Deep Self-Taught Hashing for Image Retrieval

Yu Liu, Jingkuan Song, Ke Zhou, Lingyu Yan, Li Liu, Fuhao Zou,
and Ling Shao, *Senior Member, IEEE*

***Abstract*—Hashing algorithm has been widely used to speed up image retrieval due to its compact binary code and fast distance calculation. The combination with deep learning boosts the performance of hashing by learning accurate representations and complicated hashing functions. So far, the most striking success in deep hashing have mostly involved discriminative models, which require labels. To apply deep hashing on datasets without labels, we propose a deep self-taught hashing algorithm (DSTH), which generates a set of pseudo labels by analyzing the data itself, and then learns the hash functions for novel data using discriminative deep models. Furthermore, we generalize DSTH to support both supervised and unsupervised cases by adaptively incorporating label information. We use two different deep learning framework to train the hash functions to deal with out-of-sample problem and reduce the time complexity without loss of accuracy. We have conducted extensive experiments to investigate different settings of DSTH, and compared it with state-of-the-art counterparts in six publicly available datasets. The experimental results show that DSTH outperforms the others in all datasets.**

***Index Terms*—Deep Learning, hashing, image retrieval, self-taught.**

I. Introduction

With the rapid development of the Internet, data amount has been increasing exponentially, leading to big data era. Methods of efficiently extracting related or similar information from a huge amount of data are the core of data perceptual technology under big data environment. With the scale of image data continuously increasing, efficient image retrieval on large image data sets has attracted much attention from researchers, among which, content-based image hashing is an effective method for image retrieval under big data environment. Actually, hashing is a special way of dimensionality reduction, mapping high dimensional feature to compact hash codes. Since the Hamming distance between two binary hash codes can be computed efficiently by using bit XOR operation and counting the number of nonzero bits, an ordinary PC today would be able to do millions of Hamming distance computation in just a few milliseconds. As a result, hashing shows incomparable superiority in the fast similarity search. Despite ensured efficiency, it is a hot topic to improve hashing accuracy.

First, effective feature extraction is the premise of ensuring the accuracy of hashing in the process of applying hashing to image retrieval. Existing hashing methods are mainly based on hand-crafted features, such as Color Histogram, GIST, scale-invariant feature transform, bag of word, etc. However, those features are limited in aspect of reflecting image semantic information, because they represent the semantical content in just one aspect, either global or local view. Meanwhile, above feature extracting models are shallow models, which intrinsically could not explore the high-level semantic information contained in feature data, showing poor performance on tackling the problem of the semantic gap between image features and hash codes.

Another critical problem in hashing is to preserve or magnify similarity relationship of extracted features. Early research on hashing focuses on data-unaware hashing methods, in which the locality sensitive hashing (LSH) methods are the most well-known representatives. LSH [1] explores the random projections followed by thresholding to embed high-dimensional features into a low-dimensional Hamming space, which are independent of the data. To make the hash codes more efficient and accurate, several researchers attempt to design data-aware hashing by introducing the machine learning tricks. Data-aware hashing methods can be categorized into unsupervised and supervised methods. Unsupervised methods only use unlabeled data to learn hash functions that map input data points to similarity-preserving hash codes. Representative methods in this category include kernelized locality-sensitive hashing [2], quantization-based hashing [3], spectral hashing [4], graph-based hashing methods [5], and iterative quantization [6]. Supervised methods try to adopt supervised information (e.g., class labels, relative similarities among data points, etc.) to generate semantic hash codes. Supervised hashing with kernels (KSH) [7] is a kernel-based method that pursues compact binary codes to minimize the Hamming distances among similar pairs and

Manuscript received April 6, 2017; revised October 30, 2017; accepted March 8, 2018. Date of publication May 4, 2018; date of current version March 28, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61672254, Grant 61502189, Grant 61572215, Grant 61502155, and Grant 61772180, and in part by the National Key Research and Development Program of China under Grant 2016YFB0800402. This paper was recommended by Associate Editor Q. Ji. *(Corresponding authors: Jingkuan Song; Ke Zhou.)*

Y. Liu and K. Zhou are with the Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System (School of Computer Science and Technology), Ministry of Education of China, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: k.zhou@hust.edu.cn).

J. Song is with the Center for Future Media and School of Computer Science, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: jingkuan.song@gmail.com).

L. Yan is with the School of Computer Science, Hubei University of Technology, Wuhan 430068, China.

L. Liu and L. Shao are with the School of Computing Sciences, University of East Anglia, Norwich NR1 1NN, U.K.

F. Zou is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China.

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

Digital Object Identifier 10.1109/TCYB.2018.2822781

maximize those among dissimilar pairs. Binary reconstruction embedding (BRE) [8] learns hash functions by constructing an objective function of reconstruction errors among original feature data and corresponding hash codes. Self-taught hashing (STH) [9] is proposed and considered as one of state-of-the-art works. However, it suffers from overfitting problem since the operations of generating hash codes for training data and hash functions for testing data are independently handled, which leads to poor generalization ability.

Luckily, these problems have been greatly relieved with the development of deep learning. Convolutional neural networks (CNNs), have already achieved great success in various visual tasks, such as image classification, retrieval, and object detection [10]–[17] due to their powerful capability on mining high-level semantic image representation. With regard to the hashing task, several researchers introduced CNN into research [18]–[28], yielding good performance.

Above researches improve the hashing performance to some extent, but there exists several problems. For higher accuracy, existing methods require classification labels to supervise the learning process. However, classification labels are difficult to acquire which leads to poor adaptability to out-of-sample data. To satisfy the label requirement and implement end-to-end learning mechanism, research, such as CNNH generates hash labels for supervised learning, which relies on classification labels and neglects the intrinsic semantic information of images. Besides, to generate accurate image features, existing methods always adopt deep network in the process of feature extraction, yielding high time consumption which is unacceptable in online retrieval process.

In view of above problems, we propose a deep STH algorithm (DSTH) that can adapt to the circumstance without labels and automatically generate hash labels for end-to-end learning. At first, we get hash labels according to features generated in a deep framework. And then, using the relatively simple deep learning structure to carry out end-to-end learning with generated hash labels.

II. Related Works

The research of hash algorithm is carried out from shallow hashing algorithms to deep hashing algorithms.

A. Shallow Hashing Algorithm

Shallow hashing provides the basic concepts of transforming data into hash codes. In addition to the classical supervised hashing algorithm [7], [8] and unsupervised hashing algorithms [1], [2], [4]–[6], [29], [30] introduced in Section I [31], [32] also have been explored by shallow methods. The core of shallow hashing is to map vectors from high-dimensional space into low-dimensional space. To maximize similarity preservation in the process of hashing, shallow learning algorithms always combine different extracted shallow features, by different mapping ways. These enlightening works have been surpassed by deep methods, because of the intrinsic drawback of the feature extraction of shallow models. However, in order to adapt to the situation without labels, the structure of self-taught in shallow learning is worth noting, like STH [9].

STH [9] focuses on the local similarity structure, i.e., k-nearest-neighbors (KNN), for each data point. Let $Y = [y_1, y_2, \dots, y_n]$ be the matrix of hash codes (binary vectors with dimensionality k) associated with n data points $X = [x_1, x_2, \dots, x_n]$ and $W_{n\times n}$ be the affinity matrix, where W is defined as

$$W(i,j) = \begin{cases} \frac{x_i^T x_j}{\|x_i\|\|x_j\|}, & \text{if } x_i \in N_k(x_j) \text{ or } x_j \in N_k(x_i) \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $N_k(x)$ represents the set of KNN of data point x. For training data, to make sure that similar items are mapped into similar hash codes with good efficiency (i.e., each bit has a 50% chance of being one or zero, and different bits are independent of each other), the hash codes must satisfy the following criteria:

$$\begin{aligned} \arg\min \ & \sum_{i,j} W_{ij} \left\| y_i - y_j \right\|^2 \\ \text{s.t.} \ & y_i \in \{-1, 1\}^k, \sum_i y_i = 0, \ \frac{1}{n}\sum_i y_i {y_i}^T = I. \end{aligned} \tag{2}$$

By introducing a $n \times k$ matrix Y whose jth row is y_j^T and a diagonal $n \times n$ matrix $D(i,i) = \sum_j W(i,j)$, the above problem is rewritten as

$$\begin{aligned} \arg\min_{Y} \ & \mathrm{Tr}\left(Y^T (D - W) Y\right) \\ \text{s.t.} \ & y(i,j) \in \{-1, 1\}, \ YY^T = I, \ Y^T\mathbf{1} = \mathbf{0}. \end{aligned} \tag{3}$$

Obviously, the above problem is NP-hard problem, but one can obtain a compromised solution by selecting the k eigenvectors of $D - W$ with minimal eigenvalues followed by thresholding these eigenvectors to obtain a binary code.

