Fast Scalable Supervised Hashing

Xin Luo
Shandong University
luoxin.lxin@gmail.com

Liqiang Nie
Shandong University
nieliqiang@gmail.com

Xiangnan He
National University of Singapore
xiangnanhe@gmail.com

Ye Wu
Shandong University
kimiwadewu@gmail.com

Zhen-Duo Chen
Shandong University
chenzd.sdu@gmail.com

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

ABSTRACT

Despite significant progress in supervised hashing, there are three common limitations of existing methods. First, most pioneer methods discretely learn hash codes bit by bit, making the learning procedure rather time-consuming. Second, to reduce the large complexity of the n by n pairwise similarity matrix, most methods apply sampling strategies during training, which inevitably results in information loss and suboptimal performance; some recent methods try to replace the large matrix with a smaller one, but the size is still large. Third, among the methods that leverage the pairwise similarity matrix, most of them only encode the semantic label information in learning the hash codes, failing to fully capture the characteristics of data. In this paper, we present a novel supervised hashing method, called Fast Scalable Supervised Hashing (FSSH), which circumvents the use of the large similarity matrix by introducing a pre-computed intermediate term whose size is independent with the size of training data. Moreover, FSSH can learn the hash codes with not only the semantic information but also the features of data. Extensive experiments on three widely used datasets demonstrate its superiority over several state-of-the-art methods in both accuracy and scalability. Our experiment codes are available at: https://lcbwlx.wixsite.com/fssh.

CCS CONCEPTS

• **Information systems** → **Image search**; • **Computing methodologies** → *Learning paradigms*;

KEYWORDS

Learning to Hash; Large-Scale Retrieval; Discrete Optimization; Supervised Hashing.

ACM Reference Format:
Xin Luo, Liqiang Nie, Xiangnan He, Ye Wu, Zhen-Duo Chen, and Xin-Shun Xu. 2018. Fast Scalable Supervised Hashing. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209978.3210035

*Corresponding author.

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3210035

1 INTRODUCTION

Similarity search lays the foundation for many fields, such as computer vision, information retrieval and recommendation [30, 31, 34, 45, 54]. In the last decades, the volume of data has increased explosively, and it is usually represented in high dimensional spaces. As an exhaustive comparison between a query and the instances in a large database is very time-consuming in the original space, traditional similarity search methods are unsuitable for scalable search. On the contrary, hashing methods transform the high dimensional data points into compact binary codes meanwhile preserving the similarity and structural information of the original data. Moreover, pairwise comparisons can be carried out extremely efficiently by using XOR operation in the Hamming space. Therefore, hashing is more suitable for the large-scale search and many promising hashing methods have been explored in recent years [6, 37, 43, 47, 49].

Roughly speaking, hashing methods can be divided into two categories, i.e., data-independent and data-dependent. The former is independent of the training data and uses random projections to generate the hash function. Locality-Sensitive Hashing [8] is one of the most representative data-independent methods. However, data-independent methods need long hash codes to achieve high precision, leading to large memory usage. In contrast, the latter learns the hash function from data and is able to achieve higher accuracy with shorter codes. To efficiently generate desirable binary codes, both unsupervised [5, 12, 46] and supervised [26, 27, 42, 53] methods have been developed for data-dependent hashing.

Instead of leveraging the label information, unsupervised methods learn the hash function by harnessing the relations in the training data. Typical examples include Spectral Hashing [40], Iterative Quantization [9], Inductive Hashing on Manifolds [36], and Scalable Graph Hashing [16]. In contrast, supervised methods take the label information into account to learn the hash function and generate the similarity-preserving binary codes. They have demonstrated higher accuracy than the unsupervised methods in many real applications, attracting increasing attention. Some representative works include Minimal Loss Hashing [32], Graph Cuts Coding [7], Fast Supervised Hashing with Decision Trees (FastHash) [24], and Scalable Supervised Discrete Hashing (SSDH) [29].

Although many supervised hashing methods have been proposed with promising results, there are several challenges remaining to be addressed. First, as the binary constraints make the hash codes learning problem hard to solve, most of them, like Kernel-Based Supervised Hashing (KSH) [28], solve a continuous optimization

problem by relaxing the binary constraints. However, the errors caused by relaxation may degrade the performance. Although some of them, such as FastHash [24], Column Sampling based Discrete Supervised Hashing (COSDISH) [17], Discrete Supervised Hashing (DISH), [51] and Scalable Supervised Discrete Hashing (SSDH) [29], can solve the binary optimization problem without relaxation, they are rather time-consuming due to the bit-wise learning strategy. That is because that one step can only optimize one bit, and making lots of steps needed to optimize all bits. Another problem comes from the large memory cost of storing the $n \times n$ pairwise similarity matrix. To solve it, most supervised methods sample a subset of the entire dataset for training [24, 25, 28]or sample several columns of the similarity matrix in each iteration [17, 48]. This inevitably causes some information loss and degrades the performance. Although SSDH [29] can reduce the space cost without sampling a subset, the reduced space cost is relevant to the amount of data which can still be very large. Besides, among the supervised methods that leverage the pairwise similarity matrix to learn hash codes, most of them fail to exploit the inherent characteristics in the training data, and only the semantic information is considered in the learning of hash codes.

In this paper, we present a novel supervised hashing method, named Fast Scalable Supervised Hashing, FSSH for short, to discretely and efficiently learn the hash codes. Our main contributions are summarized as follows:

- A new loss function is introduced in FSSH. During optimization, the large pairwise similarity matrix is replaced with a pre-computed intermediate term such that the memory space cost can be reduced remarkably, making it scalable to large datasets. Moreover, FSSH can jointly harness the label information and features of data.
- An iterative algorithm comprising of three steps is proposed to solve the optimization problem. In the algorithm, all variables have a closed-form solution, making it converge quickly. Moreover, instead of learning the hash codes bit by bit, it learns all bits simultaneously. They both contribute to the short training time of FSSH.
- Extensive experiments are conducted over three public benchmark datasets. The results demonstrate the superiority of FSSH over several state-of-the-art supervised hashing methods in both accuracy and scalability.

The rest of this paper is organized as follows. In Section 2, we briefly review the related literature. Section 3 details the proposed method. Section 4 describes the experimental settings and results on three benchmark datasets. Section 5 presents a deep variant of FSSH and comparisons with several end-to-end hashing methods followed by the conclusion and future work in Section 6.

2 RELATED WORK

2.1 Two-Step Hashing

In recent years, many supervised hashing methods have been proposed and they usually divide the learning of hash codes and hash function into two individual steps, the so-called two-step hashing methods. In the first step, they generate the hash codes by optimizing a defined loss function. In the second step, given the learnt hash codes, they learn the hash functions to transform the original features into the compact binary codes. There are plenty of two-step hashing methods [17, 24–26, 28]. Accordingly, we can call the other hashing algorithms that learn the hash codes and hash function simultaneously as one-step hashing methods. As the work [25] has revealed that, 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 modeled as a binary classification problem. Thus, any effective predictive model can be leveraged as the hash function in the second step, such as linear classifiers [17, 38, 44], SVM with RBF kernel [25], boosted decision trees [17, 24], and deep convolutional network [41]. Roughly speaking, the more powerful classifiers one uses as the hash function, the better accuracy one can achieve but the more training time will be consumed.

If two two-step hashing methods adopt the same predictive model as the hash function, which actually often happens, the difference between them is in the loss function and optimization algorithm in the first step. Moreover, the quality of the binary matrix generated in the first step determines the overall quality of the two-step hashing methods. Thus, for a new two-step hashing, the design of loss function and optimization strategy determines its novelty and performance.

