International Journal of Computer Vision (2020) 128:2204–2222
https://doi.org/10.1007/s11263-020-01327-w

A General Framework for Deep Supervised Discrete Hashing

Qi Li[1,2,4] · Zhenan Sun[1,3,4] · Ran He[1,3,4] · Tieniu Tan[1,3,4]

Received: 2 January 2019 / Accepted: 30 March 2020 / Published online: 18 April 2020

Abstract
With the rapid growth of image and video data on the web, hashing has been extensively studied for image or video search in recent years. Benefiting from recent advances in deep learning, deep hashing methods have shown superior performance over the traditional hashing methods. However, there are some limitations of previous deep hashing methods (e.g., the semantic information is not fully exploited). In this paper, we develop a general deep supervised discrete hashing framework based on the assumption that the learned binary codes should be ideal for classification. Both the similarity information and the classification information are used to learn the hash codes within one stream framework. We constrain the outputs of the last layer to be binary codes directly, which is rarely investigated in deep hashing algorithms. Besides, both the pairwise similarity information and the triplet ranking information are exploited in this paper. In addition, two different loss functions are presented: l_2 loss and hinge loss, which are carefully designed for the classification term under the one stream framework. Because of the discrete nature of hash codes, an alternating minimization method is used to optimize the objective function. Experimental results have shown that our approach outperforms current state-of-the-art methods on benchmark datasets.

Keywords Supervised discrete hashing · l_2 loss · Hinge loss · Alternating minimization method

1 Introduction

Hashing has attracted much attention in recent years because of the rapid growth of image and video data on the web. It is one of the most popular techniques for image or video search due to its low computational cost and high storage efficiency. Generally speaking, hashing is used to encode high dimensional data into a set of binary codes while preserving the similarity of images or videos. Fast image or video retrieval can be carried out by calculating the Hamming distance between different samples, which dramatically reduces computational costs.

Many hashing methods have been proposed (Gionis et al. 1999; Wang et al. 2018; Weiss et al. 2009; Gong et al. 2013; Wang and Zhang 2019). Existing hashing methods can be roughly grouped into two categories: data independent methods and data dependent methods. Data independent methods rely on random projections to construct hash functions, while data dependent methods refer to using training data to learn hash functions. Data independent methods does not rely on data samples, thus it is usually less efficient and requires longer hashing codes to obtain high accuracy. Data dependent hashing methods can be further categorized into unsupervised and supervised methods. Unsupervised hashing methods try to explore the intrinsic structure of data samples without semantic labels. One of their limitations is that the semantic structure between different data samples are not fully exploited. Supervised methods can learn the

Communicated by Li Liu, Matti Pietikäinen, Jie Qin, Jie Chen, Wanli Ouyang, Luc Van Gool.

✉ Zhenan Sun
znsun@nlpr.ia.ac.cn

Qi Li
qli@nlpr.ia.ac.cn

Ran He
rhe@nlpr.ia.ac.cn

Tieniu Tan
tnt@nlpr.ia.ac.cn

[1] Center for Research on Intelligent Perception and Computing, National Laboratory of Pattern Recognition, CASIA, Beijing, China

[2] Artificial Intelligence Research, CAS, Qingdao, China

[3] Center for Excellence in Brain Science and Intelligence Technology, CAS, Beijing, China

[4] School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China

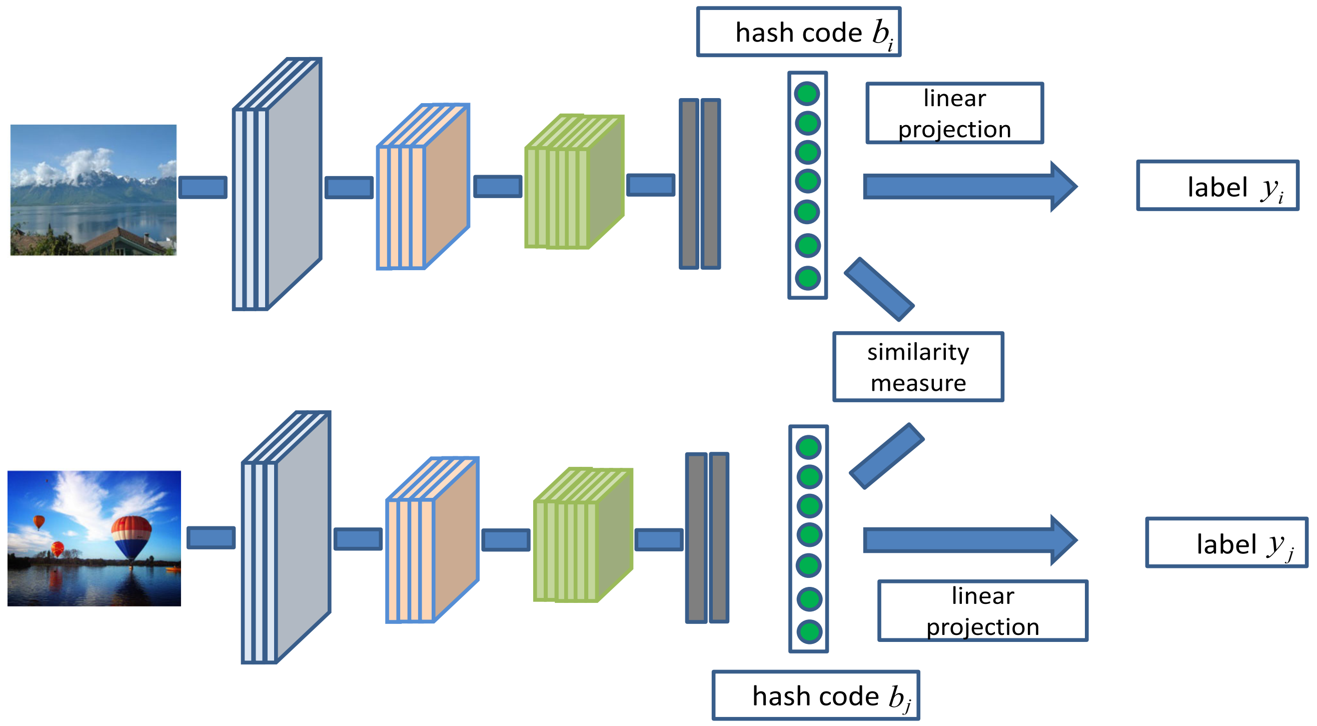

Fig. 1 The framework of our algorithm. It consists of two parts: the similarity part and the classification part. The similarity part is used to measure the similarity between different samples, and the classification part is used to measure the classification capability of the hash codes. These two parts are unified into one stream under our framework

hashing codes by incorporating semantic labels to mitigate the semantic gap and improve the retrieval performance.

Recently, Convolutional Neural Networks (CNNs) has achieved a big breakthrough in computer vision. They are capable of learning rich feature representations for image and text classification (Krizhevsky et al. 2012; Liu et al. 2019a), object detection (Girshick et al. 2014; Liu et al. 2020; Zhang et al. 2018a), segmentation or localization (Long et al. 2015; Li et al. 2016a; Wang et al. 2019), etc. Deep learning based hashing methods have also been proposed to simultaneously learn the image representation and hash coding, which have shown superior performance over the traditional hashing methods. Semantic hashing (Salakhutdinov and Hinton 2009) is one of the early works to learn the hash codes using deep neural networks. Xia et al. (2014) propose a two stage framework to learn hashing codes. It is one of the early works to incorporate deep neural networks into hash code learning, which consists of two stages to learn the image representations and hash codes respectively. The first stage is used to learn the fixed hash codes given the pairwise similarity matrix. Then the second stage is used to learn image representations and hash functions based on the learned hash codes. Lai et al. (2015) propose an end to end method for learning hashing codes, which uses triplet ranking loss to capture the relative similarities of different images. Zhao et al. (2015) also propose a deep semantic ranking method for learning hashing functions between multi-label images. Other ranking-based deep hashing methods and pairwise label based deep hashing methods have also been proposed in recent years (Yao et al. 2016; Li et al. 2016b; Zhu et al. 2016).

Although deep learning based methods have achieved great progress in image retrieval, there are still some ongoing issues for deep hashing methods. Particularly, current deep hashing methods try to divide the whole learning process into two streams under the multi-task learning framework (Lin et al. 2015; Yang et al. 2018; Yao et al. 2016). The hash stream is used to learn a hash function, while the classification stream is utilized to mine the semantic information. Although the two streams framework can improve the retrieval performance, the classification stream is only employed to learn the image representations, which does not have a direct impact on learning hash functions.

In this paper, we use CNNs to learn the image representation and hash function simultaneously. Different from previous deep hashing methods, one basic assumption of our algorithm is that the learned binary codes are ideal for classification as well. Based on this assumption, we develop a deep supervised discrete algorithm for hash coding. Both the similarity information and the classification information are incorporated into the objective function directly. Due to the discrete nature of binary codes, an alternating strategy is used to optimize the objective function. Figure 1 shows the framework of our algorithm. This paper is an extended version of Li et al. (2017b). Compared with the previous work, the differences are listed as follows. (1) In this paper, we propose a general deep supervised discrete hashing framework to learn hash codes. It mainly consists of two parts: the similarity part and classification part. Different similarity information and loss functions can be exploited under our framework. (2) The triplet ranking information and hinge loss are further exploited to measure the similarity information and the classification information under the proposed framework. The advantage o using the triplet ranking information and hinge loss is that our method can achieve more promising results on benchmark datasets when more training images are available. (3) More experimental results are presented to illustrate the effectiveness of our method. (4) Parameter sensitivity and efficiency analysis are also analyzed in this paper. The main contributions of our work can be summarized as follows.

1) We propose a general deep supervised discrete hashing framework to learn hash codes. Both the similarity information and classification information are exploited to learn hash codes under one stream framework. To the best of our knowledge, this is the first deep hashing method that uses both the similarity information and classification information to learn the hash codes under one stream framework.
2) The last layer of our method is constrained to output binary codes directly. The binary codes are learned to preserve the similarity relationship and keep the label consistent simultaneously.
3) In order to reduce the quantization error, we keep the discrete nature of hash codes during the optimization process. An alternating minimization method is proposed to optimize the objective function by using the discrete cyclic coordinate descend method.
4) Extensive experiments have shown that our method outperforms current state-of-the-art methods on three benchmark image retrieval datasets, which demonstrates the effectiveness of the proposed method.

The remainder of this paper is organized as follows. Section 2 briefly reviews some related works. Section 3 presents the details of the proposed algorithm. Experimental results are reported in Sect. 4. Finally, we give our conclusions in Sect. 5.

2 Related Work

Locality Sensitive Hashing (LSH) (Gionis et al. 1999) is one of the representative data independent hashing methods. It uses random linear projections to map nearby data in the original space into the similar binary codes. LSH is widely used for large scale image search or matching tasks. In order to generalize LSH to accommodate arbitrary kernel functions, Kenelized Locality Sensitive Hashing (KLSH) (Kulis and Grauman 2009; Raginsky and Lazebnik 2009) is proposed. It can be applied to high-dimensional kernelized data. The discriminative version of LSH is also proposed in Tomar and Rose (2013). Other variants of LSH are also proposed in recent years, such as super-bit LSH (Ji et al. 2012), non-metric LSH (Mu and Yan 2010), χ^2-LSH (Gorisse et al. 2012), ML-LSH (Kulis et al. 2009). However, data independent hashing methods suffer from a number of limitations, such as low learning efficiency caused by making no use of training data and undesirable longer hash codes with similar accuracy. Due to the limitations of the data independent hashing methods, many recent hashing methods try to exploit various machine learning techniques to learn more effective hashing function from a given dataset.