For out-of-sample data, the linear SVM, $f(x) = \mathrm{sgn}(w^T x)$ is introduced to predict hash codes in STH. Given the data points $x_1, \dots, x_n$ together with their self-taught binary labels for the pth bit ${y_1}^{(p)}, \dots, {y_1}^{(p)}$, the corresponding linear SVM can be trained by solving the following quadratic optimization problem:

$$\begin{aligned} \arg\min_{w, \epsilon_i \geq 0} \ & \frac{1}{2} w^T w + \frac{C}{n} \sum_{i=1}^{n} \epsilon_i \\ \text{s.t.} \ & {y_i}^{(p)} w^T X_i \geq 1 - \epsilon_i. \end{aligned} \tag{4}$$

STH maps feature data through the relative Euclidean distance, by preserving the semantic relationship among data and controlling the hash code length. Therefore, it provides hash codes with strong relativity and shows good performance. Although our framework seems similar to STH, we have two fundamental differences: 1) the proposed hash method is based on automatic feature extraction method (i.e., CNN), rather than hand-crafted feature used by STH and 2) the proposed method utilizes the deep model, which usually shows better performance, in contrast to the STH.

B. Deep Hashing Algorithm

Because of the end-to-end interactive learning between the feature and the task, deep learning algorithm makes the extracted image feature more suitable to the task itself. Based on this, some hashing tasks try to research with deep learning [18]–[20], [33]–[36]. In [19], the hash codes are generated by using the features extracted from the depth structure, which obtained fine results. However, it is not real end-to-end hashing learning procedure, because of using classification labels as learning standard. In [20], by referring to the network structure of slice layers, the hash value of each bit is more representative, which improves the hash accuracy. But the method is still under the premise of the known classification. Although CNNH [18] also needs to rely on classification labels, it maps classification labels into hash labels, ultimately achieved end-to-end deep hash learning.

Given n images $I = \{I_1, I_2, \ldots, I_n\}$ and a pairwise similarity matrix S defined by

$$S(i,j) = \begin{cases} +1, & I_i, I_j \text{ are semantically similar} \\ -1, & I_i, I_j \text{ are semantically dissimilar.} \end{cases} \tag{5}$$

Define n by q binary matrix H whose kth row is $H_k \in \{-1, 1\}^q$, where H_k represents the target q-bit hash code for the image I_k. The approximate hash codes for the training images in I is learned by minimizing the following reconstruction errors:

$$\begin{aligned} \arg\min_{H} \ & \sum_{i=1}^{n}\sum_{j=1}^{n}\left(S_{ij} - \frac{1}{q}H_i H_j\right)^2 \\ \text{s.t.} \ & H \in \{-1, 1\}^{n\times q}. \end{aligned} \tag{6}$$

The above optimization problem is solved by relaxing the range constraint of H and a coordinate descent algorithm using Newton directions.

After getting the learned hash codes matrix H with each of its rows being a q-bit hash code for a training image, CNNH defines an output layer with q output units, each of which corresponds to one bit in the target hash code for an image. A network is adopted to get the target hash codes which has three convolution-pooling layers with rectified linear activation, max pooling, and local contrast normalization, a standard fully connected layer and an output layer with softmax activation. There are 32, 64, 128 filters (with the size 5 × 5) in the first, second, and third convolutional layers, respectively.

CNNH generates hash labels through calculating H, solving the problem of lacking labels for generating hash codes through deep learning, which is a milestone for hashing algorithms. However, there is a weakness of independence from hand-crafted labels resulting in negligence of the relativity among image semantic by using classification labels to generate hash labels.

Inspired by previous works, we realized that better feature extracting methods generate more representative feature vectors, which yields more accurate hash labels contributing to get better hash functions. Therefore, we propose our improvement.

III. Deep Self-Taught Hashing

The proposed deep STH framework is composed of hash label generating stage and hash function learning stage. In hash label generating stage, we use the deep and shallow mixed learning. First, we extract image features by finetuning an existing model (e.g., AlexNet or GoogLeNet trained on ImageNet). After that, graph model of features is constructed using KNN algorithm and mapped using LE algorithm [37], [38]. Then we apply binarization to map results to hash codes as hash labels. The advantage of this method is that it combines the advantages of feature extraction using deep learning and the advantages of shallow hash algorithms when dealing with unlabeled data. Moreover, these hash labels are more accurate than CNNH's because they contain semantics information of images. In hash function learning stage, we apply deep learning framework to learn hash function through the hash labels generated in hash label generating stage, which is real end-to-end hashing learning and used to map new images into the hash codes for image retrieval. The whole process is similar to the encoding and decoding, but because of the use of different mapping methods, it will not trigger the overfitting problem. To accelerate the hash codes generating and try not to lose accuracy, we select complex nets in label generating stage and simple nets in function learning stage. The framework of our DSTH is shown in Fig. 1. In the following, we introduce our DSTH in detail.

A. Hash Label Generating Stage

In this stage, we apply the deep and shallow mixed learning to get labels. For more accurate features, we introduce a deeper CNNs (e.g., AlexNet or GoogLeNet), the model of which is trained on a large image dataset (e.g., ImageNet). In the absence of labels, we use the trained deep model to get features. In another case, we finetune the network with labels to extract features.

After that, we use LE and binarization to transform the extracted CNN features to hash labels, retaining relative distances among data from high dimension vector space to low dimension hamming space. Mathematically, we let n m-dimensional vectors $\{x_i\}_{i=1}^{n} \in \mathbb{R}^m$ denote the image features. We use x_i and y_i to represent ith sample and its hash label, where $y_i \in \{0, 1\}^l$. We set $y_i^{\rho} \in \{0, 1\}$ as the ρth element of y_i. The hash codes set for n samples can be represented as $[y_1, \ldots, y_n]^T$.

This method is based on graph structure and focus on the local similarity one. Therefore, we apply KNN algorithm to construct data graph. Our $n \times n$ local similarity matrix W is

$$W_{ij} = \begin{cases} 0 & \text{if } N_k(x_i, x_j) \text{ is false} \\ \frac{x_i^T x_j}{\|x_i\|\cdot\|x_j\|} & \text{otherwise} \end{cases} \tag{7}$$

where $N_k(x_i, x_j)$ represents the ith and the jth samples, which are neighbors of each other in KNNs set. Furthermore, we apply diagonal matrix

