a balanced module designed to address the ability to quickly search inside the Hamming space. It generates the binary codes distributed as uniform as possible from two aspects: the balanced substrings and the balanced hash buckets. In the following, we will present the details of these parts, respectively.

A. Efficient Representation via Deep Network

The deep network, e.g., AlexNet [33], VGG [34], GoogleLeNet [35] or residual network [36], is used for learning the powerful efficient image representation, which is made

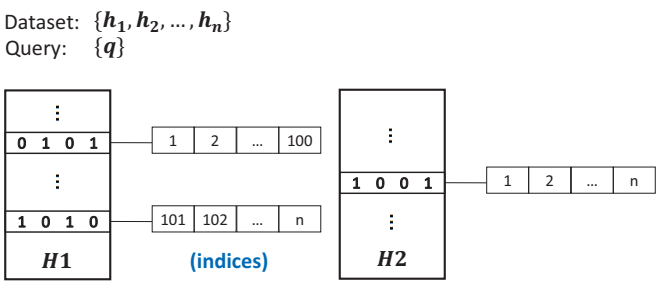


Fig. 3. An extreme example of powerful binary codes while bad searching in codes space. In such case, MIH performs worse than linear scan. Specifically, given a query vector $\mathbf{q} = [0101\ 1001]$, its substrings are firstly used to retrieve the relevant example indices: $[1, \cdots, 100]$ for the first substring and $[1, \cdots, n]$ for the second substring. The union of the two sets are the neighbour candidates: $[1, \cdots, 100] \bigcup [1, \cdots, n] = [1, \cdots, n]$. Finally, MIH computes the full Hamming distance between $\mathbf{q}$ and each candidate to retain the true neighbors of $\mathbf{q}$. The complexity of the candidates checking is $O(n)$, which is equal to that of the linear scan. Please note that (1) the substrings are not balanced, e.g., $D(\mathbf{h}_1^{(1)}, \mathbf{h}_{101}^{(1)}) = 4$ while $D(\mathbf{h}_1^{(2)}, \mathbf{h}_{101}^{(2)}) = 0$; and (2) the hash buckets are not balanced, e.g., the bucket with a hashkey "0001" has none codes, while there are 100 binary codes in the "0101" bucket in the first substring.

following structural modifications for image retrieval task. The first modification is to remove the last fully-connected layer (e.g., fc8). The second is to add a fully-connected layer with l dimensional intermediate features. The intermediate features are then fed into a tanh layer that restricts the values in the range $[-1, 1]$. The MIH contains m separate hash tables. Inspired by that, the third modification is to add a slice layer to divide the features into m slices with equal length l/m. Finally, the output of network is denoted as $\mathbf{h}_i = \mathcal{F}(I_i), \mathbf{h}_i = [\mathbf{h}_i^{(1)}, \cdots, \mathbf{h}_i^{(m)}]$, where I_i is the input image and $\mathcal{F}$ is the deep network.

The deep-network-based methods can learn very powerful hash codes for the image retrieval, while they do not consider the ability to efficiently search inside the representation space. An extreme example of powerful image representation for binary codes while bad searching is shown in Figure 3. Here the number of substrings is $m = 2$, and $\mathbf{q}$ is a query code and $\{\mathbf{h}_i\}_{i=1}^{n}$ are the database codes. Suppose that the query and the first 100 codes are similar, and dissimilar with other codes. Two hash tables are built for the n learned binary codes shown in Figure 3. The similar codes locate in the same bucket (with a similar key) and the dissimilar codes have a large Hamming distance in the first hash table. In the second hash table, all codes locate in the same bucket. The learned binary codes are very good for accuracy since the similarities are preserved and can achieve high performance that measured in terms of the MAP. However, the learned codes are very bad for searching. There are two cases cause the inefficient search: 1) the unbalanced substrings. According to Proposition 2, it will increase the number of candidate codes to be checked, and thus increase the retrieval time. And 2) the unbalanced hash buckets. In the first hash table, two hash buckets contain large of binary codes and other buckets contain none binary codes. If the first substring of query code locates in the two buckets with many codes, MIH needs to check large of neighbor candidates. It is time-consuming. Otherwise, it needs more index lookups

and it is also time-consuming.

Hence, it is necessary for finding a new way to generate more balanced binary codes.

B. Fast Search via the Deep Multi-Index Hashing

We first give the following proposition, and the proof can be found in the appendices.

Proposition 1. *When the buckets in the substring hash tables that differ from $\mathbf{q}^{(m')}$ within r' bits, i.e., $D(\mathbf{h}^{(m')}, \mathbf{q}^{(m')}) \leq r'$, then we have $\mathcal{R}^r(\mathbf{q}) \subseteq \bigcup_{m'=1}^{a+1} \mathcal{N}_{m'}^{r'}(\mathbf{q}) \subseteq \mathcal{N}^r(\mathbf{q})$, where $r = r'm + a$.*

For example, suppose that $r' = 0$, when searching in the first substring hash table, we obtain a set of candidates $\mathcal{N}_1^0(\mathbf{q})$, then the 0-neighbor (i.e., $r = r'm+a = 0$) of the query $\mathbf{q}$ is the subset of the neighbour candidates, that is $\mathcal{R}^0(\mathbf{q}) \subseteq \mathcal{N}_1^0(\mathbf{q})$. Similar with that, we have $\mathcal{R}^1(\mathbf{q}) \subseteq \bigcup_{m'=1}^{2} \mathcal{N}_{m'}^0(\mathbf{q})$, $\mathcal{R}^2(\mathbf{q}) \subseteq \bigcup_{m'=1}^{3} \mathcal{N}_{m'}^0(\mathbf{q})$ and etc. When searching in the substring hash tables differs by r' bits or less, we can obtain all r-neighbor of the query, where $r'm \leq r < (r'+1)m$.

The running time of MIH for kNN problem mainly contains two parts: candidate codes finding and checking. To achieve faster search, we should reduce 1) the number of candidate codes to be checked (the cost for candidate checking), i.e., the smaller $\bigcup_{m'=1}^{a+1} \mathcal{N}_{m'}^{r'}(\mathbf{q}) - \mathcal{R}^r(\mathbf{q})$, the better, 2) the number of distinct hash buckets to examine (the cost for finding), i.e., the smaller r', the better. To achieve these two goals, we propose to add two balanced constraints into the loss function: 1) the balanced substrings and 2) the balanced hash buckets, respectively.

1) Balanced Substrings: We first give the definition of the balanced substrings.

Balanced Substrings: the Hamming distances of each substring are equal for any two binary codes. That is suppose that $D(\mathbf{q}, \mathbf{h}) = r$ for two binary codes $\mathbf{q}$ and $\mathbf{h}$, then we have $D(\mathbf{q}^{(1)}, \mathbf{h}^{(1)}) = D(\mathbf{q}^{(2)}, \mathbf{h}^{(2)}) = \cdots = D(\mathbf{q}^{(m)}, \mathbf{h}^{(m)}) = \frac{r}{m}$. If r is not divisible by m, i.e., $r = m \times r' + a$, we can let $D(\mathbf{q}^{(1)}, \mathbf{h}^{(1)}) = \cdots = D(\mathbf{q}^{(a)}, \mathbf{h}^{(a)}) = r' + 1$ and $D(\mathbf{q}^{(a+1)}, \mathbf{h}^{(a+1)}) = \cdots = D(\mathbf{q}^{(m)}, \mathbf{h}^{(m)}) = r'$.

