Scalable Supervised Discrete Hashing for Large-Scale Search

Xin Luo
Shandong University
luoxin.lxin@gmail.com

Ye Wu
Shandong University
kimiwadewu@gmail.com

Xin-Shun Xu*
Shandong University
xuxinshun@sdu.edu.cn

ABSTRACT

Supervised hashing methods have attracted much attention in these years. However, most existing supervised hashing algorithms have some of the following problems. First, most of them leverage the pairwise similarity matrix, whose size is quadratic to the number of training samples, to supervise the learning of hash codes. Thus, they are not scalable when dealing with large data. Second, most of them relax the discrete constraints for easy optimization and then quantize the learnt real-valued solution to binary hash codes. Therefore, the quantization error caused by the relaxation may lead to a decline of retrieval performance. To address these issues and make the supervised method scalable to large datasets, we present a novel hashing method, named Scalable Supervised Discrete Hashing (SSDH). Specifically, based on a new loss function, SSDH bypasses the direct optimization on the n by n pairwise similarity matrix. In addition, SSDH adopts no relaxation optimization scheme in the learning procedure and avoids the large quantization error problem. Moreover, during learning, it leverages both the pairwise similarity matrix and label matrix; thus, more semantic information can be embedded to the learning of hash codes. Extensive experiments are conducted on six benchmark datasets including two large-scale datasets, i.e., NUS-WIDE and ImageNet. The results show that SSDH can outperform state-of-the-art baselines on these datasets, demonstrating its effectiveness and efficiency.

KEYWORDS

Learning-to-Hash; Supervised Hashing; Scalable Search; Discrete Optimization

ACM Reference Format:

Xin Luo, Ye Wu, and Xin-Shun Xu. 2018. Scalable Supervised Discrete Hashing for Large-Scale Search. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France*. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3178876.3186072>

1 INTRODUCTION

With the exponential growth of data, real-world applications, such as recommendation [13, 14], retrieval [39, 40], and multimedia analysis [38, 41, 56] have experienced great changes. Thereinto, many important applications utilize the similarity search technique, which can return similar data instances to a given query instance.

As the amount of data increases explosively [50, 51], the exhaustive comparisons among the query instance and candidate ones in original high dimensional space make traditional similarity search methods inappropriate for scalable search. To do fast similarity search, many promising hashing methods have been proposed in recent years [1, 15, 23, 27, 47, 63]. And they have been successfully used for many problems including web image retrieval [28, 35], mobile landmark search [64], video retrieval [12], recommendation [59, 62], person re-identification [61], sequential or online data search [2, 3, 54], cross-modal or cross-view retrieval [22, 31, 36, 37], search in a distributed setting [26] and so on.

Hashing methods learn the hash function to transform original data into compact binary hash codes (typically ≤ 128 bits); by storing the hash codes instead of the original high-dimensional data, storage cost can be highly reduced. When queries come, they will first be transformed into binary hash codes by the learnt function; then, similarity search can be simply done by calculating the Hamming distances between the hash codes of available data instances and queries. As the Hamming distance between two binary codes is simply the number of bits that differ and can be calculated using the bitwise operation XOR, the retrieval process can be efficiently conducted. Due to the advantages mentioned above, i.e., the low storage cost and high efficiency of pairwise comparison, hashing methods are suitable for scalable data search.

Hashing methods can be divided into data-independent and data-dependent ones. Data-independent hashing methods do not consider the specific data, and leverage random projections to learn the hash function. Locality-Sensitive Hashing (LSH) [8] is one of the most popular data-independent methods. With the success of LSH, it has been extended to a variety of similarity measures, e.g., p -norm distances [7], the Mahalanobis metric [21], and kernel similarity [20]. The main drawback of these methods is that they need long codes to guarantee high precision, which leads to a huge storage overhead. In contrast to data-independent hashing, data-dependent ones take the specific data into consideration and learn compact binary codes, which can effectively and highly efficiently index and organize massive data. A good deal of such methods have been proposed in the literature, and can be further summarized into two categories: unsupervised and supervised methods.

Unsupervised methods do not leverage the label/semantic information and learn hash codes or hash functions through exploiting the relations of training data. There are a number of representative unsupervised hashing methods, e.g. Spectral Hashing [52], Iterative Quantization [9], Inductive Hashing on Manifolds [46], Circulant Binary Embedding [57], Discrete Graph Hashing [33], and Scalable Graph Hashing [17], etc. When the label information is available, supervised hashing methods can embed the semantic information into the learning of hash codes and have demonstrated better performance than that of unsupervised ones in many real-world applications. Thus, supervised hashing has attracted more

*Corresponding author.

This paper is published under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.

WWW 2018, April 23–27, 2018, Lyon, France

© 2018 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC BY 4.0 License.

ACM ISBN 978-1-4503-5639-8/18/04.

<https://doi.org/10.1145/3178876.3186072>

and more attention recently. Some representative supervised hashing methods include Semi-Supervised Hashing [48], Minimal Loss Hashing [42], Two-Step Hashing [30], Supervised Hashing with Latent Factor [60], Fast Supervised Hashing with Decision Trees [29], Supervised Discrete Hashing [45], and Column Sampling Based Discrete Supervised Hashing [18], etc. More methods can be found in surveys on hashing in [4, 5, 49].

Although existing supervised hashing has offered promising performance, there are some problems that need to be further considered. First, the binary constraints of hash codes lead to a discrete optimization problem which is hard to solve. To tackle it, most existing methods first relax the discrete constraints and transform the original problem into a continuous one which is easier to solve; then, they quantize the continuous solution into the binary hash codes space. However, the solution may be sub-optimal and the error caused by relaxation usually degrades the performance. Second, as the supervised methods need to embed the label information, most of them construct a $n \times n$ instance-pairwise similarity matrix which is leveraged in the learning procedure. Such instance-pairwise similarity matrix has an obvious drawback, i.e., the space cost. Especially when dealing with large-scale data, the $n \times n$ space cost is not affordable. Thus, these hashing methods are time-consuming and unscalable. To tackle this, they have to sample a small subset of instances for training and discard the others. Apparently, this may cause information loss and result in poor performance.