$$D_{ii} = \sum_{j=1}^{n} W_{ij}. \tag{8}$$

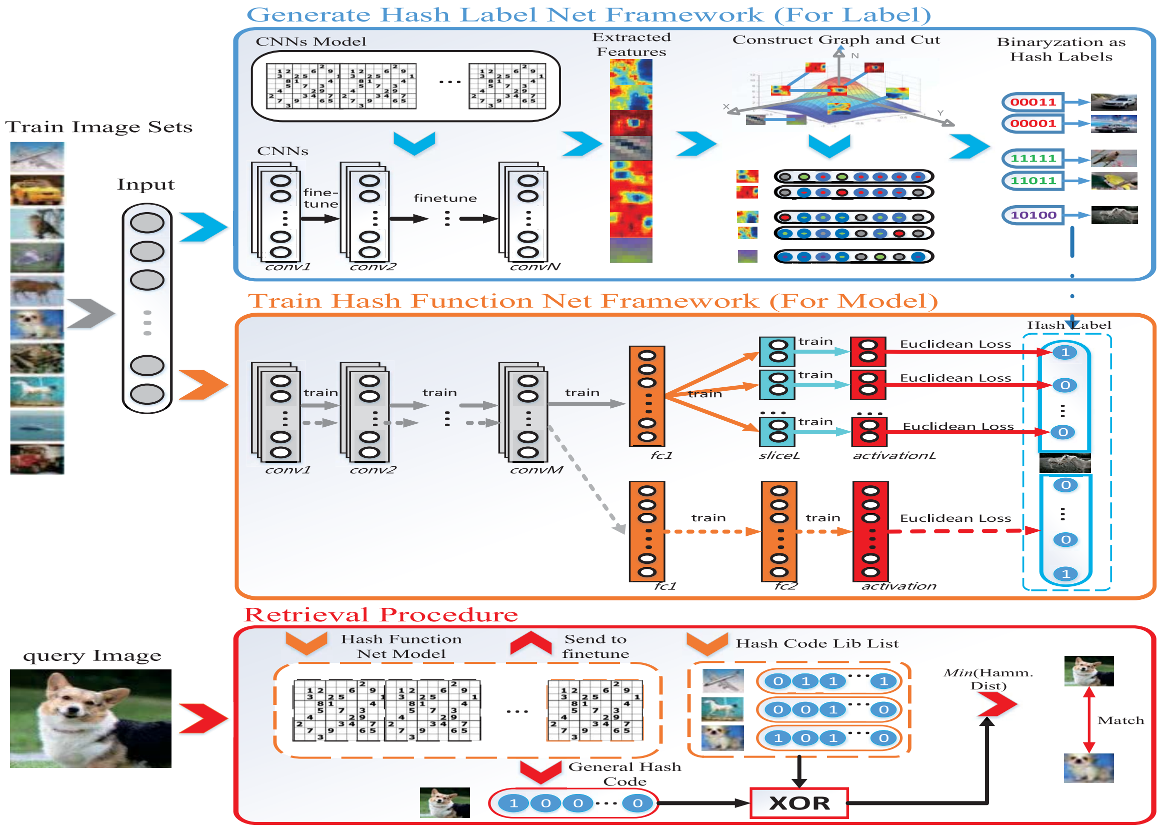


Fig. 1. DSTH framework. The structure we propose is divided into three stages: hash label generating stage (blue), hash function learning stage (orange), and image retrieval stage (red). In the first stage, we apply CNNs and trained model to abstract the features. Subsequently, hash labels will be generated by using graph construction, min cut, and binarization. In the second stage, we use two different deep learning frameworks to train the hash functions with hash labels and then obtain the net model. At last, we use the network and the net model to generate hash codes for new images and find similar images by Hamming distance.

Meanwhile, we use the number of different bits for calculating Hamming distance between y_i and y_j as

$$H_{ij} = \|y_i - y_j\|^2/4. \tag{9}$$

Similar to SpH, we define an object function ζ to minimize the weighted average Hamming distance

$$\zeta = \sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}H_{ij}. \tag{10}$$

To calculate ζ, we transform it to $\xi = \mathrm{tr}(Y^T LY)/4$, where $L = D - W$ is Laplacian matrix and $\mathrm{tr}(\cdot)$ means trace of matrix. At last, we transform ξ to LapEig problem ψ with slacking constraint $y_i \in \{0, 1\}^t$, and obtain the optimal t-dimensional real-valued vector $\tilde{y}$ to represent each sample. ψ is the following:

$$\psi = \underset{\tilde{Y}}{\mathrm{argmin}}\ \mathrm{Tr}\big(\tilde{Y}^T L\tilde{Y}\big)\ \text{s.t.} \begin{cases} \tilde{Y}^T D\tilde{Y} = I \\ \tilde{Y}^T D1 = 0 \end{cases} \tag{11}$$

where $\mathrm{Tr}(\tilde{Y}^T L\tilde{Y})$ gives the real relaxation of the weighted average Hamming distance $\mathrm{Tr}(T^T LY)$. The solution of this optimization problem is given by $\tilde{Y} = [v_1, \ldots, v_t]$ whose columns are the t eigenvectors corresponding to the smallest eigenvalues of following generalized eigenvalue problem. The solution of ψ can be transformed to

$$Lv = \lambda Dv \tag{12}$$

where vector v are the t eigenvectors which are corresponding to the t smallest eigenvalues (nonzero).

Then, we convert the t-dimensional real-valued vectors $\tilde{y_1}, \ldots, \tilde{y_n}$ into binary codes according to the threshold. We set ε^p to present threshold and $\tilde{y_i^p}$ equivalent to pth element of $\tilde{y}_i$. The hash label as final result value of y_i^p is

$$y_i^p = \begin{cases} 1 & \tilde{y_i^p} \geq \varepsilon^p \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where

$$\varepsilon^p = \frac{1}{n}\sum_{i=1}^{n} \tilde{y_i^p}. \tag{14}$$

B. Hash Function Learning Stage

In this stage, we implement an end-to-end hashing deep learning to learn hash function. First, using hash labels acquired in hash label generating stage, we employ CNNs again to receive fine-grained features. After that, we try to adopt two different kinds of encoding module and two different kinds of loss function to approximate hash labels. Before calculating the loss function, we try several different activation functions.

The first kind of encoding module we introduce is MLP. We adopt single-hidden-layer MLP, which is an artificial neural network (ANN) and consists of the input layers, hidden layers, and output layers. It can adjust the input to approximate output according to different weights on different nodes in hidden layers. The reason why we select single-hidden-layer MLP to learn hash labels after CNNs is that CNNs is also a kind of transformation model of MLP. Therefore, we can build an end-to-end deep learning framework of ANN on the multioutput condition. The detail of net structure is shown in Table I.

Formally, we set a function $f : \mathbb{R}^I \to \mathbb{R}^O$, where I is the input set, O is the output set, and x is the input vector. The formulation is

$$\begin{aligned} f^{(1)}(x) &= b^{(2)} + W^{(2)}h\Big(b^{(1)} + W^{(1)}x\Big) \\ f^{(2)}(x) &= b^{(4)} + W^{(4)}h\Big(b^{(3)} + W^{(3)}f^{(1)}(x)\Big) \\ &\dots \end{aligned} \tag{15}$$

where b is bias vector, W is weight matrix of convolution, and $h(*)$ is ReLU and BatchNorm function. Specially, $f^{(n)}$ presents output through $2 \times n$ layers and other $*^{(n)}$ presents parameters on nth layer. In practice, we set $n = 3$ for image size 32×32 and $n = 5$ for image size 96×96.

In the learning process, MLP converges according to the perceptron rules. We set train set $D = \{(I_1, O_1), \dots, (I_m, O_m)\}$, where I_i denotes ith input to MLP as $f_i^{(n)}(x)$ and O_i denotes ith objective output. The perceptron function is

$$E(W) = \sum_{I_i \in M} \big(W^T I_i\big) O_i \tag{16}$$

where W is weight matrix of full connection and M is a set of input vectors who were classified wrong. Moreover, this function subjects to

$$O_i = \begin{cases} +1 & W^T I_i \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{17}$$

Therefore, $E(W)$ is the summary of positive numbers. If all input vectors were classified correctly, $E(W) = 0$. In practice, we use upper bound of $E(W)$ and times of iteration to control MLP convergence. Denote the output as one $m \times d$ matrix (m is the number of samples in batch and d is the number of output in last full connection layer), x is the output vector, y is corresponding label. We define the loss function as follows:

$$F(x) = \min \sum_{i=1}^{m} \sum_{j=1}^{d} \left\| x_i^{(j)} - y_i^{(j)} \right\|_2^2. \tag{18}$$

The other kind of encoding module we refer to is divide and encode module [20]. It is similar to MLP, except that the first full connection layer maps the input into several groups, and each group is followed by one fully connected layer. Different from [20], we split a 1024-D vector into 16 groups, and each group is mapped to q elements. The output number $16 \times q$ is the hash code length. We define the loss function as (18).

Due to the unsteady distribution of output in loss function, the biases are large usually by thresholding values directly. Therefore, we adopt activation function to decrease the bias. The candidates are Sigmoid and BatchNorm [39]. When we use sigmoid, the core of $h(x)$ in (15) is the logistic function $\text{sigmoid}(\alpha) = 1/(1 + e^{-\alpha})$. When we use BatchNorm, the function is calculated as follows:

$$\tilde{x}^{(k)} = \frac{x^{(k)} - E\big(x^{(k)}\big)}{\sqrt{\text{Var}\big(x^{(k)}\big)}} \tag{19}$$

where

$$E(x) = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{20}$$

$$\text{Var}(x) = \frac{1}{m} \sum_{i=1}^{m} (x_i - E(x))^2. \tag{21}$$

When we choose Sigmoid as activation function, the threshold function is defined as follows:

$$x_i = \begin{cases} 1 & x_i \geq 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

On the other hand, using BatchNorm as activation function, we define the threshold function as the same as (13) and (14). Usually, we apply the threshold values of each bit calculated in the hash label generating stage.