In this paper, our proposed method has three variants, i.e., one-step FSSH named FSSH_os and two-step FSSH named FSSH_ts and FSSH_deep.

2.2 Scalable Supervised Discrete Hashing

The work that is most relevant with ours is the Scalable Supervised Discrete Hashing (SSDH) [29], which is designed to reduce the space cost of $n \times n$ pairwise similarity matrix and to discretely solve the binary constraint of hash codes. We highlight three weaknesses of SSDH which are addressed by this work. 1) The reduced space cost of similarity matrix is still large and relevant to the amount of data n. 2) SSDH adopts the bit-wise learning strategy to generate binary codes, making it time-consuming. 3) SSDH only leverages the semantic information to learn the hash codes, making it not robust to noise.

Despite the similar motivations of FSSH and SSDH, our proposed method is much better than SSDH. 1) By embedding the large similarity matrix into an intermediate term whose size is independent of the amount of data n, FSSH is more scalable to large datasets. 2) Instead of learning the hash codes bit by bit, it learns all bits simultaneously making the training time of FSSH robust to the hash code lengths. 3) FSSH can jointly harness the relations in the data and the semantic information.

3 OUR PROPOSED FSSH METHOD

3.1 Notations

Given n instances $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \in \mathbb{R}^{n \times d}$, hashing methods aim to learn the r-bit binary hash codes $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n] \in \{-1, 1\}^{n \times r}$ while preserving the similarities in the original space, where $\mathbf{b}_i$ is the hash code of $\mathbf{x}_i$ and d is the dimension of each instance. Without losing generality, we assume that the labels for all training instances are $\mathbf{L} = [\mathbf{l}_1, \mathbf{l}_2, \cdots, \mathbf{l}_n] \in \{0, 1\}^{n \times c}$, where c is the number of classes and $\mathbf{l}_{ik}$ is the k-th element of $\mathbf{l}_i$ ($\mathbf{l}_{ik} = 1$ if $\mathbf{x}_i$ is in class k and 0 otherwise). In addition, $\mathbf{S} \in \{-1, 1\}^{n \times n}$ denotes

the instance pairwise semantic similarity, where $S_{ij} = 1$ means instance i and instance j are semantically similar, and $S_{ij} = -1$ otherwise. As most existing hashing methods do, we assume the input instances to be zero centered, i.e., $\sum_{i=1}^{n} \mathbf{x}_i = 0$.

As kernelization can better capture the underlying nonlinear structure from the original features, it has become a popular and powerful technique in supervised hashing literature [4, 11, 15, 28, 35]. In this paper, we adopt the RBF kernel mapping function. Specifically, we randomly sample m points as anchors, i.e., $\mathbf{o}_1, \mathbf{o}_2, \cdots, \mathbf{o}_m$. Then data point $\mathbf{x} \in \mathbb{R}^d$ can be mapped into a m-dimensional kernel feature representation, using $\phi(\mathbf{x}) = [exp(\| \mathbf{x} - \mathbf{o}_1 \|^2/\sigma), \cdots, exp(\| \mathbf{x} - \mathbf{o}_m \|^2/\sigma)]$. Here, σ is the kernel width estimated according to the average Euclidean distances between the training samples.

3.2 Motivation

For supervised hashing methods, the desired binary codes should preserve the semantic similarity. Although several different objective functions can be leveraged to achieve this goal, the commonly used one is to leverage the inner product, reflecting the opposite of the Hamming distance of binary codes, to approximate the semantic similarity with square loss. Concretely, it is defined as follows,

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

where $\| \cdot \|_F$ denotes the Frobenius norm.

Eq. (1) shows a discrete optimization problem, hard to get solved. Most existing methods optimize it by relaxing the discrete constraint [25, 28]. Some recent studies have discretely solved the problem in Eq. (1) [17, 24, 29, 51] and the experiments demonstrate they can achieve better accuracy compared to those with relaxation. However, they all adopt the bit-wise learning strategy and learn the hash codes one bit at a time, making their training time not robust to bit length. In other words, they may be time-consuming when generating long hash codes [17, 24, 51]. Moreover, they adopt a pair-wise semantic similarity matrix $\mathbf{S}$ and both the computation complexity and memory space complexity of $\mathbf{S}$ are $O(n^2)$ if all the training data is used for learning. Therefore, most existing methods only sample a small subset with m ($m \leq n$) points for training. These methods cannot achieve satisfactory accuracy since sampling may cause information loss. Some methods, like LFH [48] and COSDISH [17], adopt a strategy called column sampling to sample several columns from $\mathbf{S}$ in each iteration and several iterations are performed for training. They can leverage all the supervised information and experimental results have demonstrated their superiority. However, as shown in [48], there is still some performance degradation. SSDH [29] tries to embed the large similarity matrix into a matrix with the size of $n \times c$ and the space cost of $\mathbf{S}$ can be reduced. However, $n \times c$ is still space-consuming when n is large. Beyond that, LFH, COSDISH, and SSDH generate the hash codes without considering the specific data (only with semantic information), which may be not robust to noise.

To address the above issues, we need to solve the problem in Eq. (1) from two aspects. First, we need to effectively exploit all the n available points for training without any sampling. Second, we need to design strategies for discrete optimization and learn the hash codes in a faster way rather than by the bit-wise learning. Third, we need to take both semantic information and the specific data into consideration. These motivate our work in this paper.

3.3 Objective Function

Kernel Features Mapping. The hash functions, mapping data from the original features into the binary codes, can be formulated as $F(\mathbf{X}) = sgn(\phi(\mathbf{X})\mathbf{W}) = \mathbf{B}$. Here $\mathbf{W} \in \mathbb{R}^{m \times r}$ is a projection matrix transforming the data onto a r-bit real-valued space, and $sgn(\cdot)$ is an element-wise sign function defined as $sgn(x) = 1$ if $x \geq 0$, and -1 otherwise. Thereafter, we can ensure the quality of kernel features mapping by defining the square loss as follows,

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

In addition, replacing the binary matrix $\mathbf{B}$ in problem (1) with real-valued information could permit more accurate approximation of similarity [4, 10]. Therefore, we further leverage the real-valued kernel features mapping $\phi(\mathbf{X})\mathbf{W}$ to better embed the similarity by defining the following problem,

$$\begin{aligned} &\min_{\mathbf{B},\mathbf{W}} \| r\mathbf{S} - \left(\phi(\mathbf{X})\mathbf{W}\right)\mathbf{B}^\top \|_F^2, \\ &s.t. \quad \mathbf{B} \in \{-1, 1\}^{n \times r}. \end{aligned} \tag{3}$$

Putting Eq. (2) and Eq. (3) together, we have:

$$\begin{aligned} &\min_{\mathbf{B},\mathbf{W}} \| r\mathbf{S} - \left(\phi(\mathbf{X})\mathbf{W}\right)\mathbf{B}^\top \|_F^2 + \mu \| \mathbf{B} - \phi(\mathbf{X})\mathbf{W} \|_F^2 \\ &s.t. \quad \mathbf{B} \in \{-1, 1\}^{n \times r}, \end{aligned} \tag{4}$$

where μ is a balance parameter.

Label Matrix Embedding. We argue that although the pairwise similarity matrix $\mathbf{S}$ can be constructed from the label matrix, it is still informative. For example, suppose there are three instances with labels $\mathbf{l}_1 = [1, 0, 0]$, $\mathbf{l}_2 = [1, 1, 1]$ and $\mathbf{l}_3 = [1, 1, 1]$, and then the values of S_{13} and S_{23} are both 1. However, the fact that the instance 2 is more similar to 3 than the first instance, cannot be found from the pairwise similarity matrix. We thus further embed the label matrix into the learning of binary codes by assuming that the labels can also be mapped into the binary codes, i.e.,

$$\begin{aligned} &\min_{\mathbf{B},\mathbf{G}} \| \mathbf{B} - \mathbf{L}\mathbf{G} \|_F^2, \\ &s.t. \quad \mathbf{B} \in \{-1, 1\}^{n \times r}, \end{aligned} \tag{5}$$

where $\mathbf{G} \in \mathbb{R}^{c \times r}$ is a projection from $\mathbf{L}$ to $\mathbf{B}$.

