This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2789887, IEEE Transactions on Pattern Analysis and Machine Intelligence



# Unsupervised Deep Hashing with Similarity-Adaptive and Discrete Optimization

Fumin Shen, Yan Xu, Li Liu, Yang Yang, Zi Huang, Heng Tao Shen*

**Abstract**—Recent vision and learning studies show that learning compact hash codes can facilitate massive data processing with significantly reduced storage and computation. Particularly, learning deep hash functions has greatly improved the retrieval performance, typically under the semantic supervision. In contrast, current unsupervised deep hashing algorithms can hardly achieve satisfactory performance due to either the relaxed optimization or absence of similarity-sensitive objective. In this work, we propose a simple yet effective unsupervised hashing framework, named Similarity-Adaptive Deep Hashing (SADH), which alternatingly proceeds over three training modules: deep hash model training, similarity graph updating and binary code optimization. The key difference from the widely-used two-step hashing method is that the output representations of the learned deep model help update the similarity graph matrix, which is then used to improve the subsequent code optimization. In addition, for producing high-quality binary codes, we devise an effective discrete optimization algorithm which can directly handle the binary constraints with a general hashing loss. Extensive experiments validate the efficacy of SADH, which consistently outperforms the state-of-the-arts by large gaps.

**Index Terms**—Binary codes, unsupervised deep hashing, image retrieval

## 1 Introduction

Hashing has been recognized as an effective technique for large-scale retrieval [9], [52], [2], [39], [11], [44], [45]. Hashing methods map the high-dimensional features (e.g., of documents, images, videos) into low-dimensional binary codes while preserving the similarities of data from original space. It is thus an effective solution using binary codes to represent and search in massive data, due to the substantial reduction on storage space (typically tens to hundreds of bits per datum) and the low complexity of Hamming distance computation. Owing to the desiring property, hashing can be potentially applied to various applications, such as visual recognition [21], [42], [43], [36], machine learning [33], [55], [32], recommendation systems [56].

Depart from the Locality Sensitive Hashing (LSH) like data-independent methods [9], recent hashing research concentrates more on data-dependent hashing (a.k.a. *learning to hash*), which aims to generate compact binary codes by *learning* from data [48]. Many learning-based hashing algorithms have been developed according to different scenarios, including the supervised methods [30], [47], [22], [37], [40], unsupervised methods [10], [49], [31], [29], [12] and cross-modal hashing methods [17], [58], [53], [24], *etc*.


- *F. Shen, Y. Xu, Y. Yang and H. T. Shen are with Center for Future Media and School of Computer Science and Engineering, University of Electronic Science and Technology of China. E-mail: {fumin.shen, xuyan5533, dlyyang}@gmail.com, shenhengtao@hotmail.com. Corresponding author: Heng Tao Shen.*
- *L. Liu is with School of Computing Sciences, University of East Anglia, UK. E-mail: liuli1213@gmail.com.*
- *Z. Huang is with School of Information Technology & Electrical Engineering, The University of Queensland. E-mail: huang@itee.uq.edu.au.*


The deep learning based supervised hashing methods which simultaneously learn image representations and coding functions [18], [26], [59], [27], have recently shown great performance improvement compared to previous hand-crafted feature based shallow models. In contrast, only very few unsupervised deep hashing methods are proposed [25], [23], [6]. These unsupervised models are trained by minimizing either the quantization loss [25], [23] or data reconstruction error [6], which fail to explicitly preserve the data similarities in the learned binary codes and thus achieve far below satisfactory performance. We note that unsupervised hashing is potentially more useful in practical applications, since most data in real-world scenarios may not have semantic labels. This is also the focus of this work.

Another challenge is that the binary constraints imposed on the hash codes make the associated optimization problem very difficult to solve (NP-hard generally). To ease the optimization, most of the hashing methods adopt the following "relaxation + rounding" approach: first optimize a relaxed problem by discarding the binary constraints, and then obtain the approximate binary solution by thresholding the continuous results. Although this relaxation scheme greatly reduces the hardness of the optimization, the resulting approximate solution may accumulate large quantization error, which yields less effective binary codes [29], [37], [15]. To address the above problem, a few efforts have recently been devoted to developing effective discrete optimization methods for the hashing problem [22], [8], [29], [37]. However, these discrete solvers are either designed for a specific problem or very computationally expensive and thus limit their application.

To overcome the aforementioned problems, in this work, we propose an effective unsupervised deep hash-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2789887, IEEE Transactions on Pattern Analysis and Machine Intelligence



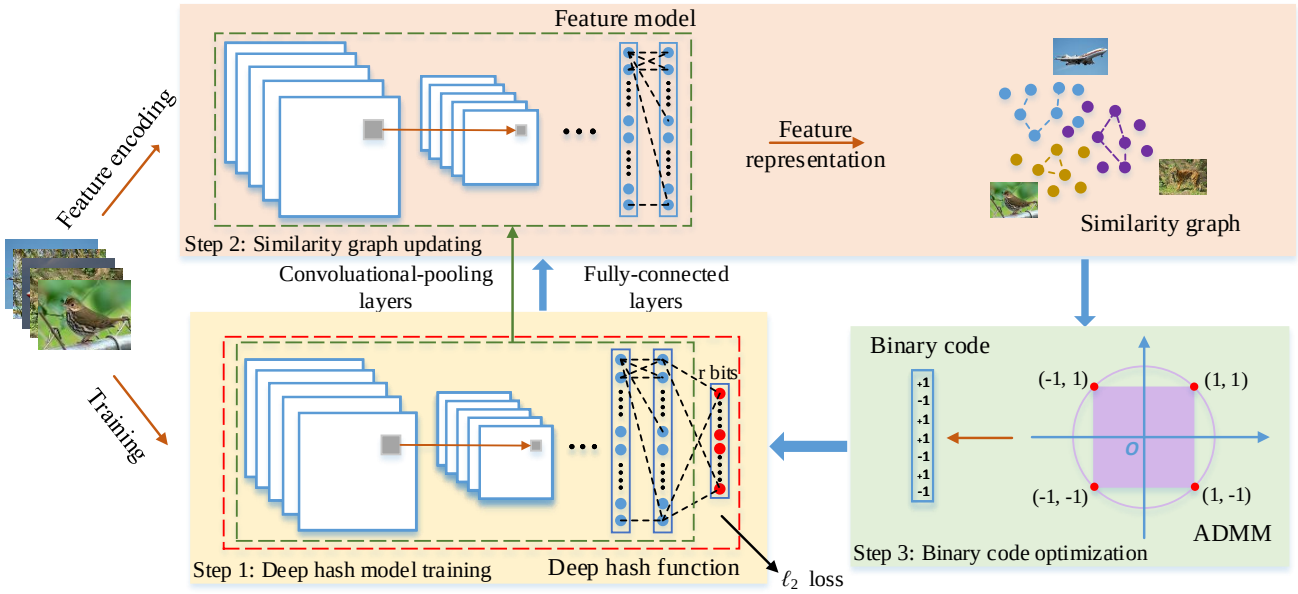



Fig. 1: An overview of the proposed Similarity-Adaptive Deep Hashing. The deep hash model (in red dash box) is trained to fit the learned binary codes. Part of the trained model is taken to generate feature representations for the images, which help update the similarity graph among data. With the updated similarity graph, the binary codes are optimized by ADMM over the intersection (red dots) of two continuous areas (in purple). *Best viewed in color.*

ing framework, dubbed Similarity-Adaptive Deep Hashing (SADH), for learning similarity-preserving binary codes from raw images. The overview training procedure of the proposed SADH is shown in Figure 1. The contributions of this work mainly include:

- We propose an effective training strategy for unsupervised deep hashing, which includes three alternating modules: deep hash model training, similarity graph updating and binary code optimization. As a key difference from the popular two-step hashing[1] [51], [59], [57], SADH updates the similarity graph matrix using the output representations from the learned deep model, which substantially improves the subsequent binary code optimization.
- For the sub-problem of binary code optimization, inspired by [50], we equivalently transform the original problem in Hamming space into the intersection of two continuous spaces. The simple reformulation greatly simplifies the original non-smooth discrete hashing problem, which is able to solve by the efficient alternating direction method of multipliers (ADMM) algorithm. The proposed algorithm supports a large family of loss functions, while in this work we particularly instantiate it with unsupervised graph hashing.
- Comprehensive evaluations are conducted on several large image datasets, and the results consistently validate the efficacy of the proposed unsupervised deep hashing algorithm, which significantly outperforms the state-of-the-art methods.

The rest of the paper is organized as follows. Section 2 briefly reviews the related hashing work in the literature. In Section 3, we elaborate the proposed Similarity-Adaptive Deep Hashing method in detail, including the training strategy and binary code optimization algorithm. Section 4 evaluates the effectiveness of the SADH model on the aspects of training strategy, binary code optimization algorithm and objective components. The experimental results with comparison to sate-of-the-arts are reported in Section 5, followed by the conclusion in Section 6.

## 2 Related Work

In this section, we briefly review some representative learning based hashing algorithms and recent developments of binary code optimization for hashing.

