

Deep Discrete Supervised Hashing

Qing-Yuan Jiang, Xue Cui, and Wu-Jun Li[✉], *Member, IEEE*

Abstract—Hashing has been widely used for large-scale search due to its low storage cost and fast query speed. By using supervised information, supervised hashing can significantly outperform unsupervised hashing. Recently, discrete supervised hashing and feature learning based deep hashing are two representative progresses in supervised hashing. On one hand, hashing is essentially a discrete optimization problem. Hence, utilizing supervised information to directly guide discrete (binary) coding procedure can avoid sub-optimal solution and improve the accuracy. On the other hand, feature learning based deep hashing, which integrates deep feature learning and hash-code learning into an end-to-end architecture, can enhance the feedback between feature learning and hash-code learning. The key in discrete supervised hashing is to adopt supervised information to directly guide the discrete coding procedure in hashing. The key in deep hashing is to adopt the supervised information to directly guide the deep feature learning procedure. However, most deep supervised hashing methods cannot use the supervised information to directly guide both discrete (binary) coding procedure and deep feature learning procedure in the same framework. In this paper, we propose a novel deep hashing method, called deep discrete supervised hashing (DDSH). DDSH is the first deep hashing method which can utilize pairwise supervised information to directly guide both discrete coding procedure and deep feature learning procedure and thus enhance the feedback between these two important procedures. Experiments on four real datasets show that DDSH can outperform other state-of-the-art baselines, including both discrete hashing and deep hashing baselines, for image retrieval.

Index Terms—Image retrieval, deep learning, deep supervised hashing.

I. INTRODUCTION

DUE to the explosive increasing of data in real applications, nearest neighbor (NN) [1] search plays a fundamental role in many areas including image retrieval, computer vision, machine learning and data mining. In many real applications, there is no need to return the exact nearest neighbors for every given query and approximate nearest neighbor (ANN) is enough to achieve satisfactory search (retrieval) performance. Hence, ANN search has attracted much attention in recent years [2]–[9].

Manuscript received July 31, 2017; revised April 5, 2018 and May 30, 2018; accepted July 17, 2018. Date of publication August 10, 2018; date of current version September 4, 2018. This work was supported in part by the NSFC under Grant 61472182, in part by the Key Research and Development Program of Jiangsu under Grant BE2016121, and in part by a fund from Tencent. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Gang Hua. (*Corresponding author: Wu-Jun Li.*)

The authors are with the National Key Laboratory for Novel Software Technology, Collaborative Innovation Center of Novel Software Technology and Industrialization, Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China (e-mail: jiangqy@lamda.nju.edu.cn; cuix@lamda.nju.edu.cn; liwujun@nju.edu.cn).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TIP.2018.2864894

Over the last decades, hashing has become an active sub-area of ANN search [5], [10], [11]. The goal of hashing is to map the data points to binary codes with hash functions which tries to preserve the similarity in the original space of the data points. With the binary hash code representation, the storage cost for the data points can be dramatically reduced. Furthermore, hashing based ANN search is able to achieve a constant or sub-linear time complexity [12]. Hence, hashing has become a promising choice for efficient ANN search in large-scale datasets because of its low storage cost and fast query speed [2]–[4], [6], [12]–[18].

Existing hashing methods can be divided into two main categories: data-independent methods and data-dependent methods. Data-independent hashing methods usually adopt random projections as hash functions to map the data points from the original space into a Hamming space of binary codes. That is to say, these methods do not use any training data to learn hash functions and binary codes. Representative data-independent hashing methods include locality-sensitive hashing (LSH) [2], [19], kernelized locality-sensitive hashing (KLSH) [14]. Typically, data-independent hashing methods need long binary code to achieve satisfactory retrieval performance. Data-dependent hashing methods, which are also called *learning to hash* methods, try to learn the hash functions from data. Recent works [12], [16], [20]–[25] have shown that data-dependent methods can achieve comparable or even better accuracy with shorter binary hash codes, compared with data-independent methods. Therefore, data-dependent methods have received more and more attention.

Existing data-dependent hashing methods can be further divided into unsupervised hashing methods and supervised hashing methods, based on whether supervised information is used for learning or not. Unsupervised hashing methods aim to preserve the metric (Euclidean neighbor) structure among the training data. Representative unsupervised hashing methods include spectral hashing (SH) [26], iterative quantization (ITQ) [20], isotropic hashing (IsoHash) [10], spherical hashing (SPH) [16], inductive manifold hashing (IMH) [21], anchor graph hashing (AGH) [27], discrete graph hashing (DGH) [28], latent semantic minimal hashing (LSMH) [29] and global hashing system (GHS) [30]. Due to the semantic gap [31], unsupervised hashing methods usually can not achieve satisfactory retrieval performance in real applications. Unlike unsupervised hashing methods, supervised hashing methods aim to embed the data points from the original space into the Hamming space by utilizing supervised information to facilitate hash function learning or hash-code learning. Representative supervised hashing methods include semantic hashing [32], self-taught hashing (STH) [3], supervised

hashing with kernels (KSH) [12], latent factor hashing (LFH) [33], fast supervised hashing (FastH) [23], supervised discrete hashing (SDH) [25], column sampling based discrete supervised hashing (COSDISH) [34] and nonlinear discrete hashing (NDH) [17]. By using supervised information for learning, supervised hashing can significantly outperform unsupervised hashing in real applications such as image retrieval. Detailed surveys on *learning to hash* can be found in [7]–[9].

Hashing is essentially a discrete optimization problem. Because it is difficult to directly solve the discrete optimization problem, early hashing methods [15], [20], [22] adopt relaxation strategies to solve a proximate continuous problem which might result in a sub-optimal solution. Specifically, relaxation based hashing methods utilize supervised information to guide continuous hash code learning at the first stage. Then they convert continuous hash code into binary code by using rounding technology at the second stage. Recently, several discrete hashing methods, e.g., DGH [28], FastH [23], SDH [25] and COSDISH [34], which try to directly learn the discrete binary hash codes, have been proposed with promising performance. In particular, several discrete supervised hashing methods, including FastH [23], SDH [25], and COSDISH [34], have shown better performance than traditional relaxation-based continuous hashing methods. The key in discrete supervised hashing is to adopt supervised information to *directly* guide the discrete coding procedure, i.e., the discrete binary code learning procedure.

Most existing supervised hashing methods, including those introduced above and some deep hashing methods [17], [32], [35], are based on hand-crafted features. One major shortcoming for these methods is that they cannot perform feature learning. That is, these hand-crafted features might not be optimally compatible with the hash code learning procedure. Hence, besides the progress about discrete hashing, there has appeared another progress in supervised hashing, which is called feature learning based deep hashing [11], [36]–[43]. Representative feature learning based deep hashing methods includes convolutional neural network hashing (CNNH) [36], network in network hashing (NINH) [37], deep semantic ranking hashing (DSRH) [38], deep similarity comparison hashing (DSCH) [11], deep pairwise-supervised hashing (DPSH) [42], deep hashing network (DHN) [41], deep supervised hashing (DSH) [40], deep quantization network (DQN) [41] and deep supervised discrete hashing (DSDH) [44]. Deep hashing adopts deep learning [45], [46] for supervised hashing. In particular, most deep hashing methods adopt deep feature learning to learn a feature representation for hashing. Many deep hashing methods integrate deep feature representation learning and hashing code learning into an end-to-end architecture. Under this architecture, feature learning procedure and hash-code learning procedure can give feedback to each other in learning procedure. Many works [40], [42] have shown that using the supervised information to *directly* guide the deep feature learning procedure can achieve better performance than other strategies [36] which do not use supervised information to directly guide the deep feature learning procedure. Hence, the key in deep

TABLE I
NOTATION

Notation	Meaning
B	boldface uppercase, matrix
b	boldface lowercase, vector
B_{ij}	the (i, j) th element in matrix B
\mathbf{B}^T	transpose of matrix B
$\ \mathbf{b}\ _2$	Euclidean norm of vector b
Ω	capital Greek letter, set of indices
\bullet	Hadamard product (i.e., element-wise product)
\mathbf{b}^2	element-wise square of vector, i.e., $\mathbf{b}^2 = \mathbf{b} \bullet \mathbf{b}$

hashing is to adopt the supervised information to directly guide the deep feature learning procedure.

Both discrete supervised hashing and feature learning based deep hashing have achieved promising performance in many real applications. However, most deep supervised hashing methods cannot use the supervised information to *directly* guide both discrete (binary) coding procedure and deep feature learning procedure in the same framework. In this paper, we propose a novel deep hashing method, called deep discrete supervised hashing (DDSH). The main contributions of DDSH are outlined as follows:

- DDSH is the first deep hashing method which can utilize pairwise supervised information to *directly* guide both discrete coding procedure and deep feature learning procedure.
- By integrating the discrete coding procedure and deep feature learning procedure into the same end-to-end framework, these two important procedures can give feedback to each other to make the hash codes and feature representation more compatible.
- Experiments on four real datasets show that our proposed DDSH can outperform other state-of-the-art baselines, including both discrete hashing and deep hashing baselines.

The rest of this paper is organized as follows. In Section II, we briefly introduce the notations and problem definition in this paper. Then we describe DDSH in Section III. We discuss the difference between DDSH and existing deep hashing methods with pairwise labels in Section IV. In Section V, we evaluate DDSH on four datasets by carrying out the Hamming ranking task and hash lookup task. Finally, we conclude the paper in Section VI.

II. NOTATION AND PROBLEM DEFINITION

A. Notation

Some representative notations we use in this paper are outlined in Table I. More specifically, we use boldface uppercase letters like **B** to denote matrices. We use boldface lowercase letters like **b** to denote vectors. The (i, j) th element in matrix **B** is denoted as B_{ij} . \mathbf{B}^T is the transpose of **B** and $\|\mathbf{b}\|_2$ is the Euclidean norm of vector **b**. We use the capital Greek letter like Ω to denote the set of indices. We use the symbol \bullet to denote the Hadamard product (i.e., element-wise product). The square of a vector (or a matrix) like \mathbf{b}^2 indicates element-wise square, i.e., $\mathbf{b}^2 = \mathbf{b} \bullet \mathbf{b}$.