Overall Objective Function. Combining Eq. (4) and Eq. (5), and replacing the other $\mathbf{B}$ with the real-valued label matrix embedding, we have the following final objective function of our method,

$$\begin{aligned} &\min_{\mathbf{B},\mathbf{G},\mathbf{W}} \| \mathbf{S} - \left(\phi(\mathbf{X})\mathbf{W}\right)(\mathbf{L}\mathbf{G})^\top \|_F^2 \\ &+ \mu \| \mathbf{B} - \mathbf{L}\mathbf{G} \|_F^2 + \theta \| \mathbf{B} - (\phi(\mathbf{X})\mathbf{W}) \|_F^2, \\ &s.t. \quad \mathbf{B} \in \{-1, 1\}^{n \times r}, \end{aligned} \tag{6}$$

where μ and θ are balance parameters.

It is obvious that both pairwise similarity matrix $\mathbf{S}$ and label matrix $\mathbf{L}$ are embedded into the learning of binary codes, which makes the learnt $\mathbf{B}$ more likely to preserve the semantic similarity. It is worth noting that, although $\mathbf{S}$ is included in the final objective

function, it does not mean that $\mathbf{S}$ will be directly leveraged. As presented in the following section, by introducing an intermediate term pre-computed from $\mathbf{S}$ and some other terms, with much smaller space cost, we can circumvent the direct use of $\mathbf{S}$ during the optimization.

3.4 Optimization Algorithm

To solve the problem in (6), we present an iterative optimization algorithm, in which each iteration contains three steps, i.e., $\mathbf{W}$ Step, $\mathbf{G}$ Step and $\mathbf{B}$ Step. More specifically, in the first step, $\mathbf{W}$ is updated by fixing $\mathbf{G}$ and $\mathbf{B}$; second, $\mathbf{G}$ is optimized with $\mathbf{W}$ and $\mathbf{B}$ fixed; finally, $\mathbf{B}$ is obtained by fixing $\mathbf{W}$ and $\mathbf{G}$. Each step is detailed in the following paragraphs.

W Step. When $\mathbf{G}$ and $\mathbf{B}$ are fixed, problem (6) is then simplified as,

$$\min_{\mathbf{W}} \parallel \mathbf{S} - \Big(\phi(\mathbf{X})\mathbf{W}\Big)(\mathbf{LG})^\top \parallel_F^2 + \theta \parallel \mathbf{B} - \phi(\mathbf{X})\mathbf{W} \parallel_F^2 . \tag{7}$$

We can get a closed-form solution to the above problem by setting the derivative of (7) regarding $\mathbf{W}$ to zero,

$$\begin{aligned} \mathbf{W} &= \Big(\phi(\mathbf{X})^\top \phi(\mathbf{X})\Big)^{-1} \\ &\Big(\phi(\mathbf{X})^\top \mathbf{SLG} + \theta\phi(\mathbf{X})^\top \mathbf{B}\Big)(\mathbf{G}^\top \mathbf{L}^\top \mathbf{LG} + \theta \mathbf{I}_{r\times r})^{-1}, \\ &= \mathbf{C}^{-1}\Big(\mathbf{AG} + \theta\phi(\mathbf{X})^\top \mathbf{B}\Big)(\mathbf{G}^\top \mathbf{DG} + \theta \mathbf{I}_{r\times r})^{-1}, \end{aligned} \tag{8}$$

where $\mathbf{A} = \phi(\mathbf{X})^\top \mathbf{SL}$, $\mathbf{C} = \phi(\mathbf{X})^\top \phi(\mathbf{X})$ and $\mathbf{D} = \mathbf{L}^\top \mathbf{L}$. Note that both $\mathbf{C}^{-1}$ and $\mathbf{D}$ can be computed only once before the optimization.

It is worth noting that, we introduce **an intermediate term** $\mathbf{A} \in \mathbb{R}^{m\times c}$ into the solution. By leveraging this term, we can circumvent the direct optimization of the large pairwise matrix $\mathbf{S}$. Apparently, $\mathbf{A}$ is a constant and can be efficiently pre-computed before training. By leveraging the pre-computed term $\mathbf{A}$ in the optimization, $O(n^2)$ memory space cost of $\mathbf{S}$ can be dramatically reduced to $O(mc)$ and the training can become more efficient.

G Step. When $\mathbf{W}$ and $\mathbf{B}$ are fixed, we can rewrite Eq. (6) as,

$$\min_{\mathbf{G}} \parallel \mathbf{S} - \Big(\phi(\mathbf{X})\mathbf{W}\Big)(\mathbf{LG})^\top \parallel_F^2 + \mu \parallel \mathbf{B} - \mathbf{LG} \parallel_F^2 . \tag{9}$$

Similarly, setting the derivative of Eq. (9) w.r.t. G to zero, we reach the closed-form solution,

$$\begin{aligned} \mathbf{G} &= (\mathbf{L}^\top \mathbf{L})^{-1} \\ &\Big(\mu \mathbf{L}^\top \mathbf{B} + \mathbf{L}^\top \mathbf{S}^\top \phi(\mathbf{X})\mathbf{W}\Big)\Big(\mathbf{W}^\top \phi(\mathbf{X})^\top \phi(\mathbf{X})\mathbf{W} + \mu \mathbf{I}_{r\times r}\Big)^{-1}, \\ &= \mathbf{D}^{-1}(\mu \mathbf{L}^\top \mathbf{B} + \mathbf{A}^\top \mathbf{W})(\mathbf{W}^\top \mathbf{CW} + \mu \mathbf{I}_{r\times r})^{-1}, \end{aligned} \tag{10}$$

where $\mathbf{A} = \phi(\mathbf{X})^\top \mathbf{SL}$, $\mathbf{C} = \phi(\mathbf{X})^\top \phi(\mathbf{X})$ and $\mathbf{D} = \mathbf{L}^\top \mathbf{L}$. As mentioned previously, $\mathbf{C}$ and $\mathbf{D}^{-1}$ can be computed only once before the training. Therefore, the computation of this solution is also efficient.