Data-dependent hashing methods fall into two parts: supervised methods and unsupervised methods. Supervised hashing tries to embed similarities extracted from both original data and labels into hash codes. The representative algorithms in this category include Minimal Loss Hashing (MLH) [34], Kernel-Based Supervised Hashing (KSH) [30], Semi-supervised Hashing (SSH) [47], FastHash [22], Supervised Discrete Hashing (SDH) [37], *etc*. Unsupervised hashing methods map original data into the Hamming space without any label

1. The "two-step" method refers to the widely applied hashing strategy that first learns binary codes and then learns hash functions to predict the learned codes from data.

information. Representative unsupervised methods include Iterative Quantization (ITQ) [10], Spectral Hashing (SH) [49], Anchor Graph Hashing (AGH) [31], Inductive Manifold-Hashing (IMH) [38] Discrete Graph Hashing (DGH) [29], Scalable Graph Hashing (SGH) [14], Large Graph Hashing with Spectral Rotation (LGHSR) [20] and Simultaneous Feature Aggregating and Hashing (SAH) [7].

All the previous hashing methods are based on hand-crafted features which may limit their performance in practical applications. Since Krizhevsky et al. [16] demonstrate the advantages of deep convolutional neural networks (CNNs), many supervised deep hashing methods have been proposed. CNNH [51] is a two-step hashing method, which learns a deep CNN based hash function with the pre-computed binary codes. However, its learned image representations cannot give any feedbacks to the hash code learning step. To overcome this shortcoming, the recently proposed methods [18], [59], [19] perform simultaneous feature learning and hash code learning. Yang et al. [54] propose to jointly learn image representations, hash codes and classification in a point-wise manner. Cao et al. [3] propose a new end-to-end deep method dubbed HashNet, which addresses the ill-posed gradient and data imbalance problem. Liu et al. [28] design a novel three-network framework for the sketch-based image retrieval (SBIR) task. Venkateswara et al. [46] propose a deep hashing method to address the unsupervised domain adaptation problem.

While most of these deep learning based hashing methods are supervised, only a handful of unsupervised deep hashing approaches are proposed. Liong et al. [25] develop a three-layer neural network to seek multiple hierarchical non-linear transformations to learn binary codes. Lin et al. [23] propose an unsupervised deep hashing method to learn discriminative binary descriptor by introducing three criterions at the last layer of the network. Do et al. [6] try to design a novel deep hashing network to efficiently learn hash codes by careful relaxations of the binary constraints. Compared to supervised deep hashing, the performance of current unsupervised deep hashing is far below satisfactory, which is the focus of this work.

Another major challenge of learning to hash stems from the binary nature of hash codes: the binary constraints imposed on hash codes yield an NP-hard problem in general. Recently, some effective approaches have been proposed. Gong et al. [10] propose the Iterative Quantization (ITQ), which is an effective approach to decrease the quantization distortion by applying an orthogonal rotation to project training data. One limitation of ITQ is that it learns orthogonal rotations over pre-computed mappings (e.g., PCA or CCA) and the separate learning procedure usually makes ITQ suboptimal. Liu et al. [29] propose an unsupervised discrete hashing method named Discrete Graph Hashing (DGH), which produces binary codes by signed gradient ascent (SGA). One main limitation of SGA is that it can only handle the convex *maximization* problem and it is unclear how to apply this method to other problems. In addition, it suffers from an expensive optimization due to the involvement of singular value decomposition in each iteration. Another discrete optimization method for hashing is recently proposed in [37], which efficiently learns binary codes in a bit-by-bit fashion by discrete cyclic coordinate descent (DCC). However, DCC is restricted to the standard binary quadratic program (BQP). For instance, DCC is not ready to optimize with the uncorrelation and balance constraints which are widely-used in the hashing literature.

## 3 Similarity-Adaptive Deep Hashing

### 3.1 Problem formulation

First, let us introduce some notations. We use boldface uppercase letters like $\mathbf{A}$ to denote matrices, boldface lowercase letters with subscript like $\mathbf{a}_i$ denote the $i$-th column of $\mathbf{A}$, and $\mathbf{0}$ and $\mathbf{1}$ denote the vectors with all zeros and ones, respectively. $\mathbf{I}_r$ denotes $r \times r$ identity matrix. $\mathrm{Tr}(\cdot)$ means trace operation on the square matrix. $\mathrm{sgn}(\cdot)$ is the sign function, which outputs +1 for positive numbers and -1 otherwise. $\mathrm{diag}(\cdot)$ is the diagonal matrix operator. $||\cdot||$ denotes the $\ell_2$ norm for a vector or Frobenius norm for a matrix.

Suppose that we are given a set of training data $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\} \in \mathbb{R}^{d \times n}$. Our goal is to learn a set of compact binary codes $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n\} \in \{-1, 1\}^{r \times n}$ which well preserve semantic similarities among data. Besides, we need an effective hash function $\mathbf{b} = h(\mathbf{x}) = \mathrm{sgn}(\mathcal{F}(\mathbf{x}; \mathcal{W}))$ to encode novel images to the Hamming space, where $\mathcal{W}$ is the model parameters. To achieve a better mapping from data to binary codes, we adopt the deep CNN model for our hash function, which has obtained large performance gains in recent hashing studies [51], [18]. The VGG-16 model [41] is simply adapted to be our deep hash model in this work. In detail, we modify the number of neurons as target code length in the output layer and activate them with a **tanH** function that makes the activations fall in $[-1, 1]$.

As we will discuss later, to train our deep hash model, we need to first generate the similarity-preserving binary codes for training data. In this work, we are particularly interested in the well-known graph hashing problem [49], [31], [29]:

$$\begin{aligned} \min_{\mathbf{B}} \ & \mathrm{Tr}(\mathbf{B}\mathbf{L}\mathbf{B}^\top) + \frac{\mu_1}{4}||\mathbf{B}\mathbf{B}^\top - n\mathbf{I}_r||^2 + \frac{\mu_2}{2}||\mathbf{B}\mathbf{1}||^2 \\ \text{s.t.} \ & \mathbf{B} \in \{-1, 1\}^{r \times n}. \end{aligned} \tag{1}$$

The graph Laplacian is defined as $\mathbf{L} = \mathrm{diag}(\mathbf{A}\mathbf{1}) - \mathbf{A}$, where the affinity matrix entry $\mathbf{A}_{ij}$ represents the similarity between data pair $(\mathbf{x}_i, \mathbf{x}_j)$ in the input feature space, and $\mathbf{A}\mathbf{1}$ returns the sum of columns of $\mathbf{A}$ with the all-one column vector $\mathbf{1}$. The last two terms force the learned binary codes to be uncorrelated and balanced [49], respectively. In practice, to avoid compute the $n \times n$ matrices $\mathbf{A}$ and $\mathbf{L}$, we adopt the anchor graph scheme

[31], that is, $\mathbf{L} = \mathbf{I} - \mathbf{Z}\Lambda^{-1}\mathbf{Z}^\top$. Here $\mathbf{Z} \in \mathbb{R}^{n\times m}$ is the anchor graph matrix computed by the similarities between all data and $m$ anchor points and $\Lambda = \mathrm{diag}(\mathbf{Z}^\top\mathbf{1})$.

Problem (1) has been extensively studied and shown to achieve very promising performance among shallow model based hashing methods [31], [29]. Unfortunately, it has been largely ignored by recent unsupervised deep hashing study. This is mainly due to the difficult optimization (involving sign functions and the uncorrelation constraint) and lack of effective deep hashing training strategy with the objective in (1). Consequently, we will focus on this problem in this work, as discussed below.

We now describe the training strategy of Similarity-Adaptive Deep Hashing (SADH). As illustrated in Figure 1, the overall training procedure of the proposed SADH includes three parts: deep hash function training, similarity graph updating and binary code optimization. The training algorithm proceeds alternatingly over these three parts.

### 3.2 Step 1: Deep hash model training

We train the deep hash model with a Euclidean loss layer, which measures the discrepancy between the outputs of the deep model and the binary codes learned during the last iteration. Formally, the problem is written as

$$\min_{\mathcal{W}} ||\tanh\big(\mathcal{F}(\mathbf{X};\{\mathbf{W}_l\})\big) - \mathbf{B}||^2. \tag{2}$$

Here $\mathcal{W} = \{\mathbf{W}_l\}$ denotes the parameters of the deep network architecture and $\mathbf{W}_l$ represents the weight parameters in each layer.

We use the pre-trained model on the large-scale ImageNet dataset [5] as the initialization of our deep hash function and fine tune it by the hashing problem for different datasets. The parameters $\{\mathbf{W}_l\}$ of the proposed model are optimized through the standard back-propagation and stochastic gradient descent (SGD).

For comparison, we also apply the widely used linear hash model $h(\mathbf{x}) = \mathrm{sgn}(\mathbf{W}^\top\mathbf{x})$, which is usually learned by solving a linear regression system with training data and generated binary codes, i.e., $\mathbf{W} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{B}^\top$. For clarity, we refer to our method with linear hash function as SADH-L.

### 3.3 Step 2: Data similarity graph updating

As shown in Figure 1, once the deep network is trained, we obtain a hash function as well as a feature representation model (shown at top). By encoding images with this deep model, we obtain more powerful deep representations. Subsequently, we update the pairwise similarity graph by

$$\mathbf{A}_{ij} = \exp||\bar{\mathcal{F}}(\mathbf{x}_i,\bar{\mathcal{W}}) - \bar{\mathcal{F}}(\mathbf{x}_j,\bar{\mathcal{W}})||^2/(2\sigma^2). \tag{3}$$

Here $\bar{\mathcal{F}}(\mathbf{x},\bar{\mathcal{W}})$ is the feature representation model, which is formed from $\mathcal{F}(\mathbf{x},\mathcal{W})$ by excluding the last fully connected layer; $\sigma$ is bandwidth parameter. The updated similarity graph is then taken into the next binary code optimization step. As a bridge between the deep hash model and binary codes, similarity graph updating makes them more compatible and, in the meanwhile, more effectively captures the semantic structure of images with the output deep representations.