To show that the balanced substrings will have the minimum number of candidates checked, we give the following proposition.

Proposition 2. *Suppose that the Hamming distances of each substring are equal for any two binary codes, then the neighbor candidates are the same as the r-neighbors. Formally, suppose that $D(\mathbf{q}^{(1)}, \mathbf{h}^{(1)}) = \cdots = D(\mathbf{q}^{(a)}, \mathbf{h}^{(a)}) = r' + 1$, and $D(\mathbf{q}^{(a+1)}, \mathbf{h}^{(a+1)}) = \cdots = D(\mathbf{q}^{(m)}, \mathbf{h}^{(m)}) = r'$ for any two binary codes $\mathbf{q}$ and $\mathbf{h}$, then we have $\bigcup_{m'=1}^{a+1} \mathcal{N}_{m'}^{r'}(\mathbf{q}) - \mathcal{R}^r(\mathbf{q}) = \emptyset$.*

The proof can be found in the appendices.

When the equality holds, the number of the neighbor candidates is minimum, which is equal to the number of the r-neighbors. Hence, it has the minimum cost for neighbor candidates checking when the substrings are balanced.

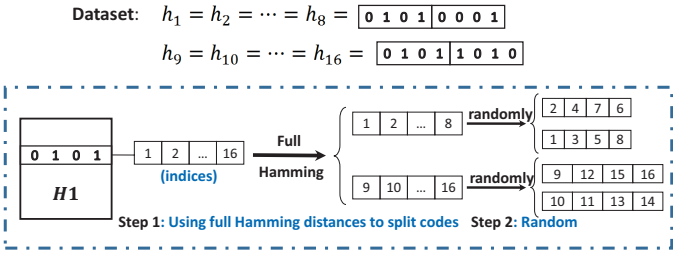


Fig. 4. An example for rebalancing the items in the buckets. Here the binary codes are split into 2 substrings of length 4. The bucket with a hash key "0101" has 16 binary codes in the first hash table. To require each bucket contains at most k' items, e.g., $k' = 4$, we first use full Hamming distance to split these items into several groups, where each group has the same binary codes. For example, the binary codes with the indices $(1, 2, \cdots, 8)$ are divided into the first group since they have the same codes. Then the items in groups are further randomly split into subgroups until all subgroups contain not more than k' items.

To reduce the cost of the candidate checking, the substrings should be balanced. We add the following new balanced constraint to our objective

$$\begin{aligned}\psi(\mathbf{h}_i, \mathbf{h}_j) = &\sum_{m'=1}^{m} \max(0, r' - D(\mathbf{h}_i^{(m')}, \mathbf{h}_j^{(m')})) \\ &+ \max(0, D(\mathbf{h}_i^{(m')}, \mathbf{h}_j^{(m')}) - (r'+1)),\end{aligned} \tag{1}$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ are any two binary codes, $r' = \lfloor D(\mathbf{h}_i, \mathbf{h}_j)/m \rfloor$. The above formulation requires the distances of each substring should be less or equal to $r'+1$ and larger or equal to r'. With this, the Hamming distances of each substring will be almost equal, i.e., $D(\mathbf{h}_i^{(1)}, \mathbf{h}_j^{(1)}) \approx D(\mathbf{h}_i^{(2)}, \mathbf{h}_j^{(2)}) \approx \cdots \approx D(\mathbf{h}_i^{(m)}, \mathbf{h}_j^{(m)})$.

2) Balanced Hash Buckets: To reduce the running time for the candidates finding, the binary codes of similar images should be indices with a similar key. In such case, $r' = 0$. Unfortunately, we need to check so many candidate binary codes (e.g., many similar codes locate in the same bucket), making the inefficient search. Thus, the number of each bucket should be not too small and not too large. In this paper, we require that each bucket in the hash table contains at most k' items.

Formally, all the buckets in all substring hash tables that have more than k' items were found. Let $P_i^{(m')} = \{p_1, \cdots, p_c\}$ is denoted as the items in the i-th bucket of the m'-th substring hash table, where c is the number of items. We use the following steps to rebalance these items as shown in Figure 4.

(1) The full Hamming distances are used to split these items into several groups, each group contains the samples which have the same binary codes. If the numbers of all groups are less than k', stop the procedure.

(2) Otherwise, if the number of the group is more than k', we further randomly split it into n_g/k' subgroups with the equal sizes, making sure each subgroup consists of at most k' items, where n_g is the number of items in the group.

To obtain the same level accuracy, a key principle should be ensured is that do not change the similarities among these images, that is the distance between the p_a-th and p_b-th

binary codes, i.e., $D(\mathbf{h}_{p_a}^{(m')}, \mathbf{h}_{p_b}^{(m')})$, should preserve relative similarities of the form "$(p_a, p_b$ in the same subgroup$) \leq (p_a, p_b$ in the same group$) \leq (p_a, p_b$ in the different groups)". Since the $D(\mathbf{h}_{p_a}^{(m')}, \mathbf{h}_{p_b}^{(m')}) = 0$ for any two p_a-th and p_b-th binary codes in current model, we need to rebalance these binary codes while preserving the performance. To achieve this, we move away these binary codes in minimum cost, which is formulated as: $\phi(P^{(m')}(i)) =$

$$\begin{cases} D(\mathbf{h}_{p_a}^{(m')}, \mathbf{h}_{p_b}^{(m')}) & \text{if } p_a, p_b \text{ in same subgroup} \\ \max(0, 1 - D(\mathbf{h}_{p_a}^{(m')}, \mathbf{h}_{p_b}^{(m')})) & \text{if } p_a, p_b \text{ in same group} \\ \max(0, 2 - D(\mathbf{h}_{p_a}^{(m')}, \mathbf{h}_{p_b}^{(m')})) & \text{if } p_a, p_b \text{ in different groups,} \end{cases} \tag{2}$$

where we let the Hamming distances between two binary codes in same subgroup to be zero (with the same hashkey). The Hamming distances in same group to be at least one and the Hamming distances in different groups to be at least two (locate in different buckets and preserve the similarities). Please note that 1 and 2 are the minimum values of the Hamming distances to move the binary codes in the same hash bucket away.