B Step. When $\mathbf{W}$ and $\mathbf{G}$ are fixed, the optimization problem is formulated as follows,

$$\begin{aligned} &\min_{\mathbf{B}} \ \mu \parallel \mathbf{B} - \mathbf{LG} \parallel_F^2 + \theta \parallel \mathbf{B} - \phi(\mathbf{X})\mathbf{W} \parallel_F^2, \\ &s.t. \quad \mathbf{B} \in \{-1, 1\}^{n\times r}. \end{aligned} \tag{11}$$

Then, we rewrite Eq. (11) as follows,

$$\begin{aligned} &\min_{\mathbf{B}} \ \mu Tr\Big((\mathbf{B} - \mathbf{LG})^\top (\mathbf{B} - \mathbf{LG})\Big) \\ &\qquad + \theta Tr\Big(\Big(\mathbf{B} - \phi(\mathbf{X})\mathbf{W}\Big)^\top \Big(\mathbf{B} - \phi(\mathbf{X})\mathbf{W}\Big)\Big), \\ &= (\mu + \theta) \parallel \mathbf{B} \parallel_F^2 - 2\mu Tr(\mathbf{B}^\top \mathbf{LG}) \\ &+ \mu \parallel \mathbf{LG} \parallel_F^2 - 2\theta Tr\Big(\mathbf{B}^\top \phi(\mathbf{X})\mathbf{W}\Big) + \theta \parallel \phi(\mathbf{X})\mathbf{W} \parallel_F^2, \\ &s.t. \quad \mathbf{B} \in \{-1, 1\}^{n\times r}, \end{aligned} \tag{12}$$

where $Tr(\cdot)$ is the trace norm. Since $\parallel \mathbf{B} \parallel_F^2$, $\parallel \mathbf{LG} \parallel_F^2$ and $\parallel \phi(\mathbf{X})\mathbf{W} \parallel_F^2$ are constants, Eq. (12) is equivalent to the following problem,

$$\begin{aligned} &\min_{\mathbf{B}} -Tr\Big(\mathbf{B}^\top \Big(\mu \mathbf{LG} + \theta\phi(\mathbf{X})\mathbf{W}\Big)\Big), \\ &s.t. \quad \mathbf{B} \in \{-1, 1\}^{n\times r}. \end{aligned} \tag{13}$$

Thus, $\mathbf{B}$ can also be solved with a closed-form solution stated as follows,

$$\mathbf{B} = sgn\Big(\mu \mathbf{LG} + \theta\phi(\mathbf{X})\mathbf{W}\Big), \tag{14}$$

where $sgn(\cdot)$ is an element-wise sign function defined as $sgn(x) = 1$ if $x \geq 0$, and -1 otherwise. It is easy to see from Eq. (14) that, only one single step is needed to simultaneously learn all bits of binary codes in each iteration. Thus, compared to those methods learning the hash codes by a bit-wise strategy, our method can learn the binary codes much faster.

The Whole Algorithm. By repeating the above three steps until it is convergent, we can obtain the final solution. It is worth noting that the closed-form solutions can offer two main advantages. 1) Precise hash codes can be learnt. 2) The whole algorithms may converge quickly. Besides, all bits of $\mathbf{B}$ can be efficiently computed by only a single step, making it much faster compared to the bit-wise learning strategy especially for long bit length. The above optimization procedure is summarized in Algorithm 1.

3.5 Out-of-Sample Extension

Note that we can design different variants of FSSH, i.e., one-step FSSH named FSSH_os and two-step FSSH named FSSH_ts. The difference between them lies in the design of the hash functions of FSSH. Suppose $\mathbf{X}_{query}$ and $\mathbf{B}_{query}$ are the original features and corresponding hash codes of the queries, respectively. Then, they are described as follows.

FSSH_os. One-step FSSH uses the learnt projection matrix $\mathbf{W}$ as the hash projection. It means that FSSH_os can simultaneously learn its hash functions and hash codes so that Algorithm 1 can return $\mathbf{W}$ and $\mathbf{B}$ at the same time.

Thereafter, for new queries, the binary codes $\mathbf{B}_{query}$ can be obtained by,

$$\mathbf{B}_{query} = sgn(\phi(\mathbf{X}_{query})\mathbf{W}). \tag{15}$$

FSSH_ts. Two-step FSSH does not leverage $\mathbf{W}$ but follows the two-step hashing strategy to learn its hash functions. Accordingly, it trains r binary classifiers based on the feature matrix $\phi(\mathbf{X})$ and the learnt hash codes matrix $\mathbf{B}$ with each bit corresponding to one classifier. As mentioned previously, any arbitrary classifier can be adopted as the hash function. For FSSH_ts, we use the linear regression on account of its efficiency. Then, the projection matrix

Algorithm 1 Optimization algorithm of FSSH.

Input: The label matrix $\mathbf{Y} \in \{0,1\}^{n \times c}$, the intermediate term $\mathbf{A} = \phi(\mathbf{X})^\top \mathbf{SL}$ computed offline, the balance parameters μ and θ, iteration number t and the hash code length r.

1: Randomly initialize $\mathbf{W}$, $\mathbf{G}$ and $\mathbf{B}$.
2: **for** $iter = 1 \rightarrow t$ **do**
3: **W step:** Fix $\mathbf{G}$ and $\mathbf{B}$, update $\mathbf{W}$ using Eq. (8).
4: **G step:** Fix $\mathbf{W}$ and $\mathbf{B}$, update $\mathbf{G}$ using Eq. (10).
5: **B step:** Fix $\mathbf{W}$ and $\mathbf{G}$, update $\mathbf{B}$ using Eq. (13).
6: **end for**

Output: $\mathbf{W}$ and $\mathbf{B}$.

$\mathbf{P}$, which transforms $\phi(\mathbf{X})$ to $\mathbf{B}$, can be obtained by optimizing the following squared loss with a regularization term,

$$\| \mathbf{B} - \phi(\mathbf{X})\mathbf{P} \|_F^2 + \lambda_e \| \mathbf{P} \|_F^2, \tag{16}$$

where λ_e is a balance parameter, and $\| \mathbf{P} \|_F^2$ is the regularization term. And the optimal $\mathbf{P}$ can be computed as,

$$\mathbf{P} = \left(\phi(\mathbf{X})^\top \phi(\mathbf{X}) + \lambda_e \mathbf{I}\right)^{-1} \phi(\mathbf{X})^\top \mathbf{B}. \tag{17}$$

For new queries, FSSH_ts generates the hash codes with the following equation,

$$\mathbf{B}_{query} = sgn(\phi(\mathbf{X}_{query})\mathbf{P}). \tag{18}$$

4 EXPERIMENTS

We conducted extensive experiments on three benchmark datasets to evaluate our method in terms of both search accuracy and efficiency. We also compared it with several state-of-the-art shallow supervised hashing methods.

4.1 Experimental Settings

4.1.1 Datasets. The three benchmark datasets are CIFAR-10 [18], MNIST [19] and NUS-WIDE [3]. They all comprise of a large number of images with semantic labels, and have been widely used in hashing literature [17, 25, 35].

CIFAR-10 is collected from 80 million tiny images. It has a number of 60,000 manually labeled images and 10 classes with 6,000 images for each class and each image is represented by a 512-dimension GIST [33] feature vector extracted from the original color image with the size of 32×32. Two images are considered to be semantically similar if they share the same class label; otherwise, they are semantically dissimilar. For CIFAR-10, we randomly picked up 1,000 images as queries and the remained 59,000 instances as the training set.

MNIST is a subset of a larger set available from NIST. It consists of 70,000 images with handwritten digits from '0' to '9'. It thus has 10 classes corresponding to the numbers ranging from '0' to '9'. Each image is represented by a 784-dimension feature vector. If two images are in the same class, they are viewed to be semantically similar, and vice versa. For this dataset, the whole dataset is split into a query set with 1,000 images and a training set with all the remaining images.

NUS-WIDE is a real-world image dataset collected from Flickr. It originally contains 269,648 samples with 81 ground-truth labels (tags). Following the setting in [35], we collected the 21 most frequent labels for evaluation and 195,834 images are left. Each image is represented by a 500-dimensional bag-of-words feature vector. For multi-label dataset NUS-WIDE, we considered two images to be semantically similar if they share at least one common label; otherwise, they are semantically dissimilar. On this dataset, for each label, 100 images are randomly sampled (2,100 images with duplication) as the query set.

4.1.2 Baselines and Implementation Details. The superiority of supervised hashing methods over unsupervised ones has been demonstrated by some literature [4, 17, 25]. Thus, we only compared our method with several state-of-the-art representative supervised hashing ones, including KSH [28], SDH [35], FSDH [11], TSH [25], LFH [48] and COSDISH [17]. Among them, KSH, SDH and FSDH are one-step hashing methods while TSH, LFH and COSDISH are two-step. Since FSSH_os and FSSH_ts are shallow models, we did not compare them with deep hashing models for fairness.