Unsupervised data dependent hashing methods try to retrieve the neighbors under some kind of distance metric. Iterative Quantization (ITQ) (Gong et al. 2013) is one of the representative unsupervised hashing methods, in which the projection matrix is optimized by iterative projection and thresholding according to the given training samples. In order to utilize the semantic labels of data samples, the supervised data dependent hashing methods are proposed. Supervised Hashing with Kernels (KSH) (Liu et al. 2012) is a well-known method of this kind, which learns the hash codes by minimizing the Hamming distances between similar pairs, and at the same time maximizing the Hamming distances between dissimilar pairs. Binary Reconstruction Embedding (BRE) (Kulis and Darrell 2009) learns the hash functions by explicitly minimizing the reconstruction error between the original distances and the reconstructed distances in Hamming space. Order Preserving Hashing (OPH) (Wang et al. 2013) learns the hash codes by preserving the supervised ranking list information, which is calculated based on the semantic labels. A new class of hash functions based on the ranking structure of feature spaces have been proposed in Li et al. (2017a)). Shen et al. (2015) propose Supervised Discrete Hashing (SDH) method based on the assumption that the learned binary codes should be optimal for linear classification. One contributions of their algorithm is that they use discrete cyclic coordinate descend method to directly optimize the binary hash codes, which proves to be more effective for learning the binary codes. Several following works try to further improve the accuracy and efficiency of SDH (Gui et al. 2016; Koutaki et al. 2016; Cui et al. 2018).

In the above methods, the input images are all represented by the hand-crafted features (e.g., GIST), which can not capture the semantic information of the images effectively. Inspired by recent advances in deep learning, some deep learning hashing methods have been proposed to take advantage of the superior power of the deep neural networks. Convolutional Neural Network Hashing (CNNH) (Xia et al. 2014) is one of the early works to incorporate deep neural networks into hash coding. One drawback of CNNH is that the learned image representation can not give feedback for learning better hash codes. To overcome the shortcomings of CNNH, many approaches have been proposed to perform feature learning and hash coding simultaneously. Network In Network Hashing (NINH) (Lai et al. 2015) presents a triplet ranking loss to capture the relative similarities of images. The image representation learning and hash coding can benefit each other within one stage framework. Deep Semantic Ranking Hashing (DSRH) (Zhao et al. 2015) learns the hash functions by preserving semantic similarity between multi-label images. Other ranking-based deep hashing methods have also been proposed in recent years (Yao et al. 2016; Wang et al. 2016).

Besides the triplet ranking based methods, some pairwise based deep hashing methods are also exploited (Li et al. 2016b; Zhu et al. 2016; Liu et al. 2016). A novel and efficient training algorithm inspired by alternating direction method of multipliers (ADMM) is proposed to train very deep neural networks for supervised hashing in Zhang et al. (2016). It first relaxes the binary constraint to be continuous, then thresholds the obtained continuous variables to be binary codes. Yu et al. (2018) propose a novel product quantization network, which mainly consists of a differentiable soft production quantization layer and an asymmetric triplet loss. Compared with previous methods, their method is immune to over-fitting and is more effective. A hash coding layer is proposed in Su et al. (2018) to maintain the discrete constraints of hash codes. It uses the sign function in forward propagation and transmits the gradients to front layers intactly. Liu et al. (2018) design a novel triplet section approach to randomly select hard triplets in the image group. Besides, a triplet quantization with weak orthogonality is is used during triplet ranking. A deep variational and structural hashing is proposed in Liong et al. (2018), which consists of a probabilistic framework to infer latent feature representation and a structure layer to obtain the hash codes through a simple encoding procedure. A simple and effective hashing method is proposed in Shen et al. (2018) to solve the unsupervised hash coding problem.

Generative Adversarial Networks are also widely used in hashing. A novel deep semantic hashing with Generative Adversarial Networks is presented in Qiu et al. (2017). Besides the hash stream and classification stream, an adversary stream is also used to distinguish synthetic images from real ones. A binary Generative Adversarial Networks (BGAN) is used to embed images to binary codes in an unsupervised way in Song (2018). It mainly consists of an adversarial loss, a content loss and a neighborhood structure loss to learn the hash codes. Due to the rapid development of deep learning, many other deep hashing methods have also been proposed in recent years (Liong et al. 2017; Zhao et al. 2017; Zhang et al. 2018b). Please refer to Wang et al. (2018) for a survey of different hashing methods.

3 Deep Supervised Discrete Hashing

3.1 Problem Definition

Given N image samples $X = \{x_i\}_{i=1}^{N} \in \mathbb{R}^{d\times N}$, hash coding learns a collection of K-bit binary codes $B \in \{-1, 1\}^{K\times N}$, where the i-th column $b_i \in \{-1, 1\}^K$ represents the binary codes for the i-th sample x_i. The binary codes are generated by the hash function $h(\cdot)$, which can be rewritten as $[h_1(\cdot), ..., h_K(\cdot)]$. For image sample x_i, its hash codes can be represented as $b_i = h(x_i) = [h_1(x_i), ..., h_K(x_i)]$. Generally speaking, hashing is to learn a hash function to project image samples to a set of binary codes.

3.2 Hash Coding as a Bayesian Process

In supervised hashing, the label information is given as $Y = \{y_i\}_{i=1}^{N} \in \mathbb{R}^{C\times N}$, where $y_i \in \{0, 1\}^C$ corresponds to the sample x_i, C is the number of categories, $y_{ic} = 1$ if the sample x_i comes from the c-th category and 0 otherwise. Note that one sample may belong to multiple categories. Given the semantic label information, the triplet label information or pairwise label information can be derived. In this paper, we mainly focus on the pairwise label information. However, it can be easily extended to the triplet label information. The pairwise label information can be represented as: $S = \{s_{ij}\}$, $s_{ij} \in \{0, 1\}$, where $s_{ij} = 1$ when x_i and x_j are semantically similar, $s_{ij} = 0$ when x_i and x_j are semantically dissimilar. For two binary codes b_i and b_j, the relationship between their Hamming distance $\text{dist}_H(\cdot, \cdot)$ and their inner product $\langle\cdot, \cdot\rangle$ is formulated as follows: $\text{dist}_H(b_i, b_j) = \frac{1}{2}(K - \langle b_i, b_j\rangle)$. If the inner product of two binary codes is small, their Hamming distance will be large, and vice versa. Therefore the inner product of different hash codes can be used to quantify their similarity.

Given the pairwise similarity relationship $S = \{s_{ij}\}$, the Maximum a Posterior estimation of hash codes can be represented as:

$$p(B|S) \propto p(S|B)\,p(B) = \prod_{s_{ij}\in S} p(s_{ij}|B)\,p(B) \quad (1)$$

where $p(S|B)$ denotes the likelihood function, $p(B)$ is the prior distribution. For each pair of the images, $p(s_{ij}|B)$ is the conditional probability of s_{ij} given their hash codes B,

Table 1 The details of the network structure parameters in our algorithm

Layer	Kernal	Stride	Padding
Conv1	$11 \times 11 \times 64$	4×4	0
Conv2	$5 \times 5 \times 256$	1×1	2
Conv3	$3 \times 3 \times 256$	1×1	1
Conv4	$3 \times 3 \times 256$	1×1	1
Conv5	$3 \times 3 \times 256$	1×1	1
Fc1	4096	–	–
Fc2	4096	–	–

It mainly consists of 5 convolutional layers and 2 fully connected layers

which is defined as follows:

$$p\left(s_{ij}|B\right) = \begin{cases} \sigma\left(\Phi_{ij}\right), & s_{ij}=1 \\ 1-\sigma\left(\Phi_{ij}\right), & s_{ij}=0 \end{cases} \tag{2}$$

where $\sigma(x) = 1/\left(1+e^{-x}\right)$ is the sigmoid function, $\Phi_{ij} = \frac{1}{2}\left\langle b_i, b_j\right\rangle = \frac{1}{2}b_i^T b_j$. From Eq. (2) we can see that, the larger the inner product $\left\langle b_i, b_j\right\rangle$ is, the larger $p\left(1|b_i, b_j\right)$ will be, which implies that b_i and b_j should be classified as similar, and vice versa. Therefore Eq. (2) is a reasonable similarity measure for hash codes.

3.3 Loss Function

Deep learning based methods have shown their superior performance over the traditional methods on object detection, image classification and image segmentation, etc. In this section, we take advantage of recent advances in CNNs, and use it for learning the hashing function. In order to have a fair comparison with existing deep hashing methods, we choose the CNN-F network architecture (Chatfield et al. 2014) as a basic component of our algorithm. This architecture is widely used for learning the hashing function in recent works (Li et al. 2016b; Wang et al. 2016). The main differences of various deep hashing methods are the design of loss functions and their optimization strategies, which play an important role for the final performance. Specifically, we have two separate CNNs for learning the hashing function, which share the same weights. The pairwise samples are used as the input for these two separate CNNs. The CNNs model consists of 5 convolutional layers and 2 fully connected layers. The details of the network architecture are shown in Table 1. Because the ReLU layer, the pooling layer and the local response normalization layer contain few learned parameters, we omit them for convenience. The number of neurons in the last fully connected layer is equal to the length of the output hash codes. Note that other networks, such as, AlexNet (Krizhevsky et al. 2012) and ResNet (He et al. 2016), can also be used in our algorithm.

Considering the pairwise similarity measure, the following loss function is used to learn the hash codes:

$$\begin{aligned} J &= -\log p\left(S|B\right) = -\sum_{s_{ij}\in S} \log\ p\left(s_{ij}|B\right) \\ &= -\sum_{s_{ij}\in S}\left(s_{ij}\Phi_{ij} - \log\left(1+e^{\Phi_{ij}}\right)\right). \end{aligned} \tag{3}$$

Equation (3) is the negative log likelihood function, which makes the Hamming distance of two similar points as small as possible, and at the same time makes the Hamming distance of two dissimilar points as large as possible. Although pairwise label information is used to learn the hash function in Eq. (3), the label information is not fully exploited. Most of the previous works make use of the label information under a two streams multi-task learning framework (Yang et al. 2018; Yao et al. 2016). The classification stream is used to measure the classification error, while the hash stream is employed to learn the hash function. One basic assumption of our algorithm is that the learned binary codes should be ideal for classification. In order to take advantage of the label information directly, we expect the learned binary codes to be optimal for the jointly learned linear classifier.

We use a simple linear classifier to model the relationship between the learned binary codes and the label information:

$$Y = W^T B, \tag{4}$$

where $W = \left[w_1, w_{2,\ldots,} w_C\right]$ is the classifier weight, $Y = \left[y_1, y_{2,\ldots,} y_N\right]$ is the ground-truth label vector. The loss function can be calculated as:

$$\begin{aligned} Q &= L\left(Y, W^T B\right) + \lambda \left\|W\right\|_F^2 \\ &= \sum_{i=1}^{N} L\left(y_i, W^T b_i\right) + \lambda \left\|W\right\|_F^2, \end{aligned} \tag{5}$$

where $L(\cdot)$ is the loss function, λ is the regularization parameter, $\left\|\cdot\right\|_F$ is the Frobenius norm of a matrix. Combining Eqs. (3) and (5), we have the following formulation to learn hash codes:

$$\begin{aligned} F &= J + \mu Q \\ &= -\sum_{s_{ij}\in S}\left(s_{ij}\Phi_{ij} - \log\left(1+e^{\Phi_{ij}}\right)\right) \\ &\quad + \mu \sum_{i=1}^{N} L\left(y_i, W^T b_i\right) + \nu \left\|W\right\|_F^2, \end{aligned} \tag{6}$$

where μ is the trade-off parameters, $\nu = \lambda\mu$.

3.4 Optimization with l_2 Loss

Suppose that we choose l_2 loss for the linear classifier, Eq. (6) can be rewritten as follows:

$$F = -\sum_{s_{ij}\in S}\left(s_{ij}\Phi_{ij} - \log\left(1+e^{\Phi_{ij}}\right)\right) + \mu\sum_{i=1}^{N}\left\|y_i - W^T b_i\right\|_2^2 + \nu\|W\|_F^2, \tag{7}$$

where $\|\cdot\|_2$ is the l_2 norm of a vector. The hypothesis for Eq. (7) is that the learned binary codes should make the pairwise label likelihood as large as possible, and at the same time should be optimal for the jointly learned linear classifier.

The minimization of Eq. (7) is a discrete optimization problem, which is difficult to optimize directly. There are several ways to solve this problem. (1) In the training stage, the sigmoid or tanh activation function is utilized to replace the ReLU function after the last fully connected layer, and then the continuous outputs are used as a relaxation of the hash codes. In the testing stage, the hash codes are obtained by applying a thresholding function on the continuous outputs. One limitation of this method is that the convergence of the algorithm is slow. Besides, there will be a large quantization error. (2) The sign function is directly applied after the outputs of the last fully connected layer, which constrains the outputs to be binary variables strictly. However, the sign function is non-differentiable, which is difficult to back propagate the gradient of the loss function.