C. Full Objective

Overall, our loss function contains three terms: 1) the similarity-preserving loss, which learns to keep the similarities between the data, 2) the objective for obtaining the balanced substrings and 3) the objective to obtain the balanced hash buckets. Our full objective can be formulated as:

$$\begin{aligned} &\min \frac{1}{n_{tri}} \sum_{i,i^+,i^-} \max(0, \epsilon + D(\mathbf{h}_i, \mathbf{h}_{i^+}) - D(\mathbf{h}_i, \mathbf{h}_{i^-})), \\ &+ \frac{1}{n_{bs}} \sum_{i,j} \psi(\mathbf{h}_i, \mathbf{h}_j) + \frac{1}{n_{bb}} \sum_{i,m'} \phi(P^{(m')}(i)) \end{aligned} \tag{3}$$

where ϵ is the margin. The n_{tri} is the number of triplets, n_{bs} is the number of pairs, and n_{bb} is the number of the unbalanced codes in the hash buckets, where these three values are used to compute the mean losses of the three objectives, respectively. The first term of the objective is the triplet ranking loss which preserves relative similarities of the form "$\mathbf{h}_i$ is more similar to $\mathbf{h}_{i^+}$ than to $\mathbf{h}_{i^-}$". The second term is to generate the balanced substrings and the third term is for the balanced hash buckets.

In training, we minimize the sum of these losses via back propagation. For ease of optimization, we replace the Hamming distance with the euclidean distance, i.e., $D(\mathbf{h}_i, \mathbf{h}_j) = ||\mathbf{h}_i - \mathbf{h}_j||_2^2$, and replace the integer constraints with the range constraints, i.e., the tanh function, as suggested by [24]. In the following we present the gradients of these three kinds of losses.

The gradients of the first term with respect to $\mathbf{h}_i, \mathbf{h}_{i^+}$ and $\mathbf{h}_{i^-}$ are

$$\begin{aligned} \frac{\partial \ell_{tri}}{\partial \mathbf{h}_i} &= 2(\mathbf{h}_{i^-} - \mathbf{h}_{i^+}) \times I_{||\mathbf{h}_i - \mathbf{h}_{i^+}||_2^2 - ||\mathbf{h}_i - \mathbf{h}_{i^-}||_2^2 + \epsilon > 0}, \\ \frac{\partial \ell_{tri}}{\partial \mathbf{h}_{i^+}} &= 2(\mathbf{h}_{i^+} - \mathbf{h}_i) \times I_{||\mathbf{h}_i - \mathbf{h}_{i^+}||_2^2 - ||\mathbf{h}_i - \mathbf{h}_{i^-}||_2^2 + \epsilon > 0}, \\ \frac{\partial \ell_{tri}}{\partial \mathbf{h}_{i^-}} &= 2(\mathbf{h}_i - \mathbf{h}_{i^-}) \times I_{||\mathbf{h}_i - \mathbf{h}_{i^+}||_2^2 - ||\mathbf{h}_i - \mathbf{h}_{i^-}||_2^2 + \epsilon > 0}, \end{aligned} \tag{4}$$

where ℓ_{tri} is the objective of the first term in E.q. 3. The indicator function $I_{condition} = 1$ if $condition$ is true; otherwise $I_{condition} = 0$.

The gradients of the second term w.r.t $\mathbf{h}_i, \mathbf{h}_j$ are

$$\begin{aligned} \frac{\partial \psi(\mathbf{h}_i, \mathbf{h}_j)}{\partial \mathbf{h}_i^{(m')}} =& 2(\mathbf{h}_j^{(m')} - \mathbf{h}_i^{(m')}) \times I_{||\mathbf{h}_i^{(m')} - \mathbf{h}_j^{(m')}||_{\mathcal{H}} < r'} \\ &+ 2(\mathbf{h}_i^{(m')} - \mathbf{h}_j^{(m')}) \times I_{||\mathbf{h}_i^{(m')} - \mathbf{h}_j^{(m')}||_{\mathcal{H}} > r'+1}, \\ \frac{\partial \psi(\mathbf{h}_i, \mathbf{h}_j)}{\partial \mathbf{h}_j^{(m')}} =& 2(\mathbf{h}_i^{(m')} - \mathbf{h}_j^{(m')}) \times I_{||\mathbf{h}_i^{(m')} - \mathbf{h}_j^{(m')}||_{\mathcal{H}} < r'} \\ &+ 2(\mathbf{h}_j^{(m')} - \mathbf{h}_i^{(m')}) \times I_{||\mathbf{h}_i^{(m')} - \mathbf{h}_j^{(m')}||_{\mathcal{H}} > r'+1}, \end{aligned} \tag{5}$$

where $m' = 1, \cdots, m$.

The gradients of the third term w.r.t $\mathbf{h}_{p_a}^{(m')}, \mathbf{h}_{p_b}^{(m')}$ are

$$\begin{aligned} \frac{\partial \phi(P^{(m')}(i))}{\partial \mathbf{h}_{p_a}^{(m')}} =& 2(\mathbf{h}_{p_a}^{(m')} - \mathbf{h}_{p_b}^{(m')}) \times I_{p_a, p_b \text{ in same subgroup}} \\ &+ 2(\mathbf{h}_{p_b}^{(m')} - \mathbf{h}_{p_a}^{(m')}) \\ &\times I_{p_a, p_b \text{ in same group } \&\ ||\mathbf{h}_{p_b}^{(m')} - \mathbf{h}_{p_a}^{(m')}||_{\mathcal{H}} < 1} \\ &+ 2(\mathbf{h}_{p_b}^{(m')} - \mathbf{h}_{p_a}^{(m')}) \\ &\times I_{p_a, p_b \text{ in different group } \&\ ||\mathbf{h}_{p_b}^{(m')} - \mathbf{h}_{p_a}^{(m')}||_{\mathcal{H}} < 2}, \\ \frac{\partial \phi(P^{(m')}(i))}{\partial \mathbf{h}_{p_b}^{(m')}} =& 2(\mathbf{h}_{p_b}^{(m')} - \mathbf{h}_{p_a}^{(m')}) \times I_{p_a, p_b \text{ in same subgroup}} \\ &+ 2(\mathbf{h}_{p_a}^{(m')} - \mathbf{h}_{p_b}^{(m')}) \\ &\times I_{p_a, p_b \text{ in same group } \&\ ||\mathbf{h}_{p_b}^{(m')} - \mathbf{h}_{p_a}^{(m')}||_{\mathcal{H}} < 1} \\ &+ 2(\mathbf{h}_{p_a}^{(m')} - \mathbf{h}_{p_b}^{(m')}) \\ &\times I_{p_a, p_b \text{ in different group } \&\ ||\mathbf{h}_{p_b}^{(m')} - \mathbf{h}_{p_a}^{(m')}||_{\mathcal{H}} < 2}. \end{aligned} \tag{6}$$

V. Experiments

In this section, we evaluate and compare the performance of the proposed method with several state-of-the-art algorithms.

A. Datasets

- **SVHN [37]** [2] is a real-world image dataset which obtained from house numbers in Google street view images. It contains over 630,000 digit images.
- **NUS-WIDE [38]** [3] consists of 269,648 images and the associated tags from Flickr, which includes ground-truth for 81 concepts extracted from the tags.
- **MS COCO [39]** [4] is a large-scale dataset for scene understanding, which contains 80 common object categories with most of them having more than 5,000 labeled instances.