### 3.4 Step 3: Binary code optimization

As aforementioned, problem (1) is known to be NP-hard due to the involvement of binary constraints. In this work, we solve this problem by introducing a novel binary code optimization algorithm. First, inspired by the recent advance in integer programming [50], we show that the binary code learning problem can be *equivalently* transformed to an optimization problem over the intersection of two continuous domains.

Through a simple modification from [50], we have the following result to better meet the hashing problem here: the constraint of binary code matrix $\mathbf{B} \in \{-1,1\}^{r\times n}$ is equivalent to $\mathbf{B} \in [-1,1]^{r\times n}$ and $||\mathbf{B}||_p^p = nr$. The result can be easily verified as follows. 1) Given $\mathbf{B} \in \{-1,1\}^{r\times n}$, we can trivially obtain $\mathbf{B} \in [-1,1]$ and $||\mathbf{B}||_p^p = nr$. 2) Given $\mathbf{B} \in [-1,1]^{r\times n}$, we know $||\mathbf{B}||_p^p = \sum_{i,j}|\mathbf{B}_{ij}|^p \le nr$ and the equation holds if and only if $\mathbf{B}_{ij}$ reaches its extreme value, i.e., $\mathbf{B}_{ij} = 1$ or $-1$. So if $\mathbf{B} \in [-1,1]^{r\times n}$ and $||\mathbf{B}||_p^p = nr$, we obtain $\mathbf{B} \in \{-1,1\}^{r\times n}$. Combining the above two results, we get the constraint $\mathbf{B} \in \{-1,1\}^{r\times n}$ if and only if $\mathbf{B} \in [-1,1]^{r\times n}$ and $||\mathbf{B}||_p^p = nr$.

Let denote by $\mathcal{L}(\mathbf{B})$ the objective in (1). With the above result, we equivalently rewrite problem (1) as follows.

$$\begin{aligned} &\min\ \mathcal{L}(\mathbf{B}) && \text{(4a)}\\ &\text{s.t.}\ \ \mathbf{B} \in \mathcal{S}_b, \mathbf{B} \in \mathcal{S}_p. && \text{(4b)}\end{aligned}$$

For clarity, we denote $\mathcal{S}_b$ and $\mathcal{S}_p$ as the set $[-1,1]^{r\times n}$ and $\{\mathbf{B} : ||\mathbf{B}||_p^p = nr\}$, respectively.

#### 3.4.1 Binary code optimization with ADMM

We now show how to optimize problem (4) with the well-known ADMM algorithm [1]. First, we introduce two auxiliary variables $\mathbf{Z}_1$ and $\mathbf{Z}_2$ to absorb the constraints with $\mathcal{S}_b$ and $\mathcal{S}_p$, respectively. That is, we rewrite the constraints (4b) as $\mathbf{Z}_1 = \mathbf{B},\ \mathbf{Z}_1 \in \mathcal{S}_b$ and $\mathbf{Z}_2 = \mathbf{B},\ \mathbf{Z}_2 \in \mathcal{S}_p$. Following the ADMM method [1], we solve

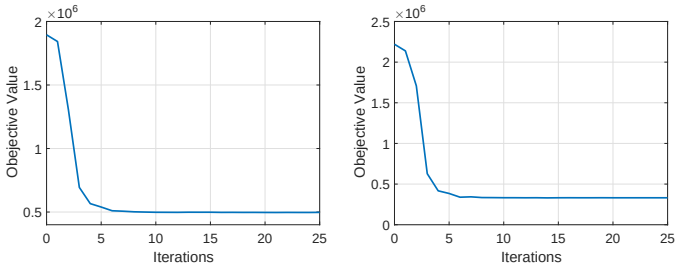
$$\begin{aligned}\min_{\mathbf{B},\mathbf{Z}_1,\mathbf{Z}_2,\mathbf{Y}_1,\mathbf{Y}_2}\ & \mathcal{L}(\mathbf{B}) + \delta_{\mathcal{S}_b}(\mathbf{Z}_1) + \delta_{\mathcal{S}_p}(\mathbf{Z}_2)\\ &+\frac{\rho_1}{2}||\mathbf{B}-\mathbf{Z}_1||^2 + \frac{\rho_2}{2}||\mathbf{B}-\mathbf{Z}_2||^2\\ &+\mathrm{Tr}(\mathbf{Y}_1^\top(\mathbf{B}-\mathbf{Z}_1)) + \mathrm{Tr}(\mathbf{Y}_2^\top(\mathbf{B}-\mathbf{Z}_2)).\end{aligned} \tag{5}$$

Here $\delta_{\mathcal{S}}(\mathbf{Z})$ is the indicator function, which outputs 0 if $\mathbf{Z} \in \mathcal{S}$ and $+\infty$ otherwise. $\mathbf{Y}_1, \mathbf{Y}_2$ and $\rho_1, \rho_2$ are the dual and penalty variables, respectively.

Following the ADMM process, we iteratively update the primal variables $\mathbf{B}, \mathbf{Z}_1, \mathbf{Z}_2$ by minimizing the augmented Lagrangian. Then, we perform gradient ascent on the dual problem to update $\mathbf{Y}_1, \mathbf{Y}_2$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2789887, IEEE Transactions on Pattern Analysis and Machine Intelligence






(a) CIFAR-10 (b) MNIST

Fig. 2: Convergence of our binary code optimization algorithm on CIFAR-10 and MNIST with 32 bits.

**Update B:** At the $(k+1)$th iteration, with all variables but $\mathbf{B}$ fixed, $\mathbf{B}^{k+1}$ is updated with the following problem,

$$\min_{\mathbf{B}} \mathcal{L}(\mathbf{B}) + \frac{\rho_1+\rho_2}{2}||\mathbf{B}||^2 + \mathrm{Tr}(\mathbf{B}\mathbf{G}^\top), \tag{6}$$

where $\mathbf{G} = \mathbf{Y}_1^k + \mathbf{Y}_2^k - \rho_1\mathbf{Z}_1^k - \rho_2\mathbf{Z}_2^k$. This sub-problem can be solved efficiently by the LBFGS-B method, with the gradient of object in (6) calculated as

$$2\mathbf{B}\mathbf{L} + \mu_1(\mathbf{B}\mathbf{B}^\top - n\mathbf{I}_r)\mathbf{B} + \mu_2\mathbf{B}\mathbf{1}\mathbf{1}^\top + (\rho_1+\rho_2)\mathbf{B} + \mathbf{G}. \tag{7}$$

**Update $\mathbf{Z}_1$ and $\mathbf{Z}_2$:** With $\mathbf{B}^{k+1}$, $\mathbf{Y}_1^k$ fixed, $\mathbf{Z}_1^{k+1}$ is updated with the following problem,

$$\min_{\mathbf{Z}_1} \delta_{\mathcal{S}_b}(\mathbf{Z}_1) + \frac{\rho_1}{2}||\mathbf{Z}_1 - \mathbf{B}^{k+1}||^2 - \mathrm{Tr}(\mathbf{Y}_1^{k\top}\mathbf{Z}_1), \tag{8}$$

which is solved by the *proximal minimization* method with the following solution:

$$\mathbf{Z}_1^{k+1} = \Pi_{\mathcal{S}_b}(\mathbf{B}^{k+1} + \frac{1}{\rho_1}\mathbf{Y}_1^k). \tag{9}$$

Here $\Pi$ is the projection operator. The operation in (9) projects those entries of $(\mathbf{B}^{k+1} + 1/\rho_1\mathbf{Y}_1^k)$ out of $[-1, 1]$ into the boundaries of the interval, i.e., -1 or 1. Similarly, $\mathbf{Z}_2^{k+1}$ is updated by $\mathbf{Z}_2^{k+1} = \Pi_{\mathcal{S}_p}(\mathbf{B}^{k+1} + 1/\rho_2\mathbf{Y}_2^k)$. In this work, we set $p = 2$ for simplicity. It is easy to solve the above problem with a closed-form solution:

$$\mathbf{Z}_2^{k+1} = \sqrt{nr} \times \frac{\mathbf{B}^{k+1} + \frac{1}{\rho_2}\mathbf{Y}_2^k}{||\mathbf{B}^{k+1} + \frac{1}{\rho_2}\mathbf{Y}_2^k||}. \tag{10}$$

**Update $\mathbf{Y}_1$ and $\mathbf{Y}_2$:** The two dual variables are updated by the gradient ascent:

$$\mathbf{Y}_1^{k+1} = \mathbf{Y}_1^k + \gamma\rho_1(\mathbf{B}^{k+1} - \mathbf{Z}_1^{k+1}), \tag{11}$$
$$\mathbf{Y}_2^{k+1} = \mathbf{Y}_2^k + \gamma\rho_2(\mathbf{B}^{k+1} - \mathbf{Z}_2^{k+1}). \tag{12}$$

Here $\gamma$ is the parameter used to accelerate convergence. The optimization alternatingly proceeds over each variable until convergence. Figure 2 illustrates the binary code optimization sub-problem, which shows that it converges within about ten iterations.