To consider these problems mentioned above, in this paper, we propose a novel method named Scalable Supervised Discrete Hashing (SSDH). It can discretely and efficiently learn the hash codes by making full use of all training samples and semantic information. To summarize, the main contributions of SSDH include:

- It takes both the pairwise similarity $\mathbf{S}$ and the label matrix $\mathbf{L}$ into consideration, which can ensure more precise hash codes.
- By representing the hash codes matrix $\mathbf{B}$ with a real-valued transformation from label matrix $\mathbf{L}$, SSDH avoids the direct optimization on $n \times n$ large matrix $\mathbf{S}$, which makes the space cost acceptable when dealing with large-scale data. Besides, using real-valued matrix and hash codes rather than only binary matrix can permit more accurate approximation of $\mathbf{S}$.
- An alternating discrete optimization algorithm is proposed to efficiently learn the hash codes, which is scalable to deal with the large-scale datasets.
- Extensive experiments are conducted on six widely used datasets. The results demonstrate the superiority of SSDH to the state-of-the-art supervised hashing methods.

The rest of this paper is organized as follows. The highly related work is discussed in Section 2. Section 3 introduces the proposed SSDH model including the loss function, optimization algorithm and its extension to out-of-sample data. Section 4 presents the experimental results and analysis on several benchmark datasets. Finally, Section 5 concludes this paper.

2 RELATED WORK: TWO-STEP HASHING

Most hashing methods learn hash codes and hash functions simultaneously, which we can call one-step hashing. Differently, two-step

hashing methods divide the learning of hash codes and hash functions into two steps. In the first step, the two-step hashing methods generate hash codes by the supervision of loss functions. In the second step, given the learnt hash codes, two-step hashing methods learn hash functions which can transform the original features into the compact binary codes. There have been plenty of two-step hashing methods [18, 29–31, 34].

Usually, the performance of a two-step hashing method highly depends on the quality of the hash codes learnt in the first step. Thus, two-step hashing focuses more on the criterion of generating hash codes, i.e. the design of loss functions. In fact, as revealed in [30], once the hash codes are learnt, for any bit of the hash codes, learning the corresponding hash function to project features into it can be modelled as a binary classification problem. Thus, any effective predictive model can be leveraged as hash function in the second step, such as linear classifiers [18, 48, 58], SVM with RBF kernel [30], deep convolutional network [53], boosted decision trees [18, 29]. Generally, the more powerful classifiers one uses as hash function, the better accuracy one can achieve and also the more training time will be consumed.

The proposed SSDH belongs to two-step hashing and its loss function as well as the corresponding optimization algorithm make SSDH different from other two-step methods.

3 OUR METHOD

This section presents the details of SSDH. It is also a two-step hashing method, which means that the training of SSDH can be divided into two steps, i.e., the hash codes learning step and hash function learning step.

3.1 Notations and Problem Definition

Assume we have n labeled instances, i.e., $\mathbf{x}_i \in \mathbb{R}^d$ with its label vector $\mathbf{y}_i \in \{0, 1\}^c$ ($i = 1, 2, \dots, n$). d is the dimension of each instance and c is the number of classes. Correspondingly, $\mathbf{X} \in \mathbb{R}^{n \times d}$ and $\mathbf{Y} \in \{0, 1\}^{n \times c}$ are the feature and label matrices, respectively. For example, $\mathbf{X}_{i*}^\top = \mathbf{x}_i$ and $\mathbf{Y}_{i*}^\top = \mathbf{y}_i$. Moreover, $\mathbf{Y}_{ik}$ is the k -th element of $\mathbf{y}_i^\top$; $\mathbf{Y}_{ik} = 1$ if x_i belongs to class k and $\mathbf{Y}_{ik} = 0$ otherwise. In addition, $\mathbf{S} \in \{-1, 1\}^{n \times n}$ denotes the instance-pairwise semantic similarity, where $\mathbf{S}_{ij} = 1$ means point i and point j are semantically similar, and $\mathbf{S}_{ij} = -1$ otherwise.

SSDH aims to learn a r -bit binary hash code $\mathbf{b}_i \in \{-1, 1\}^r$ for each instance, and $\mathbf{B} \in \{-1, 1\}^{n \times r}$ is the hash code matrix with each row $\mathbf{B}_{i*}^\top = \mathbf{b}_i$. Hash function $F(\cdot)$ transforms $\mathbf{X}$ to $\mathbf{B}$, i.e., $\mathbf{B} = F(\mathbf{X}) = \text{sgn}(\mathbf{X}\mathbf{W})$, where $\mathbf{W}$ is the projection matrix and $\text{sgn}(\cdot)$ is an element-wise sign function defined as $\text{sgn}(x) = 1$ if $x \geq 0$, and -1 otherwise. Without loss of generality, we assume the input instances to be zero centered, i.e., $\sum_{i=1}^n \mathbf{x}_i = 0$.

3.2 Hash Codes Learning – Design of Loss Function

To leverage the semantic information to learn the hash codes, many hashing methods [18, 25, 29, 30, 34, 53, 60] embed the similarity matrix into the following loss function:

$$\begin{aligned} \min_{\mathbf{B}} \quad & \|\mathbf{r}\mathbf{S} - \mathbf{B}\mathbf{B}^\top\|_F^2, \\ \text{s.t.} \quad & \mathbf{B} \in \{-1, 1\}^{n \times r}. \end{aligned} \quad (1)$$

As proved in [34], the inner product of hash codes reflects the opposite of the Hamming distance. Thus, Eq. (1) uses the inner product to approximate the semantic similarities with the square loss. The main drawback of this model is that it is time-consuming for optimization when n is large.

To tackle this problem, we propose the following novel loss function:

$$\min_{\mathbf{B}, \mathbf{G}} \|r\mathbf{S} - \mathbf{B}(\mathbf{Y}\mathbf{G})^\top\|_F^2 + \mu \|\mathbf{B} - \mathbf{Y}\mathbf{G}\|_F^2 \quad (2)$$

$$s.t. \quad \mathbf{B} \in \{-1, 1\}^{n \times r},$$

where we introduce two matrices into the loss function, i.e., $\mathbf{G}$ and $\mathbf{Y}$. $\mathbf{G} \in \mathbb{R}^{c \times r}$ is a projection from $\mathbf{Y}$ to $\mathbf{B}$; $\mathbf{Y}$ is the label matrix, and μ is a balance parameter.

The design of Eq. (2) has several major advantages. First, we can supervise the learning of hash codes by leveraging both pairwise similarity $\mathbf{S}$ and label matrix $\mathbf{Y}$ rather than only one of them, which can ensure more precise hash codes. Second, as revealed in [10], using the real-valued information in Eq. (1) rather than the binary $\mathbf{B}$ can obtain more accurate approximation of similarity $\mathbf{S}$. Thus, we use real-valued $\mathbf{Y}\mathbf{G}$ to replace one binary matrix $\mathbf{B}$ in Eq. (1), and $\mathbf{Y}\mathbf{G}$ has been proved effective to approximate $\mathbf{B}$ in [11, 45]. Third, by introducing $\mathbf{Y}\mathbf{G}$ and replacing one $\mathbf{B}$, we can tactfully avoid the direct optimization on large $\mathbf{S}$. As will be discussed in the optimization section, we can compute $\mathbf{S}\mathbf{Y}$ offline and directly use the term of $\mathbf{S}\mathbf{Y}$ instead of $\mathbf{S}$ in the learning. It is notable that the size of $\mathbf{S}\mathbf{Y}$ is only $n \times c$ (c is usually much smaller than n); thus, solving the optimization problem can become much more efficient.