Because of the discrepancy between the Euclidean space and the Hamming space, it would result in suboptimal hashing codes if one totally ignores the binary constraints. Similar to Shen et al. (2015), we emphasize that it is essential to keep the discrete nature of binary codes. Note that in our formulation, we constrain the outputs of the last layer of the CNN to be binary codes directly, thus Eq. (7) is difficult to optimize directly. Similar to Yao et al. (2016), Wang et al. (2016) and Li et al. (2016b), we solve this problem by introducing an auxiliary variable. Then we approximate Eq. (7) as:

$$\begin{aligned} F = &-\sum_{s_{ij}\in S}\left(s_{ij}\Psi_{ij} - \log\left(1+e^{\Psi_{ij}}\right)\right) \\ &+\mu\sum_{i=1}^{N}\left\|y_i - W^T b_i\right\|_2^2 + \nu\|W\|_F^2 \\ &s.t.\ \ b_i = \mathrm{sgn}(h_i)_i,\quad h_i\in\mathbb{R}^{K\times 1},\quad (i=1,\ldots,N) \end{aligned} \tag{8}$$

where $\Psi_{ij} = \frac{1}{2}{h_i}^T h_j$. h_i $(i=1,\ldots,N)$ can be seen as the output of the last fully connected layer, which is represented as:

$$h_i = M^T\Theta(x_i;\theta) + n \tag{9}$$

where θ denotes the parameters of the previous layers before the last fully connected layer, $\Theta(x_i;\theta)$ is the output of the fully connected layer with respect to the sample x_i, $M\in R^{4096\times K}$ represents the weight matrix, $n\in R^{K\times 1}$ is the bias term.

According to the Lagrange multipliers method, Eq. (8) can be reformulated as:

$$\begin{aligned} F = &-\sum_{s_{ij}\in S}\left(s_{ij}\Psi_{ij} - \log\left(1+e^{\Psi_{ij}}\right)\right) \\ &+\mu\sum_{i=1}^{N}\left\|y_i - W^T b_i\right\|_2^2 + \nu\|W\|_F^2 + \eta\sum_{i=1}^{N}\|b_i - \mathrm{sgn}(h_i)\|_2^2 \\ &s.t.\quad b_i\in\{-1,1\}^K,\quad (i=1,\ldots,N) \end{aligned} \tag{10}$$

where η is the Lagrange Multiplier. Equation (10) can be further relaxed as:

$$\begin{aligned} F = &-\sum_{s_{ij}\in S}\left(s_{ij}\Psi_{ij} - \log\left(1+e^{\Psi_{ij}}\right)\right) \\ &+\mu\sum_{i=1}^{N}\left\|y_i - W^T b_i\right\|_2^2 + \nu\|W\|_F^2 + \eta\sum_{i=1}^{N}\|b_i - h_i\|_2^2 \\ &s.t.\quad b_i\in\{-1,1\}^K,\quad (i=1,\ldots,N) \end{aligned} \tag{11}$$

The last term actually measures the constraint violation caused by the outputs of the last fully connected layer. If the parameter η is set sufficiently large, the constraint violation is penalized severely. Therefore the outputs of the last fully connected layer are forced closer to the binary codes, which are employed for classification directly. The benefit of introducing auxiliary variable is that we can decompose Eq. (11) into three sub optimization problems, which can be iteratively solved by the alternating optimization method.

First, when fixing b_i, W, we have:

$$\begin{aligned} \frac{\partial F}{\partial h_i} = &-\frac{1}{2}\sum_{j:s_{ij}\in S}\left(s_{ij} - \log\left(\frac{e^{\Psi_{ij}}}{1+e^{\Psi_{ij}}}\right)\right)h_j \\ &-\frac{1}{2}\sum_{j:s_{ji}\in S}\left(s_{ji} - \log\left(\frac{e^{\Psi_{ji}}}{1+e^{\Psi_{ji}}}\right)\right)h_j \\ &-2\eta(b_i - h_i) \end{aligned} \tag{12}$$

Then we update parameters M, n and Θ as follows:

$$\frac{\partial F}{\partial M} = \Theta(x_i;\theta)\left(\frac{\partial F}{\partial h_i}\right)^T$$

$$\begin{aligned}\frac{\partial F}{\partial n} &= \frac{\partial F}{\partial h_i}\\ \frac{\partial F}{\partial \Theta\left(x_i;\theta\right)} &= M\frac{\partial F}{\partial h_i}\end{aligned} \tag{13}$$

The gradient will propagate to previous layers by Back Propagation (BP) algorithm (Rumelhart et al. 1988).

Second, when fixing M, n, Θ and b_i, we solve W as:

$$F = \mu \sum_{i=1}^{N} \left\| y_i - W^T b_i \right\|_2^2 + \nu \left\| W \right\|_F^2 \tag{14}$$

Equation (14) is a least squares problem, which has a closed form solution:

$$W = \left(BB^T + \frac{\nu}{\mu} I \right)^{-1} BY^T \tag{15}$$

where $B = \{b_i\}_{i=1}^N \in \{-1,1\}^{K\times N}$, $Y = \{y_i\}_{i=1}^N \in \mathbb{R}^{C\times N}$

Finally, when fixing M, n, Θ and W, Eq. (11) becomes:

$$\begin{aligned} F &= \mu \sum_{i=1}^{N} \left\| y_i - W^T b_i \right\|_2^2 + \eta \sum_{i=1}^{N} \left\| b_i - h_i \right\|_2^2\\ s.t.\quad & b_i \in \{-1,1\}^K, (i = 1, \ldots, N) \end{aligned} \tag{16}$$

In this paper, we use discrete cyclic coordinate descend method to iteratively solve B row by row:

$$\begin{aligned} &\min_B \ \left\| W^T B \right\|^2 - 2\,\mathrm{Tr}\left(B^T P\right)\\ &s.t.\quad B \in \{-1,1\}^{K\times N} \end{aligned} \tag{17}$$

where $P = WY + \frac{\eta}{\mu} H$, $H = \{h_i\}_{i=1}^N \in \mathbb{R}^{K\times N}$. Let g_k^T be the k^{th} row of B, $k = 1, \ldots, K$, B_1 be the matrix of B excluding g_k, p^T be the k^{th} row of matrix P, P_1 be the matrix of P excluding p, w^T be the k^{th} row of matrix W, W_1 be the matrix of W excluding w, then

$$g_k = \mathrm{sgn}\left(p - B_1^T W_1 w \right) \tag{18}$$

It is easy to see that each bit of the hash codes is computed based on the pre-learned $K-1$ bits B_1. We can iteratively update each bit until the algorithm converges.

3.5 Optimization with Hinge Loss

Similar to Shen et al. (2015), we can also choose hinge loss for the linear classifier. Compared with l_2 loss, hinge loss tries to separate different data samples from a maximum margin view. If we choose hinge loss for the linear classifier, Eq. (6) can be rewritten as follows:

$$\begin{aligned} F = &-\sum_{s_{ij}\in S} \left(s_{ij}\Phi_{ij} - \log\left(1+e^{\Phi_{ij}}\right)\right) + \mu \sum_{i=1}^{N} \xi_i + \nu \left\| W \right\|_F^2\\ s.t.\quad & \forall i, c \quad w_{l_i}^T b_i + y_{ic} - w_c^T b_i \geq 1 - \xi_i,\\ & b_i \in \{-1,1\}^K, \quad i = 1, \ldots, N, \quad c = 1, \ldots, C \end{aligned} \tag{19}$$

where l_i is the category label of x_i, ξ_i is the slack variable. Equation (19) makes the pairwise label likelihood as large as possible, and at the same time learns the linear classifier that can separate data samples from a maximum margin view.

Similar to Sect. 3.4, we introduce an auxiliary variable and then Eq. (19) can be approximated as:

$$\begin{aligned} F = &-\sum_{s_{ij}\in S} \left(s_{ij}\Psi_{ij} - \log\left(1+e^{\Psi_{ij}}\right)\right) + \mu \sum_{i=1}^{N} \xi_i + \nu \left\| W \right\|_F^2\\ s.t.\quad & \forall i, c \quad w_{l_i}^T b_i + y_{ic} - w_c^T b_i \geq 1 - \xi_i,\\ & b_i = \mathrm{sgn}\left(h_i\right),\ h_i \in \mathbb{R}^{K\times 1},\ i = 1, \ldots, N,\ c = 1, \ldots, C \end{aligned} \tag{20}$$

According to the Lagrange multipliers method, Eq. (20) can be relaxed as:

$$\begin{aligned} F = &-\sum_{s_{ij}\in S} \left(s_{ij}\Psi_{ij} - \log\left(1+e^{\Psi_{ij}}\right)\right) + \mu \sum_{i=1}^{N} \xi_i\\ &+ \nu \left\| W \right\|_F^2 + \eta \sum_{i=1}^{N} \left\| b_i - h_i \right\|_2^2\\ s.t.\quad & \forall i, c \quad w_{l_i}^T b_i + y_{ic} - w_c^T b_i \geq 1 - \xi_i,\\ & b_i \in \{-1,1\}^K, \quad i = 1, \ldots, N, \quad c = 1, \ldots, C \end{aligned} \tag{21}$$

There are totally three unknown parts in Eq. (21), which can be divided into three sub optimization problems and iteratively solved using the alternating optimization method.

First, when fixing b_i, W, we have:

$$\begin{aligned} \frac{\partial F}{\partial h_i} = &-\frac{1}{2} \sum_{j:s_{ij}\in S} \left(s_{ij} - \log\left(\frac{e^{\Psi_{ij}}}{1+e^{\Psi_{ij}}} \right)\right) h_j\\ &-\frac{1}{2} \sum_{j:s_{ji}\in S} \left(s_{ji} - \log\left(\frac{e^{\Psi_{ji}}}{1+e^{\Psi_{ji}}} \right)\right) h_j\\ &-2\eta\left(b_i - h_i\right) \end{aligned} \tag{22}$$

The parameters M, n and Θ can be updated as follows:

$$\begin{aligned}\frac{\partial F}{\partial M} &= \Theta\left(x_i;\theta\right)\left(\frac{\partial F}{\partial h_i}\right)^T\\ \frac{\partial F}{\partial n} &= \frac{\partial F}{\partial h_i}\\ \frac{\partial F}{\partial \Theta\left(x_i;\theta\right)} &= M\frac{\partial F}{\partial h_i}\end{aligned} \tag{23}$$

Second, when fixing M, n, Θ and b_i, Eq. (21) becomes:

$$F = \mu \sum_{i=1}^{N} \xi_i + \nu \left\| W \right\|_F^2 \quad s.t. \quad \forall i, c \quad w_{l_i}^T b_i + y_{ic} - w_c^T b_i \geq 1 - \xi_i, \quad i = 1, ..., N, \; c = 1, ..., C \tag{24}$$

Equation (24) is actually a multi-class Support Vector Machine problem, which can be efficiently solved by many algorithms (Hsu and Lin 2002).

Finally, when fixing M, n, Θ and W, Eq. (21) becomes:

$$F = \eta \sum_{i=1}^{N} \left\| b_i - h_i \right\|_2^2 \quad s.t. \quad \forall i, c \quad w_{l_i}^T b_i + y_{ic} - w_c^T b_i \geq 1 - \xi_i, \quad b_i \in \{-1, 1\}^K, \; i = 1, ..., N, \; c = 1, ..., C \tag{25}$$

According to the Lagrange multipliers method, Eqn (25) can be rewritten as:

$$F = \eta \sum_{i=1}^{N} \left\| b_i - h_i \right\|_2^2 - \gamma \sum_{i=1}^{N} \sum_{c=1}^{C} \left(w_{l_i}^T b_i + y_{ic} - w_c^T b_i - 1 + \xi_i, \right) \quad s.t. \quad b_i \in \{-1, 1\}^K \tag{26}$$

where γ is the regularization parameter. Minimizing Eq. (26) is equivalent to maximizing the following equation:

$$\sum_{i=1}^{N} \left(b_i^T \left(h_i + \frac{\gamma}{2\eta} \sum_{c=1}^{C} \left(w_{l_i} - w_c \right) \right) \right) \quad s.t. \quad b_i \in \{-1, 1\}^K \tag{27}$$

From Eq. (27), we can easily obtain the final solution for b_i as:

$$b_i = \operatorname{sgn} \left(h_i + \frac{\gamma}{2\eta} \sum_{c=1}^{C} \left(w_{l_i} - w_c \right) \right) \tag{28}$$

The whole algorithm is summarized in Algorithm 1.