**Remarks** We have presented our whole method including three steps: binary code optimization, deep hash function training and similarity graph updating. We would note that each step contributes to learning better binary codes and capturing intrinsic data similarities: the hash function training step minimizes the discrepancy between learned codes and hash function outputs; the graph updating step computes more accurate similarity graph with updated deep representations; the binary code optimization step learns more effective hash codes based on the updated graph matrix. In addition, we show in Table 1 the computational time of the three steps in one training iteration on CIFAR-10 dataset, where the main time is taken by the deep hash model training part. We summarize our algorithm in Algorithm 1.

**Algorithm 1:** Similarity-Adaptive Deep Hashing

**Input**: Training data $\{\mathbf{x}_i\}_{i=1}^n$; code length $r$; parameters $\rho_1, \rho_2, \mu_1, \mu_2, \gamma$.
**Output**: Binary codes $\mathbf{B}$; deep hash function $h(\mathbf{x})$.
Initialize the deep model parameters $\{\mathbf{W}_l\}$ by the pre-trained VGG-16 model on ImageNet;
**repeat**
  // Data similarity updating
  Construct graph similarity matrix $\mathbf{A}$ by (3);
  // Binary code optimization
  Initialize $\mathbf{B}^0, \mathbf{Y}_1^0, \mathbf{Y}_2^0$ randomly and let $\mathbf{Z}_1^0 \leftarrow \mathbf{B}^0$, $\mathbf{Z}_2^0 \leftarrow \mathbf{B}^0$ and $k \leftarrow 0$;
  **repeat**
    Update $\mathbf{B}^{k+1}$ by solving (6) with LBFGS-B;
    Update $\mathbf{Z}_1^{k+1}$ by projecting $\mathbf{B}^{k+1}+\frac{1}{\rho_1}\mathbf{Y}_1^k$ on $\mathcal{S}_b$ with (9);
    Update $\mathbf{Z}_2^{k+1}$ by projecting $\mathbf{B}^{k+1}+\frac{1}{\rho_2}\mathbf{Y}_2^k$ on $\mathcal{S}_p$ with (10);
    Update $\mathbf{Y}_1^{k+1}$ by (11);
    Update $\mathbf{Y}_2^{k+1}$ by (12);
    $k = k + 1$;
  **until** *converged*;
  // Deep hash function learning
  Learn the deep model parameters $\{\mathbf{W}_l\}$ with optimized $\mathbf{B}$.
**until** *maximum iterations*;
**return** $\mathbf{B}, h(\mathbf{x}) = \mathrm{sgn}(\mathcal{F}(\mathbf{x}; \{\mathbf{W}_l\}))$.

TABLE 1: Time cost of our method for the three training steps and encoding a query on CIFAR-10 dataset.

| Step 1 | Step 2 | Step 3 | Encoding |
|---|---|---|---|
| 147m17s | 53s | 19m24s | 0.01s |

## 4 Model Analysis

In this part, we evaluate the proposed Similarity-Adaptive Deep Hashing (SADH) method on the training strategy, binary code optimization algorithm and objective components. The analysis is mainly conducted on the CIFAR-10 and NUS-WIDE datasets, whose descriptions can be found in Section 5.

### 4.1 Effectiveness of similarity graph updating

To validate the effectiveness of the graph matrix updating step in our training procedure, we compare our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2789887, IEEE Transactions on Pattern Analysis and Machine Intelligence



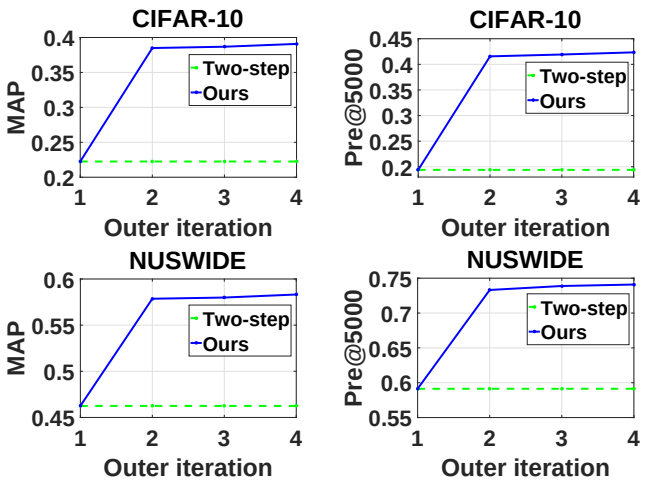



Fig. 3: Traditional two-step method vs. the proposed method with similarity graph updating.

method with the two-step training method: binary code optimization + hash function learning. The retrieval results about MAP and Precision are shown in Figure 3. As can be seen, this step effectively improves the retrieval performance. This is mainly because by the iterative training procedure, more effective semantic graph matrices are generated with the output representations of the learned deep models. This step connects the learning of hash function and binary codes, which makes them more compatible and easier to preserve the updated semantic similarity.

### 4.2 Effectiveness of proposed discrete optimization

We first evaluate our SADH-L on the impact of discrete optimization by keeping vs. discarding the sign functions. The compared results are shown in Table 2, where our discrete optimization method significantly improves its relaxed counterpart.

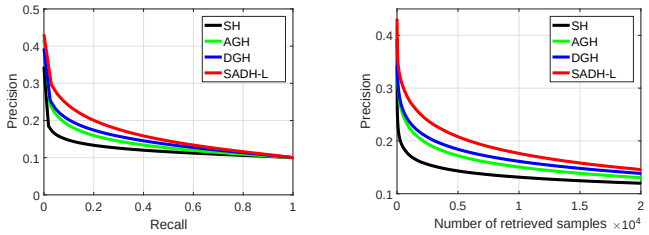
We then compare our binary code optimization method with other methods with the same unsupervised graph hashing loss, i.e., SH, AGH and DGH. We test the performance of Precision-recall curves and Precision at top 20,000 retrieved images. The results on the CIFAR-10 dataset with 64 bits are shown in Figure 4. We can see that the proposed method achieves the best performance. The proposed algorithm has clear advantages over the relaxed methods (e.g., SH, AGH) and the recent discrete optimization based DGH. These results validate the effectiveness of our ADMM based binary code optimization algorithm.

### 4.3 Impact of bit uncorrelation and balance

The performance of the proposed method on CIFAR-10 with/without the bit uncorrelation or balance terms in objective (1) is shown in Table 3. We can clearly see that better results can be obtained by imposing both two constraints than discarding them or keeping only one constraint with all code lengths, which validate their important roles in binary code optimization for hashing.

TABLE 2: Our discrete algorithm vs. relaxation method.

| Method | MAP (%) | | | Precision@5,000 (%) | | |
|---|---|---|---|---|---|---|
| | 16-bit | 32-bit | 64-bit | 16-bit | 32-bit | 64-bit |
| CIFAR-10 | | | | | | |
| Relaxed | 15.80 | 16.06 | 16.63 | 18.52 | 18.91 | 19.73 |
| Discrete | **16.45** | **16.48** | **17.61** | **19.11** | **19.42** | **19.96** |
| NUSWIDE | | | | | | |
| Relaxed | 36.05 | 36.24 | 36.38 | 36.02 | 36.79 | 37.17 |
| Discrete | **39.22** | **40.64** | **40.44** | **46.92** | **50.05** | **50.48** |




(a) Precision-recall (b) Precision

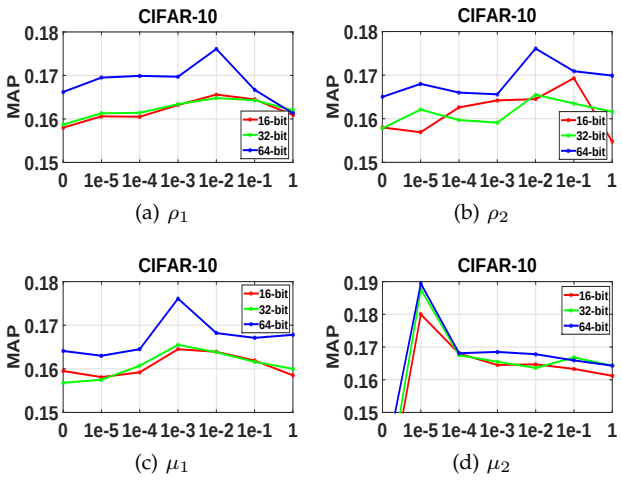
Fig. 4: Comparison of the shallow hashing models with the graph hashing loss (SH, AGH, DGH and SADH-L) on CIFAR-10 with GIST features. Results with the compared methods with 64 bits in precision-recall curve and precision curve with top 20,000 returned samples are reported.

### 4.4 Parameter sensitive analysis

The parameter $\gamma$ of our algorithm mainly influences the convergence speed. The other parameters $\rho_1, \rho_2, \mu_1, \mu_2$ can be easily tuned by the standard cross-validation. Here we conduct the sensitivity analysis of these parameters on CIFAR-10. The results are shown in Figure 5 indicate that the algorithm obtains stable performance with $\rho_1$ and $\rho_2$ in [1e-4 1e-1], $\mu_1$ and $\mu_2$ in [1e-4, 1e-2].