3.3 Hash Codes Learning – Optimization of Loss Function

As shown above, there are two variables in Eq. (2), i.e., $\mathbf{B}$ and $\mathbf{G}$. To tackle this optimization problem, in this section, we propose an alternating strategy for it, which has two alternating steps: Updating $\mathbf{G}$ by fixing $\mathbf{B}$ and updating $\mathbf{B}$ by fixing $\mathbf{G}$.

3.3.1 Updating $\mathbf{G}$ by Fixing $\mathbf{B}$. Apparently, once $\mathbf{B}$ is fixed, the optimization problem has a closed-form solution. Then, we can get $\mathbf{G}$ by setting the derivative of (2) with respect to $\mathbf{G}$ equal to zero, and we obtain:

$$\mathbf{G} = (\mathbf{Y}^\top \mathbf{Y})^{-1} (r(\mathbf{S}\mathbf{Y})^\top \mathbf{B} + \mu \mathbf{Y}^\top \mathbf{B}) (\mathbf{B}^\top \mathbf{B} + \mu \mathbf{I}_{r \times r})^{-1}. \quad (3)$$

If we directly use $\mathbf{S}$ during the optimization, the space and time cost is unacceptable when training data set is large. Fortunately, we can compute $\mathbf{S}\mathbf{Y}$ offline before the training and directly load $\mathbf{S}\mathbf{Y}$ during the optimization. Denoting $\mathbf{S}\mathbf{Y}$ as $\mathbf{A}$, Eq. (3) can be further rewritten as:

$$\mathbf{G} = (\mathbf{Y}^\top \mathbf{Y})^{-1} (r\mathbf{A}^\top \mathbf{B} + \mu \mathbf{Y}^\top \mathbf{B}) (\mathbf{B}^\top \mathbf{B} + \mu \mathbf{I}_{r \times r})^{-1}, \quad (4)$$

where the size of $\mathbf{A}$ is $n \times c$, and c is usually much smaller than n . Thus, during the optimization, SSDH can bypass the drawback of using large pairwise similarity matrix.

3.3.2 Updating $\mathbf{B}$ by Fixing $\mathbf{G}$. When updating $\mathbf{B}$, we need to consider two scenarios, i.e., single-label data and multi-label data. Both of them are discussed in the following paragraphs.

Single-Label Data: To consider the first term in Eq. (2), inspired by the work in [18], we first replace the Frobenius norm of it with

the L_1 norm; therefore, the problem becomes:

$$\min_{\mathbf{B}} \|r\mathbf{S} - \mathbf{B}(\mathbf{Y}\mathbf{G})^\top\|_1, \quad s.t., \mathbf{B} \in \{-1, 1\}^{n \times r}. \quad (5)$$

Then we have the solution $\mathbf{B} = \text{sgn}(\mathbf{S}\mathbf{Y}\mathbf{G})$. For single-label data, we can easily find the value of $\text{sgn}(\mathbf{S}\mathbf{Y}\mathbf{G})$ is equivalent to $\text{sgn}(\mathbf{Y}\mathbf{G})$. Thus, for the first term in Eq. (2), $\mathbf{B} = \text{sgn}(\mathbf{Y}\mathbf{G})$ is the solution. For the second term in Eq. (2), we can easily find that $\mathbf{B} = \text{sgn}(\mathbf{Y}\mathbf{G})$ is also the solution.

Thus, for single-label data, the solution to Eq. (2) is:

$$\mathbf{B} = \text{sgn}(\mathbf{Y}\mathbf{G}). \quad (6)$$

Multi-Label Data: For this scenario, $\text{sgn}(\mathbf{S}\mathbf{Y}\mathbf{G})$ is not equivalent to $\text{sgn}(\mathbf{Y}\mathbf{G})$ any more, which means that $\mathbf{B} = \text{sgn}(\mathbf{Y}\mathbf{G})$ is no longer a solution to Eq. (2). Instead, we use discrete cyclic coordinate descent (DCC) to obtain $\mathbf{B}$ with a closed-form solution. First, by omitting constant terms $\|r\mathbf{S}\|_F^2$, $\mu \|\mathbf{B}\|_F^2$ and $\mu \|\mathbf{Y}\mathbf{G}\|_F^2$, we rewrite Eq. (2) as follows:

$$\begin{aligned} \min_{\mathbf{B}} & \| \mathbf{Y}\mathbf{G}\mathbf{B}^\top \|_F^2 - 2\text{Tr}(\mathbf{B}^\top \mathbf{S}\mathbf{Y}\mathbf{G} + \mu \mathbf{B}^\top \mathbf{Y}\mathbf{G}) \\ &= \| \mathbf{Y}\mathbf{G}\mathbf{B}^\top \|_F^2 - 2\text{Tr}(\mathbf{B}^\top (\mathbf{A}\mathbf{G} + \mu \mathbf{Y}\mathbf{G})) \\ &= \| \mathbf{C}\mathbf{B}^\top \|_F^2 - 2\text{Tr}(\mathbf{B}^\top \mathbf{Q}) \end{aligned} \quad (7)$$

$$s.t. \quad \mathbf{B} \in \{-1, 1\}^{n \times r},$$

where $\mathbf{A} = \mathbf{S}\mathbf{Y}$, $\mathbf{C} = \mathbf{Y}\mathbf{G}$ and $\mathbf{Q} = \mathbf{A}\mathbf{G} + \mu \mathbf{C}$. Then, we solve $\mathbf{B}$ bit by bit, which means that we iteratively learn every column of $\mathbf{B}$ by fixing all other columns. To do this, we suppose $\mathbf{b}$ is the l -th column of $\mathbf{B}$, i.e., $\mathbf{b}$ is the vector composed of the l -th bits of all samples, $l = 1, 2, \dots, r$ and $\mathbf{B}'$ is the matrix excluding $\mathbf{b}$. Similarly, $\mathbf{q}$ is the l -th row of $\mathbf{Q}$, $\mathbf{Q}'$ is the matrix of $\mathbf{Q}$ excluding $\mathbf{q}$; $\mathbf{c}$ is the l -th row of $\mathbf{C}$, $\mathbf{C}'$ the matrix of $\mathbf{C}$ excluding $\mathbf{c}$. Then we have:

$$\begin{aligned} \| \mathbf{C}\mathbf{B}^\top \|_F^2 &= \text{Tr}(\mathbf{B}\mathbf{C}^\top \mathbf{C}\mathbf{B}^\top) \\ &= \| \mathbf{b}\mathbf{c} \|_F^2 + 2\mathbf{c}\mathbf{C}'\mathbf{B}'^\top \mathbf{b} + \text{const} \\ &= 2\mathbf{c}^\top \mathbf{C}'\mathbf{B}'^\top \mathbf{b} + \text{const}. \end{aligned} \quad (8)$$