Algorithm 1 Deep Supervised Discrete Hashing (DSDH).

```
Input:
    The set of training images $X \in \mathbb{R}^{d \times N}$, and their training labels
    $Y \in \mathbb{R}^{C \times N}$;
    Hash code length $K$;
    Total epoch number $T$;
    Batch size $M$;
1: for $epoch = 1, 2, ..., T$ do
2:     Randomly permute training images $X$;
3:     for $iter = 1 : round(N/M) - 1$ do
4:         Sample $M$ data points sequentially from $X$ to construct a mini-
           batch;
5:         Calculate the gradient according to Eqs. (12) or (22), and then
           update the network parameters according to Eqs. (13) or (23);
6:         Update the classifier weight $W$ according to Eqs. (15) or (24);
7:         Update the hash code $B$ according to Eqs. (18) or (28);
8:     end for
9: end for
Output:
    Parameters of the deep convolution neural network.
```

3.6 Relation to Previous Works

Our method is inspired by recent advances in deep hashing methods (Xia et al. 2014; Lai et al. 2015; Li et al. 2016b; Wang et al. 2016) and discrete hashing methods (Kang et al. 2016; Shen et al. 2015; Gui et al. 2016). Among the above methods, DPSH (Li et al. 2016b) and SDH (Shen et al. 2015) are the most similar ones. It is worthwhile to mention the difference between these methods and our method.

First, we will discuss the difference between SDH and our method. SDH also insists that the optimal hash codes should be ideal for classification. However, the pairwise label information and deep learned features are not exploited in SDH. There is also a simple variant of SDH, whose inputs are the learned features extracted by deep CNN. One drawback of the simple combination of deep CNN and SDH is that it contains two separate learning processes. Compared with the straightforward combination of deep CNN and SDH, our method is an end to end learning framework. The learned hashing codes can be used to guide the learning of image representations, and the learned image representations can give feedback for learning better hash codes. Experimental results have also shown that our method outperforms the simple combination of deep CNN and SDH by a large margin.

Our work also has some relationships with DPSH. One major difference is that the learned binary codes are ideal for classification in our method. Thus a linear classifier is exploited to guide both the feature learning and hash coding procedure. Besides, we constrain the outputs of the last layer to be binary codes, which aims to reduce the quantization error directly. Note that our method is not restricted to the pairwise label information. Other constraints, such as the triplet ranking information, can also be incorporated into our method easily. In the following, we will briefly illustrate how to utilize the triplet ranking information to generate hash codes under our framework.

Given N training images $X = \{x_i\}_{i=1}^N$ with U triplet labels $T = \left\{ \left(x_u^a, x_u^p, x_u^n \right) | \; u = 1, 2, ..., U \right\}$, where x_u^a denotes the anchor image, x_u^p denotes the positive image and x_u^n denotes the negative image. Suppose that the inner product of hash codes is $\Phi_{ij} = \frac{1}{2} b_i^T b_j$, similar to Eq. (2), we can define the conditional probability $p\left(\left(x_u^a, x_u^p, x_u^n \right) | B \right)$ as: $p\left(\left(x_u^a, x_u^p, x_u^n \right) | B \right) = \sigma \left(\Phi_{x_u^a x_u^p} - \Phi_{x_u^a x_u^n} - \alpha \right)$, where $\sigma(x) = 1 / \left(1 + e^{-x} \right)$ is the sigmoid function. Then the likelihood function can be summarized as:

$$p(T|B) = \sum_{u=1}^{U} p\left(\left(x_u^a, x_u^p, x_u^n\right)|B\right) = \sum_{u=1}^{U} \sigma\left(\Phi_{x_u^a x_u^p} - \Phi_{x_u^a x_u^n} - \alpha\right) \tag{29}$$

The final objective function becomes:

$$\begin{aligned} F &= J + \mu Q \\ &= -\log\ p(T|B) + \mu L\left(Y, W^T B\right) + \nu \|W\|_F^2 \\ &= -\sum_{u=1}^{U}\left(\Phi_{x_u^a x_u^p} - \Phi_{x_u^a x_u^n} - \alpha - \log\left(1 + e^{\Phi_{x_u^a x_u^p} - \Phi_{x_u^a x_u^n} - \alpha}\right)\right) \\ &\quad + \mu \sum_{i=1}^{N} L\left(y_i, W^T b_i\right) + \nu \|W\|_F^2 \end{aligned} \tag{30}$$

The optimization process of Eq. (30) is similar to Sects. 3.4 and 3.5.

4 Experiments

4.1 Datasets and Experimental Settings

We conduct extensive experiments on the following public benchmark datasets.

- *CIFAR-10* CIFAR-10 is a dataset containing 60,000 color images in 10 classes, and each class contains 6000 images with a resolution of 32 × 32. The entire dataset is partitioned into a training set with 50,000 images and a test set with 10,000 images. We follow the previous experimental setting in Lai et al. (2015), Li et al. (2016b) and Wang et al. (2016). 100 images per class (1000 images in total) are randomly selected as the test query set. For unsupervised hashing methods, the rest of all images are used as the training set. For supervised hashing methods, 500 images per class are chosen (5000 images in total) as the training set. CIFAR-10 is one of the most popular datasets for evaluating different hashing methods.
- *NUS-WIDE* Different from CIFAR-10, NUS-WIDE is a public multi-label image dataset. There are 269,648 color images in total with 5018 unique tags. Each image is annotated with one or multiple class labels from the 5018 tags. Similar to Lai et al. (2015), Liu et al. (2011), Xia et al. (2014) and Zhang et al. (2015), we use a subset of 195,834 images which are associated with the 21 most frequent concepts. Each concept consists of at least 5000 color images in this dataset. For NUS-WIDE dataset, we randomly sample 100 images per class (2100 images in total) as the test query set. The rest of all images are used as the training set for unsupervised hashing methods. While 500 images per class (10,500 images in total) are further selected as the training set for supervised hashing methods.
- *MIRFLICKR* MIRFLICKR consists of 25,000 images downloaded from the social photography site Flickr, where each image is labeled with at least one of the 38 semantic concepts. Following previous experimental settings (Zhu et al. 2016; Zhang and Peng 2017; Cao et al. 2016), 1000 images are randomly selected as the query set. The rest of all images are used as the training set for unsupervised hashing methods. While we randomly select 5000 images from the rest of the images to form the training set for supervised hashing methods.

Deep hashing methods usually need many training images to learn the hash function. Thus we also conduct experiments on CIFAR-10 and NUS-WIDE dataset under a different experimental setting. In CIFAR-10, 1000 images per class (10,000 images in total) are selected as the test query set, the remaining 50,000 images are used as the training set. In NUS-WIDE, 100 images per class (2100 images in total) are randomly sampled as the test query images, the remaining images (193,734 images in total) are used as the training set. The similar pairs are constructed according to the image labels: two images will be considered similar if they share at least one common semantic label. Otherwise, they will be considered dissimilar.

As for the comparison methods, we roughly divide them into two groups: traditional hashing methods and deep hashing methods. The compared traditional hashing methods consist of unsupervised and supervised methods. Unsupervised hashing methods include SH (Weiss et al. 2009), ITQ (Gong et al. 2013). Supervised hashing methods include SPLH (Wang et al. 2010), KSH (Liu et al. 2012), FastH (Lin et al. 2014), LFH (Zhang et al. 2014), and SDH (Shen et al. 2015). The deep hashing methods include CNNH (Xia et al. 2014), NINH (Lai et al. 2015), DSRH (Zhao et al. 2015), SSDH (Zhang and Peng 2017), DRCSH (Zhang et al. 2015), DSCH (Zhang et al. 2015), DHN (Zhu et al. 2016), DPSH (Li et al. 2016b), DTSH (Wang et al. 2016). These methods are briefly introduced as follows.

- *SH* (Weiss et al. 2009) SH is a data dependent unsupervised hashing method, which learns hash codes by thresholding a subset of eigenvectors of the Laplacian of the similarity graph.
- *ITQ* (Weiss et al. 2009) ITQ is also a data dependent unsupervised hashing method. It first projects the data samples into a low dimensional space by using princi-

ple component analysis. Then the quantization error is minimized to learn hash codes.

- *SPLH* (Wang et al. 2010) SPLH uses linear projection coupled with mean thresholding as a hash function. It learns the projection matrix to correct the errors made by previous one sequentially.
- *LSH* (Gionis et al. 1999) The basic idea of LSH is to compute randomized hash functions that guarantee a high probability of collision for similar examples.
- *KSH* (Liu et al. 2012) KSH is a kernel-based supervised hashing method, which maps the data samples to compact hash codes by minimizing the Hamming distances between similar pairs and maximizing the Hamming distances between dissimilar pairs at the same time.
- *FastH* (Lin et al. 2014) A two step learning strategy is employed in FastH, which decomposes the learning process into the binary code inference and the simple binary classification training of decision trees.
- *LFH* (Zhang et al. 2014)LFH learns similarity-preserving binary codes based on latent factor models. A linear time variant with stochastic learning is proposed for training LFH.
- *SDH* (Shen et al. 2015) SDH is a supervised hashing method, which leverages class label information to learn discrete hash codes.
- *CNNH* (Xia et al. 2014) CNNH is a two stage deep hashing methods. It learns hash codes in the first stage, and then train the deep neural networks in the second stage.
- *NINH* (Lai et al. 2015) NINH learns bitwise hash codes for images via a carefully designed one stage deep neural network. Triplet loss is designed to measure the ranking information and preserve relative similarities.
- *SSDH* (Zhao et al. 2015) A deep neural network with online graph construction strategy is used to make full use of unlabeled data. Both the semi-supervised loss function and the supervised loss function are used to minimize the embedding error on the labeled and unlabeled dataset.
- *DRSCH* (Zhao et al. 2015) DRSCH is also a triplet based deep hashing method, which utilizes triplet loss function to maximize the margin between matched pairs and mismatched pairs in the Hamming space. A Laplacian regularization term is also introduced to enforce the adjacency consistency.
- *DSRH* (Zhao et al. 2015) DSRH is one simplified variant of DRSCH by removing the Laplacian regularization term.
- *DHN* (Zhu et al. 2016) DHN learns hash codes in a Bayesian learning framework, which simultaneously optimizes the pairwise cosine loss on semantic similarity pairs and the product quantization loss on compact hash codes.
- *DPSH* (Li et al. 2016b) DPSH simultaneously learns hash codes and image feature with the pairwise labels. It is also an end-to-end hashing memthod.
- *DTSH* (Wang et al. 2016) DTSH utilizes the triplet labels to simultaneously learn hash codes and image features. It directly learns hash codes by maximizing the likelihood of the given triplet labels.

In order to have a fair comparison with traditional hashing methods, both the hand-crafted features and the features extracted by CNN-F network architecture are used as the input for traditional hashing methods. Similar to previous works, the handcrafted features include a 512-dimensional GIST descriptor to represent images of CIFAR-10 dataset and MIRFLICKR dataset, and a 1134-dimensional feature vector to represent images of NUS-WIDE dataset.

As for deep hashing methods, the images are resized to be a resolution of 224×224 firstly. Then the raw image pixels are used as the input for different hashing methods. Note that DPSH, DTSH and DSDH are based on the CNN-F network architecture, while DQN, DHN, DSRH are based on AlexNet architecture. Both the CNN-F network architecture and AlexNet architecture consist of five convolutional layers and two fully connected layers. In order to have a fair comparison, most of the results are directly reported from previous works (Li et al. 2016b; Zhang and Peng 2017; Zhu et al. 2016; Cao et al. 2016), and for the other methods, we re-run the codes provided by the authors, and try our best to tune the parameters. The parameters of our algorithm are set based on the standard cross-validation procedure. μ, ν and η in Eq. (11) are set to 1, 0.1 and 55, respectively. The mini-batch size is fixed to be 128, and the learning rate is tuned from 10^{-2} to 10^{-6} with a step size of 20.