IV. Experiments

In this section, we first design the comparative experiments to compare the effect of different networks combined with different configuration parameters without hand-crafted labels. In detail, we try MLP and Slice net structure with different mapping functions and activation functions. Afterward, using hand-crafted labels or not, we compare our method with some supervised algorithms and some unsupervised algorithms, respectively. At last, we give the experimental analysis and sum up some experience for different datasets. The results show the superiority of our algorithm on traditional datasets and our ability of solving out-of-sample problem on challenging datasets.

A. Experiments Setting

Some abbreviations for evaluation and configuration will be used in this paper as follows.

1) *RP:* Recall and Precision.
2) *RIP:* Returned Image Precision.
3) *NC3P:* Precision for searched images in nearest 3 classes.
4) *HD3P:* Precision for searched images of which the Hamming distance is less than or equal to 3.
5) *s:* SimpleNet.
6) *g:* GoogLeNet.
7) *a:* AlexNet.
8) *mb:* MLP+BatchNorm.
9) *ms:* MLP+Sigmoid.
10) *sb:* Slice+BatchNorm.
11) *ss:* Slice+Sigmoid.

The experiments are implemented on benchmark data sets of CIFAR-10, CIFAR-100, and STL-10. We summarize these data sets and corresponding preprocessing as follows.

1) *CIFAR-10:* CIFAR-10 are labeled subsets of the 80 million tiny images dataset, which consists of 60 000 32×32

TABLE I
SIMPLE CNNS FOR DIFFERENT DATASET

	step	type	kernel size	pad	stride	output	pool style
CIFAR-10 CIFAR-100	1	conv	5×5	2	1	32	
	2	pool	3×3	0	2	1	*max*
	3	conv	5×5	2	1	32	
	4	pool	3×3	0	2	1	*ave*
	5	conv	5×5	2	1	64	
	6	pool	3×3	0	2	1	*ave*
STL-10 Oxford 105K Holidays+1M Flickr Logo32	1	conv	5×5	2	1	32	
	2	pool	3×3	0	2	1	*max*
	3	conv	5×5	2	1	32	
	4	pool	3×3	0	2	1	*ave*
	5	conv	5×5	2	1	64	
	6	pool	3×3	0	2	1	*ave*
	7	conv	5×5	2	1	64	
	8	pool	3×3	0	2	1	*ave*
	9	conv	5×5	2	1	128	
	10	pool	3×3	0	2	1	*ave*

color images in ten classes, with 6000 images per class. There are 5000 training images and 1000 test images in each class.

2) *CIFAR-100:* CIFAR-100 is similar to the CIFAR-10, except that it has 100 classes containing 600 images in each class. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 super-classes. Each image comes with a *fine* label (the class to which it belongs) and a *coarse* label (the superclass to which it belongs).

3) *STL-10:* STL-10 is a subset of ImageNet dataset for image recognition, which consists of 10 classes, 5000 training images (500 images per class), 8000 test images (800 images per class), and 1 000 000 unlabeled images, each of size 96×96 pixels. We select 5000 training images and 8000 test images to complete our experiments.

4) *Oxford Buildings:* This dataset contains two subdatasets: Oxford 5k, which contains 5062 high revolutionary (1024×768) images, including 11 different landmarks, and each represented by several possible queries; and Oxford 105K, which combines Oxford 5K with 100 000 distractors to allow for evaluation of scalability [40].

5) *INRIA Holidays:* This dataset includes a very large variety of scene types (natural, man-made, water and fire effects, etc.) and contains 500 image groups, each of which represents a distinct scene or object. Dataset size is 1491 which contains 500 queries and 991 corresponding relevant images. The other supplemental dataset is one million images, which are stored in 1000 archives of 10 000 feature files each [41].

6) *Flickr Logo32:* This dataset is a subset of Flickr, which consists of 32 different logos, 320 training images, 960 validation images and 960 test images. At the same time, there are 3000 nonlogo images in validation and test, respectively, [42].

We adopt SimpleNet to extract features in hash function learning stage. For different datasets, the SimpleNet structures are shown in Table I. In particular, we resized images size to 96×96 for Oxford Buildings, INRIA Holidays, and Flickr Logo32 using cubic interpolation when we apply SimpleNet.

TABLE II
CLASSIFICATION PRECISION USING DIFFERENT NETS

	AlexNet	GoogLeNet	SimpleNet
CIFAR-10	0.9058	0.9059	0.8169
STL-10	0.8323	0.8192	0.6046
CIFAR-100 coarse	0.7231	0.7462	0.6381
CIFAR-100 fine	0.6136	0.6272	0.5355
Oxford Buildings	-	-	-
INRIA Holidays	-	-	-
Flickr Logo32	-	-	0.5216

The classification precision results generated by deep learning using different nets are listed in Table II. In particular, we resized image size to 256×256 using cubic interpolation when we apply AlexNet, GoogLeNet, and VGG16.

B. Component Analysis

In order to verify the effect of the algorithm on different datasets in the absence of classification labels, we configure different networks (Slice and MLP) and activation functions (BatchNorm and Sigmoid) to execute DSTH. In the hash label generating stage, we will try applying two different CNNs (AlexNet and GoogLeNet) with model (trained on ImageNet) to generating features.

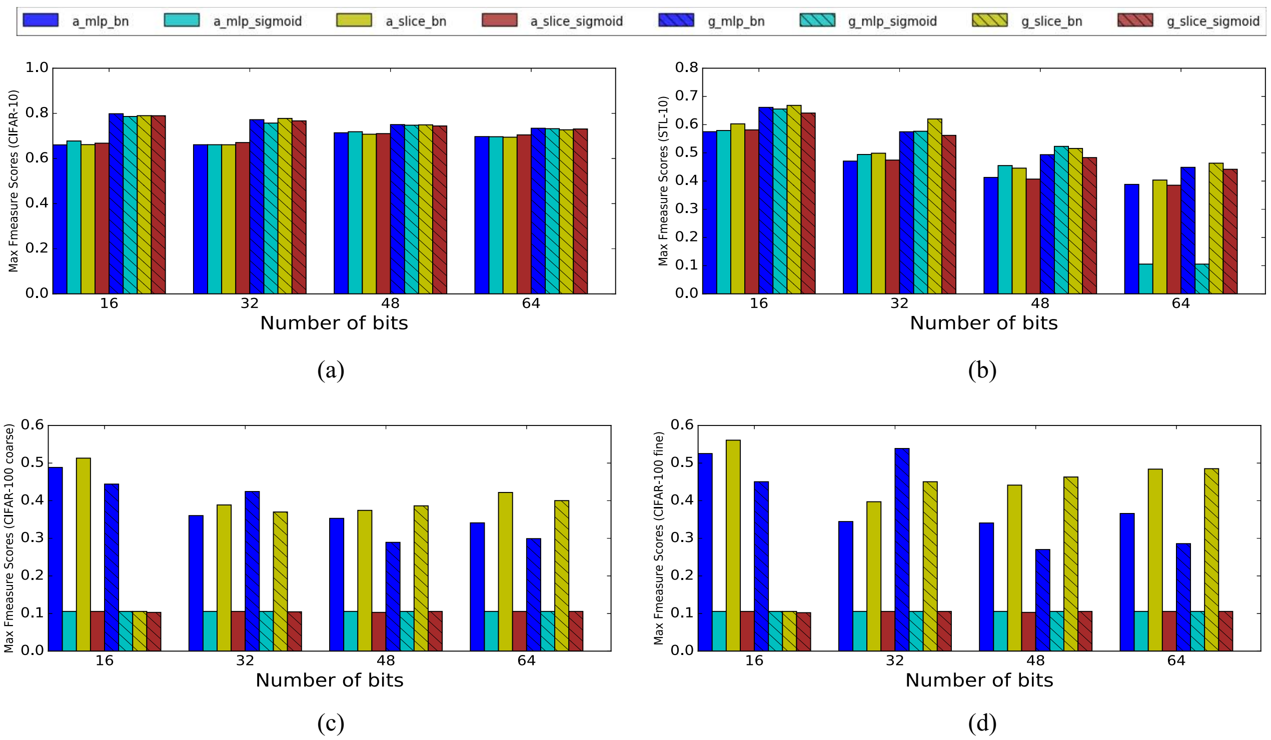
In Fig. 2, we show Max F-measure score by using different datasets and hash code lengths with $\beta = 0.5$. We show the performance with 48-bits hash codes on CIFAR-10 in Fig. 3, 32-bits hash codes on STL-10 in Fig. 4, 48-bits hash codes on CIFAR-100 coarse in Fig. 5, and 48-bits hash codes on CIFAR-100 fine in Fig. 6, respectively. The most representative details of PR and F-measure comparison on different datasets are shown in Table III. Besides, to highlight the curve, the data of ave_RP, max _RPFscore, ave_RIP, and max _RIPFscore in the radar figure of Fig. 3(b) are multiplied by 1.23.