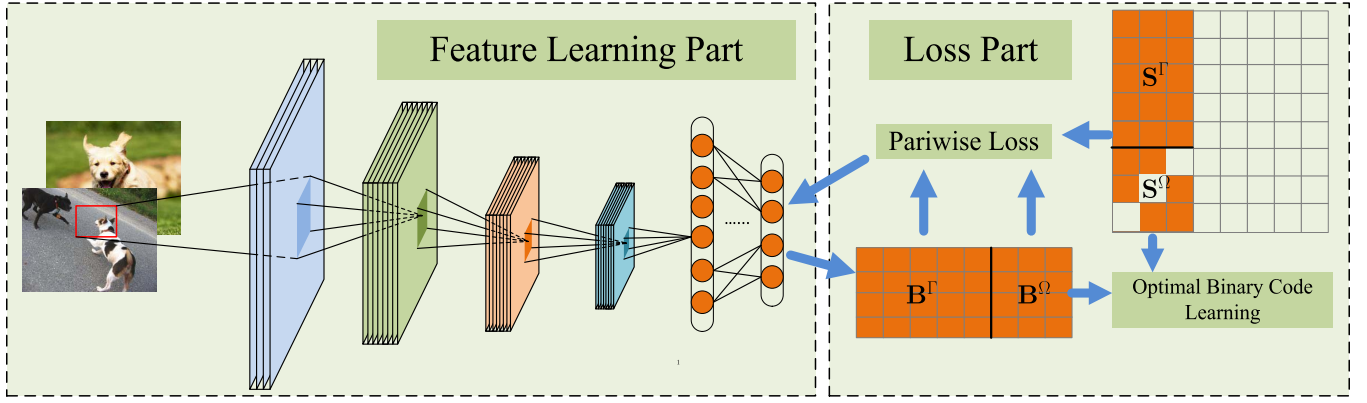


Fig. 1. The model architecture of DDSH. DDSH is an end-to-end deep learning framework which consists of two main components: loss part and feature learning part. The loss part contains the discrete coding procedure (to learn the binary codes \mathbf{B}^Ω), and the feature learning part contains the deep feature learning procedure (to learn the $F(\mathbf{x}; \Theta)$ for \mathbf{x} indexed by Γ). During each iteration, we adopt an alternating strategy to learn binary codes and feature representation alternatively, both of which are directly guided by supervised information. Hence, DDSH can enhance the feedback between the discrete coding procedure and the deep feature learning procedure.

B. Problem Definition

Although supervised information can also be triplet labels [11], [37]–[39] or pointwise labels [25], in this paper we only focus on the setting with pairwise labels [36], [40]–[43] which is a popular setting in supervised hashing. The technique in this paper can also be adapted to settings with triplet labels, which will be pursued in our future work.

We use $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ to denote a set of training points. In supervised hashing with pairwise labels, the supervised information $\mathbf{S} = \{-1, 1\}^{n \times n}$ between data points is also available for training procedure, where S_{ij} is defined as follows:

$$S_{ij} = \begin{cases} 1, & \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are similar.} \\ -1, & \text{otherwise.} \end{cases}$$

Supervised hashing aims at learning a hash function to map the data points from the original space into the binary code space (or called Hamming space), with the semantic (supervised) similarity in \mathbf{S} preserved in the binary code space. We define the hash function as: $h(\mathbf{x}) \in \{-1, +1\}^c, \forall \mathbf{x} \in \mathbf{X}$, where c is the binary code length. The Hamming distance between binary codes $\mathbf{b}_i = h(\mathbf{x}_i)$ and $\mathbf{b}_j = h(\mathbf{x}_j)$ is defined as follows:

$$\text{dist}_H(\mathbf{b}_i, \mathbf{b}_j) = \frac{1}{2}(c - \mathbf{b}_i^T \mathbf{b}_j).$$

To preserve the similarity between data points, the Hamming distance between the binary codes $\mathbf{b}_i = h(\mathbf{x}_i)$ and $\mathbf{b}_j = h(\mathbf{x}_j)$ should be relatively small if the data points \mathbf{x}_i and \mathbf{x}_j are similar, i.e., $S_{ij} = 1$. On the contrary, the Hamming distance between the binary codes $\mathbf{b}_i = h(\mathbf{x}_i)$ and $\mathbf{b}_j = h(\mathbf{x}_j)$ should be relatively large if the data points \mathbf{x}_i and \mathbf{x}_j are dissimilar, i.e., $S_{ij} = -1$. In other words, the goal of supervised hashing is to solve the following problem:

$$\begin{aligned} \min_h \mathcal{L}(h) &= \sum_{i,j=1}^n L(h(\mathbf{x}_i), h(\mathbf{x}_j); S_{ij}) \\ &= \sum_{i,j=1}^n L(\mathbf{b}_i, \mathbf{b}_j; S_{ij}), \end{aligned} \quad (1)$$

where $L(\cdot)$ is a loss function.

There have appeared various loss functions in supervised hashing. For example, KSH [12] uses L_2 function, which is defined as follows:

$$L(\mathbf{b}_i, \mathbf{b}_j; S_{ij}) = (S_{ij} - \frac{1}{c} \sum_{m=1}^c b_i^m b_j^m)^2,$$

where b_i^m is the m th element in vector \mathbf{b}_i . Please note that our DDSH is able to adopt many different kinds of loss functions. In this paper, we only use L_2 loss function as an example, and leave other loss functions for further study in future work.

III. DEEP DISCRETE SUPERVISED HASHING

In this section, we describe the details of DDSH, including the model architecture and learning algorithm.

A. Model Architecture

DDSH is an end-to-end deep hashing method which is able to simultaneously perform feature learning and hash code learning in the same framework. The model architecture of DDSH is shown in Figure 1, which contains two important components: *loss part* and *feature learning part*. The loss part contains the discrete coding procedure which aims to learn optimal binary code to preserve semantic pairwise similarity. The feature learning part contains the deep feature learning procedure which tries to learn a compatible deep neural network to extract deep representation from scratch. For DDSH, discrete coding procedure and deep feature learning are integrated into an end-to-end framework. More importantly, both procedures are *directly* guided by supervised information.

1) *Loss Part*: Inspired by COSDISH [34], we use column-sampling method to split the whole training set into two parts. More specifically, we randomly sample a subset Ω of $\Phi = \{1, 2, \dots, n\}$ and generate $\Gamma = \Phi \setminus \Omega$ ($|\Gamma| \gg |\Omega|$). Then we split the whole training set \mathbf{X} into two subsets \mathbf{X}^Ω and \mathbf{X}^Γ , where \mathbf{X}^Ω and \mathbf{X}^Γ denote the training data points indexed by Ω and Γ respectively.

Similarly, we sample $|\Omega|$ columns of \mathbf{S} with the corresponding sampled columns indexed by Ω . Then, we approximate the

original problem in (1) by only using the sampled columns of \mathbf{S} :

$$\begin{aligned} \min_h \mathcal{L}(h) &= \sum_{i \in \Omega} \sum_{j=1}^n L(h(\mathbf{x}_i), h(\mathbf{x}_j); S_{ij}) \\ &= \sum_{\mathbf{x}_i \in \mathbf{X}^\Omega} \sum_{\mathbf{x}_j \in \mathbf{X}^\Gamma} L(h(\mathbf{x}_i), h(\mathbf{x}_j); S_{ij}) \\ &\quad + \sum_{\mathbf{x}_i, \mathbf{x}_j \in \mathbf{X}^\Omega} L(h(\mathbf{x}_i), h(\mathbf{x}_j); S_{ij}). \end{aligned} \quad (2)$$

Then we introduce auxiliary variables to solve problem (2). More specifically, we utilize auxiliary variables $\mathbf{B}^\Omega = \{\mathbf{b}_i | i \in \Omega\}$ with $\mathbf{b}_i \in \{-1, +1\}^c$ to replace part of the binary codes generated by the hash function, i.e., $h(\mathbf{X}^\Omega)$. Here, $h(\mathbf{X}^\Omega) = \{h(\mathbf{x}_i) | \mathbf{x}_i \in \mathbf{X}^\Omega\}$. Then we rewrite the problem (2) as follows:

$$\begin{aligned} \min_{h, \mathbf{B}^\Omega} \mathcal{L}(h, \mathbf{B}^\Omega) &= \sum_{i \in \Omega} \sum_{\mathbf{x}_j \in \mathbf{X}^\Gamma} L(\mathbf{b}_i, h(\mathbf{x}_j); S_{ij}) \\ &\quad + \sum_{i, j \in \Omega} L(\mathbf{b}_i, \mathbf{b}_j; S_{ij}) \\ \text{s.t. } \mathbf{b}_i &\in \{-1, +1\}^c, \quad \forall i \in \Omega \end{aligned} \quad (3)$$

The problem in (3) is the final loss function (objective) to learn by DDSH. We can find that the whole training set is divided into two subsets \mathbf{X}^Ω and \mathbf{X}^Γ . The binary codes of \mathbf{X}^Ω , i.e., \mathbf{B}^Ω , are directly learned from the objective function in (3), but the binary codes of \mathbf{X}^Γ are generated by the hash function $h(\cdot)$. $h(\cdot)$ is defined based on the output of the deep feature learning part, which will be introduced in the following subsection.

The learning of \mathbf{B}^Ω contains the discrete coding procedure, which is directly guided by the supervised information. The learning of $h(\cdot)$ contains the deep feature learning procedure, which is also directly guided by the supervised information. Hence, our DDSH can utilize supervised information to directly guide both discrete coding procedure and deep feature learning procedure in the same end-to-end deep framework. This is different from existing deep hashing methods which either use relaxation strategy without discrete coding or do not use the supervised information to directly guide the discrete coding procedure.