Here $\| \mathbf{b}\mathbf{c} \|_F^2 = \text{Tr}(\mathbf{c}^\top \mathbf{b}\mathbf{b}^\top \mathbf{c}) = \mathbf{nc}^\top \mathbf{c} = \text{const}$. Correspondingly, we have:

$$2\text{Tr}(\mathbf{B}^\top \mathbf{Q}) = \mathbf{q}^\top \mathbf{b} + \text{const}. \quad (9)$$

Then, the optimization problem can be further reduced to the following form:

$$\begin{aligned} \min_{\mathbf{b}} & (\mathbf{c}^\top \mathbf{C}'\mathbf{B}'^\top + \mathbf{q}^\top) \mathbf{b} \\ s.t. & \quad \mathbf{b} \in \{-1, 1\}^n. \end{aligned} \quad (10)$$

And the optimal solution is:

$$\mathbf{b} = \text{sgn}(\mathbf{q} - \mathbf{B}'\mathbf{C}'^\top \mathbf{c}). \quad (11)$$

We can see that each bit b is computed based on the pre-learned bits of $\mathbf{B}$. We can iteratively update each bit until the procedure converges to a set of better codes $\mathbf{B}$.

3.3.3 The Whole Algorithm. The above optimization procedure of SSDH is summarized in Algorithm 1.

Algorithm 1: Discrete optimization in SSDH.

Input: $\mathbf{A} = \mathbf{SY}$, which is computed offline; the label matrix $\mathbf{Y} \in \{0, 1\}^{n \times c}$; the parameter μ ; the iteration number t ; the hash bit length r .

Output: Hash codes $\mathbf{B} \in \{-1, 1\}^{n \times r}$; the projection matrix $\mathbf{G}$. Initialize $\mathbf{G}$ and $\mathbf{B}$ by randomization.

for $iter = 1 \rightarrow t$ **do**

Update $\mathbf{G}$ by fixing $\mathbf{B}$, using Eq. (4).

if *single-label* **then**

Update $\mathbf{B}$ by fixing $\mathbf{G}$, using Eq. (6).

end

if *multi-label* **then**

Update $\mathbf{B}$ by fixing $\mathbf{G}$, using Eq. (11).

end

end

3.4 Hash Function Learning

Once the hash codes are learnt, SSDH needs to learn hash function which can transform the original features into binary codes, e.g., for out-of-sample instances. As mentioned previously, the hash function learning can be viewed as a binary classification problem for any bit of the codes and any arbitrary classifiers can be adopted as hash function. SSDH chooses linear regression on account of its efficiency¹.

When the linear regression is adopted, the hash function can be obtained by solving the following square loss problem:

$$L_e = \|\mathbf{B} - \mathbf{X}\mathbf{W}\|_F^2 + \lambda_e \|\mathbf{W}\|_F^2, \quad (12)$$

where λ_e is a balance parameter, and $\|\mathbf{W}\|_F^2$ is a regularization term. Then, we can get the optimal solution as:

$$\mathbf{W} = (\mathbf{X}^\top \mathbf{X} + \lambda_e \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{B}. \quad (13)$$

3.5 Out-of-Sample Extension

Once the hash function is learnt, SSDH can easily generate the hash codes for new queries. Suppose X_{query} and B_{query} are the original features and corresponding hash codes of queries, respectively. Then, B_{query} can be obtained by

$$\mathbf{B}_{query} = F(\mathbf{X}_{query}) = \text{sgn}(\mathbf{X}_{query} \mathbf{W}). \quad (14)$$

3.6 Kernelization

Kernelization is a popular and powerful technique in supervised hashing literature. For data point $\mathbf{x} \in \mathbb{R}^d$, we randomly sampled m anchor points, i.e., $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_m$ and mapped $\mathbf{x}$ into an m -dimensional kernel feature representation, using

$$\phi(\mathbf{x}) = [\exp(\|\mathbf{x} - \mathbf{o}_1\|^2 / \sigma), \dots, \exp(\|\mathbf{x} - \mathbf{o}_m\|^2 / \sigma)].$$

Here, σ is the kernel width. We set $m = 1,000$ and estimated σ according to the average Euclidean distances between the training samples.

Table 1: General statistics of six datasets.

dataset	label kind	training size	query size	number of labels
MNIST	single-label	69,000	1,000	10
MIRFlickr	multi-label	15,738	1,000	24
CIFAR-10	single-label	59,000	1,000	10
CIFAR-100	single-label	59,000	1,000	100
NUS-WIDE	multi-label	193,833	2,100	21
ImageNet	single-label	1,198,336	63,070	1,000

4 EXPERIMENTAL RESULTS

4.1 Experimental Setup

Datasets: We conducted experiments on six publicly available and widely used datasets to evaluate the proposed method.

- *MNIST*² [24]: This dataset is a subset of a larger set available from NIST. It consists of 70,000 images of handwritten digits from ‘0’ to ‘9’, and thus has 10 classes. Each image in this dataset is represented by 784-dimension feature vectors.
- *MIRFlickr*³ [16]: It is a real-word dataset collected from Flickr website with 25,000 instances. Each instance is manually annotated with at least one of 24 labels. Following the pretreatment in [31, 55], we remove the instances without labels or textual tags appearing less than 20 times. Finally, there are 16,738 instances are left. For each instance, its image is represented by 150-dimension edge histogram.
- *CIFAR-10*⁴ [19]: It is collected from 80 million tiny images. Every image in CIFAR-10 is manually labeled with one of 10 classes; with each class containing 6,000 samples, CIFAR-10 has a number of 60,000 images in all. Images are represented by 512-dimension GIST [43] feature vectors extracted from the original color image of size 32×32 .
- *CIFAR-100*⁵ [19]: It is also collected from the 80 million tiny images. It consists of 100 classes with each class containing 600 color images, which results in 60,000 images in total. Every image is represented by a 512-dimensional GIST feature vector.
- *NUS-WIDE*⁶ [6]: It is a real-world image dataset originally containing 269,648 samples collected from Flickr. As in [45], we collect the 21 most frequent label for evaluating and 195,834 images are left. The provided 500-dimensional bag-of-words features are used.
- *ImageNet*⁷ [44]: We also evaluate our method on the training set of ILSVRC2010. This dataset is a subset of ImageNet and contains 1.2 million images. These images belong to 1,000 categories, with each category having at least 600 samples. We use the released 1,000-dimensional BoW features, which are computed from SIFT features of original images, to represent instances.