The codes of most baselines are kindly provided by the authors. As the codes of FSDH are not publicly available, we implemented it exactly following its algorithm. For all baselines, we carefully tuned their parameters according to the scheme suggested by the authors. Due to the high time complexity, KSH and TSH cannot use the entire training set for training. Following the setting of [17, 25, 28], we randomly sampled 2,000 instances for training. The parameters of FSSH are selected by a validation procedure. Specifically, for FSSH_os, $\mu = 10000$ and $\theta = 100$; for FSSH_ts, $\mu = 10000$ and $\theta = 0.01$. We set $\lambda_e = 1$ and sample $m = 1,000$ anchors. All experiments are repeated several times with random data partitions, and the averaged results are reported. All the experiments are conducted on a workstation with Intel(R) XEON(R) CPU E5-2650@2.30GHz, 128GB memory.

4.1.3 Evaluation Metrics. The widely used criteria, i.e., mean average precision (MAP), top-N precision curves and precision-recall curves, are adopted to evaluate the retrieval performance of various models. For all the three evaluation metrics, a larger value indicates the better retrieval performance.

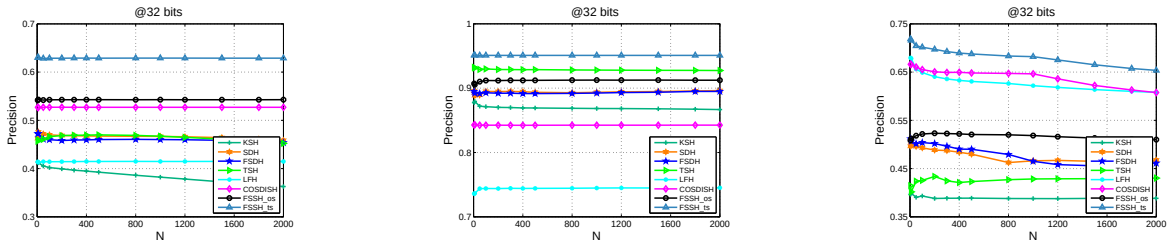
4.2 Accuracy

4.2.1 MAP. The MAP results of various methods on CIFAR-10, MNIST and NUS-WIDE are summarized in Table 1. To make further comparison, we divided the eight methods in Table 1 into two categories. The first contains all one-step hashing, i.e., KSH, SDH, FSDH and FSSH_os; the second contains the remaining two-step hashing methods including TSH, LFH, COSDISH and FSSH_ts. Then, we had the following observations:

- We found that FSSH_os outperforms all one-step baselines. More specifically, compared with the highest MAP values of all baselines, FSSH_os obtains 10.5%, 47.6%, 63.0% and 63.5% performance gains in the cases of 16-bit, 32-bit, 64-bit and 96-bit codes on CIFAR-10, respectively. On MNIST, performance gains offered by FSSH_os are 3.53%, 6.84%, 4.49% and 3.78%, respectively. On NUS-WIDE, they are 17.0%, 27.7%, 41.9% and 43.2% with code lengths varying from 16 to 96.

Table 1: The MAP results of various methods on the three datasets. The best MAP values of each case are shown in boldface.

Method	CIFAR-10				MNIST				NUS-WIDE			
	16 bits	32 bits	64 bits	96 bits	16 bits	32 bits	64 bits	96 bits	16 bits	32 bits	64 bits	96 bits
KSH	0.2626	0.2897	0.3108	0.3185	0.8017	0.8233	0.8390	0.8400	0.3703	0.3728	0.3785	0.3786
SDH	0.3525	0.3788	0.3986	0.4135	0.8718	0.8873	0.8958	0.8972	0.4113	0.4114	0.4135	0.4211
FSDH	0.3284	0.3661	0.3986	0.4110	0.8562	0.8804	0.8907	0.8962	0.4052	0.4115	0.4155	0.4166
TSH	0.3202	0.3551	0.3711	0.3843	0.9007	0.9215	0.9281	0.9323	0.4424	0.4479	0.4555	0.4545
LFH	0.3803	0.5061	0.6133	0.6288	0.5564	0.7577	0.8593	0.8575	0.5767	0.5974	0.6042	0.6124
COSDISH	0.5737	0.6109	0.6310	0.6368	0.8551	0.8754	0.8818	0.8888	0.5719	0.5913	0.5916	0.6027
FSSH_os	0.3896	0.5592	0.6497	0.6760	0.9023	0.9480	0.9360	0.9311	0.4813	0.5255	0.5867	0.6029
FSSH_ts	**0.6350**	**0.6829**	**0.7071**	**0.7108**	**0.9443**	**0.9649**	**0.9713**	**0.9721**	**0.5846**	**0.6060**	**0.6105**	**0.6225**



(a) CIFAR-10 @32 bits (c) MNIST @32 bits (e) NUS-WIDE @32 bits

(b) CIFAR-10 @96 bits (d) MNIST @96 bits (f) NUS-WIDE @96 bits

Figure 1: The top-N precision curves of various methods in the cases of 32-bit and 96-bit code length on the three datasets.

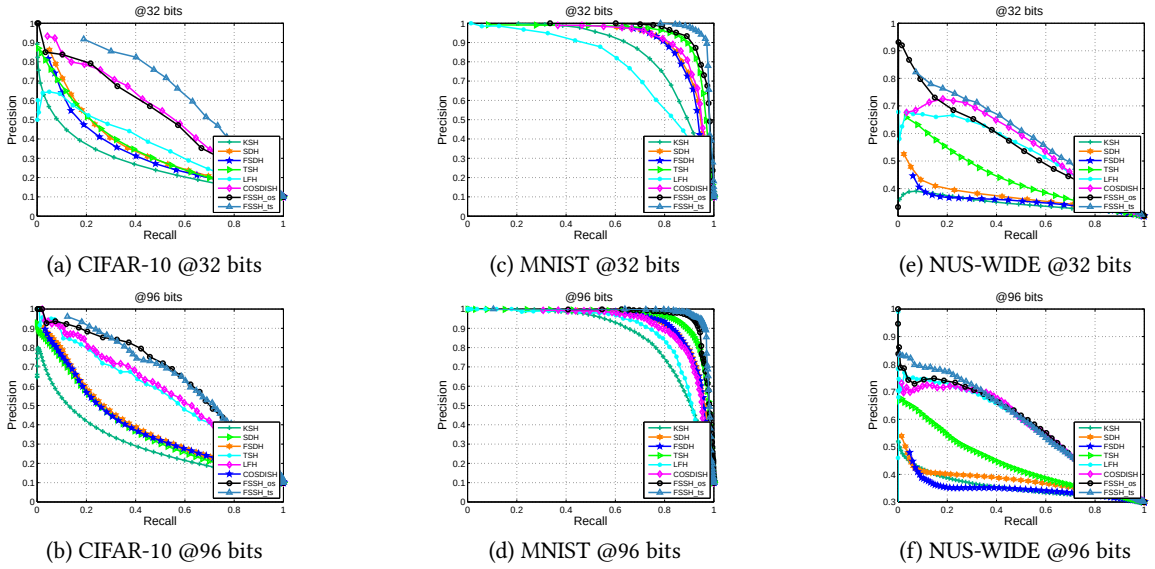
- Comparing FSSH_ts with other two-step baselines, i.e., TSH, LFH and COSDISH, we can see that FSSH_ts achieves much better results in all cases on the three datasets.
- FSSH_ts obtains the best results in all cases. Especially, FSSH_ts always outperforms FSSH_os, demonstrating the superiority of two-step hashing.