In NUS-WIDE, we follow the settings in [40], [14] for a fair comparison. The 21 most frequent labels are selected, where

[2] http://ufldl.stanford.edu/housenumbers/
[3] http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm
[4] http://cocodataset.org/

each label associated with at least 5,000 images. The images that do not contain any label are removed. We randomly select 100 images from each of the selected 21 classes to form the query set of 2,100 images. The rest images are used as the retrieval database. In the retrieval database, 500 images from each of the selected 21 classes are randomly chosen as the training set.

In SVHN, we randomly select 1,000 images (100 images per class) as the query set, and the rest images are used as the retrieval database. We also randomly choose 5,000 images (500 images per class) from the rest images as the training set.

We use the COCO 2017 in this paper. The COCO 2017 dataset has more than 120,000 labeled images for train/val, 41,000 test images and 123,000 unlabeled images that follow the same class distribution as the labeled images. In total this dataset has about 287,000 images. Similar to SVHN and NUS-WIDE, we also uniformly sample 100 images per class from the train/val images to form the query set of 8,000 images. And the rest images are used as the retrieval database. In the retrieval database, 500 images per class from the labeled images are randomly selected as the training set. Note that we only use the labeled images in the retrieval database to measure the search accuracy, e.g., MAP.

B. Experimental Setting

Implementation details. We implement the proposed method using the open-source *Caffe* [41] framework. In this paper, we use AlexNet [33] as our basic network, which the following modifications are made: 1) the last fully-connected layer (e.g., fc8) is removed since it is for 1000 image classification, 2) another fully-connected layer with ℓ dimensional output is added (the parameters are randomly initialized), 3) a slice layer is added to divide the ℓ dimensional feature into m slices, each slice has equal length. The weights of layers are firstly initialized by the pre-trained AlexNet model [5]. The batch size is 256 and the total epoch is 60. The base learning rate is 0.0001 and it is changed to one-tenth of the current value after every 20 epochs. In all our experiments, the ϵ is setted to be $l/2$ and $k' = 5$. The numbers of substring hash tables are set to be $m = 1, 2, 3, 4$ for 24 bits, 48 bits, 64 bits and 96 bits, respectively.

Since the main goal of our method is to accelerate the search time, we mainly compare the following two methods to make a fair comparison:

- **Deep Hashing (DH)**. The hash functions are learned without the assistance of the balanced constraints, i.e., only using the first term of the objectives in E.q. (3).
- **Deep Multi-Index Hashing (DMIH)**. The hash functions are learned with the assistance of the balanced constraints, i.e., using all terms in the E.q. (3).

Since the two methods use the same network and the only difference is that using or not using the proposed balanced constraints, these comparisons can show us whether the balanced constraints can contribute to the speed or not.

In training, we firstly use DH to train a hash model. Then we fine-tune the proposed DMIH by adding two balanced constraints. To obtain the balanced substrings, we add the E.q. 1 in the loss function. To obtain the balanced hash buckets, we first use the hash model of DH to find all training images that indices in the hash buckets which have more than 5 binary codes, then add the E.q. 2 into the loss function.

Evaluations. For search accuracy, MAP is used as the main evaluation metric. It is the mean of averaged precisions over a set of queries, which can be calculated by

$$\text{MAP} = \sum_{j=1}^{n} P@j \times pos(j)/N_{pos}, \tag{7}$$

where $pos(j)$ is an indicator function. If the number of shared labels between two images is larger than or equal to one, these two images are relevant. If the image at the position j is relevant, $pos(j)$ is 1, otherwise $pos(j)$ is 0. N_{pos} represents the total number of relevant images w.r.t. the query image. $P@j = \frac{N_{pos}(j)}{j}$, where $N_{pos}(j)$ represents the number of relevant images within the top j images.

For retrieval time, we use MIH [6] to report the accelerated ratios compared to linear scan. The speed-up factors of MIH over linear scan are used as the evaluation metric. The larger, the faster.

C. Results on Search Accuracy

In the first set of experiments, we evaluate and compare the performance of the proposed method with several state-of-the-art algorithms.

LSH [6], ITQ [1], ITQ-CCA [1], SH [15] and DH are selected as the baselines. The results of LSH, ITQ, ITQ-CCA, and SH are obtained by the implementations provided by their authors, respectively. Note that DH is very similar to the existing work one-stage hashing [24], which also divides the feature into several slices and uses the triplet ranking loss for preserving the similarities. Since the results of DH and one-stage hashing are almost the same, thus we only report the results of DH. For a fair comparison, all of the methods use identical training and test sets, and the AlexNet model pre-trained on ImageNet dataset [42] is used to extract deep features (i.e., 4096-dimensional features) for LSH, ITQ, ITQ-CCA, and SH.

Fig. 5 shows the comparison results of the MAP on the three datasets. We can see that 1) the deep-network-based methods show improvements over the baselines that use fixed deep features. Specifically, on NUS-WIDE, our method obtains a MAP of 0.8215 on 24 bits, which shows a relative increase of 13% compared to the ITQ-CCA. 2) DMIH shows comparable performance against the most related baseline DH. For example, the MAP of our method is 0.6648, compared to 0.6607 of the DH on COCO dataset. Fig. 6 shows the precision-recall curves on the three datasets. Again, our method yields comparable accuracy against DH and beats all other baselines.

Recently, Sablayrolles et al. [25] proposed new protocols for hashing and showed that the current evaluation protocols

[5]http://dl.caffe.berkeleyvision.org/bvlc_alexnet.caffemodel

[6]https://github.com/norouzi/mih

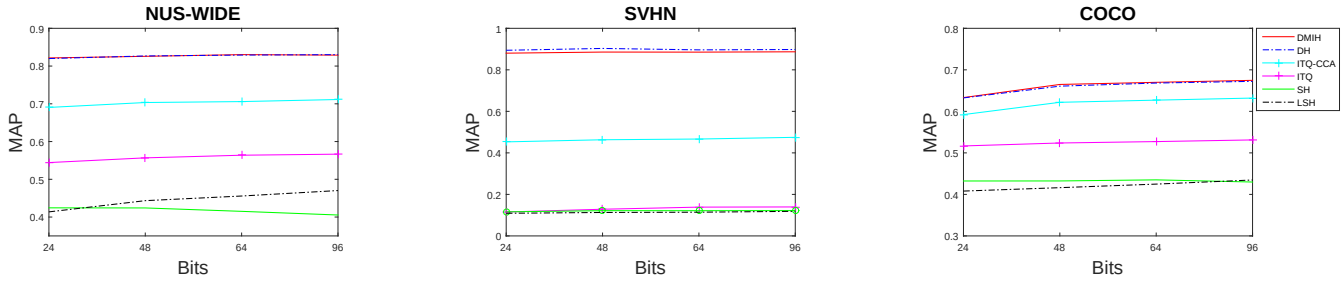


Fig. 5. The comparison results of MAP w.r.t. different number of bits.