Please note that “*directly guided*” in this paper means that the supervised information is directly included in the corresponding terms in the loss function. For example, the supervised information S_{ij} is directly included in all terms about the discrete codes \mathbf{B}^Ω in (3), which means that the discrete coding procedure is directly guided by the supervised information. Furthermore, the supervised information S_{ij} is also directly included in the term about the deep feature learning function $h(\mathbf{x}_j)$ in (3), which means that the deep feature learning procedure is also directly guided by the supervised information. To the best of our knowledge, DDSH is the first deep hashing method which can utilize pairwise supervised information to directly guide both discrete coding procedure and deep feature learning procedure, and thus enhance the feedback between these two important procedures.

TABLE II

CONFIGURATION OF THE CONVOLUTIONAL LAYERS IN DDSH

Layer	Configuration				
	filter size	stride	pad	LRN	pool
conv1	$64 \times 11 \times 11$	4×4	0	yes	2×2
conv2	$256 \times 5 \times 5$	1×1	2	yes	2×2
conv3	$256 \times 3 \times 3$	1×1	1	no	-
conv4	$256 \times 3 \times 3$	1×1	1	no	-
conv5	$256 \times 3 \times 3$	1×1	1	no	2×2

TABLE III

CONFIGURATION OF THE FULLY-CONNECTED LAYERS IN DDSH

Layer	Configuration
full6	4096
full7	4096
full8	hash code length c

2) *Feature Learning Part*: The binary codes of \mathbf{X}^Γ are generated by the hash function $h(\cdot)$, which is defined based on the output of the deep feature learning part. More specifically, we define our hash function as: $h(\mathbf{x}) = \text{sign}(F(\mathbf{x}; \Theta))$, where $\text{sign}(\cdot)$ is the element-wise sign function. $F(\mathbf{x}; \Theta)$ denotes the output of the feature learning part and Θ denotes all parameters of the deep neural network.

We adopt a convolutional neural network (CNN) from [47], i.e., CNN-F, as our deep feature learning part. We replace the last layer of CNN-F as one fully-connected layer to project the output of the second last layer to \mathbb{R}^c space. More specifically, the feature learning part contains 5 convolutional layers (“conv1-conv5”) and 3 fully-connected layers (“full6-full8”). The detailed configuration of the 5 convolutional layers is shown in Table II. In Table II, “filter size” denotes the number of convolutional filters and their receptive field size. “stride” specifies the convolutional stride. “pad” indicates the number of pixels to add to each size of the input. “LRN” denotes whether Local Response Normalization (LRN) [45] is applied or not. “pool” denotes the down-sampling factor. The detailed configuration of the 3 fully-connected layers is shown in Table III, where the “configuration” shows the number of nodes in each layer.

We adopt the Rectified Linear Unit (ReLU) [45] as activation function for all the first seven layers. For the last layer, we utilize identity function as the activation function.

B. Learning

After randomly sampling Ω at each iteration, we utilize an alternating learning strategy to solve problem (3).

More specifically, each time we learn one of the variables \mathbf{B}^Ω and $h(F(\mathbf{x}; \Theta))$ with the other fixed. When $h(F(\mathbf{x}; \Theta))$ is fixed, we directly learn the discrete hash code \mathbf{B}^Ω over binary variables. When \mathbf{B}^Ω is fixed, we update the parameter Θ of the deep neural network.

1) *Learn \mathbf{B}^Ω With $h(F(\mathbf{x}; \Theta))$ Fixed*: When $h(F(\mathbf{x}; \Theta))$ is fixed, it’s easy to transform problem (3) into a binary quadratic programming (BQP) problem as that in TSH [22]. Each time we optimize one bit for all points. Then, the optimization of

the k th bit of all points in \mathbf{B}^Ω is given by:

$$\begin{aligned} \min_{\mathbf{b}^k} & [\mathbf{b}^k]^T \mathbf{Q}^k \mathbf{b}^k + [\mathbf{b}^k]^T \mathbf{p}^k \\ \text{s.t. } & \mathbf{b}^k \in \{-1, +1\}^{|\Omega|} \end{aligned} \quad (4)$$

where \mathbf{b}^k denotes the k th column of \mathbf{B}^Ω , and

$$\begin{aligned} Q_{ij}^k &= -2(cS_{ij}^\Omega - \sum_{m=1}^{k-1} b_i^m b_j^m) \\ Q_{ii}^k &= 0 \\ p_i^k &= -2 \sum_{l=1}^{|\Gamma|} B_{lk}^\Gamma (cS_{li}^\Gamma - \sum_{m=1}^{k-1} B_{lm}^\Gamma b_i^m). \end{aligned}$$

Here, b_i^m denotes the m th bit of \mathbf{b}_i and p_i^k denotes the i th element of \mathbf{p}^k .

Following COSDISH, we can easily solve problem (4) by transforming the BQP problem into an equally clustering problem [48].

2) *Learn $h(F(\mathbf{x}; \Theta))$ With \mathbf{B}^Ω Fixed:* Because the derivative of the hash function $h(\mathbf{x}) = \text{sign}(F(\mathbf{x}; \Theta))$ is 0 everywhere except at 0, we cannot use back-propagation (BP) methods to update the neural network parameters. So we relax $\text{sign}(\cdot)$ as $h(\mathbf{x}) = \tanh(F(\mathbf{x}; \Theta))$ inspired by Song *et al.* [24]. Then we optimize the following problem:

$$\begin{aligned} \min_h & \mathcal{L}(h) = \sum_{i \in \Omega} \sum_{\mathbf{x}_j \in \mathbf{X}^\Gamma} L(\mathbf{b}_i, h(\mathbf{x}_j); S_{ij}) \\ \text{s.t. } & h(\mathbf{x}_j) = \tanh(F(\mathbf{x}_j; \Theta)) \end{aligned} \quad (5)$$

To learn the CNN parameter Θ , we utilize a back-propagation algorithm. That is, each time we sample a mini-batch of data points, and then use BP algorithm based on the sampled data.

We define the output of CNN as $\mathbf{z}_j = F(\mathbf{x}_j; \Theta)$ and $\mathbf{a}_j = \tanh(\mathbf{z}_j)$. Then we can compute the gradient of \mathbf{a}_j and \mathbf{z}_j as follows:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{a}_j} &= \sum_{i \in \Omega} \frac{\partial L(\mathbf{b}_i, \mathbf{a}_j; S_{ij})}{\partial \mathbf{a}_j} \\ &= \sum_{i \in \Omega} 2(\mathbf{a}_j^T \mathbf{b}_i - S_{ij}) \mathbf{b}_i \end{aligned} \quad (6)$$

and

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{z}_j} &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}_j} \bullet (1 - \mathbf{a}_j^2) \\ &= \sum_{i \in \Omega} 2(\mathbf{a}_j^T \mathbf{b}_i - S_{ij}) \mathbf{b}_i \bullet (1 - \mathbf{a}_j^2) \end{aligned} \quad (7)$$

Then, we can use chain rule to compute $\frac{\partial \mathcal{L}}{\partial \Theta}$ based on $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_j}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{z}_j}$.

We summarize the whole learning algorithm for DDSH in Algorithm 1.

C. Out-of-Sample Extension for Unseen Data Points

After training our DDSH model, we can adopt the learned deep hashing framework to predict the binary code for any unseen data point during training.

Algorithm 1 The Learning Algorithm for DDSH

Input:

Training set \mathbf{X} ;
Code length c ;
Supervised information \mathbf{S} .

Output:

Parameter Θ of the deep neural network.

Initialization

Initialize neural network parameter Θ , mini-batch size M and iteration number T_{out}, T_{in}

Initialize $\mathbf{B} = \{\mathbf{b}_i | i = 1, 2, \dots, n\}$

for $iter = 1, 2, \dots, T_{out}$ **do**

Randomly sample Ω and set $\Gamma = \Phi \setminus \Omega$

Split training set \mathbf{X} into \mathbf{X}^Ω and \mathbf{X}^Γ .

Split \mathbf{B} into \mathbf{B}^Ω and \mathbf{B}^Γ .

for $epoch = 1, 2, \dots, T_{in}$ **do**

for $k = 1, 2, \dots, c$ **do**

Construct the BQP problem for the k th bit using (4).

Construct the clustering problem to solve the BQP problem for the k th bit.

end for

for $t = 1, 2, \dots, |\Gamma|/M$ **do**

Randomly sample M data points from \mathbf{X}^Γ to construct a mini-batch.

Calculate $h(\mathbf{x}_j)$ for each data point \mathbf{x}_j in the mini-batch by forward propagation.

Calculate the gradient according to (7).

Update the parameter Θ by using back propagation.

Update $\mathbf{b}_j = \text{sign}(h(\mathbf{x}_j))$ for each data point \mathbf{x}_j in the mini-batch.

end for

end for

end for

More specifically, given any point $\mathbf{x}_q \notin \mathbf{X}$, we use the following formula to predict its binary code:

$$\mathbf{b}_q = h(\mathbf{x}_q) = \text{sign}(F(\mathbf{x}_q; \Theta)),$$

where Θ is the deep neural network parameter learned by DDSH model.