One possible reason for the significant performance gain of FSSH_os over SDH and FSDH is that, FSSH_os leverages both the pairwise similarity matrix and label matrix while SDH and FSDH only embed the label matrix in their loss functions. Meanwhile, LFH and COSDISH using the pairwise similarity matrix as supervision can outperform SDH and FSDH as well. These results, to some extent, demonstrate that the pairwise similarity can better supervise the binary codes than the label matrix does. There are two reasons that FSSH_ts outperforms LFH and COSDISH. The first one would be that both LFH and COSDISH adopt the column sampling strategy to solve the huge $n \times n$ space cost problem of the pairwise similarity matrix and sampling may lead to information loss and the decrease of accuracy. The other one is that FSSH_ts leverages not only the pairwise similarity matrix but also the label matrix to supervise the learning, and more supervision information ensures more precise hash codes.

In summary, our FSSH_os and FSSH_ts can achieve the state-of-the-art accuracy, which demonstrates that the loss function and optimization algorithm in this work are effective.

4.2.2 Top-N Precision Curves. Figure 1 illustrates the top-N precision curves in the cases of 32-bit and 96-bit length on the three datasets. From this figure, we had the following observations:

- On CIFAR-10, FSSH_os obtains better precision than that of other one-step baselines, and meanwhile, FSSH_ts also outperforms other two-step baselines. Especially for FSSH_ts, its top-N precision curve is much higher compared with the others.
- On MNIST, compared with KSH, SDH and FSDH, FSSH_os is always the best. Compared with TSH, LFH and COSDISH, FSSH_ts is the best among the two-step hashing methods.
- On NUS-WIDE, we had similar observations, i.e., FSSH_os surpasses the other one-step hashing methods and FSSH_ts outperforms the other two-step ones.



(a) CIFAR-10 @32 bits (c) MNIST @32 bits (e) NUS-WIDE @32 bits

(b) CIFAR-10 @96 bits (d) MNIST @96 bits (f) NUS-WIDE @96 bits

Figure 2: The precision-recall curves of various methods in the cases of 32-bit and 96-bit code length on the three datasets.

- On all datasets, FSSH_ts performs the best in all cases.

To summarize, both FSSH_os and FSSH_ts outperform the state-of-the-art one-step and two-step shallow supervised methods, respectively, in terms of the top-N precision curves evaluation criterion. It further verifies the effectiveness of FSSH.

4.2.3 Precision-recall Curves. The precision-recall curves in the cases of 32-bit and 96-bit code length are plotted in Figure 2. From this figure, we can observe that:

- On both CIFAR-10 and MNIST, FSSH_os outperforms the other one-step baselines, i.e., KSH, SDH and FSDH. FSSH_ts outperforms the other two-step baselines, i.e., TSH, LFH and COSDISH.
- All methods perform well on MNIST. The reason is that MNIST is an 'easy' dataset, not as challenging as CIFAR-10 and NUS-WIDE.
- On NUS-WIDE, we had similar observations, i.e., FSSH_os outperforms the other one-step hashing baselines. In the case of 96 bits, curve crossover occurs; however, FSSH_os's curve is mostly above that of COSDISH.
- On all datasets, FSSH_ts obtains the best or competitive results.

In a nutshell, from the results on the three benchmark datasets, we can conclude that both FSSH_os and FSSH_ts are effective.

4.3 Scalability

To verify the efficiency of our approach, we reported the training and testing time of all methods on CIFAR-10 and MNIST in Table 2. From this table, we observed the following points:

- Even though only 2,000 samples are used in training, KSH and TSH are still time-consuming because they directly use the $n \times n$ similarity matrix $\mathbf{S}$ during the optimization.
- Although SDH and COSDISH are much faster than KSH and TSH, their training time is much longer than that of FSSH_os and FSSH_ts.
- All baselines except FSDH learn the hash codes bit by bit; therefore, there is a significant increment in the cases of longer code length. FSDH is a method based on SDH; but it is much faster than SDH because it learns all bits simultaneously.
- Both FSSH_os and FSSH_ts are faster than FSDH.
- Comparing FSSH_os and FSSH_ts, FSSH_ts needs slightly longer time because it spends extra time to learn its hash functions.

To further verify the scalability of our approach, we reported the training time on the larger dataset, i.e., NUS-WIDE, with 193K training samples. In this experiment, KSH and TSH are not used for comparison because they can only work on a subset of this dataset. The training time of SDH, FSDH, LFH, COSDISH, FSSH_os and FSSH_ts is summarized in Table 3. Clearly, both FSSH_os and FSSH_ts are faster than the compared baselines. Especially, in the case of 16 bits, FSSH_os only needs 9.78 seconds to finish the training of 193,833 images. Moreover, there is no significant increment for neither FSSH_os nor FSSH_ts when the code length becomes longer.

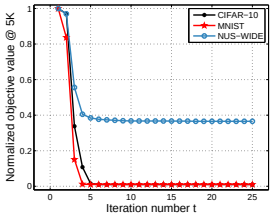
From both Table 2 and 3, we can conclude that FSSH_os and FSSH_ts are scalabe to large-scale datasets and long code length.

4.4 Convergence

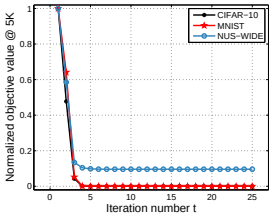
To demonstrate that the proposed alternative optimization method can converge, we further plotted the convergence curves of the objective function in the case of 32-bit code length on the three datasets in Figure 3. For convenience, the objective values are normalized by dividing the maximum on each dataset.

Table 2: The training and testing time of various methods on CIFAR-10 and MNIST.

Method	CIFAR-10						MNIST					
	Training Time (second)			Test Time (millisecond)			Training Time (second)			Test Time (millisecond)		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
KSH	111.31	235.29	359.24	2.57	3.91	5.33	169.80	302.47	480.94	5.20	5.69	7.75
SDH	8.96	25.16	83.80	2.62	3.74	5.35	15.74	71.82	141.27	3.90	6.47	7.21
FSDH	6.25	6.62	6.87	2.60	3.80	5.78	9.01	9.20	9.88	4.08	5.82	7.03
TSH	105.37	206.98	340.37	2.56	4.02	6.06	173.20	343.73	483.02	4.33	5.41	7.68
LFH	4.06	7.80	13.79	2.28	3.57	5.74	7.44	12.19	18.41	2.88	4.33	5.93
COSDISH	15.99	78.99	189.59	2.45	3.89	5.54	20.88	104.08	224.01	2.75	4.19	5.65
FSSH_os	2.98	3.40	3.88	2.32	3.62	5.68	5.06	5.20	5.23	4.19	5.80	7.06
FSSH_ts	4.10	4.85	5.19	2.52	3.89	5.70	5.55	5.78	6.72	2.81	4.23	6.56

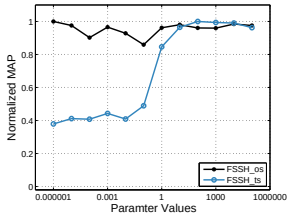


(a) FSSH_os

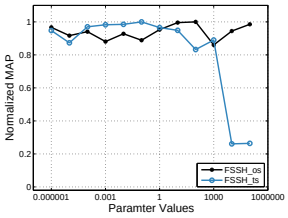


(b) FSSH_ts

Figure 3: The convergence curves of FSSH_os and FSSH_ts in the case of 32-bit code length on the three datasets.



(a) μ



(b) θ

Figure 4: The parameter sensitivity analysis of FSSH_os and FSSH_ts in terms of μ and θ in the case of 32-bit code length on the three datasets.