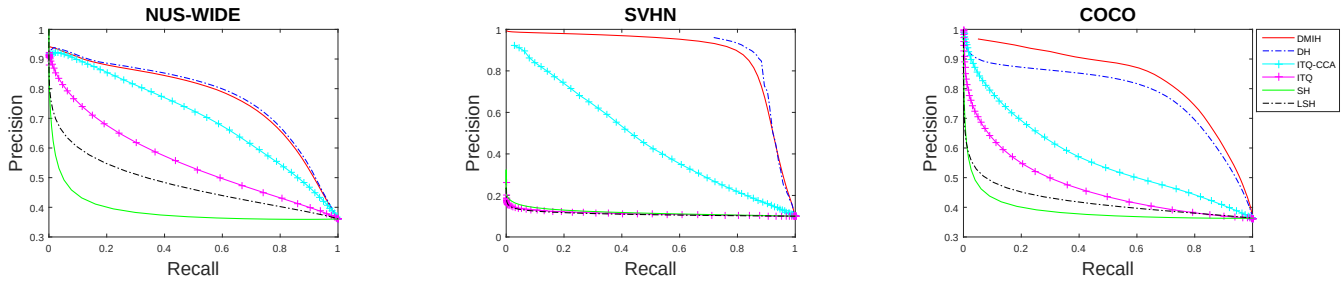


Fig. 6. The comparison results of precision-recall curves of Hamming ranking with 64 bits.

are not satisfactory. In this set of experiments, we follow the setting of [25] to retrieval of the unseen classes. In this setting, the first 75% [7] of the classes are assumed to be known when learning the hashing function, and the 25% remaining classes are used to evaluate the encoding hashing scheme. The *learn - database - query* split is as follow. In training, we use all the images from the first 75% classes to learn the hash model. In testing, we randomly select 100 images per class from the 25% classes as the query set, and the rest images from the 25% classes are used as the retrieval database.

Table I shows the performances of all comparing approaches. We can see that our method can achieve very comparable accuracy against DH and other baselines in the new protocol. For instant, the results of DMIH is 0.5858 when the number of bits is 48, compared to 0.5840 of DH on COCO database.

All these results verify that adding new balanced constraints does not drop the performance.

TABLE I
RETRIEVAL OF THE UNSEEN CLASSES

Method	24 bits	48 bits	64 bits	96 bits
SVHN				
LSH	0.3528	0.3498	0.3566	0.3600
SH	0.3334	0.3346	0.3406	0.3406
ITQ	0.3446	0.3502	0.3494	0.3519
ITQ-CCA	0.4471	0.4625	0.4564	0.4536
DH	0.4845	0.6769	0.6785	0.6737
DMIH	0.4867	0.6746	0.6702	0.6878
NUS-WIDE				
LSH	0.6066	0.6241	0.6188	0.6392
SH	0.5721	0.5730	0.5730	0.5730
ITQ	0.6907	0.6956	0.7026	0.7103
ITQ-CCA	0.7278	0.7299	0.7302	0.7310
DH	0.7481	0.7504	0.7543	0.7544
DMIH	0.7426	0.7524	0.7520	0.7522
COCO				
LSH	0.4615	0.4796	0.4891	0.5053
SH	0.4246	0.4250	0.4250	0.4250
ITQ	0.5314	0.5497	0.5514	0.5599
ITQ-CCA	0.5368	0.5530	0.5554	0.5601
DH	0.5774	0.5840	0.5865	0.5834
DMIH	0.5716	0.5858	0.5870	0.5839

D. Results on Retrieval Time

In the second set of experiments, we show the retrieval times of the proposed DMIH and the DH, which is to answer us whether the two constraints can accelerate the search time in the Hamming space or not. Note that DMIH and DH have the same performances in all the following experiments.

After obtaining the binary codes, we use MIH [8] to report the accelerated ratios of the two methods compared to linear scan on all the above databases. Note that the linear scan does not depend on the underlying distribution of the binary codes, thus the running times of linear scan of two methods are the same. The speed-up factors of MIH and DH over linear scan for different kNN problems are shown in Table II.

The results show that DMIH is more efficient than DH,

[7] Suppose that nc is the number of classes, we set number of the seen class to be $\lfloor 0.75 \times nc \rfloor$ when it is not an integer. The remaining classes are used to be the unseen classes.

[8] https://github.com/norouzi/mih

TABLE II
SPEED-UP FACTORS FOR MIH VS. LINEAR SCAN.

Dataset	nbits	Method	1-NN	2-NN	5-NN	10-NN	20-NN	50-NN	100-NN	linear scan
NUS-WIDE	24	DMIH	**115.59**	**100.17**	**95.54**	**87.58**	**60.60**	**52.97**	**41.41**	2.01 ms
		DH	75.66	69.34	60.58	60.01	55.11	50.19	39.77	
	48	DMIH	**107.17**	**95.16**	**87.98**	**70.18**	**59.29**	**42.29**	**32.89**	3.81 ms
		DH	60.15	55.39	52.45	45.82	37.31	31.30	27.49	
	64	DMIH	**104.23**	**95.37**	**75.62**	**66.59**	**58.75**	**45.01**	**35.83**	5.09 ms
		DH	64.17	59.28	49.71	44.79	38.82	30.61	25.11	
	96	DMIH	**77.59**	**70.51**	**61.78**	**51.02**	**44.93**	**32.96**	**26.08**	7.49 ms
		DH	40.09	36.76	31.14	26.04	22.71	18.38	15.56	
SVHN	24	DMIH	**151.64**	**147.39**	**136.50**	**128.78**	**121.37**	**110.32**	**98.76**	6.78 ms
		DH	10.83	10.06	10.68	10.57	10.47	10.72	10.22	
	48	DMIH	**112.76**	**111.39**	**100.56**	**95.49**	**92.21**	**75.15**	**63.04**	12.89 ms
		DH	10.36	10.84	10.49	10.52	10.09	10.63	10.03	
	64	DMIH	**97.32**	**94.81**	**90.47**	**84.83**	**80.67**	**72.29**	**54.00**	16.92 ms
		DH	10.48	10.82	11.02	10.61	10.44	10.60	10.07	
	96	DMIH	**78.51**	**78.18**	**70.24**	**60.38**	**49.90**	**38.88**	**29.18**	25.74 ms
		DH	10.66	10.04	10.89	10.55	9.71	9.58	9.10	
COCO	24	DMIH	**190.54**	**173.18**	**161.67**	**150.55**	**147.08**	**129.78**	**106.63**	3.12 ms
		DH	16.47	16.36	16.16	16.48	16.02	15.91	15.75	
	48	DMIH	**168.93**	**158.18**	**131.54**	**100.98**	**63.48**	**48.82**	**41.98**	5.73 ms
		DH	16.53	16.42	16.26	16.62	16.18	14.91	13.66	
	64	DMIH	**88.98**	**74.32**	**61.65**	**55.48**	**46.88**	**36.28**	**29.61**	6.90 ms
		DH	19.35	19.01	18.92	17.74	15.57	14.54	12.81	
	96	DMIH	**54.47**	**44.75**	**33.13**	**29.02**	**24.74**	**20.98**	**18.56**	10.12 ms
		DH	14.96	13.51	12.36	11.73	10.96	9.58	8.95	

especially for the small kNN problems. For instance, for 1-NN in 96-bits codes on SVHN, the speed-up factor for DMIH is 78.51, compared to 10.66 for DH. In NUS-WIDE, our method shows about 2 speed-up ratio in comparison with DH. In summary, our method performs 2 to 15 times faster than DH with the comparable performance.