¹SSDH can choose more powerful classifiers and the accuracy can be further improved. But the choice of classifiers is not the focus of our paper, and we leave the study of other candidate classifiers for future pursuit.

²<http://yann.lecun.com/exdb/mnist/>

³<http://press.liacs.nl/mirflickr/mirdownload.html>

⁴<https://www.cs.toronto.edu/~kriz/cifar.html>

⁵<https://www.cs.toronto.edu/~kriz/cifar.html>

⁶<http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm>

⁷<http://image-net.org/challenges/LSVRC/2010/>

Table 2: The MAP results and training time of all methods on MNIST. The best MAPs for each category are shown in boldface.

	MAP					Training Time (in second)				
Method	8 bits	16 bits	32 bits	64 bits	96 bits	8 bits	16 bits	32 bits	64 bits	96 bits
LSH	0.1731	0.2202	0.2686	0.3267	0.3553	0.040	0.044	0.062	0.091	0.129
ITQ	0.3818	0.4132	0.4367	0.4601	0.4673	0.892	1.42	2.72	5.27	7.18
KSH	0.7103	0.7880	0.8258	0.8434	0.8428	48.02	90.72	179.82	337.53	508.68
SDH	0.5956	0.8544	0.8861	0.8945	0.8963	6.45	7.13	17.27	59.60	127.06
NSH	0.7314	0.8667	0.8934	0.9102	0.9105	4.33	5.25	5.38	6.89	8.51
FSDH	0.8701	0.8940	0.9139	0.9200	0.9253	8.60	7.93	8.71	9.94	10.46
COSDISH	0.7895	0.8498	0.8740	0.8866	0.8861	4.59	8.08	30.31	107.01	241.08
SSDH	0.9338	0.9557	0.9639	0.9669	0.9683	3.68	3.80	3.81	3.93	4.09

Table 3: The MAP results and training time of all methods on MIRFlickr. The best MAPs for each category are shown in boldface.

	MAP					Training Time (in second)				
Method	8 bits	16 bits	32 bits	64 bits	96 bits	8 bits	16 bits	32 bits	64 bits	96 bits
LSH	0.5669	0.5760	0.5712	0.5862	0.5885	0.005	0.005	0.007	0.013	0.016
ITQ	0.5760	0.5721	0.5794	0.5796	0.5806	0.134	0.154	0.305	0.589	0.925
KSH	0.5780	0.5775	0.5794	0.5782	0.5790	28.69	57.02	109.05	219.11	329.99
SDH	0.6023	0.6071	0.6138	0.6207	0.6220	1.31	1.90	3.99	13.19	30.38
NSH	0.6143	0.6180	0.6158	0.6243	0.6270	0.873	0.932	1.03	1.31	1.59
FSDH	0.5908	0.5957	0.6053	0.6124	0.6152	2.01	2.02	2.07	2.13	2.20
COSDISH	0.6819	0.6939	0.7036	0.7173	0.7185	0.639	1.66	6.04	29.85	69.99
SSDH	0.7099	0.7128	0.7154	0.7198	0.7258	0.722	0.849	1.32	3.10	6.29

Table 4: The MAP results and training time of all methods on CIFAR-10. The best MAPs for each category are shown in boldface.

	MAP					Training Time (in second)				
Method	8 bits	16 bits	32 bits	64 bits	96 bits	8 bits	16 bits	32 bits	64 bits	96 bits
LSH	0.1077	0.1165	0.1199	0.1270	0.1360	0.021	0.025	0.032	0.044	0.064
ITQ	0.1429	0.1482	0.1569	0.1487	0.1551	0.483	0.743	1.32	2.54	3.98
KSH	0.2315	0.2623	0.2905	0.3095	0.3199	29.91	59.59	121.00	243.73	359.11
SDH	0.2373	0.3528	0.3868	0.4049	0.4159	4.58	5.55	9.09	29.54	83.13
NSH	0.2902	0.2936	0.3220	0.3390	0.3687	2.99	3.675	3.70	4.81	5.75
FSDH	0.3311	0.3884	0.4270	0.4522	0.4646	6.05	6.09	6.26	6.65	7.05
COSDISH	0.5012	0.5745	0.6094	0.6321	0.6394	2.10	4.33	15.19	76.99	180.34
SSDH	0.5151	0.6698	0.6991	0.7046	0.7063	2.61	2.69	2.66	2.67	2.73

Table 5: The MAP results and training time of all methods on CIFAR-100. The best MAPs for each category are shown in boldface.

	MAP					Training Time (in second)				
Method	8 bits	16 bits	32 bits	64 bits	96 bits	8 bits	16 bits	32 bits	64 bits	96 bits
LSH	0.0117	0.0125	0.0139	0.0153	0.0169	0.021	0.025	0.027	0.0485	0.059
ITQ	0.0125	0.0151	0.0170	0.0193	0.0203	0.468	0.711	1.28	2.46	4.01
KSH	0.0225	0.0268	0.0290	0.0307	0.0338	32.63	63.60	128.91	257.12	386.89
SDH	0.0217	0.0339	0.0473	0.0589	0.0392	3.83	5.77	9.69	80.92	190.57
NSH	0.0323	0.0483	0.0649	0.0758	0.0837	3.24	3.53	4.07	5.58	6.99
FSDH	0.0256	0.0419	0.0598	0.0690	0.0884	6.61	6.55	6.73	7.01	7.57
COSDISH	0.0379	0.0752	0.1449	0.2019	0.2292	1.99	6.39	16.76	79.82	185.28
SSDH	0.1001	0.1640	0.2348	0.2824	0.2674	3.03	3.11	3.16	3.30	3.34

Each dataset is randomly split into a training set and a query set. On NUS-WIDE, for each label, 100 images are randomly sampled (2,100 images with duplication) for the query set. On ImageNet, we construct a query set by randomly sampling 5% instances. The statistics of these datasets are given in Table 1.

For single-label datasets, if two samples have the same class label, they are considered to be semantically similar, and dissimilar otherwise. For multi-label datasets, if two samples share at least one semantic label, they are considered to be semantically similar.