Similar to Lai et al. (2015), we adopt the widely used evaluation metrics to evaluate the image retrieval quality: Mean Average Precision (MAP) for different number of bits, precision curves within Hamming distance 2, precision curves with different number of top returned samples, precision within top 500 retrieved samples and precision-recall curves. These evaluation metrics are defined as follows.

- *Mean Average Precision (MAP)* MAP is an indicator of the overall performance of hash functions. Specifically, given a query point q, its average precision (AP) can be calculated as:

$$\mathrm{AP} = \frac{1}{l}\sum_{r=1}^{R} p_q(r)\,\delta_q(r) \tag{31}$$

where l is the number of ground-truth neighbors of the query sample, $p_q(r)$ denotes the precision at cutoff r for the ranking list, $\delta_q(r)$ indicates whether the r-th data item is relevant to the query sample. Then given Q query samples, the Mean Average Precision is defined as:

$$\text{MAP} = \frac{1}{Q}\sum_{i=1}^{Q} \text{AP}(q_i) \tag{32}$$

where Q is the number of query samples.

– *Precision curves with Hamming distance 2* It is computed as the precision curve of returned images in the buckets that fall within the Hamming radius 2 of the query image.
– *Precision curves with different number of top returned samples* The percentage of true neighbors among the top k retrieved images. This metric is averaged over all queries.
– *precision within top 500 retrieved samples* The average precision of the top 500 returned images for each query.
– *Precision recall curves* The precisions at certain levels of recall. It is a good indicator of the overall performance of different algorithms.

Since NUS-WIDE dataset contains a large amount of images, when computing MAP for NUS-WIDE dataset under the first experimental setting, we only consider the top 5000 returned neighbors. While we consider the top 50,000 returned neighbors under the second experimental setting for this dataset.

4.2 Empirical Analysis

In order to verify the effectiveness of our method, several variants of our method are also proposed. First, we only consider the pairwise label information while neglecting the classification information in Eq. (6), which is named DSDH-A (similar to Li et al. 2016b). Then we design a two-stream deep hashing algorithm to learn the hash codes. One stream is designed based on the pairwise label information in Eq. (3), and the other stream is constructed based on the classification information. These two streams share the same image representations except for the last fully connected layer. We denote this method as DSDH-B. Besides, we also design another approach directly applying the sign function after the outputs of the last fully connected layer in Eq. (7), which is denoted as DSDH-C. The loss function of DSDH-C can be represented as:

$$\begin{aligned} F = &-\sum_{s_{ij}\in S}\left(s_{ij}\Psi_{ij} - \log\left(1+e^{\Psi_{ij}}\right)\right) \\ &+\mu\sum_{i=1}^{N}\left\|y_i - W^T h_i\right\|_2^2 + \nu\|W\|_F^2 + \eta\sum_{i=1}^{N}\|b_i - \text{sgn}(h_i)\|_2^2, \\ &s.t.\ \ h_i \in R^{K\times 1},\ \ (i=1,\dots,N) \end{aligned} \tag{33}$$

Then we use the alternating minimization method to optimize DSDH-C. We denote our method as DSDH-$l2$ (optimization with l_2 loss) and DSDH-hinge (optimization with hinge loss). Besides, we also design a deep supervised hashing algorithm with the triplet ranking information, which is named DSDH-triplet. Our algorithm with the triplet ranking information and l_2 loss based linear classifier is denoted as DSDH-triplet-$l2$. Similarly, the triplet ranking information and hinge loss based linear classifier is represented as DSDH-triplet-hinge.

The results of different methods on CIFAR-10 under the first experimental setting are shown in Fig. 2. From Fig. 2 we can see the following information. (1) The performance of DSDH-C is better than DSDH-A. DSDH-B is better than DSDH-A in terms of precision with Hamming radius 2 and precision-recall curves. More information is exploited in DSDH-C than in DSDH-A, which demonstrates the classification information is helpful for learning the hash codes. (2) The improvement of DSDH-C over DSDH-A is marginal. The reason is that the classification information in DSDH-C is only used to learn the image representations, which is not fully exploited. Both DSDH-triplet-$l2$ and DSDH-triplet-hinge perform better than DSDH-triplet, which indicates the effectiveness of our algorithm with the triplet ranking information. (4) DSDH-$l2$ and DSDH-hinge achieve a similar performance under the one stream classification framework. Due to the violation of the discrete nature of hash codes, DSDH-C suffers from a large quantization loss. Note that both DSDH-$l2$ and DSDH-hinge beat DSDH-B and DSDH-C by a large margin.

Both the semantic information part and the classification information part play an important role in our algorithm. Our algorithm with the pairwise label information usually achieves a more satisfied result than with the triplet ranking information. As for the classification part, both $l2$ loss and hinge loss based linear classifier achieve a similar retrieval performance with limited training samples. However, if we have more training images, the large margin based linear classifier usually performs better than the $l2$ loss based linear classifier in our algorithm.

4.3 Results on CIFAR-10 Dataset

The MAP results of all methods on CIFAR-10 under the first experimental setting are listed in Table 2. From Table 2 we can see that DSDH-$l2$ and DSDH-hinge achieve a similar performance on this dataset due to the usage of classification information. Both of them substantially outperform the traditional hashing methods. The average MAP result of DSDH-$l2$ is 0.787, and the average MAP result of DSDH-hinge is 0.790, which is slightly better than DSDH-$l2$. Both of them are more than twice as much as SDH, FastH and ITQ. Due to the random sampling strategy, DSDH-triplet-$l2$ and DSDH-triplet-hinge perform inferior than DSDH-$l2$ and DSDH-hinge. However, they still perform better than most

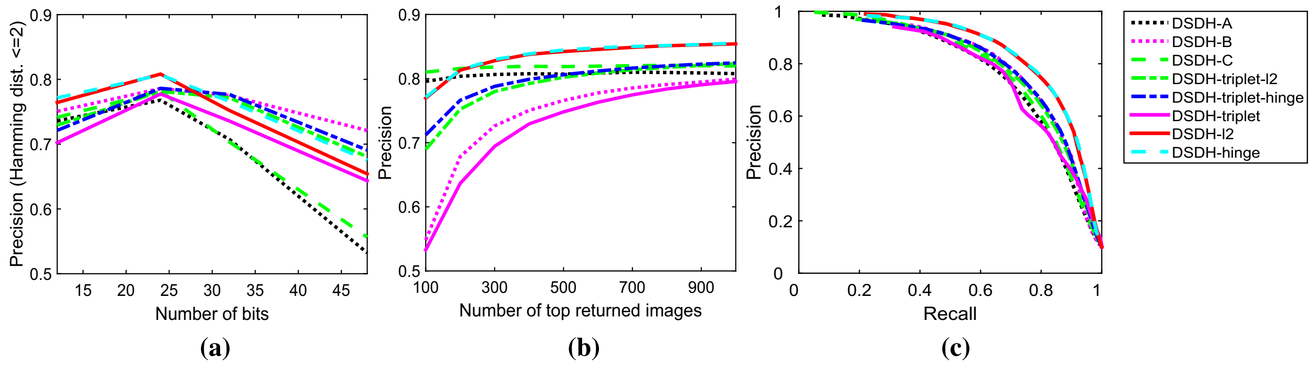

Fig. 2 The results of DSDH-A, DSDH-B, DSDH-C and DSDH on CIFAR-10 dataset: **a** precision curves within Hamming radius 2; **b** precision curves with respect to different number of top returned images; **c** precision-recall curves of Hamming ranking with 48 bits

Table 2 MAP for different methods on CIFAR-10 dataset under the first experimental setting

Method	CIFAR-10			
	12 bits	24 bits	32 bits	48 bits
DSDH-*l*2	0.740	0.786	0.801	0.820
DSDH-hinge	0.751	0.785	0.801	0.822
DSDH-triplet-*l*2	0.720	0.763	0.774	0.780
DSDH-triplet-hinge	0.710	0.772	0.775	0.785
DPSH	0.713	0.727	0.744	0.757
DHN	0.555	0.594	0.603	0.621
DTSH	0.710	0.750	0.765	0.774
NINH	0.552	0.566	0.558	0.581
CNNH	0.439	0.511	0.509	0.522
FastH	0.305	0.349	0.369	0.384
SDH	0.285	0.329	0.341	0.356
KSH	0.303	0.337	0.346	0.356
LFH	0.176	0.231	0.211	0.253
SPLH	0.171	0.173	0.178	0.184
ITQ	0.162	0.169	0.172	0.175
SH	0.127	0.128	0.126	0.129

of the other hashing algorithms. We believe with a much more efficient sampling strategy, the performance of DSDH-triplet-*l*2 and DSDH-triplet-hinge can be further improved. Besides, most of the deep hashing methods perform better than the traditional hashing methods. In particular, DTSH achieves the best performance among all the other methods except DSDH-*l*2 and DSDH-hinge on CIFAR-10 dataset. Compared with DTSH, our method further improves the performance by 3–6%. These results verify that learning the hash function and classifier within one stream framework can boost the retrieval performance.

Figure 3 further shows: (a) precision curves with Hamming radius 2; (b) precision curves with different number of top returned images when the 48 bits hash codes are used; (c) precision-recall curves of Hamming ranking with 48 bits hash codes on CIFAR-10 dataset under the first experimental setting. From Fig. 3 we can see that, similar messages are conveyed through this figure as observed in Table 2. Deep hashing methods have shown superior performance over traditional hashing methods in terms of precision curves with Hamming radius 2, precision curves with different number of top returned images and precision-recall curves. Both DSDH-triplet-*l*2 and DSDH-triplet-hinge perform better than most of the other hashing algorithms due to the combination of the triplet ranking information and classification information. Among all of the deep hashing methods, both DSDH-hinge and DSDH-*l*2 perform more favorably against others on this dataset.

Deep hashing methods usually need more training images to learn the hash function. In order to have a fair comparison, we further compare with other deep hashing methods under the second experimental setting, which contains more training images. Table 3 lists MAP results for different methods under the second experimental setting. As shown in Table 3, with more training images, most of the deep hashing methods perform better than in the previous experimental setting. The average MAP result of DRSCH is 0.624 on CIFAR-10 dataset, and the average MAP results of DPSH and DTSH are 0.787, 0.922, respectively. The average MAP results of DSDH-*l*2 and DSDH-hinge are 0.938, 0.926, respectively. DTSH and DPSH have a significant advantage over other deep hashing methods. Our method further outperforms DTSH and DPSH by 2–10%. Table 3 also shows that DSDH-hinge is slightly better than DSDH-*l*2 with more training images. The reason is that the experimental setting is different with the previous one. If there are fewer training images, the supervised information may play a limited role in learning the hash codes. Thus DSDH-*l*2 and DSDH-hinge achieve a similar performance. However, with more training images, our method with the hinge loss based linear classifier may perform better than with the l_2 loss based linear classifier.