IV. COMPARISON TO RELATED WORK

Existing feature learning based deep hashing methods with pairwise labels either use relaxation strategy without discrete coding or do not use the supervised information to directly guide the discrete coding procedure. For example, CNNH [36] is a two-step method which adopts relaxation strategy to learn continuous code in the first stage and performs feature learning in the second stage. The feature learning procedure in CNNH is not directly guided by supervised information. NINH [37], DHN [41] and DSH [40] adopt relaxation strategy to learn continuous code. DPSH [42] and DQN [41] can learn binary code in the training procedure. However, DSPH and DQN do not utilize the supervised information to directly guide the

discrete coding procedure. The objective function of DPSH¹ can be written as:

$$\mathcal{L}_{\text{DPSH}} = - \sum_{S_{ij} \in \mathbf{S}} (S_{ij} \Theta_{ij} - \log(1 + e^{\Theta_{ij}})) + \eta \sum_{i=1}^n \|\mathbf{b}_i - \mathbf{u}_i\|_F^2,$$

where $\Theta_{ij} = \frac{1}{2} \mathbf{u}_i^T \mathbf{u}_j$ and \mathbf{u}_i denotes the output of the deep neural network. We can find that in DPSH the discrete coding procedure is not directly guided by supervised information, i.e., the supervised information is not directly included in the terms of $\{\mathbf{b}_i\}$ in the objective function. The objective function of DQN can be written as:

$$\mathcal{L}_{\text{DQN}} = \sum_{S_{ij} \in \mathbf{S}} (S_{ij} - \frac{\mathbf{z}_i^T \mathbf{z}_j}{\|\mathbf{z}_i\| \|\mathbf{z}_j\|})^2 + \lambda \sum_{i=1}^n \|\mathbf{z}_i - \mathbf{C}\mathbf{h}_i\|_F^2,$$

where \mathbf{z}_i denotes the output of the deep neural network and $\sum_{i=1}^n \|\mathbf{z}_i - \mathbf{C}\mathbf{h}_i\|_F^2$ denotes the product quantization loss. The discrete coding procedure is only contained in the term $\sum_{i=1}^n \|\mathbf{z}_i - \mathbf{C}\mathbf{h}_i\|_F^2$. We can find that in DQN the discrete coding procedure is not directly guided by supervised information either.

There appears one deep hashing method called DSDH [44]. Unlike DDSH, DSDH utilizes pointwise supervised information to guide the discrete coding procedure and utilize pairwise similarity to guide feature learning procedure. Then DSDH bridges discrete coding procedure and feature learning procedure by using the method of auxiliary coordinates (MAC) technique in AFFHash [49]. Due to the requirements of pointwise labels and pairwise labels, the application scenarios of DSDH might be limited.

To the best of our knowledge, our DDSH is the first deep hashing method which can utilize pairwise supervised information to directly guide both discrete coding procedure and deep feature learning procedure in the same framework.

V. EXPERIMENT

We evaluate DDSH and other baselines on datasets from image retrieval applications. The open source deep learning library MatConvNet [50] is used to implement our model. All experiments are performed on an NVIDIA K40 GPU server.

A. Experimental Setting

1) *Datasets*: We adopt four widely used image datasets to evaluate our proposed method. They are CIFAR-10² [45], SVHN³ [51], NUS-WIDE⁴ [52] and Clothing1M⁵ [53].

The CIFAR-10 dataset contains 60,000 images which are manually labeled into 10 classes including “airplane”, “automobile”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship” and “truck”. It’s a single-label dataset. The size of each image is 32×32 pixels. Two images are treated as similar if they share the same label, i.e., they belong to the same class. Otherwise, they are considered to be dissimilar.

The SVHN dataset consists of 73,257 digits for training, 26,032 digits for testing and 531,131 additional samples. It is a real-world image dataset for recognizing digital numbers in natural scene images. The images are categorized into 10 classes, each corresponding to a digital number. SVHN is also a single-label dataset. Two images are treated as similar if they share the same label. Otherwise, they are considered to be dissimilar.

The NUS-WIDE dataset is a large-scale image dataset which includes 269,648 images and the associated tags from Flickr website. It’s a multi-label dataset where each image might be annotated with multi labels. We select 186,577 data points that belong to the 10 most frequent concepts from the original dataset. Two images are treated as similar if they share at least one label. Otherwise, they are considered to be dissimilar.

The Clothing1M dataset is a relatively large-scale dataset which contains 1,037,497 images which belong to 14 classes including “T-shirt”, “shirt”, “knitwear”, “chiffon”, “sweater”, “hoodie”, “windbreaker”, “jacket”, “downcoat”, “suit”, “shawl”, “dress”, “vest” and “underwear”. Clothing1M dataset is a single-label dataset. Two images are treated as similar if they share the same label, i.e., they belong to the same class. Otherwise, they are considered to be dissimilar.

Table IV illustrates some example points from the above four datasets.

For CIFAR-10 dataset, we randomly take 1,000 images (100 images per class) as query set and the remaining images as retrieval set. Furthermore, we randomly select 5,000 images (500 images per class) from retrieval set as training set. For SVHN dataset, we randomly select 1,000 images (100 images per class) from testing set as query set and utilize the whole training set as retrieval set. We randomly select 5,000 images (500 images per class) from retrieval set as training set. For NUS-WIDE dataset, we randomly select 1,867 data points as query set and the remaining data points as retrieval set. We randomly select 5,000 data points from retrieval set as training set. For Clothing1M dataset, after removing the images whose links are invalid, we randomly select 7,000 images (500 images per class) as query set and 1,028,083 images as retrieval set. Furthermore, we randomly sample 14,000 images (1,000 images per class) from retrieval set to construct training set.

2) *Baselines and Evaluation Protocol*: We compare DDSH with eleven state-of-the-art baselines, including LSH [19], ITQ [20], LFH [33], FastH [23], SDH [25], COSDISH [34], NDH [17], DHN [43], DSH [40], DPSH [42] and DSDH [44]. These baselines are briefly introduced as follows:

- Locality-sensitive hashing (LSH) [19]: LSH is a representative data-independent hashing method. LSH utilizes random projection to generate hash function.
- Iterative quantization (ITQ) [20]: ITQ is a representative unsupervised hashing method. ITQ first projects data points into low space by utilizing principal component analysis (PCA). Then ITQ minimizes the quantization error to learn binary code.

¹For DPSH, supervised information S_{ij} is defined on $\{0, 1\}$.


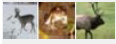

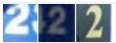
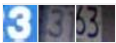
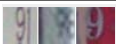





²<https://www.cs.toronto.edu/~kriz/cifar.html>

³<http://ufldl.stanford.edu/housenumbers/>

⁴<http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm>

⁵https://github.com/Cysu/noisy_label

TABLE IV
EXAMPLE POINTS OF THE DATASETS

Dataset	Example	Label
CIFAR-10		"frog".
		"deer".
		"truck".
SVHN		"2".
		"3".
		"9".
NUS-WIDE		"person", "sky".
		"clouds", "ocean", "person", "sky", "water".
		"road", "clouds", "sky", "buildings".
Clothing1M		"T-shirt".
		"shawl".

- Latent factor hashing (LFH) [33]: LFH is a supervised hashing method which tries to learn binary code based on latent factor models.
- Fast supervised hashing (FastH) [23]: FastH is supervised hashing method. FastH directly adopts graph-cut method to learn discrete binary code.
- Supervised discrete hashing (SDH) [25]: SDH is a point-wise supervised hashing method which utilizes the discrete cyclic coordinate descent (DCC) algorithm to learn discrete hash code.
- Column sampling based discrete supervised hashing (COSDISH) [34]: COSDISH is a supervised hashing method. COSDISH can directly learn discrete hash code.
- Nonlinear deep hashing (NDH) [17]: NDH is a deep supervised hashing method. NDH utilizes both pointwise label and pairwise similarity to guide binary coding and hash-function learning. However, NDH is a hand-crafted feature based method.
- Deep hashing network (DHN) [43]: DHN is a deep supervised hashing method. DHN minimizes both pairwise cross-entropy loss and pairwise quantization loss.
- Deep supervised hashing (DSH) [40]: DSH is a deep supervised hashing method. DSH takes pairs of points

as input and learns binary codes by maximizing the discriminability of the corresponding binary codes.

- Deep pairwise-supervised hashing (DPSH) [42]: DPSH is a deep supervised hashing method. DPSH performs deep feature learning and hash-code learning simultaneously with pairwise labels by minimizing negative log-likelihood of the observed pairwise labels.
- Deep supervised discrete hashing (DSDH) [44]: DSDH is a deep supervised hashing method. Similar to NDH, DSDH also utilizes both pointwise label and pairwise similarity to learn binary codes and deep neural network. Furthermore, DSDH adopts the method of auxiliary coordinates (MAC) technique in AFFHash [49] to bridge binary coding procedure and feature learning procedure.

Among all these baselines, LSH is a data-independent hashing method. ITQ is an unsupervised hashing method. LFH, FastH, COSDISH, and SDH are non-deep methods, which cannot perform deep feature learning. LFH is a relaxation-based method. FastH, COSDISH and SDH are discrete supervised hashing methods. NDH is a hand-crafted feature based deep supervised hashing method. DHN, DSH, and DPSH are deep hashing methods with pairwise labels which can perform feature learning and hash-code learning simultaneously. DSDH is a deep supervised hashing method which utilizes both pointwise label and pairwise similarity.

For fair comparison, all feature learning based deep hashing methods, including deep baselines and our DDSH, adopt the same pre-trained CNN-F model on ImageNet⁶ for feature learning. Because the CNN-F model is pre-trained with images of size 224×224 pixels, we first resize all images to be 224×224 pixels for four datasets. Then the raw image pixels are directly utilized as input for deep hashing methods. We carefully implement DHN and DSH on MatConvNet. We fix the mini-batch size to be 128 and tune the learning rate from 10^{-6} to 10^{-2} by using a cross-validation strategy. Furthermore, we set weight decay as 5×10^{-4} to avoid overfitting. For DDSH, we set $|\Omega| = 100$, $T_{out} = 3$ and $T_{in} = 50$ for CIFAR-10, SVHN and NUS-WIDE datasets. Because NUS-WIDE is a multi-label dataset, we reduce the similarity weight for those training points with multi labels when we train DDSH. For Clothing1M dataset, we set $|\Omega| = 500$, $T_{out} = 10$ and $T_{in} = 15$.

For other supervised hashing methods, including LFH, ITQ, LFH, FastH, SDH, COSDISH and NDH, we use 4,096-dim deep features extracted by the CNN-F model pre-trained on ImageNet as input for fair comparison. Because SDH is a kernel-based methods, we randomly sample 1,000 data points as anchors to construct the kernel by following the suggestion of Shen *et al.* [25] of SDH. For LFH, FastH and COSDISH, we utilize boosted decision tree for out-of-sample extension by following the setting of FastH. For NDH whose source code is not available, we carefully re-implement its algorithm by ourselves. Following the authors' suggestion, we use 200-dimension feature vector derived from PCA on deep features for NDH.