The main reason is that the proposed method can learn more balanced hash codes than DH. To give an intuitive comparison, we utilize two measurements for the balanced substrings or the balanced buckets, respectively.

1) Measurement for the Balanced Buckets: To give an intuitive understanding of our method, we utilize an entropy based measurement to measure the data distributions of the binary codes. It is defined as

$$I(H) = -\frac{1}{m}\sum_{m'=1}^{m}\sum_{i=1}^{2^s} p(i) \times \log(p(i)), \tag{8}$$

where $s = l/m$ is the dimension of the m'-th hash table, thus there are 2^s buckets in this table. And $p(i)$ is the probability of codes assigned to bucket i, which is defined as $p(i) = n(i)/N$, where $n(i)$ is the number of codes in the bucket i and N is the size of database. Note that the higher entropy value means better distribution of data items in hash tables.

Again, for all databases and bits, our method yields the higher entropy and beats the baseline as shown in Table III. This is also can explain why our method can obtain the faster searching. Compared to NUS-WIDE dataset, the numbers of items in hash buckets of the SVHN and the COCO datasets are more unbalanced, thus the constraint for the balanced hash buckets is very useful in these two datasets.

2) Measurement for the Balanced Substrings: We choose the *variance* between the query binary codes and the database binary codes as the measure, which is used in our method for balanced substrings optimization. It is defined as

$$V(Q, H) = \frac{1}{nn_q}\sum_{i=1}^{n}\sum_{k=1}^{n_q}[Var(\mathbf{q}_k, \mathbf{h}_i)], \tag{9}$$

where $Var(\mathbf{q}_k, \mathbf{h}_i) = \frac{1}{m}\sum_{m'=1}^{m} ||D(\mathbf{h}_i^{(j)}, \mathbf{q}_k^{(j)}) - \frac{D(\mathbf{h}_i, \mathbf{q}_k)}{m}||^2$, n and n_q are the numbers of binary codes in retrieval database and the query set, respectively. The small value, the better balanced distribution.

TABLE III
DISTRIBUTION OF DATA ITEMS IN THE HASH BUCKETS OF THE TWO METHODS.

Method	24 bits	48 bits	64 bits	96 bits
SVHN				
DH	4.56	4.35	4.11	4.68
DMIH	**10.14**	**10.78**	**9.52**	**9.51**
NUS-WIDE				
DH	9.63	9.56	9.11	9.58
DMIH	**10.51**	**10.42**	**9.71**	**9.91**
COCO				
DH	8.30	8.42	7.89	8.14
DMIH	**11.32**	**10.96**	**10.42**	**10.53**

Table IV shows the results. Since the number of substrings is one on 24 bits, we do not report the results on 24 bits. We can seen that our method yields the smaller values on three datasets.

3) Ablation Study: In this set of our experiments, we do ablation study to clarify the impact of each part of our method on the final performance. To give an intuitive comparison, we give the results of using only the balanced substrings or the balanced buckets, respectively.

TABLE IV
DISTRIBUTION OF THE SUBSTRINGS.

Method	48 bits	64 bits	96 bits
SVHN			
DH	3.78	2.55	4.76
DMIH	**1.21**	**1.71**	**1.83**
NUS-WIDE			
DH	2.51	3.31	3.68
DMIH	**1.72**	**1.50**	**1.42**
COCO			
DH	2.57	2.80	3.30
DMIH	**1.49**	**1.94**	**2.02**

In the first baseline, we remove the second term in E.q. 3 and only explore the effects of the balanced buckets. This method is refereed as Deep Multi-Index Hashing with Balanced Buckets (DMIH-BB), which is formulated as

$$\min \frac{1}{n_{tri}} \sum_{i,i^+,i^-} \max(0, \epsilon + D(\mathbf{h}_i, \mathbf{h}_{i^+}) - D(\mathbf{h}_i, \mathbf{h}_{i^-})), + \frac{1}{n_{bb}} \sum_{i,m'} \phi(P^{(m')}(i)). \tag{10}$$

In the second baseline, we only explore the effects of the balanced substrings, which is refereed as Deep Multi-Index Hashing with Balanced Substrings (DMIH-BS). It is defined as

$$\min \frac{1}{n_{tri}} \sum_{i,i^+,i^-} \max(0, \epsilon + D(\mathbf{h}_i, \mathbf{h}_{i^+}) - D(\mathbf{h}_i, \mathbf{h}_{i^-})), + \frac{1}{n_{bs}} \sum_{i,j} \psi(\mathbf{h}_i, \mathbf{h}_j). \tag{11}$$

Table V shows the results on 96 bits. The results show that balanced substrings are useful on three datasets, e.g., DMIH-BS performs 57.52 times faster than linear scan and DH performs 40.09 times faster than linear scan for the 1NN search problem. The results also show that the balanced buckets are very useful on the SVHN and COCO datasets while this constraint does a litter helpful on NUS-WIDE dataset. It depends on the data distributions of the learned binary codes.

4) Effect of the End-to-end Learning: Our framework is an end-to-end framework. To show the advantages of the end-to-end framework, we compare to the following baselines, which adopt a two-stage strategy. The pipelines of the baselines are divided into two parts: (1) the existing hashing methods are firstly used to extract the learned hash codes; and (2) the already learned hash codes are then updated to obtain more balanced binary codes. For a fair comparison, DH method is learned to encode images into binary codes in the first stage. Then, the substring optimization (SO) [3], [29] or our balanced loss are used to rebalance the binary codes, respectively. In the first baseline, the substring optimization is to rebalance the binary codes. Substring optimization is a greedy method: bits are greedy assigned to substrings one at a time based on the correlation between the bits. The second baseline also uses the two constraints in E.q. 3 but adopts a post-processing method. Given the already learned codes, we deploy a feed-forward neural network with one fully connected layer activated by *tanh* function for the nonlinearly projection, i.e., $\hat{\mathbf{h}} = W\mathbf{h} + b$, where $\mathbf{h}$ is the already learned binary code. The $W \in \mathbb{R}^{\ell \times \ell}$ and $b \in \mathbb{R}^{\ell}$ are two parameters needed to be learned, where ℓ is the length of the binary codes.

TABLE VI
SPEED-UP FACTORS FOR MIH VS. LINEAR SCAN.

Method	1-NN	5-NN	10-NN	50-NN	100-NN
NUS-WIDE					
DMIH	77.59	61.78	51.02	32.96	26.08
DH	40.09	31.14	26.04	18.38	15.56
Two-stage ([3])	43.57	34.13	29.76	19.63	17.92
Two-stage (Ours)	62.04	53.35	45.56	30.15	24.20
SVHN					
DMIH	78.51	70.24	60.38	38.88	29.18
DH	10.66	10.89	10.55	9.58	9.10
Two-stage ([3])	11.24	10.30	10.73	10.17	9.75
Two-stage (Ours)	26.70	25.59	24.23	21.90	19.39
COCO					
DMIH	54.47	33.13	29.02	20.98	18.56
DH	14.96	12.36	11.73	9.68	8.95
Two-stage ([3])	15.25	13.57	12.26	9.71	8.99
Two-stage (Ours)	38.71	29.91	25.17	17.76	14.61

Table VI shows the comparison results. We can observe that our method performs better than the DH and the two-stage methods, and our two-stage method performs better than the substring optimization method. It is desirable to learn the hash functions and the balanced procedure in the end-to-end framework.