According to Table II and the above results, we find that better classification results yield better hash labels and retrieval performance (there exist better performances when using hash labels derived from the feature extracted by GoogLeNet than that by AlexNet). Moreover, Slice+BatchNorm has a higher chance to get better results. See detailed analysis in Section IV-E.

C. Algorithm Comparison

In this part, we compare our method with other hash algorithms on CIFAR-10 and STL-10 datasets. We adopt the parameter model trained on ImageNet with GoogLeNet to extract features for hash label generation. Specially, Slice+BatchNorm are introduced in the hash function learning stage. Within the experiment, we set the number of nearest neighbor as $\mu = 12$ in LE. In the process of learning hash function, we set momentum as $\xi = 0.9$, weight decay as $\varpi = 0.004$ in SimpleNet, and execute SimpleNet 60 000 times with learning rate $lr = 0.01$, then finetune features 5000 times with $lr = 0.001$ and $lr = 0.0001$, respectively.

In order to verify the independence of our algorithm on the hand-crafted labels, we design two groups of experiments.



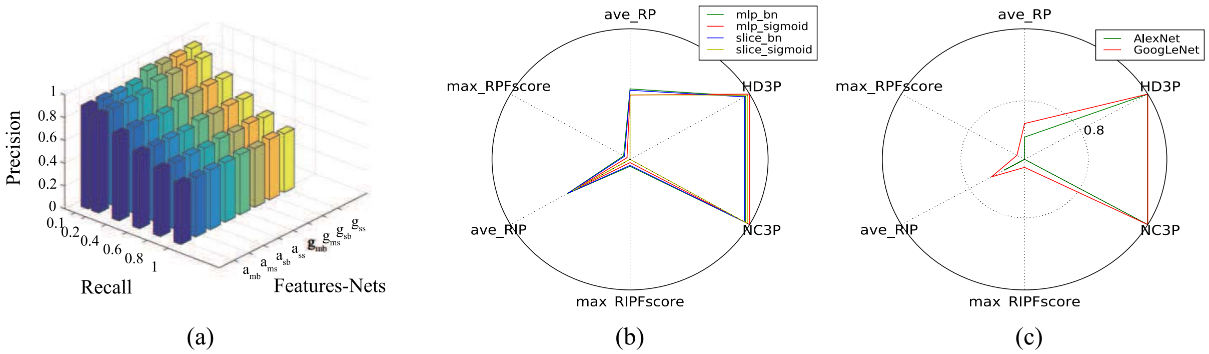
(a)

(b)

(c)

(d)

Fig. 2. Max F-measure score ($\beta = 0.5$) under different hash length on different data. (a) CIFAR-10. (b) STL-10. (c) CIFAR-100 coarse. (d) CIFAR-100 fine.

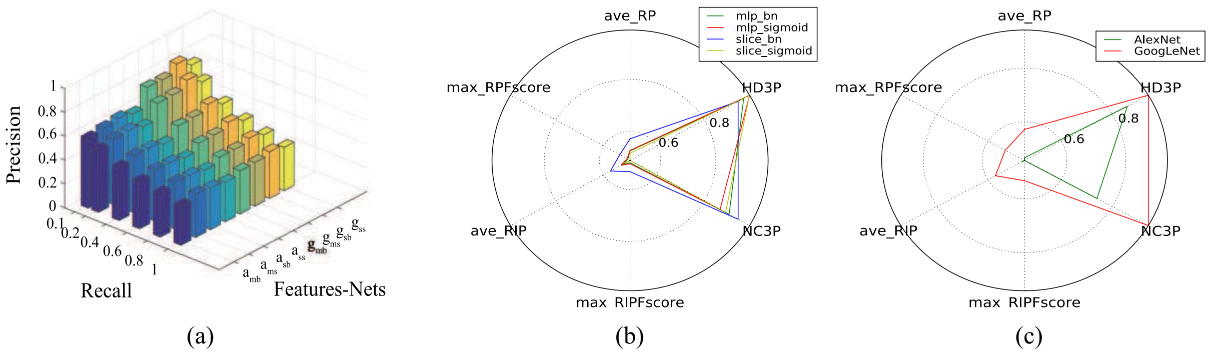


(a)

(b)

(c)

Fig. 3. Contrast on 48-bits codes CIFAR-10 without class label. (a) PR curve, (b) performance of different net and activation function config under GoogLeNet feature, and (c) performance of different feature under Slice net with BatchNorm activation function.



(a)

(b)

(c)

Fig. 4. Contrast on 32-bits codes STL-10 without class labels. (a) PR curve, (b) performance of different net and activation function config under GoogLeNet feature, and (c) performance of different feature under Slice net with BatchNorm activation function.

In the first group of experiments, we do not use hand-crafted labels in datasets. Therefore, we compare DSTH with several unsupervised hash algorithms, including SKLSH, SH, PCA-ITQ, STH, SpH, DSH, LSH. Fig. 7(a)–(d) shows the PR and RIP performance of 48-bits codes for CIFAR-10 and 32-bits codes for STL-10. Fig. 7(e)–(h) shows the