⁶We download the CNN-F model pre-trained on ImageNet from <http://www.vlfeat.org/matconvnet/pretrained/>.

TABLE V

MAP OF THE HAMMING RANKING TASK ON CIFAR-10 DATASET.
THE BEST ACCURACY IS SHOWN IN BOLDFACE

Method	CIFAR-10			
	12 bits	24 bits	32 bits	48 bits
DDSH	0.7695	0.8289	0.8352	0.8194
DSDH	0.7442	0.7868	0.7991	0.8142
DPSH	0.6844	0.7225	0.7396	0.7460
DSH	0.6457	0.7492	0.7857	0.8113
DHN	0.6725	0.7107	0.7045	0.7135
NDH	0.5620	0.6404	0.6515	0.6772
COSDISH	0.6085	0.6827	0.6959	0.7158
SDH	0.5200	0.6461	0.6577	0.6688
FastH	0.6202	0.6731	0.6870	0.7163
LFH	0.4009	0.6047	0.6571	0.6995
ITQ	0.2580	0.2725	0.2834	0.2936
LSH	0.1468	0.1725	0.1798	0.1929

In our experiment, ground-truth neighbors are defined based on whether two data points share at least one class label. We carry out Hamming ranking task and hash lookup task to evaluate DDSH and baselines. We report the Mean Average Precision (MAP), top-k precision, precision-recall curve and case study for Hamming ranking task. Specifically, given a query \mathbf{x}_q , we can calculate its average precision (AP) through the following equation:

$$AP(\mathbf{x}_q) = \frac{1}{R_k} \sum_{k=1}^N P(k) \mathbb{I}_1(k),$$

where R_k is the number of the relevant samples, $P(k)$ is the precision at cut-off k in the returned sample list and $\mathbb{I}_1(k)$ is an indicator function which equals 1 if the k th returned sample is a ground-truth neighbor of \mathbf{x}_q . Otherwise, $\mathbb{I}_1(k)$ is 0. Given Q queries, we can compute the MAP as follows:

$$MAP = \frac{1}{Q} \sum_{q=1}^Q AP(\mathbf{x}_q).$$

Because NUS-WIDE is relatively large, the MAP value on NUS-WIDE is calculated based on the top 5000 returned neighbors. The MAP values for other datasets are calculated based on the whole retrieval set.

For hash lookup task, we report mean hash lookup success rate (SR) within Hamming radius 0, 1 and 2 [28]. When at least one ground-truth neighbor is retrieved within a specific Hamming radius, we call it a lookup success. The hash lookup success rate (SR) can be calculated as follows:

$$SR = \sum_{q=1}^Q \frac{\mathbb{I}(\text{\#retrieved ground-truth for query } \mathbf{x}_q > 0)}{Q}$$

Here, $\mathbb{I}(\cdot)$ is an indicator function, i.e., $\mathbb{I}(\text{true}) = 1$ and $\mathbb{I}(\text{false}) = 0$. Q is the total number of query images.

B. Experimental Result

1) *Hamming Ranking Task:* Table V, Table VI, Table VII and Table VIII reports the MAP result on CIFAR-10, SVHN, NUS-WIDE and Clothing1M dataset, respectively. We can easily find that our DDSH achieves the state-of-the-art retrieval

TABLE VI

MAP OF THE HAMMING RANKING TASK ON SVHN DATASET.
THE BEST ACCURACY IS SHOWN IN BOLDFACE

Method	SVHN			
	12 bits	24 bits	32 bits	48 bits
DDSH	0.5735	0.6744	0.7031	0.7184
DSDH	0.5121	0.5670	0.5866	0.5839
DPSH	0.3790	0.4216	0.4337	0.4557
DSH	0.3702	0.4802	0.5232	0.5828
DHN	0.3800	0.4096	0.4158	0.4302
NDH	0.2177	0.2710	0.2563	0.2803
COSDISH	0.2381	0.2951	0.3196	0.3408
SDH	0.1509	0.2996	0.3202	0.3335
FastH	0.2516	0.2961	0.3177	0.3436
LFH	0.1933	0.2558	0.2839	0.3253
ITQ	0.1108	0.1138	0.1149	0.1159
LSH	0.1074	0.1082	0.1093	0.1109

TABLE VII

MAP OF THE HAMMING RANKING TASK ON NUS-WIDE DATASET.
THE BEST ACCURACY IS SHOWN IN BOLDFACE

Method	NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits
DDSH	0.7911	0.8165	0.8217	0.8259
DSDH	0.7916	0.8059	0.8063	0.8180
DPSH	0.7882	0.8085	0.8167	0.8234
DSH	0.7622	0.7940	0.7968	0.8081
DHN	0.7900	0.8101	0.8092	0.8180
NDH	0.7015	0.7351	0.7447	0.7449
COSDISH	0.7303	0.7643	0.7868	0.7993
SDH	0.7385	0.7616	0.7697	0.7720
FastH	0.7412	0.7830	0.7948	0.8085
LFH	0.7049	0.7594	0.7778	0.7936
ITQ	0.5053	0.5037	0.5031	0.5054
LSH	0.3407	0.3506	0.3509	0.3706

TABLE VIII

MAP OF THE HAMMING RANKING TASK ON CLOTHING1M DATASET.
THE BEST ACCURACY IS SHOWN IN BOLDFACE

Method	Clothing1M			
	12 bits	24 bits	32 bits	48 bits
DDSH	0.2763	0.3667	0.3878	0.4008
DSDH	0.2903	0.3285	0.3413	0.3475
DPSH	0.1950	0.2087	0.2162	0.2181
DSH	0.1730	0.1870	0.1912	0.2021
DHN	0.1909	0.2243	0.2120	0.2488
NDH	0.1857	0.2276	0.2338	0.2354
COSDISH	0.1871	0.2358	0.2567	0.2756
SDH	0.1518	0.1865	0.1941	0.1973
FastH	0.1736	0.2066	0.2167	0.2440
LFH	0.1548	0.1591	0.2128	0.2579
ITQ	0.1150	0.1214	0.1228	0.1259
LSH	0.0834	0.0894	0.0914	0.0920

accuracy in most cases compared with all baselines, including deep hashing methods, non-deep supervised hashing methods, non-deep unsupervised hashing methods and data-independent methods.

By comparing ITQ to LSH, we can find that the data-dependent hashing methods can significantly outperform data-independent hashing methods. By comparing NDH, COSDISH, SDH, FastH and LFH to ITQ, we can find that supervised methods can outperform unsupervised methods because of the effect of using supervised information.

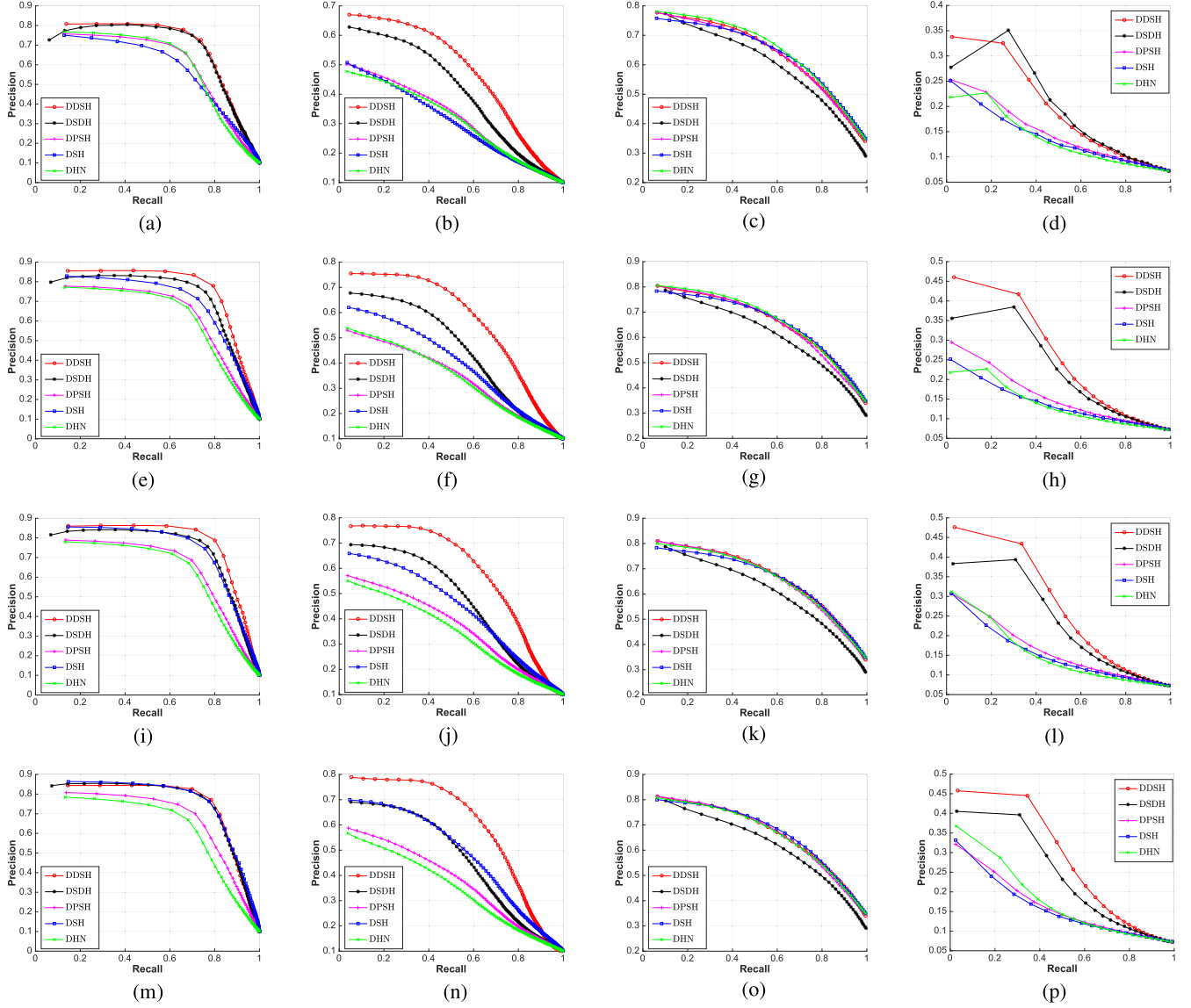


Fig. 2. Performance of precision-recall curve on four datasets. The four sub-figures in each row are the precision-recall curves for 12 bits, 24 bits, 32 bits and 48 bits, respectively. (a) 12 bits @CIFAR-10. (b) 12 bits @SVHN. (c) 12 bits @NUS-WIDE. (d) 12 bits @Clothing1M. (e) 24 bits @CIFAR-10. (f) 24 bits @SVHN. (g) 24 bits @NUS-WIDE. (h) 24 bits @Clothing1M. (i) 32 bits @CIFAR-10. (j) 32 bits @SVHN. (k) 32 bits @NUS-WIDE. (l) 48 bits @Clothing1M. (m) 48 bits @CIFAR-10. (n) 48 bits @SVHN. (o) 48 bits @NUS-WIDE. (p) 48 bits @Clothing1M.