TABLE 3: Evaluation of our linear model with/without the constraint of balance or uncorrelation. "✓" and "–" denote the corresponding operation whether or not to be applied.

| Balance | Uncorrelation | MAP | | | Precision | | |
|---|---|---|---|---|---|---|---|
| | | 16-bit | 32-bit | 64-bit | 16-bit | 32-bit | 64-bit |
| CIFAR-10 | | | | | | | |
| – | – | 12.54 | 13.25 | 12.68 | 13.86 | 15.08 | 14.37 |
| ✓ | – | 12.91 | 13.71 | 14.57 | 14.24 | 15.37 | 16.55 |
| – | ✓ | 15.86 | 15.68 | 15.77 | 18.38 | 18.44 | 18.29 |
| ✓ | ✓ | **16.45** | **16.48** | **17.61** | **19.11** | **19.42** | **19.96** |
| NUS-WIDE | | | | | | | |
| – | – | 36.13 | 36.15 | 36.35 | 36.17 | 36.56 | 37.03 |
| ✓ | – | 38.01 | 39.05 | 39.00 | 41.79 | 44.47 | 44.00 |
| – | ✓ | 39.19 | 39.37 | 39.73 | 44.72 | 46.64 | 46.88 |
| ✓ | ✓ | **39.97** | **40.64** | **40.44** | **46.92** | **50.05** | **50.48** |




(a) $\rho_1$ (b) $\rho_2$ (c) $\mu_1$ (d) $\mu_2$

Fig. 5: Parameter sensitivity analysis of the proposed SADH.

### 4.5 Complexity analysis

In this part, we analyze the time and space complexities of the proposed method. For the graph construction component, the time complexity of constructing anchor graph is $O(dmn)$ [31]. For the ADMM component, the time-cost step is updating $\mathbf{B}$ with LBFGS-B, which needs to calculate gradient by (7) and loss value by (6), for which the time complexities are $O(t(2rmn+2r^2n+rm^2))$ and $O(2r^2n+rm^2+r^2m+rmn+r^3)$, respectively. Notice that $t$ is maximum iteration number of updating B step, and other updating steps $(\mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Z}_1, \mathbf{Z}_2)$ are constant time complexity. The space complexity of the proposed method in the training phase is $O(5rn + mn)$. The time complexity of computing the linear hash functions with least square method is $O(dnr + 2d^2n)$ and the space complexity is $O(dn + rn)$. In general, since $d, m, r \ll n$ are always satisfied, our proposed method is linear as the data size $n$.

## 5 Experiments

In this section, we thoroughly compare the proposed approach with the state-of-the-art unsupervised hashing methods on several benchmark datasets.

### 5.1 Database

Four mostly-used databases in the recent unsupervised deep hashing works are taken into our evaluation.

**CIFAR-10** As a subset of the well-known 80M tiny image collection, CIFAR-10[2] consists of 60,000 color $32 \times 32$ images which are averagely divided into 10 object classes. We use both of hand-crafted shallow features and deep features to evaluate our algorithm on this dataset. The 512-D GIST features [35] are used as shallow features.

**MNIST** The MNIST[3] dataset contains 70,000 $28 \times 28$ gray-scale images of handwritten digits from '0' to '9', which are grouped into 10 classes accordingly.

**NUS-WIDE** The NUS-WIDE dataset [4] consists of 269,648 images collected from Flickr. This database contains 81 ground-truth concepts, each image is tagged multiple semantic labels. In our experiment, we use the 21 most frequent labels for evaluation as in [31].

Following the widely-used setting in the literature [37], we randomly select 100 images from each class to form the query set, and the rest images are used for the retrieval database for all these three datasets. Deep features for these datasets are extracted from the fc7 layer of the VGG-16 model.

**GIST1M** The GIST1M[4] dataset contains one million GIST features and each feature is represented by a 960-D vector. One million samples are used as training set and additional 10K samples are used for testing. The ground truth is defined as the closest 2 percent of points as measured by the Euclidean distance, as in [47].

### 5.2 Compared methods and implementation details

We compare our SADH and its linear variant SADH-L with several other methods, including eight shallow unsupervised hashing methods, i.e., LSH [9], PCAH [47], ITQ [10], SH [49], AGH [31], DGH [29], SGH [14], LGHSR [20], and three deep unsupervised hashing methods, i.e., DH [25], DeepBit [23] and UH-BDNN [6]. We implement DGH by ourselves according to their algorithm description. For all other shallow methods, we employ the implementations and suggested parameters provided by the authors. For the deep methods, we cite the results of DH on CIFAR-10 using the identical settings since the source codes are not available. For DeepBit and UH-BDNN, we run the experiments with the released codes by the authors.

For SADH, we tune the parameters by cross-validation on the training data and set $\mu_1, \mu_2$ as $10^{-3}$, $\rho_1, \rho_2$ as $10^{-2}$ and $\gamma$ as $10^{-3}$. For efficiency, we use the anchor graph [31] to construct affinity matrix $\mathbf{A}$. We train SADH using *Caffe* [13] with raw images as inputs on a single GeForce GTX TITAN X. In all experiments, we set the initial learning rate, batch size, momentum and weight decay to 0.001, 20, 0.9 and 0.0005, respectively.

### 5.3 Evaluation metrics

We measure the performance of compared methods using Mean Average Precision (MAP), precision of the top 5,000 retrieved examples (Precision), and the precision/precision-recall curves. We adopt semantic similarity (by labels) as evaluation ground truth in most our experiments, which is widely used in the unsupervised hashing literature, for instance, AGH [31], DGH [29], and the deep methods DH [25], DeepBit [23] and UH-BDNN [6]. Particularly, for NUS-WIDE, the true neighbors are defined based on whether two images share at least one common label. Note that this work focuses on unsupervised deep hashing for encoding raw

2. http://www.cs.toronto.edu/kriz/cifar.html.
3. http://yann.lecun.com/exdb/mnist/.
4. http://corpus-texmex.irisa.fr/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2789887, IEEE Transactions on Pattern Analysis and Machine Intelligence



TABLE 4: Comparison in terms of MAP (%) and Precision@5,000 (%) on CIFAR-10 with 16, 32 and 64 bits. Shallow and deep hash models are tested with both hand-crafted features (GIST) and deep features (VGGNet). Note that DeepBit and SADH take raw images as inputs of the hash functions.

| Method | Feature | MAP | | | Precision | | |
|---|---|---|---|---|---|---|---|
| | | 16-bit | 32-bit | 64-bit | 16-bit | 32-bit | 64-bit |
| LSH | GIST | 12.92 | 13.82 | 14.87 | 14.39 | 14.33 | 16.95 |
| PCAH | GIST | 13.38 | 13.00 | 12.37 | 15.50 | 15.10 | 14.25 |
| ITQ | GIST | 15.27 | 16.25 | 16.93 | 17.69 | 18.75 | 19.87 |
| SH | GIST | 12.66 | 12.52 | 12.54 | 14.37 | 14.33 | 14.31 |
| AGH | GIST | 15.75 | 15.26 | 14.48 | 16.86 | 16.93 | 16.78 |
| DGH | GIST | 16.08 | 16.05 | 15.60 | 18.79 | 18.32 | 17.95 |
| SGH | GIST | 15.80 | 16.06 | 16.63 | 18.52 | 18.91 | 19.73 |
| LGHSR | GIST | 16.08 | 15.29 | 14.51 | 18.99 | 18.12 | 17.43 |
| **SADH-L** | GIST | **16.45** | **16.48** | **17.61** | **19.11** | **19.42** | **19.96** |
| LSH | VGG | 15.13 | 16.60 | 19.51 | 20.44 | 25.86 | 31.90 |
| PCAH | VGG | 20.83 | 17.67 | 15.78 | 23.95 | 20.74 | 18.97 |
| ITQ | VGG | 24.21 | 26.67 | 28.19 | 27.05 | 29.98 | 31.77 |
| SH | VGG | 20.88 | 18.91 | 17.46 | 24.21 | 22.37 | 20.95 |
| AGH | VGG | 32.25 | 29.97 | 26.73 | 32.44 | 30.73 | 27.48 |
| DGH | VGG | 33.02 | 31.02 | 27.97 | 37.10 | 34.35 | 29.46 |
| SGH | VGG | 25.30 | 26.66 | 29.09 | 28.81 | 30.81 | **33.25** |
| LGHSR | VGG | 28.75 | 26.94 | 24.67 | 33.79 | 32.72 | 30.09 |
| **SADH-L** | VGG | **37.60** | **34.00** | **30.30** | **40.53** | **34.91** | **32.02** |
| DH | GIST | 16.17 | 16.62 | 16.96 | - | - | - |
| UH-BDNN | GIST | 15.71 | 16.15 | 16.91 | 18.18 | 18.95 | 19.78 |
| UH-BDNN | VGG | 30.10 | 30.89 | 31.18 | 33.97 | 34.48 | 35.00 |
| DeepBit* | Raw | 15.95 | 19.16 | 20.96 | 18.02 | 22.27 | 24.36 |
| **SADH** | Raw | **38.70** | **38.49** | **37.68** | **41.80** | **41.56** | **41.15** |

* Note the MAP of DeepBit is computed using top 1,000 returned images in the original paper, while we use the standard MAP here.

images into semantic similarity-preserving hash codes. We also conduct comparisons using ground truths determined by Euclidean nearest neighbors in an appropriate feature space (Section 5.7).

### 5.4 Results on CIFAR-10

Table 4 shows the retrieval results on the CIFAR-10 dataset in term of MAP and Precision@5,000. We group these hashing methods into two categories: shallow models and deep models. For shallow models, we use both hand-crafted (GIST) and deep features (VGGNet) as inputs for fair comparisons. Among the deep models, DH and UH-BDNN can take both hand-crafted features and deep features as inputs, while DeepBit and our SADH based on deep convolutional neural networks (CNNs) are designed to feed raw images.