TABLE III
RECALL-PRECISION AND MAX F-MEASURE SCORE FOR PARAMETER CONFIGURATION

			Recall						Max F-measure Score	
			0.1	0.2	0.4	0.6	0.8	1	β=0.5	β=0.25
CIFAR-10	AlexNet 48-bits	MLP+BatchNorm	0.88767	0.84391	0.75071	0.65969	0.58496	0.52461	0.64682	0.71389
		MLP+Sigmoid	0.89313	0.84808	0.75619	0.66001	0.58364	0.52302	**0.64706**	**0.71855**
		Slice+BatchNorm	0.88415	0.83454	0.74328	0.65784	0.58301	0.52167	0.64539	0.70756
		Slice+Sigmoid	0.89015	0.83938	0.74640	0.65597	0.57976	0.52023	0.64395	0.71022
	GoogLeNet 48-bits	MLP+BatchNorm	0.94992	0.89647	0.79357	0.68076	0.59624	0.52627	**0.66769**	**0.75016**
		MLP+Sigmoid	0.94352	0.88895	0.79048	0.68129	0.58734	0.51524	0.66332	0.74755
		Slice+BatchNorm	0.94169	0.89300	0.79224	0.68560	0.59855	0.59855	0.66658	0.74904
		Slice+Sigmoid	0.94953	0.89673	0.78618	0.67625	0.58462	0.51472	0.65949	0.74422
STL-10	AlexNet 32-bits	MLP+BatchNorm	0.58109	0.51425	0.44047	0.39117	0.36245	0.32596	0.43173	0.47074
		MLP+Sigmoid	0.60330	0.54419	0.47570	0.42716	0.39257	0.36308	**0.45835**	0.49417
		Slice+BatchNorm	0.61890	0.55003	0.47312	0.41947	0.38555	0.34949	0.45643	**0.49869**
		Slice+Sigmoid	0.59215	0.51883	0.44501	0.39706	0.36487	0.33193	0.43521	0.47435
	GoogLeNet 32-bits	MLP+BatchNorm	0.74947	0.65085	0.52989	0.45390	0.40595	0.36076	0.49757	0.57465
		MLP+Sigmoid	0.74691	0.65334	0.53541	0.45669	0.40493	0.36517	0.50146	0.57647
		Slice+BatchNorm	0.81520	0.71389	0.57970	0.49731	0.43375	0.39043	**0.53190**	**0.62016**
		Slice+Sigmoid	0.73990	0.63360	0.51734	0.44617	0.39684	0.36194	0.48867	0.56193
CIFAR-100 coarse	AlexNet 48-bits	MLP+BatchNorm	0.41351	0.37074	0.32315	0.29636	0.27661	0.25846	**0.33606**	0.35301
		MLP+Sigmoid	0.10000	0.10000	0.10000	0.10000	0.10000	0.10000	0.12195	0.10559
		Slice+BatchNorm	0.45167	0.38353	0.31272	0.27238	0.24542	0.22741	0.32699	**0.37425**
		Slice+Sigmoid	0.10000	0.10000	0.10000	0.10000	0.10000	0.10000	0.12195	0.10559
	GoogLeNet 48-bits	MLP+BatchNorm	0.32804	0.28509	0.25054	0.23161	0.21794	0.20829	0.27077	0.28924
		MLP+Sigmoid	0.10000	0.10000	0.10000	0.10000	0.10000	0.10000	0.12195	0.10559
		Slice+BatchNorm	0.47048	0.40016	0.32879	0.28925	0.26058	0.24204	**0.34093**	**0.38630**
		Slice+Sigmoid	0.10000	0.10000	0.10000	0.10000	0.10000	0.10000	0.12195	0.10559
CIFAR-100 fine	AlexNet 48-bits	MLP+BatchNorm	0.40111	0.30000	0.24820	0.22033	0.20443	0.18949	0.27272	0.34075
		MLP+Sigmoid	0.10000	0.10000	0.10000	0.10000	0.10000	0.10000	0.12195	0.10559
		Slice+BatchNorm	0.56111	0.42526	0.32000	0.27033	0.24278	0.21737	**0.34707**	**0.44138**
		Slice+Sigmoid	0.10000	0.10000	0.10000	0.10000	0.10000	0.10000	0.12195	0.10559
	GoogLeNet 48-bits	MLP+BatchNorm	0.30222	0.25947	0.20538	0.17372	0.15911	0.14464	0.24490	0.27009
		MLP+Sigmoid	0.10000	0.10000	0.10000	0.10000	0.10000	0.10000	0.12195	0.10559
		Slice+BatchNorm	0.59888	0.47894	0.35179	0.30677	0.26113	0.24565	**0.37448**	**0.46301**
		Slice+Sigmoid	0.10000	0.10000	0.10000	0.10000	0.10000	0.10000	0.12195	0.10559

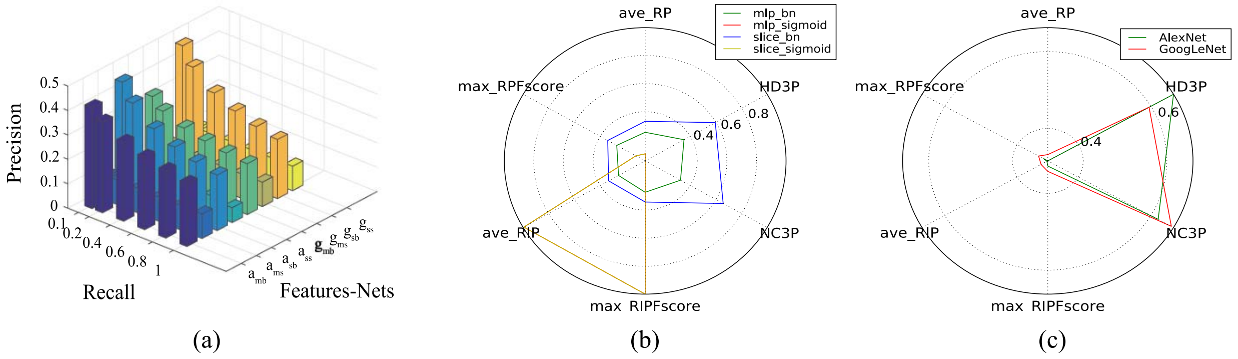
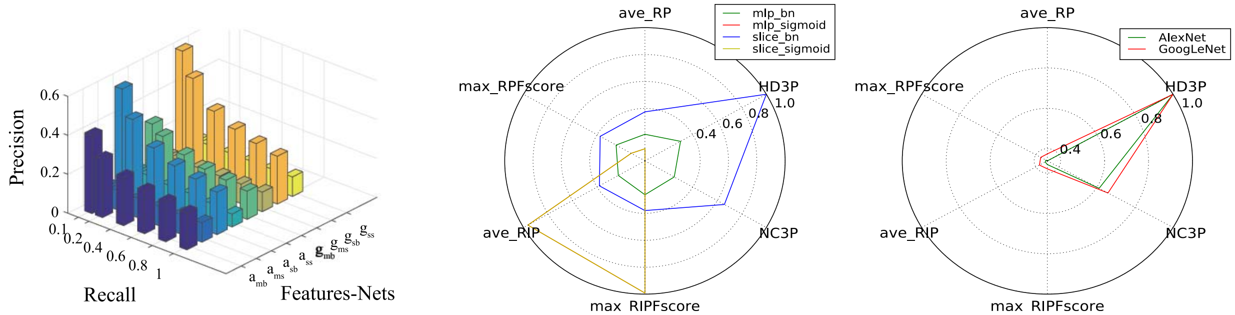


Fig. 5. Contrast on 48-bits codes CIFAR-100 coarse without class label. (a) PR curve, (b) performance of different net and activation function config under GoogLeNet feature, and (c) performance of different feature under Slice net with BatchNorm activation function.

code length analysis of CIFAR-10 and STL-10 on NC3P and HD3P.

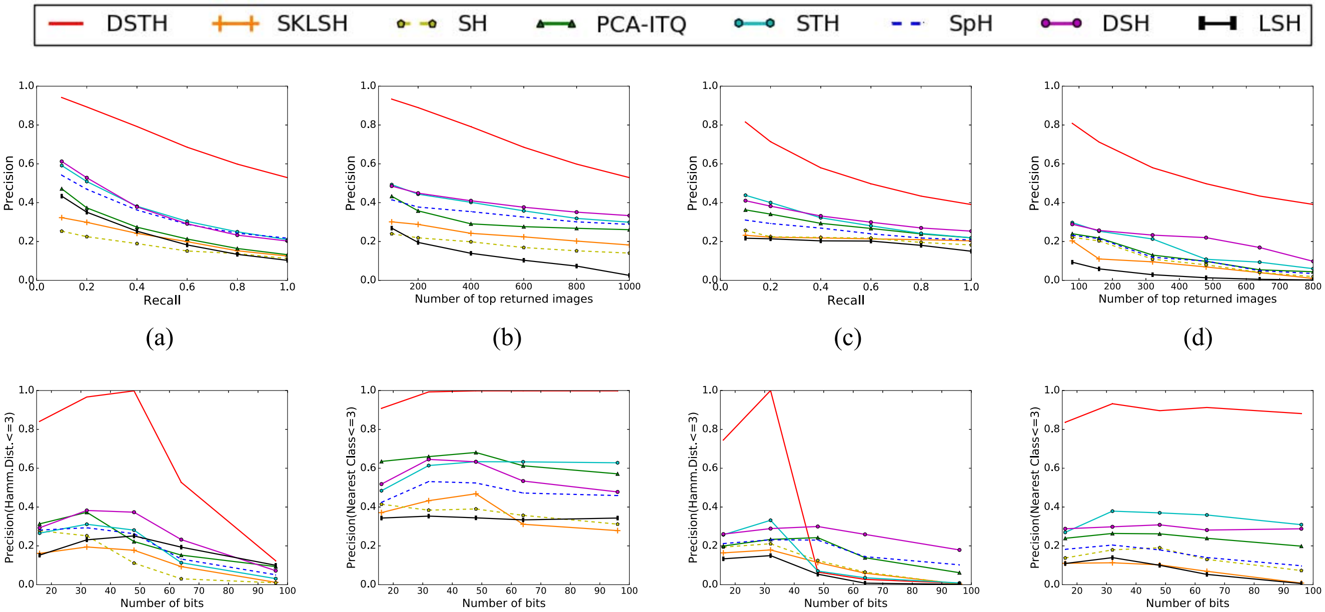
In the second group of experiments, we use classification labels in datasets to finetune features during hash label generating stage. We compare DSTH with several supervised algorithms, including CNNH, KSH, BRE, and CCA-ITQ. Fig. 7(a)–(d) shows the PR and RIP performance of 48-bits codes for CIFAR-10 and 32-bits codes for STL-10. Fig. 7(e)–(h) shows the code length analysis of CIFAR10 and STL10 on NC3P and HD3P. Besides, Table IV shows the performance of DSTH with labels or not, including the PR and score of 48-bits codes for CIFAR-10 and 32-bits codes for STL-10.