By comparing NDH, COSDISH, SDH and FastH to LFH, we can find that discrete supervised hashing can outperform relaxation-based continuous hashing, which means that discrete coding procedure is able to learn more optimal binary codes. By comparing feature learning based deep hashing methods, i.e., DDSH, DSDH, DPSH, DHN and DSH, to non-deep hashing methods, we can find that feature learning based deep hashing can outperform non-deep hashing because deep supervised hashing can perform deep feature learning compared with non-deep hashing methods. This experimental result demonstrates that deep supervised hashing is a more compatible architecture for hashing learning.

The main difference between our proposed DDSH and other discrete supervised hashing methods like COSDISH, SDH and FastH is that our DDSH adopts supervised information

to directly guide deep feature learning procedure but other discrete supervised hashing methods do not have deep feature learning ability. The main difference between our DDSH and other deep hashing methods is that DDSH adopts supervised information to directly guide the discrete coding procedure but other deep hashing methods do not have this property. Hence, the experimental results successfully demonstrate the motivation of DDSH, i.e., utilizing supervised information to *directly* guide both deep feature learning procedure and discrete coding procedure can further improve retrieval performance in real applications.

Furthermore, we select four best baselines, i.e., DSDH, DPSH, DSH and DHN, to compare the precision-recall and top-k precision results. We report the precision-recall curve on all four datasets in Figure 2. We can see that the

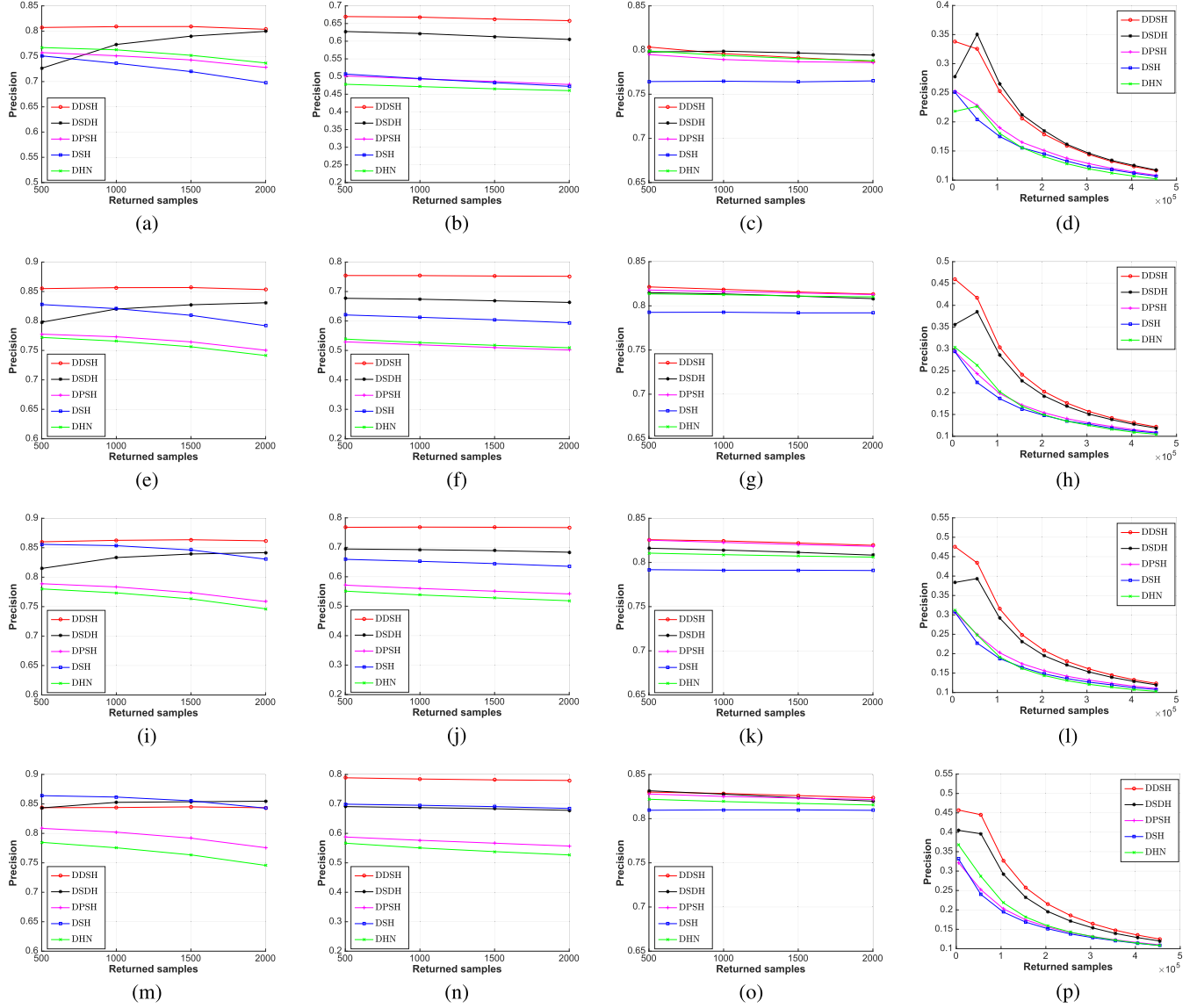


Fig. 3. Performance of top-k precision on four datasets. The four sub-figures in each row are the top-k precision curves for 12 bits, 24 bits, 32 bits and 48 bits, respectively. (a) 12 bits @CIFAR-10. (b) 12 bits @SVHN. (c) 12 bits @NUS-WIDE. (d) 12 bits @Clothing1M. (e) 24 bits @CIFAR-10. (f) 24 bits @SVHN. (g) 24 bits @NUS-WIDE. (h) 24 bits @Clothing1M. (i) 32 bits @CIFAR-10. (j) 32 bits @SVHN. (k) 32 bits @NUS-WIDE. (l) 32 bits @Clothing1M. (m) 48 bits @CIFAR-10. (n) 48 bits @SVHN. (o) 48 bits @NUS-WIDE. (p) 48 bits @Clothing1M.

proposed DDSH still achieves the best performance in terms of precision-recall curve in most cases.

In real applications, we might care about top-k retrieval results more than the whole database. Hence we report the top-k precision on four datasets based on the returned top-k samples. In Figure 3, we show the top-k precision for different k on CIFAR-10 dataset, SVHN dataset, NUS-WIDE and Clothing1M dataset respectively, where k is the number of returned samples. Again, we can find that DDSH can outperform other deep hashing methods in most cases.

2) Hash Lookup Task: In practice, retrieval with hash lookup can usually achieve constant or sub-linear search speed in real applications. Recent works like DGH [28] show that discrete hashing can significantly improve hash lookup success rate.

In Figure 4, we present the mean hash lookup success rate within Hamming radius 0, 1 and 2 on all four datasets for all deep hashing methods. We can find that DDSH can achieve the best mean hash lookup success rate on four datasets, especially for long codes.

3) Further Analysis: To further demonstrate the effectiveness of utilizing supervised information to directly guide both discrete coding procedure and deep feature learning procedure in the same end-to-end framework, we evaluate several variants of DDSH. These variants include “DDSH0”, “COSDISH-Linear”, “COSDISH-CNN” and “DDSH-MAC”.

DDSH0 denotes the variant in which we fix the parameters of the first seven layers of CNN-F in DDSH during training procedure. In other words, DDSH0 can not perform deep feature learning procedure, and all the other parts are