Competing Methods: We compared SSDH against seven state-of-the-art hashing algorithms, including unsupervised and supervised ones:

- LSH [8]: Locality-Sensitive Hashing (unsupervised). Its basic idea is to hash the points from the database so as to ensure that the probability of collision is much higher for objects that are close to each other than for those that are far apart.
- ITQ [9]: Iterative Quantization (unsupervised). ITQ contains a simple and efficient alternating minimization scheme for finding a rotation of zero-centered data so as to minimize the quantization error of mapping this data to the vertices of a zero-centered binary hypercube.
- KSH [34]: Kernel-Based Supervised Hashing (supervised). KSH utilizes the equivalence between optimizing the code inner products and the Hamming distances.
- SDH [45]: Supervised Discrete Hashing (supervised). Its learning objective is to generate the optimal binary hash codes for linear classification.
- NSH [32]: Natural Supervised Hashing (supervised). The key idea of NSH is to treat label vectors as binary codes and to learn target codes which have similar structure to label vectors.
- FSDH [11]: Fast Supervised Discrete Hashing (supervised). FSDH uses a very simple yet effective regression of the class labels of training examples to the corresponding hash code to accelerate the algorithm.
- COSDISH [18]: Column Sampling based Discrete Supervised Hashing (supervised). COSDISH directly learns the discrete hash code from semantic information.

For most baselines we used the codes kindly provided by the authors and we carefully tuned their parameters according to the scheme suggested by the authors. As the code of NSH and FSDH are not publicly available, we implemented the code exactly following the algorithms of them and tuned the parameters according to the scheme suggested in their papers. For SSDH, we set $t = 5$ which is enough to get satisfactory performance, $\mu = 1$ and $\lambda_e = 1$. All methods run on a workstation with Intel(R) XEON(R) CPU E5-2650@2.30GHz, 128GB memory.

Following [18, 30, 34], we randomly sampled 2,000 images as training sets for KSH on all datasets due to its high time complexity.

Evaluation Metrics: To fully evaluate SSDH and all of baselines, we adopted three widely used metrics, i.e., mean average precision (MAP), precision-recall curves and topN-precision curves. For all metrics, a larger value indicates the better retrieval performance.

The average precision (AP) for a query is defined as

$$AP = \frac{1}{R} \sum_{r=1}^n Precision(r) \delta(r). \quad (15)$$

where R is the number of ground-truth neighbors of the query in a database, n is the number of samples in the database, $Precision(r)$ denotes the precision of the top r retrieved entities, and $\delta(r) = 1$ if the r -th retrieved entity is a ground-truth neighbour and $\delta(r) = 0$ otherwise. For a query set whose size is M , the MAP is defined as the mean of the average precision scores for all the queries in the query set, i.e.,

$$MAP = \frac{1}{M} \sum_{i=1}^M AP_i. \quad (16)$$

Precision-recall curves can be obtained by varying the Hamming radius of the retrieved points and evaluating the precision, recall and the number of retrieved points.

TopN-precision curves reflects the change of precision with respect to the number of top-ranked N instances returned to the users, which is expressive for retrieval.

4.2 Comparison Results

4.2.1 MAP and Time. The MAP results and training time of all methods on MNIST, MIRFlickr, CIFAR-10 and CIFAR-100 are listed in Table 2, 3, 4, and 5, respectively.

From these tables, we can observe that:

- SSDH can outperform all baselines on MAP results in all cases, which verifies its effectiveness.
- Unsupervised hashing methods LSH and ITQ require the least amount of training time. However, all supervised methods outperform them.
- On single-label datasets, i.e., MNIST, CIFAR-10, and CIFAR-100, SSDH can be efficiently trained. For example, SSDH needs shorter time than other supervised baselines. Besides, SSDH's training time is robust to hash bit lengths while KSH, SDH, and COSDISH need more training time with longer bit lengths.
- On multi-label datasets, i.e., MIRFlickr, we can also train SSDH fast. As SSDH needs to learn binary codes bit by bit, it takes more time to learn longer hash codes. However, compared to baselines KSH, SDH, and COSDISH which also learn hash codes bit by bit, SSDH is still the fastest one.
- When the number of labels getting larger, data becomes more complicated and search becomes more challenging. It can be seen from that MAP results of all methods on CIFAR-100 are much worse than those on CIFAR-10.

From the MAP and time cost results in these table, we can conclude SSDH is effective and efficient on these datasets.

4.2.2 Precision-Recall and TopN-Precision Curves. To gain further insights, we plotted the precision-recall curves and topN-precision curves on MNIST, MIRFlickr, CIFAR-10, and CIFAR-100 in Figure 1 and Figure 2, respectively. Note that one column in these figures demonstrates the results on one dataset.

From these figures, we can find that:

- In terms of precision-recall curves, SSDH performs much better than all baselines in all cases.
- In terms of the topN-precision curves, in all cases, the performance gains provided by SSDH over the best baseline are significant. Especially, SSDH obtains much better results than other baselines when N is small. This means

Figure 1: Precision v.s. Recall curves on MNIST, MIRFlickr, CIFAR-10 and CIFAR-100 with code lengths from 8 to 96.

Figure 2: TopN-precision curves on MNIST, MIRFlickr, CIFAR-10 and CIFAR-100 with code lengths from 8 to 96.

Table 6: The MAP results and training time of all methods on NUS-WIDE.

Method	MAP			Training Time (in second)		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
LSH	0.3107	0.3165	0.3197	0.06	0.08	0.13
ITQ	0.3343	0.3348	0.3364	2.55	5.53	9.63
KSH	0.3703	0.3728	0.3785	52.53	105.24	208.97
SDH	0.4113	0.4114	0.4135	28.94	77.95	250.95
NSH	0.4075	0.4135	0.4133	10.97	11.93	15.26
FSDH	0.4052	0.4115	0.4155	19.86	19.26	21.48
COSDISH	0.5765	0.5947	0.5942	16.11	56.27	232.55
SSDH	0.6023	0.6035	0.6069	10.88	20.86	56.02

Table 7: The MAP results and training time of all methods on ImageNet.

Method	MAP			Training Time (in second)		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
LSH	0.0015	0.0018	0.0021	0.40	0.51	0.92
ITQ	0.0027	0.0028	0.0031	23.16	35.92	60.20
KSH	0.0023	0.0020	0.0024	41.81	92.13	228.22
SDH	0.0026	0.0038	0.0051	979.42	1859.35	5327.54
NSH	0.0045	0.0052	0.0065	92.46	112.55	145.41
FSDH	0.0027	0.0037	0.0051	222.28	230.02	248.99
COSDISH	0.0041	0.0141	0.0212	99.33	344.98	1837.24
SSDH	0.0122	0.0209	0.0335	269.38	282.78	290.51

that SSDH can return much more similar samples at the beginning, which is very important for a retrieval model.