From Table 4, we have the following observations. 1) Based on the top two sections of Table 4, it is clear that our SADH-L with linear hashing functions consistently achieves the best results among all the shallow methods, using both hand-crafted and deep features. These results further validate the efficacy of our ADMM based binary code optimization algorithm. 2) It is not surprising that the performance of shallow models is significantly improved with deep features compared to those with hand-crafted features. Consequently, we mainly compare the shallow methods with deep features in the following experiments. 3) By comparing the deep methods (bottom of Table 4) , we observe the CNNs based DeepBit and SADH tend to obtain better results than DH and UH-BDNN with GIST inputs. This is attributed to the better representation ability of CNNs models. 4) Among all the deep methods, SADH with raw images obtains highest MAPs and precisions at all code lengths, which are significantly better than the results of DeepBit. Note that both these two methods use the VGGNet as the hash function. Another interesting observation is that the deep methods DH, UH-BDNN with GIST features and DeepBit are on par with or even worse than the shallow models with deep features.

We further show in Figure 6 the precision-recall and precision curves with top 20,000 retrieved images. It clearly shows that our SADH obtains better results in most cases, which is consistent with the observations in Table 4.

TABLE 5: Results by Hamming ranking in terms of MAP (%) and Precision@5,000 (%) on MNIST.

| Method | MAP | | | Precision | | |
|---|---|---|---|---|---|---|
| | 16-bit | 32-bit | 64-bit | 16-bit | 32-bit | 64-bit |
| Deep feature + shallow model | | | | | | |
| VGG + LSH | 16.67 | 18.21 | 22.04 | 19.35 | 24.44 | 27.32 |
| VGG + PCAH | 20.56 | 19.34 | 18.54 | 24.98 | 23.93 | 23.03 |
| VGG + ITQ | 23.83 | 28.85 | 31.04 | 27.85 | 33.49 | 36.15 |
| VGG + SH | 24.86 | 24.13 | 24.20 | 29.92 | 29.63 | 29.84 |
| VGG + AGH | 42.89 | 38.40 | 30.91 | 49.27 | 45.15 | 40.13 |
| VGG + DGH | 45.37 | 40.87 | 35.04 | 51.04 | 47.95 | 42.26 |
| VGG + SGH | 30.01 | 32.30 | 38.97 | 35.24 | 38.05 | 33.44 |
| VGG + LGHSR | 41.07 | 36.32 | 30.78 | 47.48 | 44.11 | 38.00 |
| Deep model | | | | | | |
| VGG + UH-BDNN | 35.76 | 38.37 | 40.48 | 40.43 | 42.94 | 44.32 |
| **Raw + SADH** | **46.22** | **43.03** | **41.00** | **51.61** | **48.16** | **45.73** |

### 5.5 Results on MNIST

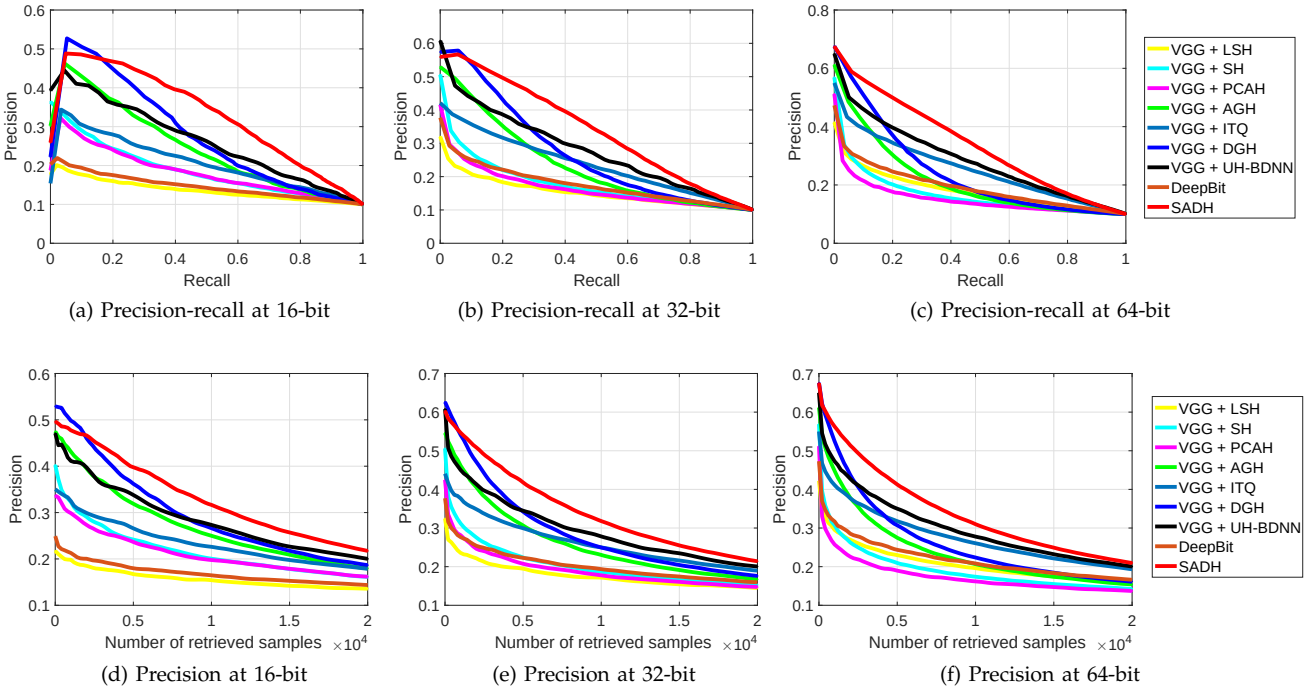
Table 5 shows the results of compared methods on MNIST with 16, 32 and 64 bits. Note that we do not report the results for DeepBit on MNIST since we do not achieve reasonable results by the provided codes. SADH obtains the best results of MAP and precision at all code lengths. In particular, SADH with raw images obtains much better precision than the UH-BDNN even with deep features.

### 5.6 Results on NUS-WIDE

Table 6 shows the retrieval results of compared methods on the NUS-WIDE dataset. We can see that the MAPs and precisions of SADH are higher than all other methods at all code lengths. For instance, with 64 bits, our method outperforms UH-BDNN by 1.59% and 2.30%

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2789887, IEEE Transactions on Pattern Analysis and Machine Intelligence






(a) Precision-recall at 16-bit (b) Precision-recall at 32-bit (c) Precision-recall at 64-bit

(d) Precision at 16-bit (e) Precision at 32-bit (f) Precision at 64-bit

Fig. 6: Precision-recall curves (top) and Precision curves (bottom) with top 20,000 retrieved images for different unsupervised hashing methods on CIFAR-10 dataset with 16, 32 and 64 bits, respectively.

TABLE 6: Results by Hamming ranking in terms of MAP (%) and Precision@5,000 (%) on NUS-WIDE.

| Method | MAP | | | Precision | | |
|---|---|---|---|---|---|---|
| | 16-bit | 32-bit | 64-bit | 16-bit | 32-bit | 64-bit |
| Deep feature + shallow model | | | | | | |
| VGG + LSH | 40.45 | 48.04 | 46.75 | 47.95 | 55.29 | 58.95 |
| VGG + PCAH | 44.10 | 42.74 | 42.21 | 57.55 | 56.68 | 55.95 |
| VGG + ITQ | 54.27 | 54.83 | 55.78 | 67.32 | 68.92 | 70.18 |
| VGG + SH | 44.74 | 42.60 | 42.36 | 59.15 | 54.98 | 54.18 |
| VGG + AGH | 49.80 | 47.14 | 44.72 | 70.43 | 70.29 | 69.29 |
| VGG + DGH | 54.03 | 52.74 | 49.64 | 71.15 | 71.91 | 70.66 |
| VGG + SGH | 48.86 | 49.10 | 51.33 | 64.92 | 66.04 | 69.01 |
| VGG + LGHSR | 50.79 | 47.72 | 45.45 | 66.79 | 68.59 | 67.60 |
| Deep model | | | | | | |
| VGG + UH-BDNN | 54.26 | 51.72 | 54.74 | 70.18 | 69.60 | 72.74 |
| Raw + DeepBit | 39.22 | 40.32 | 42.06 | 45.54 | 51.34 | 57.72 |
| **Raw + SADH** | **60.14** | **57.99** | **56.33** | **71.45** | **73.88** | **75.04** |

and DGH by 6.69% and 4.38% in MAP and precision, respectively.

In Figure 7, we show the precision-recall curves and precision curves with top 20,000 retrieved images on the NUS-WIDE dataset. Consistent with previous results, we can see that our method achieves the best results among the compared approaches in most situations.

Based on the above experiments, we conclude that the advantage of SADH is mainly due to the proposed effective binary code optimization algorithm and training strategy, by which the intrinsic data similarities are effectively encoded by the learned binary codes and hash functions. In contrast, the hash models are trained by minimizing a quantization loss in DH [25] and DeepBit [23] and by reconstructing the original data in UH-BDNN [6], which fail to explicitly preserve the similarities among data points.

### 5.7 Comparison using Euclidean ground truth

In this part, we evaluate the compared methods using Euclidean nearest neighbors as ground truths rather than semantic labels. Following [47], [29], we define nearest neighbors by the top 2% samples with minimum Euclidean distances using VGG features. Table 7 shows the results on CIFAR-10. Consistent with the above evaluations using semantic label based ground truth, SADH demonstrates higher MAPs and precisions than other compared approaches.