TABLE VII
SPEED-UP FACTORS USING DIFFERENT NUMBERS OF k'.

k'	1-NN	5-NN	10-NN	50-NN	100-NN
NUS-WIDE					
$k' = 1$	107.74	87.13	69.90	42.69	32.51
$k' = 5$	107.17	87.98	70.18	42.29	32.89
$k' = 10$	108.75	88.51	70.40	42.31	31.29
$k' = 50$	107.43	88.32	70.67	42.32	32.85
SVHN					
$k' = 1$	118.73	107.94	101.72	79.67	69.14
$k' = 5$	112.76	100.56	95.49	75.15	63.04
$k' = 10$	104.84	97.71	90.26	75.95	63.74
$k' = 50$	102.58	94.17	90.50	76.12	63.34
COCO					
$k' = 1$	169.92	131.25	101.67	48.26	39.64
$k' = 5$	168.93	131.54	100.98	48.82	41.98
$k' = 10$	168.18	132.66	100.73	49.18	40.69
$k' = 50$	159.23	122.47	95.76	42.10	40.15

5) Effects of the Maximum Number of Codes in Hash Buckets: In this paper, we use $k' = 5$ that each bucket contains at most 5 binary codes. We add experiments to explore the effect of different number of codes in the hash buckets. Table VII shows the comparison results on 48 bits. The results show that the small value of k' would perform better. The method using $k' = 1$ performs close to those using $k' = 5$, and the method using $k' = 50$ performs worst.

TABLE V
SPEED-UP FACTORS FOR MIH VS. LINEAR SCAN.

Dataset	Method	1-NN	2-NN	5-NN	10-NN	20-NN	50-NN	100-NN
NUS-WIDE	DMIH	77.59	70.51	61.78	51.02	44.93	32.96	26.08
	DMIH-BB	45.75	38.33	33.34	30.09	22.00	20.28	18.13
	DMIH-BS	57.52	49.50	45.88	38.11	25.53	24.12	23.30
	DH	40.09	36.76	31.14	26.04	22.71	18.38	15.56
SVHN	DMIH	78.51	78.18	70.24	60.38	49.90	38.88	29.18
	DMIH-BB	72.43	72.89	65.78	58.79	40.46	32.05	25.76
	DMIH-BS	17.29	16.75	16.23	16.04	16.02	15.77	14.84
	DH	10.66	10.04	10.89	10.55	9.71	9.58	9.10
COCO	DMIH	54.47	44.75	33.13	29.02	24.74	20.98	18.56
	DMIH-BB	38.86	31.68	28.74	22.33	20.10	18.27	15.04
	DMIH-BS	15.48	14.27	13.33	12.39	12.75	11.07	10.46
	DH	14.96	13.51	12.36	11.73	10.96	9.58	8.95

VI. CONCLUSION

In this paper, we proposed a deep-network-based multi-index hashing method for fast search. In the proposed deep architecture, an image goes through the deep network with stacked convolutional layers and is encoded into high-level image representation with several substrings. Then, we proposed to learn more balanced binary codes by adding two constraints. One is the balanced substrings, which is used to make the binary codes distributed as balance as possible in each hash table. Another is the balanced hash buckets, which is used to let the buckets in each substrings hash table contain equal items. Finally, the deep hash functions with triplet ranking loss and the balanced constraints are learned simultaneously. Empirical evaluations on two datasets show that the proposed method runs faster than the baseline and achieve comparable performance.

In future work, we plan to accelerate the running times of extracting the features from the deep network.

APPENDICES

Proof of Proposition 1

Suppose that there exists one binary code $\mathbf{h}$, $\mathbf{h} \in \mathcal{R}^r(\mathbf{q})$ and $\mathbf{h} \notin \bigcup_{m'=1}^{a+1} \mathcal{N}_{m'}^{r'}(\mathbf{q})$. According to Proposition 1 in paper [3], we have $D(\mathbf{h}^{(z)}, \mathbf{q}^{(z)}) \leq r'$ for a substring. We discuss in two situations: 1) if $z \leq a+1$, then $\mathbf{h} \in \bigcup_{m'=1}^{a+1} \mathcal{N}_{m'}^{r'}(\mathbf{q})$ according to the definition of $\mathcal{N}_z^{r'}(\mathbf{q})$, which contradicts the premise. 2) if $z > a+1$, all the first $a+1$ substrings is strictly greater than r', then the total number of bits that differ in the last $m-a-1$ is at most $r-(a+1)\times(r'+1) = r'm+a-ar'-a-r'-1 = (m-a-1)\times r'-1$. Using Proposition 1 in paper [3] again, we have $D(\mathbf{h}^{(z)}, \mathbf{q}^{(z)}) \leq r'-1$, thus $\mathbf{h} \in \mathcal{N}_z^{r'-1}(\mathbf{q}) \subseteq \mathcal{N}_1^{r'}(\mathbf{q})$, which contradicts the premise.

Proof of Proposition 2

Suppose that $\bigcup_{m'=1}^{a+1} \mathcal{N}_{m'}^{r'}(\mathbf{q}) - \mathcal{R}^r(\mathbf{q}) \neq \emptyset$, we have at least one binary code $\mathbf{b}$ satisfies $\mathbf{b} \in \bigcup_{m'=1}^{a+1} \mathcal{N}_i^{r'}(\mathbf{q})$ and $\mathbf{b} \notin \mathcal{R}^r(\mathbf{q})$. Since $\mathbf{b}$ is not the r-neighbor of query $\mathbf{q}$, then $\mathbf{b}$ and $\mathbf{q}$ at least differ by $r+1$ bits, i.e., $D(\mathbf{q}, \mathbf{b}) \geq r+1$. According to the assumption that the substrings are balanced and $D(\mathbf{q}, \mathbf{b}) \geq r+1$, we have $D(\mathbf{q}^{(a+1)}, \mathbf{b}^{(a+1)}) \geq r'+1$, thus $\mathbf{b} \notin \bigcup_{m'=1}^{a+1} \mathcal{N}_{m'}^{r'}(\mathbf{q})$ by the definition of $\mathcal{N}_{m'}^{r'}(\mathbf{q})$, which contradicts the premise.

ACKNOWLEDGMENT

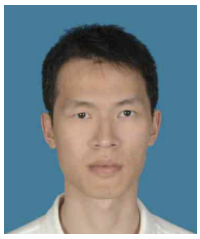
This work is supported by the National Natural Science Foundation of China under Grants (U1611264, 61602530, U1536203, 61572493, U1711262 and U1501252). This work is also supported by the Research Foundation of Science and Technology Plan Project in Guangdong Province (2017B030308007) and CCF-Tencent Open Research Fund.