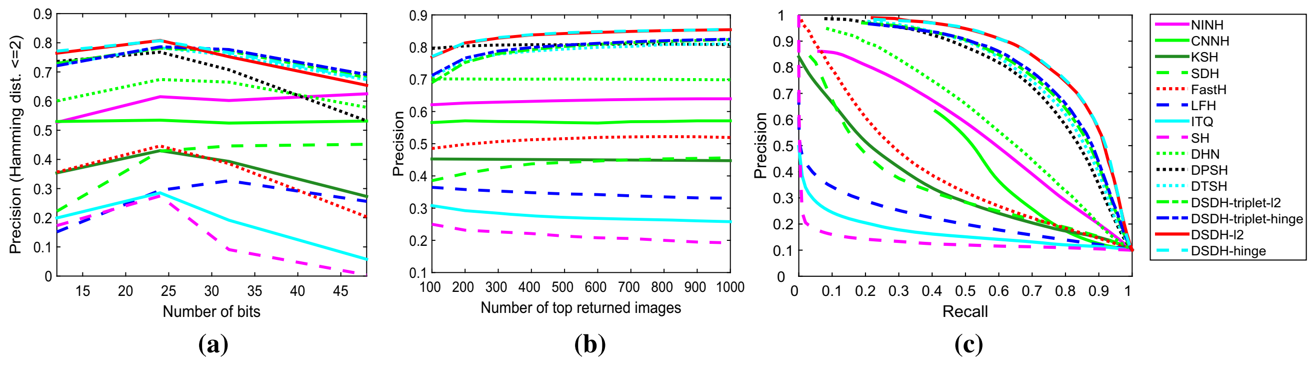

Fig. 3 The results of different methods on CIFAR-10 dataset: **a** precision curves within Hamming radius 2; **b** precision curves with respect to different number of top returned images; **c** precision-recall curves of Hamming ranking with 48 bits

Table 3 MAP for different methods under the second experimental setting

Method	CIFAR-10				Method	NUS-WIDE			
	16 bits	24 bits	32 bits	48 bits		16 bits	24 bits	32 bits	48 bits
DSDH-*l*2	0.935	0.940	0.939	0.939	DSDH-*l*2	0.815	0.814	0.820	0.821
DSDH-hinge	0.943	0.946	0.946	0.947	DSDH-hinge	0.817	0.819	0.824	0.825
DTSH	0.915	0.923	0.925	0.926	DTSH	0.756	0.776	0.785	0.799
DPSH	0.763	0.781	0.795	0.807	DPSH	0.715	0.722	0.736	0.741
DRSCH	0.615	0.622	0.629	0.631	DRSCH	0.618	0.622	0.623	0.628
DSCH	0.609	0.613	0.617	0.620	DSCH	0.592	0.597	0.611	0.609
DSRH	0.608	0.611	0.617	0.618	DSRH	0.609	0.618	0.621	0.631

The MAP for NUS-WIDE dataset is calculated based on the top 50,000 returned neighbors

4.4 Results on NUS-WIDE Dataset

In this section, we compare with other methods on NUS-WIDE dataset. The MAP results of different methods are listed in Table 4. From Table 4 we can see that the gap between deep hashing methods and traditional hashing methods is not very large on NUS-WIDE dataset, which is different from CIFAR-10 dataset. For example, the average MAP result of SDH is 0.603, while the average MAP result of NINH is 0.700. The MAP results of most of the traditional hashing methods are larger than 0.5. DTSH achieves the best performance among all of the other methods except ours.

Figure 4 further shows: (a) precision curves with Hamming radius 2; (b) precision curves with different number of top returned images when the 48 bits hash codes are used; (c) precision-recall curves of Hamming ranking with 48 bits hash codes on CIFAR-10 dataset under the first experimental setting. From Fig. 4 we can see that DHN is slightly better than NINH and CNNH due to the joint optimization of pairwise cross entropy loss and quantization loss. Our method achieves a similar performance with DPSH and DTSH. Note that Fig. 4a shows precisions with Hamming radius 2 using hash lookup. Precisions of DSDH-*l*2 and DSDH-hinge are slightly inferior to DTSH on 24 bits or 48 bits. The possible reason is that the Hamming space becomes increasingly sparse when hash codes become longer, and very few data

Table 4 MAP for different methods on NUS-WIDE dataset under the first experimental setting

Method	NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits
DSDH-*l*2	0.776	0.808	0.820	0.829
DSDH-hinge	0.772	0.809	0.819	0.832
DPSH*	0.752	0.790	0.794	0.812
DHN	0.708	0.735	0.748	0.758
DTSH	0.773	0.808	0.812	0.824
NINH	0.674	0.697	0.713	0.715
CNNH	0.611	0.618	0.625	0.608
FastH	0.621	0.650	0.665	0.687
SDH	0.568	0.600	0.608	0.637
KSH	0.556	0.572	0.581	0.588
LFH	0.571	0.568	0.568	0.585
SPLH	0.568	0.589	0.597	0.601
ITQ	0.452	0.468	0.472	0.477
SH	0.454	0.406	0.405	0.400

The MAP for NUS-WIDE dataset is calculated based on the top 5000 returned neighbors. DPSH* denotes re-running the code provided by the authors of DPSH

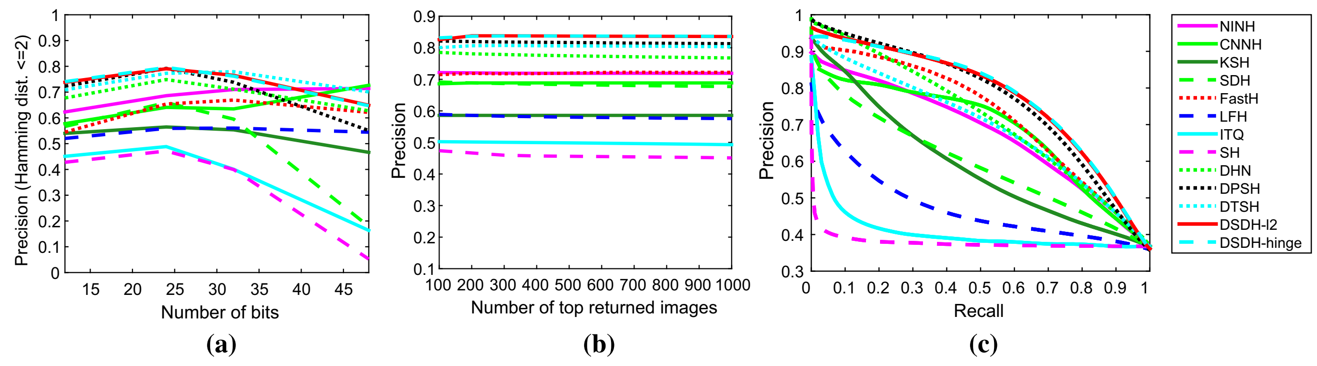

Fig. 4 The results of different methods on NUS-WIDE dataset: **a** precision curves within Hamming radius 2; **b** precision curves with respect to different number of top returned images; **c** precision-recall curves of Hamming ranking with 48 bits

points fall within the Hamming ball with radius 2 in our method.

Both DSDH-*l*2 and DSDH-hinge are slightly superior to DTSH in terms of the MAP results on NUS-WIDE dataset. The main reasons are that there exists more categories in NUS-WIDE than CIFAR-10, and each of the image contains multiple labels. Compared with CIFAR-10, there are only 500 images per class for training, which may not be enough for DSDH-*l*2 and DSDH-hinge to learn the multi-label classifier. Thus the second term in Eq. (6) plays a limited role to learn a better hash function. In the following, we will show that our method achieves a better performance than other deep hashing methods with more training images per class for the multi-label dataset.

Similar to Sect. 4.3, we also compare with other deep hashing methods with more training images under the second experimental setting on NUS-WIDE dataset. Table 3 lists MAP results for different methods on this dataset. With more training images, deep hashing methods perform better. The average MAP results of DSRH, DSCH and DRSCH are 0.620, 0.602 and 0.623, respectively. Similar with previous experimental results, the results of DPSH and DTSH are better than DSRH, DSCH and DRSCH. DSDH-*l*2 and DSDH-hinge further improve the retrieval performance by 5% in terms of the average MAP result. Note that with more training images, DSDH-hinge also performs better than DSDH-*l*2, which is consistent with previous experimental results on CIFAR-10 dataset.

4.5 Results on MIRFLICKR Dataset

In this section, MIRFLICKR dataset is used to evaluate different hashing methods. MIRFLICKR is also a multi-label dataset, which contains more semantic concepts than NUS-WIDE dataset. In the previous experimental setting, different deep hashing methods have different network structures (DPSH, DTSH and DSDH are based on the CNN-F network architecture, while DQN, DHN, DSRH are based on AlexNet architecture). Although CNN-F network architecture and AlexNet architecture have similar components: five convolutional layers and two fully connected layers, they may have different influences on the final retrieval performance. Thus in this section, we choose CNN-F network as the basic structure and re-run the source codes of different deep hashing methods, which eliminates the influence of different network structures.

Table 5 MAP for different methods on MIRFLICKR dataset

Method	MIRFLICKR			
	12 bits	24 bits	32 bits	48 bits
DSDH-*l*2	0.795	0.821	0.825	0.829
DSDH-hinge	0.798	0.815	0.819	0.825
SSDH	0.773	0.779	0.778	0.778
DRSCH	0.741	0.741	0.737	0.728
NINH	0.693	0.711	0.718	0.709
CNNH	0.667	0.688	0.654	0.626
SDH	0.595	0.601	0.608	0.605
ITQ	0.576	0.579	0.579	0.580
SH	0.561	0.562	0.563	0.562
LSH	0.557	0.564	0.562	0.569

The results of CNNH, DRSCH, NINH and SSDH are directly reported from Zhang and Peng (2017)

The results of different methods on MIRFLICKR dataset are listed in Table 5. As shown in Table 5, SH, LSH and ITQ achieve similar performance. The average result of SDH outperforms SH, LSH and ITQ by about 3%. While most of the deep hashing methods achieve better results than traditional hashing methods. The average MAP result of NINH is 0.708, which is better than CNNH (0.659). DRSCH improves the performance by about 2–5%. While SSDH further beats them due to the large scale semi-supervised information. Note that although SSDH is a semi-supervised deep hashing method, it makes use of the labeled dataset. The meaning of semi-supervised mainly refers to the usage of unlabeled dataset

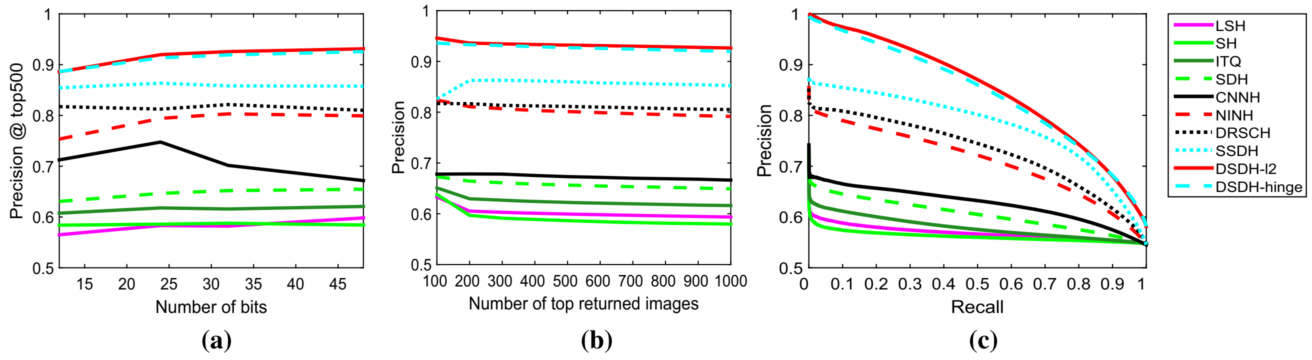

Fig. 5 The results of different methods on MIRFLICKR dataset: **a** precision curves within top 500 retrieved samples with respect to different length of hash codes; **b** precision curves with respect to different number of top returned images; **c** precision-recall curves of Hamming ranking with 48 bits

Table 6 MAP for different traditional hashing methods with deep learned features under the first experimental setting

Method	CIFAR-10				NUS-WIDE				MIRFLICKR			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
DSDH-$l2$	0.740	0.786	0.801	0.820	0.776	0.808	0.820	0.829	0.795	0.821	0.825	0.829
DSDH-hinge	0.751	0.785	0.801	0.822	0.772	0.809	0.819	0.832	0.798	0.815	0.819	0.825
FastH + CNN	0.553	0.607	0.619	0.636	0.779	0.807	0.816	0.825	0.750	0.785	0.800	0.809
SDH + CNN	0.478	0.557	0.584	0.592	0.780	0.804	0.815	0.824	0.695	0.704	0.697	0.708
KSH + CNN	0.488	0.539	0.548	0.563	0.768	0.786	0.790	0.799	0.753	0.760	0.761	0.758
LFH + CNN	0.208	0.242	0.266	0.339	0.695	0.734	0.739	0.759	0.610	0.625	0.623	0.630
ITQ + CNN	0.237	0.246	0.255	0.261	0.719	0.739	0.747	0.756	0.648	0.654	0.652	0.652
SH + CNN	0.183	0.164	0.161	0.161	0.621	0.616	0.615	0.612	0.603	0.595	0.590	0.588

during hash coding. Actually, SSDH uses more information than other deep hashing methods. The average MAP results of DSDH-$l2$ and DSDH-hinge are 0.818 and 0.814, respectively. DSDH-$l2$ outperforms SSDH by large margin of 2, 4, 5, 5% in MAP with different code length, and DSDH-hinge outperforms SSDH by large margin of 2, 4, 4, 5% in MAP with different code length. Both of them achieve a better performance than previous deep hashing methods.