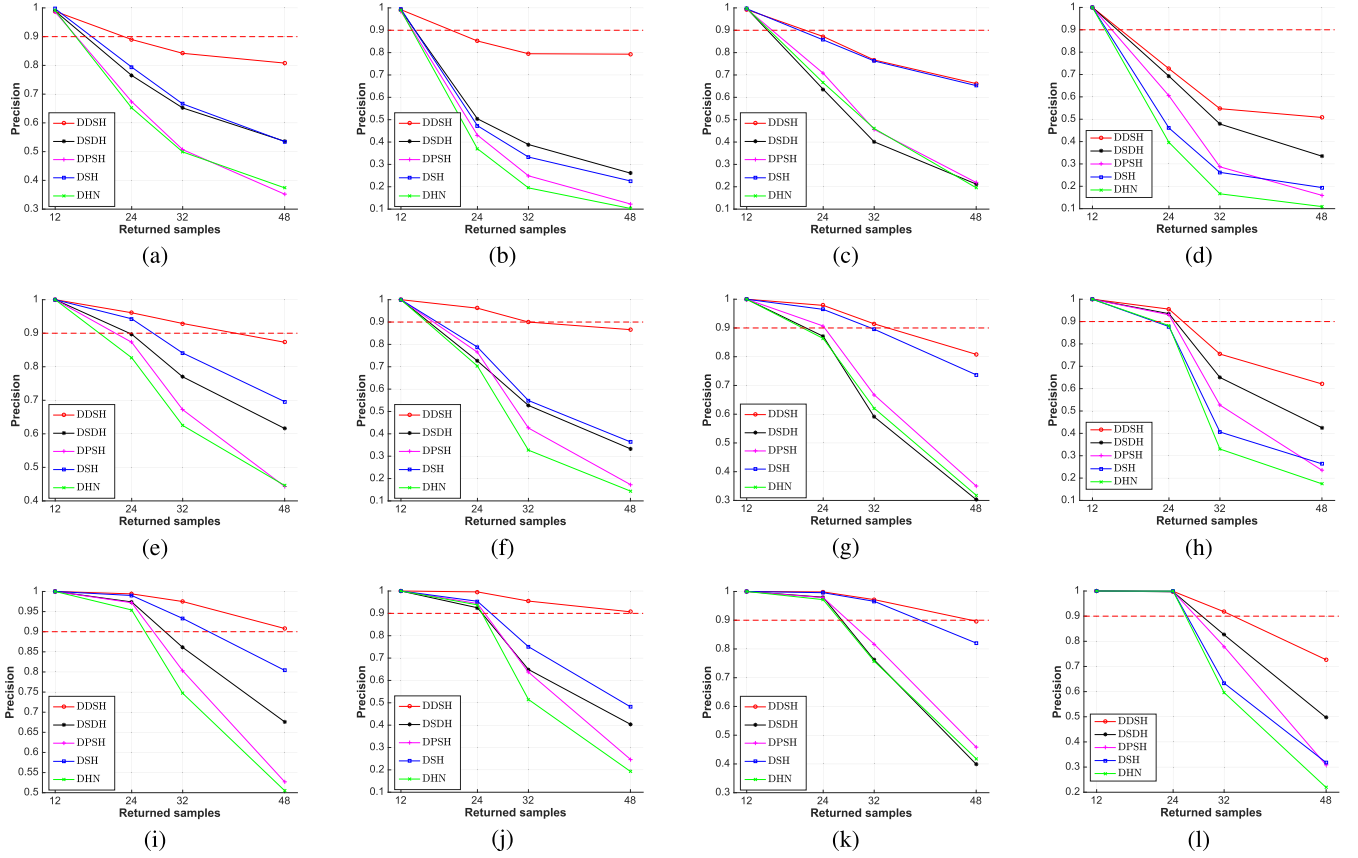


Fig. 4. Hash lookup success rate. Each row includes four sub-figures and presents the hash lookup success rate results on CIFAR-10, SVHN, NUS-WIDE and Clothing1M datasets, respectively. (a) Radius 0 @CIFAR-10. (b) Radius 0 @SVHN. (c) Radius 0 @NUS-WIDE. (d) Radius 0 @Clothing1M. (e) Radius 1 @CIFAR-10. (f) Radius 1 @SVHN. (g) Radius 1 @NUS-WIDE. (h) Radius 1 @Clothing1M. (i) Radius 2 @CIFAR-10. (j) Radius 2 @SVHN. (k) Radius 2 @NUS-WIDE. (l) Radius 2 @Clothing1M.

exactly the same as those in DDSH. Comparison between DDSH0 and DDSH is to show the importance of deep feature learning.

COSDISH-Linear denotes a variant of COSDISH in which we use linear function rather than boosted decision tree for out-of-sample extension. COSDISH-CNN denotes a variant of COSDISH in which we learn optimal binary codes using COSDISH first, and then we use the CNN-F to approximate the binary codes for out-of-sample extension. Because the discrete coding procedure in DDSH is similar to that in COSDISH, COSDISH-CNN can be considered as a two-stage variant of DDSH where the discrete coding stage is independent of the feature learning stage. The comparison between COSDISH-CNN and DDSH is to show that integrating the discrete coding procedure and deep feature learning procedure into the same framework is important.

DDSH-MAC is a variant of DDSH by using the method of auxiliary coordinates (MAC) technique in AFFHash [49]. That is to say, we use loss function $\mathcal{L}_{\text{COSDISH}}(\mathbf{B}^\Gamma, \mathbf{B}^\Omega) + \lambda \|\mathbf{B} - \tanh(F(\mathbf{X}; \Theta))\|_F^2$ to enhance the feedback between deep feature learning and discrete code learning. Here, $\mathcal{L}_{\text{COSDISH}}(\cdot)$ is the loss used in COSDISH. DDSH-MAC can integrate the discrete coding procedure and deep feature learning procedure into the same framework. However, the supervised information S_{ij} isn't directly included in the deep feature learning

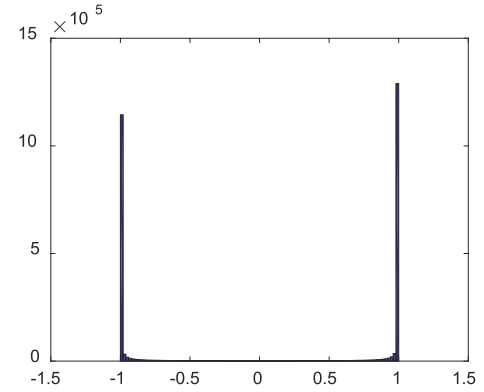


Fig. 5. The effect of $\tanh(\cdot)$ approximation on CIFAR-10.

term $\|\mathbf{B} - \tanh(F(\mathbf{X}; \Theta))\|_F^2$ in DDSH-MAC. That is to say, the supervised information is not directly used to guide the deep feature learning procedure.

The experimental results are shown in Table IX. By comparing DDSH to its variants including DDSH0, COSDISH-Linear, COSDISH-CNN and DDSH-MAC, we can find that DDSH can significantly outperform all the other variants. It means that utilizing supervised information to directly guide both discrete coding procedure and deep feature learning procedure

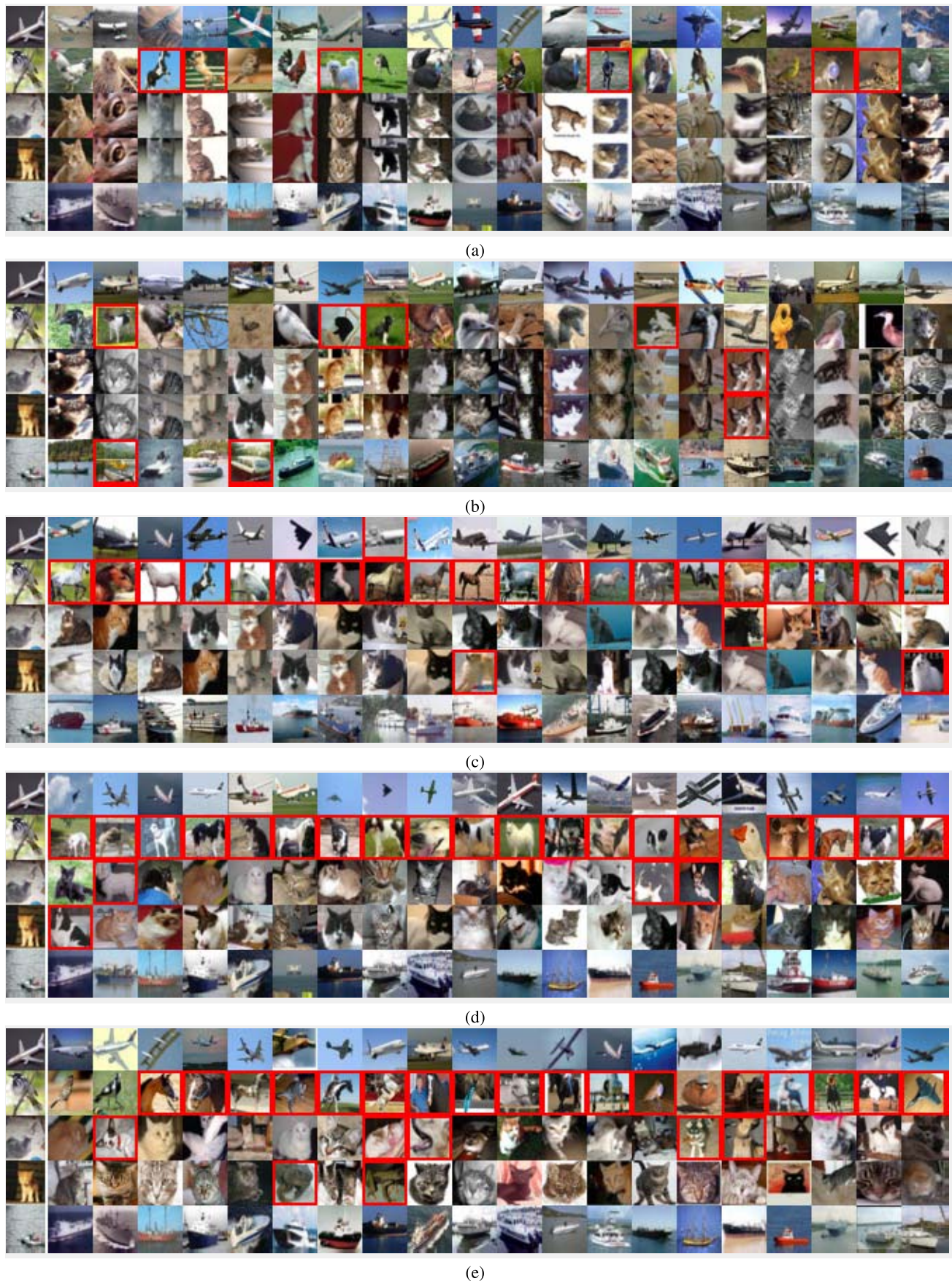


Fig. 6. Case study on CIFAR-10 with 32 bits. The first column for each sub-figure is queries and the following twenty columns denote the top-20 returned results. We use red box to denote the wrongly returned results. (a) DDSH @32 bits. (b) DSDH @32 bits. (c) DSH @32 bits. (d) DPSH @32 bits. (e) DHN @32 bits.