Above results show the superiority of DSTH. Besides, for DSTH, compared with generating hash codes without labels, the improvement of performance by using classification labels is not outstanding. Therefore, using DSTH without labels is enough. See detailed analysis on Section IV-E.



(a) (b) (c)

Fig. 6. Contrast on 48-bits codes CIFAR-100 fine without class label. (a) PR curve, (b) performance of different net and activation function config under GoogLeNet feature, and (c) performance of different feature under Slice net with BatchNorm activation function.



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 7. Performance contrast without class label (a) 48-bits codes precision-recall curve on CIFAR-10, (b) 48-bits codes return image precision curve on CIFAR-10, (c) 32-bits codes precision-recall curve on STL-10, (d) 32-bits codes return image precision curve on STL-10, (e) code length analyze with Hamming distance ≤ 3 on CIFAR-10, (f) code length analyze with nearest class ≤ 3 on CIFAR-10, (g) code length analyze with Hamming distance ≤ 3 on STL-10, and (h) code length analyze with nearest class ≤ 3 on STL-10.

TABLE IV
RECALL-PRECISION AND MAX *F*-MEASURE SCORE CONTRAST WITHOUT-LABELS AND WITH-LABELS

			Recall						Max F-measure Score	
			0.1	0.2	0.4	0.6	0.8	1	β=0.5	β=0.25
CIFAR-10	48-bits	without-labels with-labels	0.94169 0.98192	0.89300 0.93909	0.79224 0.74940	0.68560 0.61585	0.59855 0.52583	0.59855 0.46436	0.66658 0.63795	0.74904 0.77140
STL-10	32-bits	without-labels with-labels	0.81520 0.89382	0.71389 0.82180	0.57970 0.68041	0.49731 0.57600	0.43375 0.50451	0.39043 0.44584	0.53190 0.59674	0.62016 0.69474

D. Comparison With State-of-the-Art of Single Target Deep Hashing Methods

For verifying of the effectiveness of DSTH further, we compare ours with the state-of-the-art of single target deep hashing algorithms on more challenging datasets. Comparison methods include DLBH [19], DeepBit [43], AIBC [44], DSH [45], and SSDH [25]. Datasets include Oxford Buildings, INRIA Holidays, and Flickr Logo32 which have less training samples for each class and more out-of-class samples. Different from previous experiments, we select VGG16 with model [46] which has been calculated on ILSVRC to generate deep features. Table V shows results of mAP and HD3P of the compared methods with 16-bits codes and 32-bits codes.

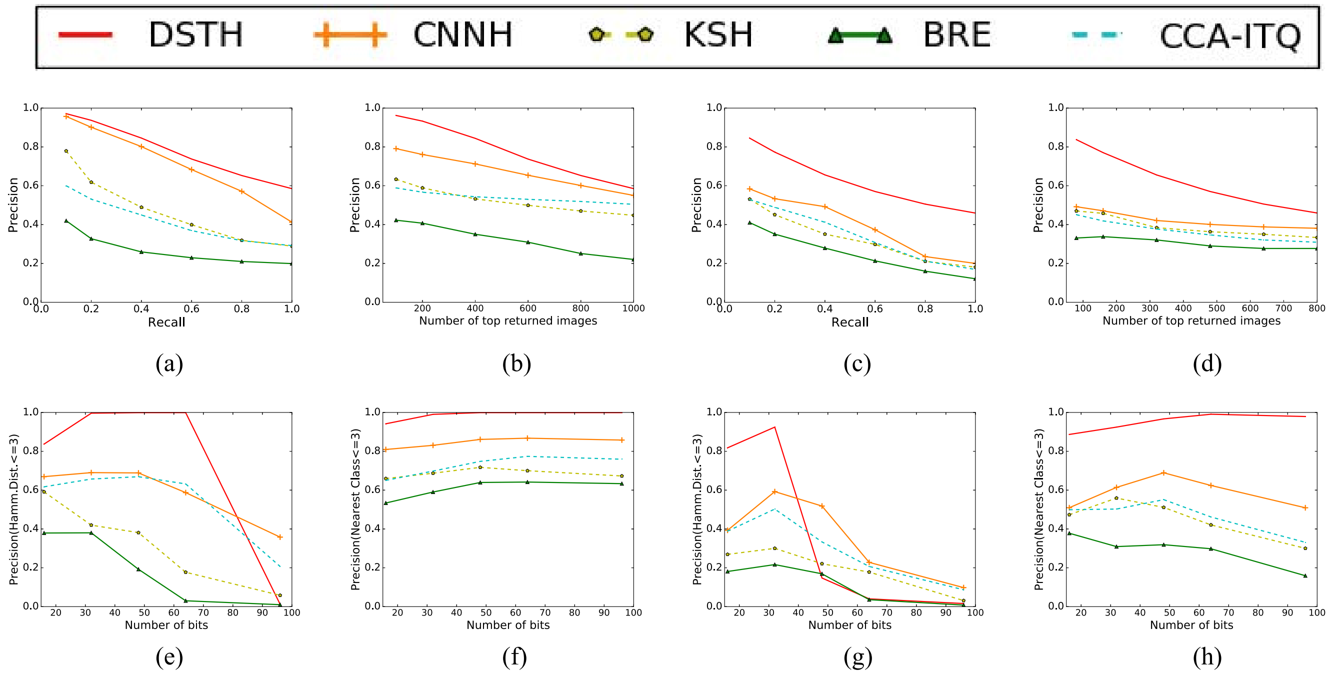


Fig. 8. Performance contrast with class label (a) 48-bits codes precision-recall curve on CIFAR-10, (b) 48-bits codes return image precision curve on CIFAR-10, (c) 32-bits codes precision-recall curve on STL-10, (d) 32-bits codes return image precision curve on STL-10, (e) code length analyze with Hamming distance ≤ 3 on CIFAR-10, (f) code length analyze with nearest class ≤ 3 on CIFAR-10, (g) code length analyze with Hamming distance ≤ 3 on STL-10, and (h) code length analyze with nearest class ≤ 3 on STL-10.

The results shown in Table V approve that DSTH has better effects on those datasets. Although it is not the best in HD3P, it has advantage in mAP because of ability of solving out-of-sample problem. See detailed analysis on Section IV-E.

E. Experimental Analysis

Experiments in Sections IV-B and IV-C demonstrate the effectiveness of our algorithm. Referring to Tables II and III, it is obvious that DSTH is sensitive to different datasets. Generally, the better classification performance is, the better performance of hash is, which derive from the effective hash labels generated by accurate features. Meanwhile, with different feature extraction methods, our algorithm shows different adaptability to different net settings. We illustrate the performance with different datasets, respectively.