Figure 5 further shows: (a) precision curves within top 500 retrieved samples with respect to different lengths of hash codes; (b) precision curves with respect to different number of top returned images; (c) precision-recall curves of Hamming ranking with 48 bits. As shown in Fig. 5a, both DSDH-$l2$ and DSDH-hinge achieve over 88% precision on all code lengths, which shows a clear advantage over SSDH and other state-of-the-art methods. Figure 5b illustrates that our method achieves over 90% search accuracy when the number of retrieved results increases. Figure 5c further shows precision recall curves of different methods, which also shows the clear advantage of our methods over other hashing methods. On all the evaluation metrics, it can be observed that both DSDH-$l2$ and DSDH-hinge outperform other state-of-the-art methods, which shows the benefit of unifying the pairwise label information and the classification information.

4.6 Comparison with Traditional Hashing Methods

In order to have a fair comparison, we also compare with traditional hashing methods using deep learned features extracted by the CNN-F network. The deep learned features are extracted from the second fully connected layer of the pre-trained CNN-F network, which contains a 4096 dimensional feature descriptors. The results of traditional hashing methods using deep learning features are denoted as FastH + CNN, SDH + CNN, KSH + CNN, LFH + CNN, ITQ + CNN, and SH + CNN. The MAP results of different methods on CIFAR-10, NUS-WIDE and MIRFLICKR datasets are listed in Table 6.

As shown in Table 6, most of the traditional hashing methods obtain a better retrieval performance using deep learned features. The average MAP results of FastH and SDH with traditional handcrafted features on CIFAR-10 dataset are 0.352 and 0.328, respectively. While the average MAP results of FastH + CNN and SDH + CNN on CIFAR-10 dataset are 0.604 and 0.553, respectively. Benefiting from the one stream framework, the average MAP result of our method on

CIFAR-10 dataset is 0.787, which outperforms the traditional hashing methods with deep learned features substantially. Note that NUS-WIDE dataset is a multi-label dataset, and the semantic information of the images from this dataset is more ambiguous than MIRFLICKR dataset. The proposed methods achieve a comparable performance with the best traditional hashing methods with deep learned features on NUS-WIDE dataset. While both DSDH-$l2$ and DSDH-hinge outperform other methods substantially on CIFAR-10 and MIRFLICKR datasets. This experiment further shows the enormous advantage of the proposed methods.

4.7 DSDH with Different Backbone Networks

Most of the existing hashing methods adopt CNN-F network architecture or AlexNet network architecture as backbone networks. Our algorithm is conducted based on CNN-F network architecture in previous experiments. In this section, we choose AlexNet network architecture as the backbone and compare with several hashing methods which use the same network architecture: GreedyHash (Su et al. 2018) and PQN (Yu et al. 2018). We denote our method, which use AlexNet network architecture, as DSDH-$l2$-alexnet and DSDH-hinge-alexnet.

We conduct experiments on CIFAR-10 dataset under the first experimental setting. Experimental results are shown in Table 7. From Table 7, we can see that our algorithm with AlexNet network architecture performs slightly better than with CNN-F network architecture, although both of them have five convolutional layers and two fully connected layers. DSDH-hinge-AlexNet achieves the highest MAP results among all of the algorithms listed in Table 7. Compared with DSDH-hinge, it further improves the retrieval performance about 2% in terms of MAP. Similar results can be observed with DSDH-$l2$-alexnet and DSDH-$l2$.

Besides, we have also compared with GreedyHash and PQN on CIFAR-10 dataset. The comparison results are also presented in Tables 7 and 8. As shown in Tables 7 and 8, although GreedyHash performs better than DSDH-$l2$ and DSDH-hinge, we can achieve comparable results if DSDH-$l2$ use AlexNet as the backbone. While DSDH-hinge-alexnet also performs better than GreedyHash. For example, the average MAP results of GreedyHash is 0.800 under the first experimental setting, while the average MAP results of DSDH-hinge-alexnet is 0.810. The average MAP results of GreedyHash is 0.943 under the second experimental setting, while the average MAP results of DSDH-hinge-alexnet is 0.949. Considering the MAP results are nearly saturate, it is a relatively large improvement. From Table 8 we can see that, PQN performs slightly better than DSDH-$l2$, and it is comparable with DSDH-hinge. If our algorithm also uses AlexNet as the backbone, we can achieve a comparable or slightly better performance than PQN.

Table 7 MAP for different methods with AlexNet network architecture on CIFAR-10 dataset under the first experimental setting

Method	CIFAR-10			
	12 bits	24 bits	32 bits	48 bits
DSDH-$l2$	0.740	0.786	0.801	0.820
DSDH-hinge	0.751	0.785	0.801	0.822
GreedyHash	0.774	0.795	0.810	0.822
DSDH-$l2$-AlexNet	0.786	0.809	0.810	0.822
DSDH-hinge-AlexNet	0.787	0.808	0.814	0.830
DSDH-$l2$-VGG	0.867	0.871	0.873	0.881
DSDH-hinge-VGG	0.845	0.857	0.857	0.853

Table 8 MAP for different methods with AlexNet network architecture on CIFAR-10 dataset under the second experimental setting

Method	CIFAR-10			
	16 bits	24 bits	32 bits	48 bits
DSDH-$l2$	0.935	0.940	0.939	0.939
DSDH-hinge	0.943	0.946	0.946	0.947
GreedyHash	0.942	0.943	0.943	0.944
PQN	0.947	0.947	0.946	0.947
DSDH-$l2$-alexnet	0.940	0.942	0.942	0.943
DSDH-hinge-alexnet	0.948	0.949	0.949	0.950
DSDH-$l2$-VGG	0.959	0.959	0.952	0.969
DSDH-hinge-VGG	0.942	0.969	0.971	0.970

Usually, using deeper network as the backbone can improve accuracy of different hashing methods. Since previous hashing methods also used VGG-16 network as their backbone (Yelamarthi et al. 2018; Yang et al. 2019), we also test our method with VGG-16 network (Simonyan and Zisserman 2015). We denote our method with VGG-16 network as DSDH-$l2$-VGG and DSDH-hinge-VGG. The experimental results are shown in Tables 7 and 8. As shown in the tables, our algorithm with VGG-16 network can improve the performance more than 5% in average compared with AlexNet network on CIFAR-10 dataset under the first experimental setting. While due to the saturate performance, our algorithm with VGG-16 network can improve the performance more than 1% in average compared with AlexNet network on CIFAR-10 dataset under the second experimental setting.

4.8 Parameter Sensitivity

In this section, the impact of different parameters on our method is evaluated. There are totally three parameters in both DSDH-$l2$ and DSDH-hinge, i.e., μ, ν and η. They are set to 1, 0.1 and 55 in all the experiments. μ is the weighting parameter of the classification term. ν is the coefficient of the classification regularization term, and η is the weighting parameter of the quantization loss term. In this section, we will tune these parameters and see the effects of these parameters on our method. First, we fix η and report the performance when μ and ν are changing. Then we fix μ and ν, and report the performance when η is changing. The results on cifar-10 under the first experimental setting are shown in Fig. 6. As shown in Fig. 6, although the performance of DSDH-$l2$ and DSDH-hinge varies when the parameters are changing, these variations are relatively small. We can obtain a better performance when $\mu = 1$, $\nu = 0.1$ and $\eta = 55$. Note that we haven't searched the optimal parameters exhaustively. We just tune the parameters by cross validation and choose suitable ones. Besides the optimal value, we can vary the parameters to achieve a satisfactory performance.

Because μ controls the weight of the classification term, a small value cannot make full of the classification information, and a large value will lead to over emphasizing the classification information while neglecting the pairwise similarity term. Thus a suitable value for μ can achieve a satisfactory. We also observe parameter ν controls the classification regularization term, so that a proper value is needed. Besides, η is also a very important parameter which controls the quantization loss, an appropriate value can boost the final performance.

4.9 Efficiency Analysis

We evaluate the testing time of different hashing methods in this section. All of the experiments are conducted on a PC with a Intel Core 2.6 GHZ CPU and a NVIDIA GTX Titan Black GPU. We assume all the images have already been represented by binary hash codes. Thus the testing time of different hashing methods is mainly caused by the query image. Feature extraction, hash code generation and image search are three key components for the testing time of different hashing methods. Note that most of the deep hashing methods are end-to-end frameworks, whose inputs are raw images pixels. While most of the traditional hashing methods focus on hash coding, whose inputs are image features. In order to have a fair comparison, we use deep learned features for traditional hashing methods, and the CNN-F network architecture is adopted for different deep hashing methods. We denote the testing time for different hashing methods as the summation of time spent on feature extraction, hashing code generation and image search.

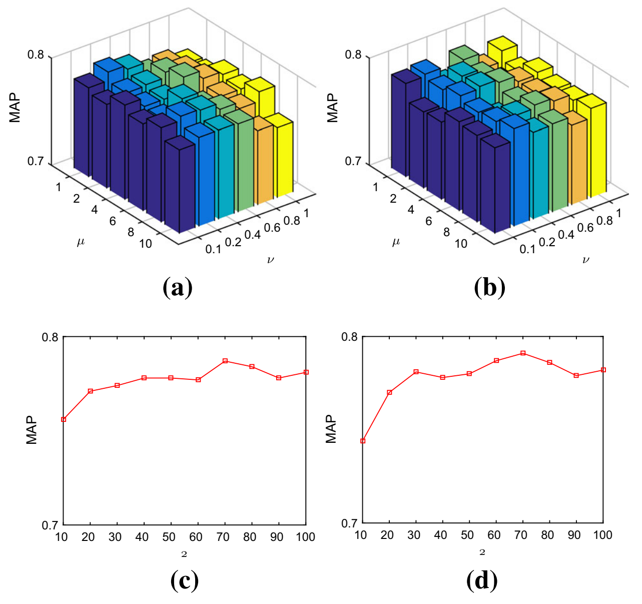

Fig. 6 Performance variation with respect to different parameters on CIFAR-10 dataset. The first row is performance variation with respect to μ and ν when we fix η. The second row is performance variation with respect to η when we fix μ and ν. **a**, **c** DSDH-$l2$. **b**, **d** DSDH-hinge

Table 9 Comparison of the testing time (millisecond per image) on three benchmark datasets by fixing the hash code length at 48

Method	CIFAR-10	NUS-WIDE	MIRFLICKR
DSDH-$l2$	2.04	2.21	1.96
DSDH-hinge	2.02	2.19	1.94
DPSH	2.08	2.25	2.02
CNNH	1.99	2.14	1.89
KSH + CNN	2.24	2.47	2.18
LFH + CNN	1.95	2.15	1.91
ITQ + CNN	2.05	2.26	2.01

The average testing time of different approaches on three benchmark datasets are listed in Table 9. From Table 9 we can see that the computational time for most of the deep hashing methods are similar, which is mainly caused by forward propagation of the same neural network. We can also see that deep hashing methods and traditional hashing methods are comparable with each other in terms of the testing time. For example, the testing time of DSDH-$l2$ on NUS-WIDE dataset is 2.21 millisecond, while the testing time of ITQ on NUS-WIDE dataset is 2.26 millisecond. Our method is comparable with most of the hashing methods in terms of the testing time. Since the testing time of our method consists of a series of convolution or inner product operations, it can be further optimized with more advanced GPU devices.

We also show the training cost of our algorithm in Fig. 7. Although there is no theoretical guarantee to ensure the convergence of our algorithm, it usually converges with less than 100 epochs.