TABLE IX

MAP COMPARISON AMONG VARIANTS OF DDSH ON CIFAR-10.
THE BEST ACCURACY IS SHOWN IN BOLDFACE

Method	CIFAR-10			
	12 bits	24 bits	32 bits	48 bits
DDSH	0.7695	0.8289	0.8352	0.8194
DDSH0	0.5793	0.6387	0.6536	0.6800
COSDISH-Linear	0.2123	0.2345	0.2578	0.2723
COSDISH-CNN	0.3742	0.4773	0.4678	0.5153
DDSH-MAC	0.4121	0.5060	0.5276	0.5335

in the same end-to-end framework is the key to make DDSH achieve state-of-the-art retrieval performance.

Furthermore, to evaluate the approximation we used when we update the parameter of deep neural network, we report the distribution of the output for the deep neural network. Figure 5 shows the distribution of the output of $\tanh(F(\mathbf{X}; \Theta))$ when we finish the training procedure of DDSH on CIFAR-10 dataset. The x-axis is the $\tanh(F(\mathbf{X}; \Theta))$, and the y-axis is the number of points having the corresponding $\tanh(F(\cdot))$ value. It's easy to see that the $\tanh(\cdot)$ can successfully approximate the $\text{sign}(\cdot)$ function in real applications.

4) *Case Study*: We randomly sample some queries and return top-20 results for each query as a case study on CIFAR-10 to show the retrieval result intuitively. More specifically, for each given query image, we return top-20 nearest neighbors based on its Hamming distance away from query. Then we use red box to indicate the returned results that are not a ground-truth neighbor for the corresponding query image.

The result is shown in Figure 6. In each sub-figure, the first column is queries, including an airplane, a bird, two cats and a ship, and the following twenty columns denote the top-20 returned results. We utilize red box to denote the wrongly returned results. It's easy to find that DDSH can achieve better retrieval performance than other deep hashing baselines.

VI. CONCLUSION

In this paper, we propose a novel deep hashing method called deep discrete supervised hashing (DDSH) with application for image retrieval. On one hand, DDSH adopts a deep neural network to perform deep feature learning from pixels. On the other hand, DDSH also adopts a discrete coding procedure to perform discrete hash code learning. Moreover, DDSH integrates deep feature learning procedure and discrete coding procedure into the same architecture. DDSH is the first deep supervised hashing method which can utilize pairwise supervised information to directly guide both discrete coding procedure and deep feature learning procedure in the same end-to-end framework. Experiments on image retrieval applications show that DDSH can significantly outperform other baselines to achieve the state-of-the-art performance.

REFERENCES

- [1] A. Andoni, "Nearest neighbor search in high-dimensional spaces," in *Proc. Int. Symp. Math. Found. Comput. Sci.*, 2011, pp. 1–33.
- [2] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *Proc. IEEE Symp. Found. Comput. Sci.*, Oct. 2006, pp. 459–468.
- [3] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2010, pp. 18–25.
- [4] K. He, F. Wen, and J. Sun, "K-means hashing: An affinity-preserving quantization method for learning binary compact codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 2938–2945.
- [5] F. Shen, X. Zhou, Y. Yang, J. Song, H. T. Shen, and D. Tao, "A fast optimization method for general binary code learning," *IEEE Trans. Image Process.*, vol. 25, no. 12, pp. 5610–5621, 2016.
- [6] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua, "Discrete collaborative filtering," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2016, pp. 325–334.
- [7] J. Wang, H. T. Shen, J. Song, and J. Ji. (Aug. 2014). "Hashing for similarity search: A survey." [Online]. Available: <https://arxiv.org/abs/1408.2927>
- [8] J. Wang, W. Liu, S. Kumar, and S.-F. Chang, "Learning to hash for indexing big data—A survey," *Proc. IEEE*, vol. 104, no. 1, pp. 34–57, Jan. 2016.
- [9] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 769–790, Apr. 2018.
- [10] W. Kong and W.-J. Li, "Isotropic hashing," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1655–1663.
- [11] R. Zhang, L. Lin, R. Zhang, W. Zuo, and L. Zhang, "Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification," *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 4766–4779, Dec. 2015.
- [12] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, "Supervised hashing with kernels," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 2074–2081.
- [13] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. Int. Conf. Very Large Data Bases*, 1999, pp. 518–529.
- [14] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Proc. Int. Conf. Comput. Vis.*, Sep/Oct. 2009, pp. 2130–2137.
- [15] J. Wang, O. Kumar, and S.-F. Chang, "Semi-supervised hashing for scalable image retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2010, pp. 3424–3431.
- [16] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2012, pp. 2957–2964.
- [17] D. Ma, J. Liang, R. He, and X. Kong, "Nonlinear discrete cross-modal hashing for visual-textual data," in *Proc. IEEE MultiMedia*, vol. 24, no. 2, Apr. 2017, pp. 56–65.
- [18] Y. Guo, G. Ding, L. Liu, J. Han, and L. Shao, "Learning to hash with optimized anchor embedding for scalable retrieval," *IEEE Trans. Image Process.*, vol. 26, no. 3, pp. 1344–1354, Mar. 2017.
- [19] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. 20th ACM Symp. Comput. Geometry*, 2004, pp. 253–262.
- [20] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2011, pp. 817–824.
- [21] F. Shen, C. Shen, Q. Shi, A. van den Hengel, and Z. Tang, "Inductive hashing on manifolds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 1562–1569.
- [22] G. Lin, C. Shen, D. Suter, and A. van den Hengel, "A general two-step approach to learning-based hashing," in *Proc. Int. Conf. Comput. Vis.*, 2013, pp. 2552–2559.
- [23] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter, "Fast supervised hashing with decision trees for high-dimensional data," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1963–1970.
- [24] D. Song, W. Liu, R. Ji, D. A. Meyer, and J. R. Smith, "Top rank supervised binary coding for visual search," in *Proc. Int. Conf. Comput. Vis.*, 2015, pp. 1922–1930.
- [25] F. Shen, C. Shen, W. Liu, and H. T. Shen, "Supervised discrete hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 37–45.
- [26] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2008, pp. 1753–1760.
- [27] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proc. Int. Conf. Mach. Learn.*, 2011, pp. 1–8.
- [28] W. Liu, C. Mu, S. Kumar, and S.-F. Chang, "Discrete graph hashing," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3419–3427.

- [29] X. Lu, X. Zheng, and X. Li, "Latent semantic minimal hashing for image retrieval," *IEEE Trans. Image Process.*, vol. 26, no. 1, pp. 355–368, Jan. 2017.
- [30] D. Tian and D. Tao, "Global hashing system for fast image search," *IEEE Trans. Image Process.*, vol. 26, no. 1, pp. 79–89, Jan. 2017.
- [31] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content-based image retrieval at the end of the early years," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 12, pp. 1349–1380, Dec. 2000.
- [32] R. Salakhutdinov and G. Hinton, "Semantic hashing," *Int. J. Approx. Reasoning*, vol. 50, no. 7, pp. 969–978, Jul. 2009.
- [33] P. Zhang, W. Zhang, W.-J. Li, and M. Guo, "Supervised hashing with latent factor models," in *Proc. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2014, pp. 173–182.
- [34] W.-C. Kang, W.-J. Li, and Z.-H. Zhou, "Column sampling based discrete supervised hashing," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 1230–1236.
- [35] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 2475–2483.
- [36] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning," in *Proc. AAAI Conf. Artif. Intell.*, 2014, pp. 2156–2162.
- [37] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 3270–3278.
- [38] F. Zhao, Y. Huang, L. Wang, and T. Tan, "Deep semantic ranking based hashing for multi-label image retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1556–1564.
- [39] B. Zhuang, G. Lin, C. Shen, and I. D. Reid, "Fast training of triplet-based deep binary embedding networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 5955–5964.
- [40] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 2064–2072.
- [41] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen, "Deep quantization network for efficient image retrieval," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 3457–3463.
- [42] W.-J. Li, S. Wang, and W.-C. Kang, "Feature learning based deep supervised hashing with pairwise labels," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 1711–1717.
- [43] H. Zhu, M. Long, J. Wang, and Y. Cao, "Deep hashing network for efficient similarity retrieval," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2415–2421.
- [44] Q. Li, Z. Sun, R. He, and T. Tan, "Deep supervised discrete hashing," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2017, pp. 2479–2488.
- [45] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
- [46] Y. LeCun *et al.*, "Handwritten digit recognition with a back-propagation network," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 1989, pp. 396–404.
- [47] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *Proc. Brit. Mach. Vis. Conf.*, 2014, pp. 1–28.
- [48] R. Yang, "New results on some quadratic programming problems," Ph.D. dissertation, Dept. Ind. Enterprise Syst. Eng., Univ. Illinois Urbana-Champaign, Champaign, IL, USA, 2013.
- [49] R. Raziperchikolaie and M. Á. Carreira-Perpiñán. (Jan. 2015). "Optimizing affinity-based binary hashing using auxiliary coordinates." [Online]. Available: <https://arxiv.org/abs/1501.05352>
- [50] A. Vedaldi and K. Lenc, "MatConvNet: Convolutional neural networks for MATLAB," in *Proc. Annu. ACM Conf. Multimedia Conf.*, 2015, pp. 689–692.
- [51] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, p. 5.
- [52] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "NUS-WIDE: A real-world Web image database from National University of Singapore," in *Proc. ACM Int. Conf. Image Video Retr.*, 2009, Art. no. 48.
- [53] T. Xiao, T. Xia, Y. Yang, C. Huang, and X. Wang, "Learning from massive noisy labeled data for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 2691–2699.



Qing-Yuan Jiang received the B.Sc. degree in computer science from Nanjing University, China, in 2014, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Technology. His research interests are in machine learning and learning to hash.



Xue Cui received the B.Sc. degree in computer science and technology from Chongqing University, China, and the M.Eng. degree in computer science from Nanjing University, China. Her research interests mainly include machine learning and data mining.



Wu-Jun Li (M'09) received the B.Sc. and M.Eng. degrees in computer science from Nanjing University, China, and the Ph.D. degree in computer science from The Hong Kong University of Science and Technology. He started his academic career as an Assistant Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. He then joined Nanjing University, where he is currently an Associate Professor with the Department of Computer Science and Technology. His research interests are in machine learning, big data, and artificial intelligence.