From the results of these figures, we can also conclude, compared to all baselines, SSDH can achieve competitive performance.

4.3 Scalable Search

We further carried out experiments on two large-scale datasets, i.e., NUS-WIDE and ImageNet.

4.3.1 Results on NUS-WIDE. The MAP results of all methods on NUS-WIDE are shown in Table 6. To further evaluate the scalability of SSDH and all baselines, training time is also reported in this table. From this table, we can find that:

- SSDH obtains the best MAP results in all cases.
- The training process of SSDH is very fast and scales well. On NUS-WIDE, which has nearly 200,000 samples, the training costs in cases of 16-bit, 32-bit, and 64-bit are about 10, 20, and 56 seconds, respectively.
- The MAP results of COSDISH are comparable to those of SSDH and both of them perform better than other baselines with large accuracy gains. But COSDISH needs much more training time than SSDH. Thus, SSDH is more scalable than COSDISH.
- With code lengths being 32 and 64, NSH and FSDH can be trained faster than SSDH because they simultaneously learn all bits. As SSDH learns hash codes bit by bit for multi-label data, it needs more time than NSH and FSDH. However,

SSDH can obtain much better MAP results than both NSH and FSDH with slightly increased time cost.

In a nutshell, SSDH is the most practical supervised hashing method, especially for large-scale datasets.

4.3.2 Results on ImageNet. All methods' MAP results and training time on ImageNet are listed in Table 7. From this table, we can observe that:

- SSDH obtains much better accuracy than all baselines in all cases on this dataset.
- Compared to the results on other datasets, the accuracy of all methods on ImageNet is a little low. The main reason is that ImageNet is a more challenging dataset. For example, it has more than 1 million instances; moreover, it has as much as 1,000 classes, which makes it complicated.
- SDH and KSH are not scalable as they need too much time. Note that KSH is trained on a subset containing 2,000 samples.
- COSDISH can outperform other baselines in cases of 32-bit and 64-bit. But with the code length increasing, it becomes time-consuming and cannot scale well.
- Compared to SSDH, NSH, and FSDH need less time for training; however, their accuracy is not satisfactory, even not acceptable. Besides, compared to them, SSDH only needs a little more time to finish its training.

From the results on ImageNet, we can further conclude that SSDH scales well on large-scale dataset (more than 1 million) and is the most practical supervised hashing method.

5 CONCLUSIONS

In this paper, we proposed a novel scalable hashing method, i.e., the Scalable Supervised Discrete Hashing (SSDH). By replacing one binary matrix $\mathbf{B}$ by $\mathbf{YG}$ ($\mathbf{Y}$ is the label matrix, $\mathbf{G}$ is a projection matrix) and computing $\mathbf{SY}$ offline, SSDH bypasses the direct optimization on $n \times n$ large matrix $\mathbf{S}$ during the optimization. Besides, as SSDH makes use of both pairwise similarity matrix $\mathbf{S}$ and label matrix $\mathbf{L}$, more semantic information can be embedded to the learning of hash codes. Moreover, based on the proposed optimization algorithm, SSDH can discretely learn hash codes. Due to the well-designed loss function and the efficient alternative optimization algorithm, SSDH achieves high accuracy with acceptable time cost; even the large-scale data can be efficiently and effectively handled. Experiments conducted on six datasets demonstrate the superiority of SSDH to the existing state-of-the-art unsupervised and supervised hashing methods.

6 ACKNOWLEDGEMENTS

This work was partially supported by the Key Research and Development Program of Shandong Province (2016GGX101044) and National Natural Science Foundation of China (61573212).

REFERENCES

- [1] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. In *NIPS*. 1225–1233.
- [2] Fatih Cakir and Stan Sclaroff. 2015. Adaptive hashing for fast similarity search. In *ICCV*. 1044–1052.