5 Conclusion

In this paper, we have proposed a general deep supervised discrete hashing framework. Both the similarity informa-

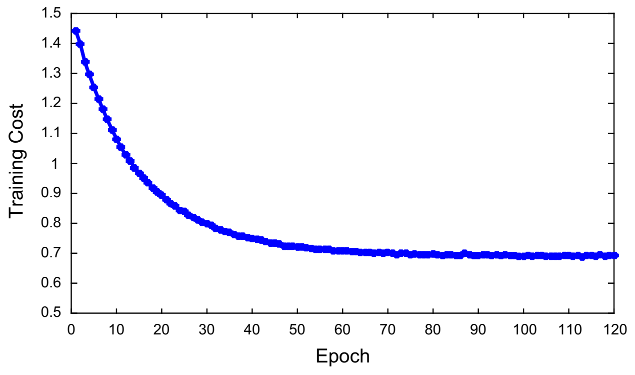

Fig. 7 The training cost of our algorithm on CIFAR-10 dataset. X-axis represents the number of epochs, and Y-axis represents the training cost

tion and classification information are used for learning hash codes under one stream framework. Besides, we constrain the outputs of the last layer to be binary codes directly, which is rarely investigated in previous deep hashing methods. Both pairwise similarity information and triplet ranking information are exploited. In addition, two different loss functions: l_2 loss and hinge loss are further analyzed. Because of the discrete nature of hash codes, we derive an alternating minimization method to optimize the loss function. Extensive experiments have shown that our method outperforms state-of-the-art methods on three benchmark image retrieval datasets.

Acknowledgements This work was partially supported by the Natural Science Foundation of China (Grant Nos. U1836217, 61702513, 61721004, and 61427811). This work was also partially supported by CAS-AIR and Shandong Provincial Key Research and Development Program (Major Scientific and Technological Innovation Project) (No. 2019JZZY010119).

References

Cao, Y., Long, M., Wang, J., Zhu, H., & Wen, Q. (2016). Deep quantization network for efficient image retrieval. In *AAAI conference on artificial intelligence*, pp. 3457–3463.

Chatfield, K., Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). Return of the devil in the details: Delving deep into convolutional nets. In *British machine vision conference*

Cui, Y., Jiang, J., Lai, Z., Hu, Z., & Wong, W. (2018). Supervised discrete discriminant hashing for image retrieval. *Pattern Recognition*, *78*, 79–90.

Gionis, A., Indyk, P., Motwani, R., et al. (1999). Similarity search in high dimensions via hashing. In *International conference on very large data bases*, Vol. 99, pp. 518–529.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE conference on computer vision and pattern recognition*, pp 580–587

Gong, Y., Lazebnik, S., Gordo, A., & Perronnin, F. (2013). Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(12), 2916–2929.

Gorisse, D., Cord, M., & Precioso, F. (2012). Locality-sensitive hashing for chi2 distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(2), 402–409.

Gui, J., Liu, T., Sun, Z., Tao, D., & Tan, T. (2016). Supervised discrete hashing with relaxation. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(3), 608–617.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition*, pp .770–778.

Hsu, C. W., & Lin, C. J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, *13*(2), 415–425.

Ji, J., Li, J., Yan, S., Zhang, B., & Tian, Q. (2012). Super-bit locality-sensitive hashing. In *Advances in neural information processing systems*, pp. 108–116.

Kang, W. C., Li, W. J., & Zhou, Z. H. (2016). Column sampling based discrete supervised hashing. In *AAAI conference on artificial intelligence*, pp. 1230–1236.

Koutaki, G., Shirai, K., & Ambai, M. (2016). *Fast supervised discrete hashing and its analysis*. arXiv:1611.10017.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

Kulis, B., & Darrell, T. (2009). Learning to hash with binary reconstructive embeddings. In *Advances in neural information processing systems*, pp. 1042–1050.

Kulis, B., & Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. In *International conference on computer vision*, pp. 2130–2137.

Kulis, B., Jain, P., & Grauman, K. (2009). Fast similarity search for learned metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*(12), 2143–2157.

Lai, H., Pan, Y., Liu, Y., & Yan, S. (2015). Simultaneous feature learning and hash coding with deep neural networks. In *IEEE conference on computer vision and pattern recognition*, pp. 3270–3278.

Li, K., Qi, G. J., Ye, J., & Hua, K. A. (2017a). Linear subspace ranking hashing for cross-modal retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(9), 1825–1838.

Li, Q., Sun, Z., He, R., & Tan, T. (2017b). Deep supervised discrete hashing. In *Advances in neural information processing systems*, pp. 2479–2488.

Li, Q., Sun, Z., Lin, Z., He, R., & Tan, T. (2016a). Transformation invariant subspace clustering. *Pattern Recognition*, *59*, 142–155.

Li, W. J., Wang, S., & Kang, W. C. (2016b). Feature learning based deep supervised hashing with pairwise labels. In *International joint conference on artificial intelligence*, pp. 1711–1717.

Lin, G., Shen, C., Shi, Q., van den Hengel, A., & Suter, D. (2014). Fast supervised hashing with decision trees for high-dimensional data. In *IEEE conference on computer vision and pattern recognition*, pp. 1963–1970.

Lin, K., Yang, H. F., Hsiao, J. H., & Chen, C. S. (2015). Deep learning of binary hash codes for fast image retrieval. In *IEEE conference on computer vision and pattern recognition workshops*, pp. 27–35.

Liong, V. E., Lu, J., Duan, L., & Tan, Y. (2018). Deep variational and structural hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(3), 580–595.

Liong, V. E., Lu, J., Tan, Y. P., & Zhou, J. (2017). Cross-modal deep variational hashing. In *IEEE conference on computer vision and pattern recognition*, pp. 4077–4085.

Liu, B., Cao, Y., Long, M., Wang, J., & Wang, J. (2018). Deep triplet quantization. In *ACM international conference on multimedia*, pp. 755–763.

Liu, H., Wang, R., Shan, S., & Chen, X. (2016). Deep supervised hashing for fast image retrieval. In *IEEE conference on computer vision and pattern recognition*, pp. 2064–2072.

Liu, L., Chen, J., Fieguth, P., Zhao, G., Chellappa, R., & Pietikäinen, M. (2019a). From bow to cnn: Two decades of texture representation for texture classification. *International Journal of Computer Vision*, *127*(1), 74–109.

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., et al. (2020). Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, *128*(2), 261–318.

Liu, W., Wang, J., Ji, R., Jiang, Y. G., & Chang, S. F. (2012). Supervised hashing with kernels. In *IEEE conference on computer vision and pattern recognition*, pp. 2074–2081.

Liu, W., Wang, J., Kumar, S., & Chang, S. F. (2011). Hashing with graphs. In *International conference on machine learning*, pp. 1–8.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

Mu, Y., & Yan, S. (2010). Non-metric locality-sensitive hashing. In *AAAI Conference on Artificial Intelligence*, pp. 539–544.

Qiu, Z., Pan, Y., Yao, T., & Mei, T. (2017). Deep semantic hashing with generative adversarial networks. In *International ACM SIGIR conference on research and development in information retrieval*, pp. 225–234.

Raginsky, M., & Lazebnik, S. (2009). Locality-sensitive binary codes from shift-invariant kernels. In *Advances in neural information processing systems*, pp. 1509–1517.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive Modeling*, *5*(3), 1.

Salakhutdinov, R., & Hinton, G. (2009). Semantic hashing. *International Journal of Approximate Reasoning*, *50*(7), 969–978.

Shen, F., Shen, C., Liu, W., & Tao Shen, H. (2015). Supervised discrete hashing. In *IEEE conference on computer vision and pattern recognition*, pp. 37–45.

Shen, F., Xu, Y., Liu, L., Yang, Y., Huang, Z., & Shen, H. T. (2018). Unsupervised deep hashing with similarity-adaptive and discrete optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(12), 3034–3044.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International conference on learning representations*

Song, J. (2018). Binary generative adversarial networks for image retrieval. In *AAAI conference on artificial intelligence*, pp. 394–401.

Su, S., Zhang, C., Han, K., & Tian, Y. (2018). Greedy hash: towards fast optimization for accurate hash coding in cnn. In *Advances in neural information processing systems*, pp. 798–807.

Tomar, V. S., & Rose, R. C. (2013). Locality sensitive hashing for fast computation of correlational manifold learning based feature space transformations. In *Interspeech*, pp. 1776–1780.

Wang, J., Kumar, S., & Chang, S. F. (2010). Sequential projection learning for hashing with compact codes. In *International conference on machine learning*, pp. 1127–1134.

Wang, J., Wang, J., Yu, N., & Li, S. (2013). Order preserving hashing for approximate nearest neighbor search. In *ACM international conference on multimedia*, pp. 133–142.

Wang, J., & Zhang, T. (2019). Composite quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *41*(6), 1308–1322.

Wang, J., Zhang, T., Sebe, N., & Shen, H. T. (2018). A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(4), 769–790.

Wang, W., Huang, Y., & Wang, L. (2019). Language-driven temporal activity localization: A semantic matching reinforcement learning model. In *IEEE conference on computer vision and pattern recognition*, pp. 334–343.

Wang, X., Shi, Y., & Kitani, K. M. (2016). Deep supervised hashing with triplet labels. In *Asian conference on computer vision*, pp. 70–84.

Weiss, Y., Torralba, A., & Fergus, R. (2009). Spectral hashing. In *Advances in neural information processing systems*, pp. 1753–1760.

Xia, R., Pan, Y., Lai, H., Liu, C., & Yan, S. (2014). Supervised hashing for image retrieval via image representation learning. In *AAAI conference on artificial intelligence*, Vol. 1, p. 2.

Yang, E., Liu, T., Deng, C., Liu, W., & Tao, D. (2019). Distillhash: Unsupervised deep hashing by distilling data pairs. In *IEEE conference on computer vision and pattern recognition*, pp. 2946–2955.

Yang, H. F., Lin, K., & Chen, C. S. (2018). Supervised learning of semantics-preserving hash via deep convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(2), 437–451.

Yao, T., Long, F., Mei, T., & Rui, Y. (2016). Deep semantic-preserving and ranking-based hashing for image retrieval. In *International joint conference on artificial intelligence*, pp .3931–3937.

Yelamarthi, S. K., Reddy, S. K., Mishra, A., & Mittal, A. (2018). A zero-shot framework for sketch based image retrieval. In *European conference on computer vision*, pp. 316–333.

Yu, T., Yuan, J., Fang, C., & Jin, H. (2018). Product quantization network for fast image retrieval. In *European conference on computer vision*, pp. 186–201.

Zhang, H., Li, Q., Sun, Z., & Liu, Y. (2018a). Combining data-driven and model-driven methods for robust facial landmark detection. *IEEE Transactions on Information Forensics and Security*, *13*(10), 2409–2422.

Zhang, J., & Peng, Y. (2017). Ssdh: semi-supervised deep hashing for large scale image retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*, *29*(1), 212–225.

Zhang, J., Peng, Y., & Yuan, M. (2018b). Unsupervised generative adversarial cross-modal hashing. In *AAAI conference on artificial intelligence*, pp. 539–546.

Zhang, P., Zhang. W., Li, WJ., & Guo, M. (2014) Supervised hashing with latent factor models. In *International ACM SIGIR conference on research and development in information retrieval*, pp. 173–182.

Zhang, R., Lin, L., Zhang, R., Zuo, W., & Zhang, L. (2015). Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE Transactions on Image Processing*, *24*(12), 4766–4779.

Zhang, Z., Chen, Y., & Saligrama, V. (2016). Efficient training of very deep neural networks for supervised hashing. In *IEEE conference on computer vision and pattern recognition*, pp. 1487–1495.

Zhao, F., Huang, Y., Wang, L., & Tan, T. (2015). Deep semantic ranking based hashing for multi-label image retrieval. In *IEEE conference on computer vision and pattern recognition*, pp. 1556–1564.

Zhao, X., Ding, G., Guo, Y., Han, J., & Gao, Y. (2017). Tuch: Turning cross-view hashing into single-view hashing via generative adversarial nets. In *International joint conferences on artificial intelligence*, pp. 3511–3517.

Zhu, H., Long, M., Wang, J., & Cao, Y. (2016). Deep hashing network for efficient similarity retrieval. In *AAAI conference on artificial intelligence*, pp. 2415–2421.