We further evaluate our linear method SADH-L on GIST1M[5]. As can be seen from Table 8, SADH-L obtains the best results in most situations. For example, the top three methods SADH-L SGH and DGH obtain 26.26%, 23.65%, 23.56% in the MAP with 32 bits, respectively.

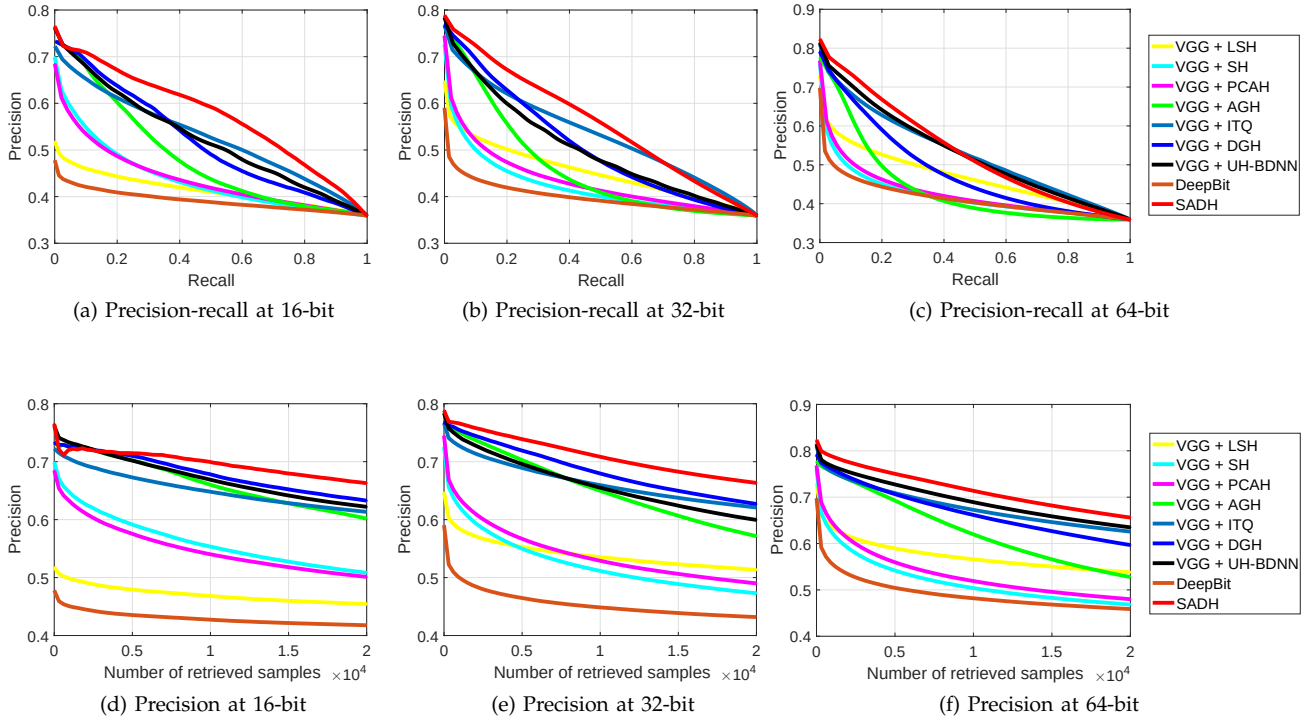
## 6 Conclusion

In this paper, we proposed a novel unsupervised deep hashing approach, SADH, which alternatingly optimized over three major components: ADMM based binary code optimization, deep CNN hash model training and data similarity graph updating with a learned deep model. The similarity graph updating step not only helped to

5. Please note GIST1M cannot be used for evaluating the CNN based SADH, which takes raw images as inputs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2789887, IEEE Transactions on Pattern Analysis and Machine Intelligence






(a) Precision-recall at 16-bit (b) Precision-recall at 32-bit (c) Precision-recall at 64-bit

(d) Precision at 16-bit (e) Precision at 32-bit (f) Precision at 64-bit

Fig. 7: Precision-recall curves (top) and Precision curves (bottom) with top 20,000 retrieved images for different unsupervised hashing methods on NUS-WIDE dataset with 16, 32 and 64 bits, respectively.

TABLE 7: Evaluation using top 2% Euclidean nearest neighbors as ground truth on CIFAR-10.

| Method | MAP (%) | | | Precision | | |
|---|---|---|---|---|---|---|
| | 16-bit | 32-bit | 64-bit | 16-bit | 32-bit | 64-bit |
| Deep feature + shallow model | | | | | | |
| VGG + LSH | 10.10 | 13.22 | 20.67 | 11.27 | 13.67 | 18.49 |
| VGG + PCAH | 16.01 | 17.54 | 18.31 | 15.74 | 15.94 | 15.86 |
| VGG + ITQ | 20.24 | 28.94 | 31.44 | 18.51 | 23.45 | 25.43 |
| VGG + SH | 18.98 | 22.08 | 24.39 | 17.19 | 18.09 | 18.26 |
| VGG + AGH | 22.93 | 26.07 | 28.00 | 20.41 | 21.01 | 20.89 |
| VGG + DGH | 23.64 | 29.79 | 32.93 | 20.27 | 22.51 | 22.52 |
| Deep model | | | | | | |
| VGG + UH-BDNN | 22.32 | 27.81 | 34.13 | 18.91 | 20.23 | 21.73 |
| Raw + DeepBit | 10.50 | 13.95 | 18.54 | 12.23 | 15.24 | 18.64 |
| **Raw + SADH** | **24.30** | **30.75** | **37.00** | **22.01** | **24.19** | **26.25** |

TABLE 8: Evaluation using top 2% Euclidean nearest neighbors as ground truth on GIST1M.

| Method | MAP (%) | | | Precision@5,000 (%) | | |
|---|---|---|---|---|---|---|
| | 16-bit | 32-bit | 64-bit | 16-bit | 32-bit | 64-bit |
| LSH | 12.94 | 13.82 | 19.82 | 17.12 | 21.09 | 31.66 |
| PCAH | 17.84 | 17.16 | 16.47 | 28.25 | 28.46 | 27.66 |
| ITQ | 21.43 | 19.66 | 25.48 | 30.44 | 26.62 | 39.32 |
| SH | 19.89 | 21.98 | 22.66 | 33.14 | 36.73 | 38.39 |
| AGH | 20.75 | 22.70 | 22.55 | 35.12 | 41.75 | 45.66 |
| DGH | 21.42 | 23.56 | 24.38 | 33.68 | 41.92 | 46.47 |
| SGH | 22.00 | 23.65 | 27.35 | **39.27** | 41.23 | 45.94 |
| LGHSR | 18.56 | 19.35 | 19.65 | 28.37 | 33.60 | 35.70 |
| UH-BDNN | 17.99 | 15.06 | 15.07 | 20.16 | 16.78 | 17.72 |
| **SADH-L** | **22.52** | **26.26** | **30.25** | 36.04 | **42.29** | **47.50** |

effectively preserve the data similarities but also enforced the learned codes to be more compatible with the deep hash functions. The experiments on the public datasets of CIFAR-10, MNIST, NUS-WIDE and GIST1M clearly demonstrated the efficacy of the binary code optimization algorithm, and the resulting deep hashing algorithm significantly outperformed the state-of-the-art unsupervised deep hashing methods.

We note that the proposed algorithm is not restricted to the unsupervised graph hashing loss. The proposed ADMM based binary code optimization algorithm provides a general solver for the hashing problems with binary constraints. In addition, the proposed deep hashing training strategy can also be potentially applied to other hashing problems involving data similarity computation. The adaptation of the proposed method to other hashing problems deserves a further study.

## ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under Project 61502081, Project 61572108, and Project 61632007, the Fundamental Research Funds for the Central Universities under Project ZYGX2015kyqd017, and Australian Research Council Project FT130101530.

## REFERENCES


[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[2] F. Cakir and S. Sclaroff. Adaptive hashing for fast similarity search. In *Proc. ICCV*, pages 1044–1052, 2015.

[3] Z. Cao, M. Long, J. Wang, and P. S. Yu. Hashnet: Deep learning to hash by continuation. In *Proc. ICCV*, pages 5608–5617, 2017.

[4] 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 CIVR*, number 48, 2009.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, pages 248–255, 2009.

[6] T.-T. Do, A.-D. Doan, and N.-M. Cheung. Learning to hash with binary deep neural network. In *Proc. ECCV*, pages 219–234, 2016.

[7] T.-T. Do, D.-K. Le Tan, T. T. Pham, and N.-M. Cheung. Simultaneous feature aggregating and hashing for large-scale image search. In *Proc. CVPR*, pages 6618–6627, 2017.

[8] T. Ge, K. He, and J. Sun. Graph cuts for supervised binary coding. In *Proc. ECCV*, pages 250–264, 2014.

[9] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. VLDB*, pages 518–529, 1999.

[10] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proc. CVPR*, pages 817–824, 2011.

[11] Y. Guo, G. Ding, L. Liu, J. Han, and L. Shao. Learning to hash with optimized anchor embedding for scalable retrieval. *IEEE TIP*, 26(3):1344–1354, 2017.

[12] M. Hu, Y. Yang, F. Shen, N. Xie, and H. T. Shen. Hashing with angular reconstructive embeddings. *IEEE TIP*, 27(2):545–555, 2018.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[14] Q.-Y. Jiang and W.-J. Li. Scalable graph hashing with feature transformation. In *Proc. IJCAI*, pages 2248–2254, 2015.