Table 3: The training time cost in terms of seconds of various methods on NUS-WIDE.

Method	NUS-WIDE (193K)			
	16 bits	32 bits	64 bits	96 bits
SDH	25.13	77.28	262.03	680.86
FSDH	17.91	19.14	20.42	26.90
LFH	16.45	19.96	28.17	37.35
COSDISH	14.06	54.22	227.01	512.14
FSSH_os	9.78	10.35	11.98	14.61
FSSH_ts	13.71	14.09	15.51	15.53

From this figure, we can see that both FSSH_os and FSSH_ts converge within a few iterations, e.g., less than 5, demonstrating the convergence of the optimization scheme. The quick convergence is a result of the well-designed loss function and optimization algorithm as all variables have closed-form solutions.

4.5 Parameter Sensitivity Analysis

To analyze the influence of parameters μ and θ on the performance, we conducted experiments on CIFAR-10. The MAP curves in the case of 32-bit code length are plotted in Figure 4. For convenience, we normalized the MAP by dividing the maximums of FSSH_os and FSSH_ts. Parameters μ and θ vary in the range of $[10^{-6}, 10^{5}]$.

From this figure, we can observe that:

- Both μ and θ impact the performance of FSSH_os and FSSH_ts.
- FSSH_os performs well when μ is in the range of $[10^{1}, 10^{4}]$ and θ is from 10 to 100, respectively.

Table 4: Experimental results of SSDH and FSSH on the three datasets with respect to MAP.

Method	CIFAR-10			
	16 bits	32 bits	64 bits	96 bits
SSDH	**0.6698**	**0.6991**	0.7046	0.7063
FSSH_ts	0.6350	0.6829	**0.7071**	**0.7108**
Method	MNIST			
	16 bits	32 bits	64 bits	96 bits
SSDH	0.9557	0.9639	0.9669	0.9683
FSSH_ts	**0.9443**	**0.9649**	**0.9713**	**0.9721**
Method	NUS-WIDE			
	16 bits	32 bits	64 bits	96 bits
SSDH	**0.6023**	0.6035	0.6069	0.6101
FSSH_ts	0.5846	**0.6060**	**0.6105**	**0.6225**

Table 5: The training time cost in terms of seconds of SSDH and FSSH on NUS-WIDE.

Method	NUS-WIDE (193K)			
	16 bits	32 bits	64 bits	96 bits
SSDH	14.47	25.60	61.54	129.84
FSSH_ts	13.71	14.09	15.51	15.53

- FSSH_ts performs well when μ is in the range of $[10^{1}, 10^{4}]$ and θ is $[10^{-4}, 10^{0}]$, respectively.

Table 6: The MAP results of various methods on CIFAR-10.

Method	CIFAR-10		
	24 bits	32 bits	48 bits
DSRH	0.611	0.617	618
DSCH	0.613	0.617	0.620
DRSCH	0.622	0.629	0.631
DPSH	0.781	0.795	0.807
VDSH	0.848	0.844	0.845
DTSH	0.923	0.925	0.926
DSDH	**0.940**	**0.939**	**0.939**
FSSH_deep	0.878	0.862	0.883

4.6 Further Comparisons with SSDH

As the motivations of SSDH and our FSSH are similar, we further compared these two methods in this section.

FSSH has several obvious advantages over SSDH. First, SSDH adopts the bit-wise learning strategy which needs lots of steps to solve all bits of hash codes while FSSH can optimize all bits in one step. Second, SSDH reduces the $n \times n$ space cost to $n \times c$ which is still large and relevant to the amount of data n while FSSH can embed $n \times n$ similarity matrix $\mathbf{S}$ into an intermediate term with a size of $m \times c$. Here, m is the number of kernel anchor points and is set to 1, 000, far less than the number of training instances n. Third, as stated above, SSDH generates the hash codes without considering the specific data (only with semantic information) while FSSH leverages both the relations of data and the label information.

Table 4 summarizes the MAP results on the three datasets and Table 5 shows the training time on large-scale dataset NUS-WIDE. As SSDH is a two-step hashing model, we only reported the two-step variant of FSSH for fair comparison. From the experimental results of SSDH and FSSH, we have the following observations::

- FSSH can obtain better MAP results in most cases. In other cases, the performance of FSSH is competitive with that of SSDH while FSSH needs less space cost.
- FSSH can be trained faster than SSDH on NUS-WIDE. While SSDH consumes longer time when the code length becomes longer, the time cost of our FSSH is robust to the hash code length.

To conclude, our FSSH is more practical than SSDH, especially in the cases with longer code lengths on large-scale datasets.

5 DEEP VARIANT OF FSSH

Recently, deep neural networks have been widely used in various fields and shown encouraging results [2, 13, 14, 20, 21]. Thus, in this section, we propose another variant of FSSH and name it as FSSH_deep. This variant follows the two-step hashing strategy and utilizes a neural network as the hash function in the second step.

Note that, all variants of FSSH including FSSH_deep are not end-to-end models. The reason is that we want to concentrate on the criterion of hash codes, i.e., the design of loss functions and the optimization algorithm; and we believe that our FSSH model can be easily transformed into an end-to-end method. We leave the idea of realizing FSSH in an end-to-end deep version for the future pursuit.

5.1 FSSH_deep

As stated above, for two-step hashing methods, any effective predictive model can be leveraged as the hash function in the second step, including deep neural networks. Here, we adopts CNN-F [1] as the hash function. Although many other neural network architectures can be used to substitute the CNN-F network, it is not the focus of this paper to study different networks and we just use CNN-F for illustrating the effectiveness of FSSH.

We train the network by solving a multi-label classification problem and optimize the following loss function,

$$L = -\frac{1}{n}\sum_{i=1}^{n}[\mathbf{b}_i \log \mathbf{p}_i + (1-\mathbf{b}_i)\log(1-\mathbf{p}_i)], \tag{19}$$

where $\mathbf{b}_i$ is the hash code of the i-th training instance[1], $\mathbf{p}_i = \sigma(\mathbf{z}_i)$, $\sigma(\cdot)$ is a sigmoid function, and $\mathbf{z}_i$ is the output of the network. We can learn the parameters θ of the network through the Back-Propagation (BP) algorithm with stochastic gradient descent (SGD). Specifically, the derivative of the loss function with respect to $\mathbf{z}_i$ is $\frac{\partial L}{\partial \mathbf{z}_i} = \frac{1}{n}(\mathbf{p}_i - \mathbf{b}_i)$. And $\frac{\partial L}{\partial \theta}$ with $\frac{\partial L}{\partial \mathbf{z}_i}$ can be obtained with the chain rule and the parameter θ can be updated. After the training, the hash codes of out-of-sample queries can be generated by,

$$\mathbf{B}_{query} = sgn(\mathbf{Z}_{query}), \tag{20}$$

where $\mathbf{Z}_{query}$ is the output of the network for queries.

Besides, for the first step of FSSH_deep, $\mathbf{X}$ are deep features, which are extracted by a pre-trained CNN-F network.

5.2 Comparisons with Deep Hashing Methods

Following the previous experimental setting [22, 23, 39], for CIFAR-10, we selected 1,000 images per class (10,000 images in total) as the queries and left the remaining 50,000 images as the training set. As for the comparison methods, several state-of-the-art deep hashing methods are adopted, including DSRH [53], DSCH [50], DRSCH [50], DPSH [23], VDSH [52], DTSH [39], and DSDH [22].In order to have a fair comparison, the results of baselines are directly reported from previous works.