REFERENCES

[1] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011, pp. 817–824.
[2] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2074–2081.
[3] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast exact search in hamming space with multi-index hashing," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 6, pp. 1107–1119, 2014.
[4] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Proceedings of Advances in Neural Information Processing Systems*, 2009, pp. 1042–1050.
[5] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for scalable image retrieval," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 3424–3431.
[6] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the International Conference on Very Large Data Bases*, 1999, pp. 518–529.
[7] K. Lin, H.-F. Yang, J.-H. Hsiao, and C.-S. Chen, "Deep learning of binary hash codes for fast image retrieval," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 27–35.
[8] W. Li, "Feature learning based deep supervised hashing with pairwise labels," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2016, pp. 3485–3492.
[9] F. Zhao, Y. Huang, L. Wang, and T. Tan, "Deep semantic ranking based hashing for multi-label image retrieval," *arXiv preprint arXiv:1501.06272*, 2015.
[10] B. Zhuang, G. Lin, C. Shen, and I. Reid, "Fast training of triplet-based deep binary embedding networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5955–5964.
[11] L. Liu, F. Shen, Y. Shen, X. Liu, and L. Shao, "Deep sketch hashing: Fast free-hand sketch-based image retrieval," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
[12] R. Salakhutdinov and G. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *Proceedings of the International Conference on Artificial*, 2007, pp. 412–419.
[13] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Proceedings of the IEEE International Conference on Computer Vision*, 2009, pp. 2130–2137.
[14] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proceedings of the International Conference on Machine Learning*, 2011, pp. 1–8.

[15] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proceedings of Advances in Neural Information Processing Systems*, 2008, pp. 1753–1760.
[16] M. Norouzi and D. M. Blei, "Minimal loss hashing for compact binary codes," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 353–360.
[17] J. Wang, T. Zhang, N. Sebe, and et al., "A survey on learning to hash," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.
[18] Q.-Y. Jiang and W.-J. Li, "Deep cross-modal hashing," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
[19] D. Mandal, K. Chaudhury, and S. Biswas, "Generalized semantic preserving hashing for n-label cross-modal retrieval," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
[20] Z. Zhang, Y. Chen, and V. Saligrama, "Efficient training of very deep neural networks for supervised hashing," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1487–1495.
[21] H.-F. Yang, K. Lin, and C.-S. Chen, "Supervised learning of semantics-preserving hashing via deep neural networks for large-scale image search," *arXiv preprint arXiv:1507.00101*, 2015.
[22] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang, "Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 4766–4779, 2015.
[23] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2064–2072.
[24] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3270–3278.
[25] A. Sablayrolles, M. Douze, N. Usunier, and H. Jégou, "How should we evaluate supervised hashing?" in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017, pp. 1732–1736.
[26] D. Greene, M. Parnas, and F. Yao, "Multi-index hashing for information retrieval," in *Proceedings of the International Conference on Fuzzy Systems and Knowledge Discovery*.
[27] Q. Liu, H. Xie, Y. Liu, C. Zhang, and L. Guo, "Data-oriented multi-index hashing," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, 2015, pp. 1–6.
[28] S. Gog and R. Venturini, "Fast and compact hamming distance index," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2016, pp. 285–294.
[29] J. Wan, S. Tang, Y. Zhang, L. Huang, and J. Li, "Data driven multi-index hashing," in *Proceedings of the IEEE International Conference on Image Processing*, 2013, pp. 2670–2673.
[30] E.-J. Ong and M. Bober, "Improved haproceedings of the acm international conference on multimediaing distance search using variable length substrings," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2000–2008.
[31] M. Wang, X. Feng, and J. Cui, "Multi-index hashing with repeat-bits in hamming space," in *Proceedings of the International Conference on Fuzzy Systems and Knowledge Discovery*, 2015, pp. 1307–1313.
[32] J. Song, H. Shen, J. Wang, Z. Huang, N. Sebe, and J. Wang, "A distance-computation-free search scheme for binary code databases," *IEEE Transactions on Multimedia*, vol. 18, no. 3, pp. 484–495, 2016.
[33] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of Advances in Neural Information Processing Systems*, 2012, pp. 1106–1114.
[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, 2014.
[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
[37] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proceedings of Advances in Neural Information Processing Systems*, vol. 2011, no. 2, 2011, p. 5.
[38] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "Nus-wide: a real-world web image database from national university of singapore," in *Proceedings of the ACM International Conference on Image and Video Retrieval*, 2009, p. 48.
[39] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Coproceedings of the acm international conference on multimediaon objects in context," in *Proceedings of the European conference on computer vision*, 2014, pp. 740–755.
[40] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning," in *Proceedings of the AAAI Conference on Artificial Intellignece*, 2014, pp. 2156–2162.
[41] Y. Jia, "Caffe: An open source convolutional architecture for fast feature embedding," *http://caffe. berkeleyvision. org*, 2013.
[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

Hanjiang Lai received his B.S. and Ph.D. degrees from Sun Yat-sen University in 2009 and 2014, respectively. He was working as a research fellow at National University of Singapore during 2014-2015. He is currently an associate professor with Sun Yat-Sen university. His research interests includes machine learning algorithms, deep learning, and computer vision.



Pan Yan received the B.S. degree in information science and the Ph.D. degree in computer science from Sun Yat-Sen university, Guangzhou, China, in 2002 and 2007, respectively.

He is currently an Associate Professor with Sun Yat-sen University. His current research interests include machine learning algorithms, learning to rank, and computer vision.

Dr. Pan has served as a reviewer for several conferences and journals. He was the winner of the object categorization task in PASCAL Visual Object Classes Challenge in 2012.

Si Liu is an associate professor in Beihang University. She received Bachelor degree from advanced class of Beijing Institute of Technology, and Ph.D. degree from Institute of Automation, Chinese Academy of Sciences. She has been Research Assistant and Postdoc in National University of Singapore. Her research interest includes computer vision and multimedia analysis. She has published over 40 cutting-edge papers on the human-related analysis including the human parsing, face editing and image retrieval. She was the recipient of Best Paper of ACM MM 2013, Best demo award of ACM MM 2012. She was the Champion of Big Data & Computing Intelligence Contest 2016 and CVPR 2017 Look Into Person Challenge.

Zhenbin Weng received the B.S. and Master degree in software engineering from Sun Yat-Sen University, Guangzhou, in 2014 and 2016, respectively. His main research interests include learning to hash, nearest neighbor search and computer vision.

Jian Yin received the B.S., M.S., and Ph.D. degrees from Wuhan University, China, in 1989, 1991, and 1994, respectively, all in computer science. He joined Sun Yat-Sen University in July 1994 and now he is a professor of Data and Computer Science School. He has published more than 100 refereed journal and conference papers. His current research interests are in the areas of Data Mining, Artificial Intelligence, and Machine Learning. He is a senior member of China Computer Federation.