[15] W.-C. Kang, W.-J. Li, and Z.-H. Zhou. Column sampling based discrete supervised hashing. In *Proc. AAAI*, pages 1230–1236, 2016.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1106–1114, 2012.

[17] S. Kumar and R. Udupa. Learning hash functions for cross-view similarity search. In *Proc. IJCAI*, pages 1360–1365, 2011.

[18] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proc. CVPR*, pages 3270–3278, 2015.

[19] W.-J. Li, S. Wang, and W.-C. Kang. Feature learning based deep supervised hashing with pairwise labels. *Proc. IJCAI*, pages 1711–1717, 2016.

[20] X. Li, D. Hu, and F. Nie. Large graph hashing with spectral rotation. In *Proc. AAAI*, pages 2203–2209, 2017.

[21] X. Li, C. Shen, A. Dick, and A. van den Hengel. Learning compact binary codes for visual tracking. In *Proc. CVPR*, pages 2419–2426, 2013.

[22] 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. CVPR*, pages 1971–1978, 2014.

[23] K. Lin, J. Lu, C.-S. Chen, and J. Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *Proc. CVPR*, pages 1183–1192, 2016.

[24] Z. Lin, G. Ding, J. Han, and J. Wang. Cross-view retrieval via probability-based semantics-preserving hashing. *IEEE TCYB*, 47(12):4342–4355, 2017.

[25] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *Proc. CVPR*, pages 2475–2483, 2015.

[26] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *Proc. CVPR*, pages 2064–2072, 2016.

[27] L. Liu, L. Shao, F. Shen, and M. Yu. Discretely coding semantic rank orders for image hashing. In *Proc. CVPR*, pages 1425–1434, 2017.

[28] L. Liu, F. Shen, Y. Shen, X. Liu, and L. Shao. Deep sketch hashing: Fast free-hand sketch-based image retrieval. In *Proc. CVPR*, pages 2862–2871, 2017.

[29] W. Liu, C. Mu, S. Kumar, and S.-F. Chang. Discrete graph hashing. In *Proc. NIPS*, pages 3419–3427, 2014.

[30] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proc. CVPR*, pages 2074–2081, 2012.

[31] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proc. ICML*, pages 1–8, 2011.

[32] X. Liu, X. Fan, C. Deng, Z. Li, H. Su, and D. Tao. Multilinear hyperplane hashing. In *Proc. CVPR*, pages 5119–5127, 2016.

[33] Y. Mu, G. Hua, W. Fan, and S.-F. Chang. Hash-svm: Scalable kernel machines for large-scale visual classification. In *Proc. CVPR*, pages 979–986, 2014.

[34] M. Norouzi and D. M. Blei. Minimal loss hashing for compact binary codes. In *Proc. ICML*, pages 353–360, 2011.

[35] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 42(3):145–175, 2001.

[36] J. Qin, L. Liu, L. Shao, B. Ni, C. Chen, F. Shen, and Y. Wang. Binary coding for partial action analysis with limited observation ratios. In *Proc. CVPR*, pages 146–155, 2017.

[37] F. Shen, C. Shen, W. Liu, and H. T. Shen. Supervised discrete hashing. In *Proc. CVPR*, pages 37–45, 2015.

[38] F. Shen, C. Shen, Q. Shi, A. van den Hengel, Z. Tang, and H. T. Shen. Hashing on nonlinear manifolds. *IEEE TIP*, 24(6):1839–1851, 2015.

[39] F. Shen, Y. Yang, L. Liu, W. Liu, and H. T. S. Dacheng Tao. Asymmetric binary coding for image search. *IEEE TMM*, 19(9):2022–2032, 2017.

[40] F. Shen, X. Zhou, Y. Yang, J. Song, H. T. Shen, and D. Tao. A fast optimization method for general binary code learning. *IEEE TIP*, 25(12):5610–5621, 2016.

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[42] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *IEEE TPAMI*, 34(1):66–78, 2012.

[43] D. Tao, X. Li, X. Wu, and S. J. Maybank. General tensor discriminant analysis and gabor features for gait recognition. *IEEE TPAMI*, 29(10):1700–1715, 2007.

[44] D. Tao, X. Tang, X. Li, and X. Wu. Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *IEEE TPAMI*, 28(7):1088–1099, 2006.

[45] D. Tian and D. Tao. Global hashing system for fast image search. *IEEE TIP*, 26(1):79–89, 2017.

[46] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan. Deep hashing network for unsupervised domain adaptation. In *Proc. CVPR*, pages 5018–5027, 2017.

[47] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large scale search. *IEEE TPAMI*, 34(12):2393–2406, 2012.

[48] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen. A survey on learning to hash. *IEEE TPAMI*, PP, 2017.

[49] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proc. NIPS*, pages 1753–1760, 2008.

[50] B. Wu and B. Ghanem. $\ell_p$-box admm: A versatile framework for integer programming. *arXiv preprint arXiv:1604.07666*, 2016.

[51] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *Proc. AAAI*, pages 2156–2162, 2014.

[52] H. Xu, J. Wang, Z. Li, G. Zeng, S. Li, and N. Yu. Complementary hashing for approximate nearest neighbor search. In *Proc. ICCV*, pages 1631–1638, 2011.

[53] X. Xu, F. Shen, Y. Yang, H. T. Shen, and X. Li. Learning discriminative binary codes for large-scale cross-modal retrieval. *IEEE TIP*, 26(5):2494–2507, 2017.

[54] H.-F. Yang, K. Lin, and C.-S. Chen. Supervised learning of semantics-preserving hash via deep convolutional neural networks. *IEEE TPAMI*, PP, 2017.

[55] Y. Yang, F. Shen, Z. Huang, , H. T. Shen, and X. Li. Discrete nonnegative spectral clustering. *IEEE TKDE*, 29(9):1834–1845, 2017.

[56] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua. Discrete collaborative filtering. In *Proc. SIGIR*, pages 325–334, 2016.

[57] Z. Zhang, Y. Chen, and V. Saligrama. Efficient training of very deep neural networks for supervised hashing. In *Proc. CVPR*, pages 1487–1495, 2016.

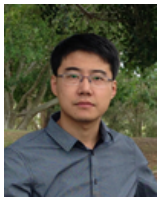
[58] X. Zhu, Z. Huang, H. T. Shen, and X. Zhao. Linear cross-modal hashing for efficient multimedia search. In *Proc. ACM Multimedia*, pages 143–152, 2013.

[59] B. Zhuang, G. Lin, C. Shen, and I. Reid. Fast training of triplet-based deep binary embedding networks. In *Proc. CVPR*, pages 5955–5964, 2016.

**Fumin Shen** received his Bachelor degree at 2007 and PhD degree at 2014 from Shandong University and Nanjing University of Science and Technology, China, respectively. Now he is with School of Computer Science and Engineering, University of Electronic Science and Technology of China. His major research interests include computer vision and machine learning. He was the recipient of the Best Paper Award Honorable Mention at ACM SIGIR 2016, ACM SIGIR 2017 and the World's FIRST 10K Best Paper Award - Platinum Award at IEEE ICME 2017.

**Yan Xu** is currently a master student in School of Computer Science and Engineering, University of Electronic Science and Technology of China. His current research interests include multimedia and computer vision.

**Li Liu** received the B.Eng. degree in electronic information engineering from Xian Jiaotong University, Xian, China, in 2011, and the Ph.D. degree from the Department of Electronic and Electrical Engineering, University of Sheffield, Sheffield, U.K., in 2014. He is currently with School of Computing Sciences, University of East Anglia, UK. His current research interests include computer vision, machine learning, and data mining.



**Yang Yang** is currently with University of Electronic Science and Technology of China. He was a Research Fellow under the supervision of Prof. Tat-Seng Chua in National University of Singapore during 2012-2014. He was conferred his Ph.D. Degree (2012) from The University of Queensland, Australia. During the PhD study, Yang Yang was supervised by Prof. Heng Tao Shen and Prof. Xiaofang Zhou. He obtained Master Degree (2009) and Bachelor Degree (2006) from Peking University and Jilin University, respectively.

**Zi Huang** received the B.Sc. degree in computer science from Tsinghua University, China, and the Ph.D. degree in computer science from the School of Information Technology and Electrical Engineering, The University of Queensland, Australia. She is an ARC Future Fellow with the School of Information Technology and Electrical Engineering, The University of Queensland. Her research interests include multimedia indexing and search, social data analysis, and knowledge discovery.

**Heng Tao Shen** is a Professor of National "Thousand Talents Plan" and the director of Center for Future Media at the University of Electronic Science and Technology of China (UESTC). He obtained his BSc with 1st class Honours and PhD from Department of Computer Science, National University of Singapore (NUS) in 2000 and 2004 respectively. He then joined the University of Queensland (UQ) as a Lecturer, Senior Lecturer, Reader, and became a Professor in late 2011. His research interests mainly include Multimedia Search, Computer Vision, and Big Data Management on spatial, temporal, and multimedia databases. He has published over 150 peer-reviewed papers, most of which are in prestigious international venues of interests. For his outstanding research contributions, he received the Chris Wallace Award in 2010 conferred by Computing Research and Education Association, Australasia, and the Future Fellowship from Australia Research Council in 2012. He is an Associate Editor of IEEE Transactions on Knowledge and Data Engineering, and has organized ICDE 2013 as Local Organization Co-Chair, and ACM Multimedia 2015 as Program Committee Co-Chair.