- [3] Fatih Cakir and Stan Sclaroff. 2015. Online supervised hashing. In *ICIP*. 2606–2610.
- [4] Yuan Cao, Heng Qi, Wenrui Zhou, Jien Kato, Keqiu Li, Xiulong Liu, and Jie Gui. 2018. Binary hashing for approximate nearest neighbor search on big data: A survey. *IEEE Access* 6 (2018), 2039–2054.
- [5] Lianhua Chi and Xingquan Zhu. 2017. Hashing Techniques: A survey and taxonomy. *CSUR* 50, 1 (2017), 11.
- [6] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao Zheng. 2009. NUS-WIDE: A real-world web image database from National University of Singapore. In *CIVR*. 48:1–48:9.
- [7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*. 253–262.
- [8] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*. 518–529.
- [9] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI* 35, 12 (2013), 2916–2929.
- [10] Albert Gordo, Florent Perronnin, Yunchao Gong, and Svetlana Lazebnik. 2014. Asymmetric distances for binary embeddings. *TPAMI* 36, 1 (2014), 33–47.
- [11] Jie Gui, Tongliang Liu, Zhenan Sun, Dacheng Tao, and Tieniu Tan. 2018. Fast supervised discrete hashing. *TPAMI* 40, 2 (2018), 490–496.
- [12] Yanbin Hao, Tingting Mu, John Y Goulermas, Jianguo Jiang, Richang Hong, and Meng Wang. 2017. Unsupervised t-distributed video hashing and its deep hashing extension. *TIP* 26, 11 (2017), 5531–5544.
- [13] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *SIGIR*. ACM, 355–364.
- [14] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*. 549–558.
- [15] Long-Kai Huang and Sinno Jialin Pan. 2016. Class-wise supervised hashing with label embedding and active bits. In *IJCAI*. 1585–1591.
- [16] Mark J. Huiskes and Michael S. Lew. 2008. The MIR flickr retrieval evaluation. In *MIR*. 39–43.
- [17] Qing-Yuan Jiang and Wu-Jun Li. 2015. Scalable graph hashing with feature transformation. In *IJCAI*. 2248–2254.
- [18] Wang-Cheng Kang, Wu-Jun Li, and Zhi-Hua Zhou. 2016. Column sampling based discrete supervised hashing. In *AAAI*. 1230–1236.
- [19] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto* (2009).
- [20] Brian Kulis and Kristen Grauman. 2009. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*. 2130–2137.
- [21] Brian Kulis, Prateek Jain, and Kristen Grauman. 2009. Fast similarity search for learned metrics. *TPAMI* 31, 12 (2009), 2143–2157.
- [22] Shaishav Kumar and Raghavendra Udupa. 2011. Learning hash functions for cross-view similarity search. In *IJCAI*. 1360–1365.
- [23] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. 2015. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*. 3270–3278.
- [24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *P IEEE* 86, 11 (1998), 2278–2324.
- [25] Cong Leng, Jian Cheng, Jiaxiang Wu, Xi Zhang, and Hanqing Lu. 2014. Supervised hashing with soft constraints. In *CIKM*. 1851–1854.
- [26] Cong Leng, Jiaxiang Wu, Jian Cheng, Xi Zhang, and Hanqing Lu. 2015. Hashing for distributed data. In *ICML*. 1642–1650.
- [27] Ping Li, Anshumali Shrivastava, Joshua L Moore, and Arnd C König. 2011. Hashing algorithms for large-scale learning. In *NIPS*. 2672–2680.
- [28] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. 2016. Feature learning based deep supervised hashing with pairwise labels. In *IJCAI*. 1711–1717.
- [29] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton Van den Hengel, and David Suter. 2014. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*. 1963–1970.
- [30] Guosheng Lin, Chunhua Shen, David Suter, and Anton van den Hengel. 2013. A general two-step approach to learning-based hashing. In *ICCV*. 2552–2559.
- [31] Zijia Lin, Guiguang Ding, Mingqing Hu, and Jianmin Wang. 2015. Semantics-preserving hashing for cross-view retrieval. In *CVPR*. 3864–3872.
- [32] Qi Liu and Hongtao Lu. 2016. Natural supervised hashing. In *IJCAI*. 1788–1794.
- [33] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. 2014. Discrete graph hashing. In *NIPS*. 3419–3427.
- [34] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. 2012. Supervised hashing with kernels. In *CVPR*. 2074–2081.
- [35] Xianglong Liu, Cheng Deng, Bo Lang, Dacheng Tao, and Xuelong Li. 2016. Query-adaptive reciprocal hash tables for nearest neighbor search. *TIP* 25, 2 (2016), 907–919.
- [36] Xianglong Liu, Lei Huang, Cheng Deng, Jiwen Lu, and Bo Lang. 2015. Multi-view complementary hash tables for nearest neighbor search. In *ICCV*. 1107–1115.
- [37] Mingsheng Long, Yue Cao, Jianmin Wang, and Philip S Yu. 2016. Composite correlation quantization for efficient multimodal retrieval. In *SIGIR*. 579–588.
- [38] Liqiang Nie, Meng Wang, Zhengjun Zha, Guangda Li, and Tat-Seng Chua. 2011. Multimedia answering: enriching text QA with media information. In *SIGIR*. 695–704.
- [39] Liqiang Nie, Meng Wang, Zheng-Jun Zha, and Tat-Seng Chua. 2012. Oracle in image search: A content-based approach to performance prediction. *TOIS* 30, 2 (2012), 13.
- [40] Liqiang Nie, Shuicheng Yan, Meng Wang, Richang Hong, and Tat-Seng Chua. 2012. Harvesting visual concepts for image search with complex queries. In *MM*. 59–68.
- [41] Liqiang Nie, Yi-Liang Zhao, Mohammad Akbari, Jialie Shen, and Tat-Seng Chua. 2015. Bridging the vocabulary gap between health seekers and healthcare knowledge. *TKDE* 27, 2 (2015), 396–409.
- [42] Mohammad Norouzi and David M Blei. 2011. Minimal loss hashing for compact binary codes. In *ICML*. 353–360.
- [43] Aude Oliva and Antonio Torralba. 2001. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV* 42, 3 (2001), 145–175.
- [44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet large scale visual recognition challenge. *IJCV* 115, 3 (2015), 211–252.
- [45] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. 2015. Supervised discrete hashing. In *CVPR*. 37–45.
- [46] Fumin Shen, Chunhua Shen, Qinfeng Shi, Anton Hengel, and Zhenmin Tang. 2013. Inductive hashing on manifolds. In *CVPR*. 1562–1569.
- [47] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *NIPS*. 2321–2329.
- [48] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2010. Semi-supervised hashing for scalable image retrieval. In *CVPR*. 3424–3431.
- [49] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. 2018. A survey on learning to hash. *TPAMI* 40, 4 (2018), 769–790.
- [50] Meng Wang, Weijie Fu, Shijie Hao, Hengchang Liu, and Xindong Wu. 2017. Learning on big graph: Label inference and regularization with anchor hierarchy. *TKDE* 29, 5 (2017), 1101–1114.
- [51] Meng Wang, Weijie Fu, Shijie Hao, Dacheng Tao, and Xindong Wu. 2016. Scalable semi-supervised learning by efficient anchor graph regularization. *TKDE* 28, 7 (2016), 1864–1877.
- [52] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *NIPS*. 1753–1760.
- [53] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. 2014. Supervised hashing for image retrieval via image representation learning. In *AAAI*. 2156–2162.
- [54] Liang Xie, Jialie Shen, and Lei Zhu. 2016. Online cross-modal hashing for web image retrieval. In *AAAI*. 294–300.
- [55] Xing Xu, Fumin Shen, Yang Yang, Heng Tao Shen, and Xuelong Li. 2017. Learning discriminative binary codes for large-scale cross-modal retrieval. *TIP* 26, 5 (2017), 2494–2507.
- [56] Xun Yang, Meng Wang, Richang Hong, Qi Tian, and Yong Rui. 2017. Enhancing person re-identification in a self-trained subspace. *TOMM* 13, 3 (2017), 27:1–27:23.
- [57] Felix Yu, Sanjiv Kumar, Yunchao Gong, and Shih-Fu Chang. 2014. Circulant binary embedding. In *ICML*. 946–954.
- [58] Dongqing Zhang and Wu-Jun Li. 2014. Large-scale supervised multimodal hashing with semantic correlation maximization. In *AAAI*. 2177–2183.
- [59] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *SIGIR*. 325–334.
- [60] Peichao Zhang, Wei Zhang, Wu-Jun Li, and Minyi Guo. 2014. Supervised hashing with latent factor models. In *SIGIR*. 173–182.
- [61] Ruimao Zhang, Liang Lin, Rui Zhang, Wangmeng Zuo, and Lei Zhang. 2015. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *TIP* 24, 12 (2015), 4766–4779.
- [62] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *SIGIR*. 183–192.
- [63] Yi Zhen, Piyush Rai, Hongyuan Zha, and Lawrence Carin. 2015. Cross-modal similarity learning via pairs, preferences, and active supervision. In *AAAI*. 3203–3209.
- [64] Lei Zhu, Zi Huang, Xiaobai Liu, Xiangnan He, Jiande Sun, and Xiaofang Zhou. 2017. Discrete multimodal hashing with canonical views for robust mobile landmark search. *TMM* 19, 9 (2017), 2066–2079.