Table 6 lists the MAP results on CIFAR-10. From this table, we can find that some baselines outperform FSSH_deep. The reason is that the deep feature construction procedure in the first step is independent of FSSH's hash code learning procedure, and the deep features $\mathbf{X}$ might not be optimally compatible with the hashing procedure. In contrast, all compared deep methods perform simultaneous feature learning and hash code learning, i.e., they are end-to-end. Although FSSH_deep is not an end-to-end method, it still performs better than most baselines which shows its effectiveness. And we believe that we can further boost the performance of FSSH by realizing our proposed method in an end-to-end deep version.

6 CONCLUSION

In this paper, we present a novel supervised hashing method, namely, Fast Scalable Supervised Hashing, FSSH for short. FSSH can be trained extremely fast. By introducing an intermediate term during the optimization, whose size is independent with the amount of

[1]Here, we replace the '-1' values in $\mathbf{B}$ with '0'.

data, FSSH can circumvent the use of the large similarity matrix **S**. Besides, by taking both the semantic information and specific data into consideration, FSSH is able to learn precise hash codes. Two shallow variants, i.e., FSSH_os and FSSH_ts, and one deep variant, i.e., FSSH_deep, are further proposed. Extensive experiments on three benchmark datasets demonstrate that all variants of FSSH outperform some state-of-the-art hashing methods in both accuracy and scalability. Since the focus of this work is on the development of loss function and optimization algorithm, we have only implemented two shallow variants and one simple deep variant of FSSH. In future, we plan to realize our proposed FSSH method in an end-to-end deep version to boost its performance.

ACKNOWLEDGMENTS

This work was partially supported by the Key Research and Development Program of Shandong Province (2016GGX101044), the National Natural Science Foundation of China (61573212, 61772310, 61702300, 61702302), and the Project of Thousand Youth Talents 2016. NExT research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative.

REFERENCES

[1] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*.
[2] Zhen-Duo Chen, Wan-Jin Yu, Chuan-Xiang Li, Liqiang Nie, and Xin-Shun Xu. 2018. Dual deep neural networks cross-modal hashing. In *AAAI*.
[3] 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.
[4] Cheng Da, Shibiao Xu, Kun Ding, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2017. AMVH: Asymmetric multi-valued hashing. In *CVPR*. 736–744.
[5] Guiguang Ding, Yuchen Guo, and Jile Zhou. 2014. Collective matrix factorization hashing for multimodal data. In *CVPR*. 2075–2082.
[6] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. 2015. Deep hashing for compact binary codes learning. In *CVPR*. 2475–2483.
[7] Tiezheng Ge, Kaiming He, and Jian Sun. 2014. Graph cuts for supervised binary coding. In *ECCV*. 250–264.
[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 Yannis Goulermas, Jianguo Jiang, Richang Hong, and Meng Wang. 2017. Unsupervised t-distributed video hashing and its deep hashing extension. *TIP* 24, 9 (2017), 2827–2840.
[13] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *SIGIR*. 355–364.
[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.
[15] Long-Kai Huang and Sinno Jialin Pan. 2016. Class-wise supervised hashing with label embedding and active bits. In *IJCAI*. 1585–1591.
[16] Qing-Yuan Jiang and Wu-Jun Li. 2015. Scalable graph hashing with feature transformation. In *IJCAI*. 2248–2254.
[17] Wang-Cheng Kang, Wu-Jun Li, and Zhi-Hua Zhou. 2016. Column sampling based discrete supervised hashing. In *AAAI*. 1230–1236.
[18] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto* (2009).
[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[20] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *ACM Conference on Information and Knowledge Management*. 1419–1428.
[21] Piji Li, Zihao Wang, Zhaochun Ren, Lidong Bing, and Wai Lam. 2017. Neural rating regression with abstractive tips generation for recommendation. In *SIGIR*. 345–354.
[22] Qi Li, Zhenan Sun, Ran He, and Tieniu Tan. 2017. Deep supervised discrete hashing. In *NIPS*. 2479–2488.
[23] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. 2016. Feature learning based deep supervised hashing with pairwise labels. In *IJCAI*. 1711–1717.
[24] 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.
[25] 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.
[26] Zijia Lin, Guiguang Ding, Mingqing Hu, and Jianmin Wang. 2015. Semantics-preserving hashing for cross-view retrieval. In *CVPR*. 3864–3872.
[27] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. 2016. Deep supervised hashing for fast image retrieval. In *CVPR*. 2064–2072.
[28] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. 2012. Supervised hashing with kernels. In *CVPR*. 2074–2081.
[29] Xin Luo, Ye Wu, and Xin-Shun Xu. 2018. Scalable Supervised Discrete Hashing for Large-Scale Search. In *WWW*. 1603–1612.
[30] 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.
[31] 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.
[32] Mohammad Norouzi and David M Blei. 2011. Minimal loss hashing for compact binary codes. In *ICML*. 353–360.
[33] 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.
[34] Mingdong Ou, Peng Cui, Fei Wang, Jun Wang, and Wenwu Zhu. 2015. Non-transitive hashing with latent similarity components. In *SIGKDD*. 895–904.
[35] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. 2015. Supervised discrete hashing. In *CVPR*. 37–45.
[36] Fumin Shen, Chunhua Shen, Qinfeng Shi, Anton Hengel, and Zhenmin Tang. 2013. Inductive hashing on manifolds. In *CVPR*. 1562–1569.
[37] Jinhui Tang, Zechao Li, Meng Wang, and Ruizhen Zhao. 2015. Neighborhood discriminant hashing for large-scale image retrieval. *TIP* 24, 9 (2015), 2827–2840.
[38] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. 2010. Semi-supervised hashing for scalable image retrieval. In *CVPR*. 3424–3431.
[39] Xiaofang Wang, Yi Shi, and Kris M Kitani. 2016. Deep supervised hashing with triplet labels. In *ACCV*. 70–84.
[40] Yair Weiss, Antonio Torralba, and Rob Fergus. 2009. Spectral hashing. In *NIPS*. 1753–1760.
[41] 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.
[42] Ting-Kun Yan, Xin-Shun Xu, Shanqing Guo, Zi Huang, and Xiao-Lin Wang. 2016. Supervised robust discrete multimodal hashing for cross-media retrieval. In *CIKM*. 1271–1280.
[43] Yang Yang, Fumin Shen, Heng-Tao Shen, Hanxi Li, and Xuelong Li. 2015. Robust discrete spectral hashing for large-scale image semantic indexing. *Trans. Big Data* 1, 4 (2015), 162–171.
[44] Dongqing Zhang and Wu-Jun Li. 2014. Large-scale supervised multimodal hashing with semantic correlation maximization. In *AAAI*. 2177–2183.
[45] Dan Zhang, Fei Wang, and Luo Si. 2011. Composite hashing with multiple information sources. In *SIGIR*. 225–234.
[46] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. 2010. Self-taught hashing for fast similarity search. In *SIGIR*. 18–25.
[47] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *SIGIR*. 325–334.
[48] Peichao Zhang, Wei Zhang, Wu-Jun Li, and Minyi Guo. 2014. Supervised hashing with latent factor models. In *SIGIR*. 173–182.
[49] Peng-Fei Zhang, Chuan-Xiang Li, Meng-Yuan Liu, Liqiang Nie, and Xin-Shun Xu. 2017. Semi-relaxation supervised hashing for cross-modal retrieval. In *MM*. 1762–1770.
[50] 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.
[51] Shifeng Zhang, Jianmin Li, Mengqing Jiang, and Bo Zhang. 2017. Scalable discrete supervised multimedia hash learning with clustering. *TCSVT* PP, 99 (2017), 1–1.
[52] Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. 2016. Efficient training of very deep neural networks for supervised hashing. In *CVPR*. 1487–1495.
[53] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. 2015. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*. 1556–1564.
[54] 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.