1) *CIFAR-10:* It shows the best classification performance with every net in Table II and the highest *F*-measure score in Table III. Apart from that, results in Fig. 3(c) shows net configurations comparison with 48-bits codes, it is obvious that GoogLeNet is superior to AlexNet, satisfying the analysis that better classification results bring better hash results. In Fig. 3(b), MLP+sigmoid generates the largest effective area, while Slice+BatchNorm leads to balanced performance, which are similar to Figs. 2(a) and 7(a), (b), (e), and (f). We use CIFAR-10 to demonstrate our algorithm on small size images, which has good classification performance. Experiment results show our algorithm brings stable and efficient performance.
2) *STL-10:* Compared with CIFAR-10, the classification performance of STL-10 on three nets decreases by 9%, because lacking training sample leads to weak hash codes. According to Table III, GoogLeNet and slice+BatchNorm show much better performance than other cases. Fig. 4 also shows this case yields largest effective area. Although Slice+BatchNorm maintains the distribution of image feature to the largest extent in the process of mapping, the accuracy of feature extraction

TABLE V
RESULTS IN TERMS OF MAP AND HD3P OF THE COMPARED METHODS ON OXFORD BUILDINGS, INRIA HOLIDAYS, AND FLICKR LOGO32 WITH 16-BITS CODES AND 32-BITS CODES, RESPECTIVELY

Datasets	Methods	mAP		HD3P	
		16-bits	32-bits	16-bits	32-bits
Oxford 105K	DLBH	0.4453	0.5073	0.4108	0.4736
	DeepBit	0.3331	0.3707	0.3091	0.3568
	AIBC	0.3809	0.4399	0.3505	0.3952
	DSH	0.5233	0.5938	0.4731	0.5099
	SSDH	0.5811	0.6576	**0.4820**	0.5316
	DSTH	**0.5881**	**0.6737**	0.4618	**0.5392**
INRIA Holidays+ 1M	DLBH	0.3603	0.4113	0.3066	0.3715
	DeepBit	0.3092	0.3579	0.2717	0.3062
	AIBC	0.3189	0.3613	0.2802	0.3177
	DSH	0.3883	0.4230	0.3125	0.3849
	SSDH	0.4090	0.4303	**0.3356**	**0.3911**
	DSTH	**0.4107**	**0.4588**	0.3197	0.3816
Flickr Logo32	DLBH	0.3699	0.3920	0.2941	0.3273
	DeepBit	0.3333	0.3814	0.2238	0.2857
	AIBC	0.3559	0.3969	0.3030	0.3582
	DSH	0.3902	0.4481	0.3166	0.3551
	SSDH	0.4297	0.5018	0.3346	0.4074
	DSTH	**0.4344**	**0.5102**	**0.3395**	**0.4229**

is limited. Figs. 2(b) and 7 (c), (d), (g), and (h) demonstrate above analysis, so does Fig. 8(g) with labels. We use STL-10 to demonstrate our algorithm on large size images, which has mediocre classification performance. Experiment results show our algorithm brings good performance with limited hash length, which is not stable.

3) *CIFAR-100:* We adopt CIFAR-100 to explore the same hash codes derived from different grain sizes of classification. From Table III and Fig. 2(c) and (d), Slice+BatchNorm mostly shows superior performance, Figs. 5(c) and 6(c) display the difference of searching nearest three classes on different grain sizes of classification. However, they show little difference through hamming distance, illustrating the adaptability of DSTH.

Experiments in Section IV-D reveal the ability of solving out-of-sample problem of our algorithm. Referring to Table V, even if HD3P results of DSTH are not dominant, it still shows superior performance in mAP contrast with state-of-the-art single target hashing methods. Of course, compared with the hashing algorithms [21], [22] with RPN [13] structure or detection process, there exists a gap between the results. We illustrate the performance on different datasets, respectively.

1) *Oxford 105K:* It shows best mAP performance with 16-bits codes and 32-bits codes in Table V. The performance is improved by 0.7% and 1.61%, respectively. Meanwhile, the HD3P performance is 0.76% higher than SSDH with 32-bits codes.
2) *INRIA Holidays+1M:* Compared with Oxford, the mAP performance of Holidays decreases by 20%, because more unlabeled samples are introduced. However, our performances are higher than that of SSDH by 0.17% and 2.85% with 16-bits codes and 32-bits codes, respectively, illustrating the ability of solving out-of-sample problem.
3) *Flickr Logo32:* We adopt FL32 to explore our ability of recognizing small targets. The results show that our performances are higher than that of SSDH by 0.47% and 0.84% in mAP as well as 0.49% and 1.55% in HD3P with 16-bits codes and 32-bits codes, respectively, illustrating the superiority of ours.

It is worth noting that, we also try a combination of using SimpleNet for feature extraction in hash label generating stage and using ReLU as the activation function in MLP and Slice nets, which shows bad performance. Apart from the factor of grain size of classification, we believe that using the same net in both stages leads to weak results, and using ReLU in MLP and Slice nets destroys the distribution when mapping features to hash codes.

V. Conclusion

In this paper, we propose a novel self-taught hash algorithm under deep learning scheme (DSTH). It is a unified model to get the hash codes of the training data, for both supervised and unsupervised cases. During stages of hash label generating and hash function learning, it not only generates hash labels with guaranteed precision under no label condition, but also does not consume more generation time for hash codes. Although deep network in hash label generating stage leads to high time consumption, the process is off-line. Comparative experiments show the superiority of our scheme on solving out-of-sample problem, especially without hand-crafted labels. Furthermore, we summarize several application principles for using DSTH according to configuration of network parameters. On the one hand, the model which brings better classification results will bring better features for hash labels generating. The better the hash labels are, the better the learned hash functions are. On the other hand, using combination of Slice network and BatchNorm activation function have a higher chance to get good results.

References

[1] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions,” in *Proc. 20th ACM Symp. Comput. Geom.*, Brooklyn, NY, USA, Jun. 2004, pp. 253–262.

[2] B. Kulis and K. Grauman, “Kernelized locality-sensitive hashing,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 6, pp. 1092–1104, Jun. 2012.

[3] J. Song, L. Gao, L. Liu, X. Zhu, and N. Sebe, “Quantization-based hashing: A general framework for scalable image and video retrieval,” *Pattern Recognit.*, vol. 75, pp. 175–187, Mar. 2018.

[4] Y. Weiss, A. Torralba, and R. Fergus, “Spectral hashing,” in *Proc. NIPS*, Vancouver, BC, Canada, 2008, pp. 1753–1760.

[5] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, “Hashing with graphs,” in *Proc. ICML*, Bellevue, WA, USA, 2011, pp. 1–8.

[6] Y. Gong and S. Lazebnik, “Iterative quantization: A procrustean approach to learning binary codes,” in *Proc. CVPR*, Colorado Springs, CO, USA, 2011, pp. 817–824.

[7] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang, “Supervised hashing with kernels,” in *Proc. CVPR*, Providence, RI, USA, 2012, pp. 2074–2081.

[8] B. Kulis and T. Darrell, “Learning to hash with binary reconstructive embeddings,” in *Proc. NIPS*, Vancouver, BC, Canada, 2009, pp. 1042–1050.

[9] D. Zhang, J. Wang, D. Cai, and J. Lu, “Self-taught hashing for fast similarity search,” in *Proc. ACM SIGIR*, Geneva, Switzerland, 2010, pp. 18–25.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Int. Conf. Artif. Neural Netw.*, 2012, pp. 1106–1114.

[11] Y. Jia *et al.*, “Caffe: Convolutional architecture for fast feature embedding,” in *Proc. ACM MM*, Orlando, FL, USA, 2014, pp. 675–678.

[12] J. Donahue *et al.*, “DeCAF: A deep convolutional activation feature for generic visual recognition,” in *Proc. ICML*, Beijing, China, 2014, pp. 647–655.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial pyramid pooling in deep convolutional networks for visual recognition,” in *Proc. ECCV*, 2014, pp. 346–361.

[14] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in *Proc. NIPS*, 2015, pp. 91–99.

[15] L. Gao, Z. Guo, H. Zhang, X. Xu, and H. T. Shen, “Video captioning with attention-based LSTM and semantic consistency,” *IEEE Trans. Multimedia*, vol. 19, no. 9, pp. 2045–2055, Sep. 2017.

[16] X. Wang, L. Gao, P. Wang, X. Sun, and X. Liu, “Two-stream 3-D convNet fusion for action recognition in videos with arbitrary size and length,” *IEEE Trans. Multimedia*, vol. 20, no. 3, pp. 634–644, Mar. 2018.

[17] X. Wang *et al.*, “Deep appearance and motion learning for egocentric activity recognition,” *Neurocomputing*, vol. 275, pp. 438–447, Jan. 2018.

[18] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, “Supervised hashing for image retrieval via image representation learning,” in *Proc. AAAI*, Quebec City, QC, Canada, 2014, pp. 2156–2162.

[19] K. Lin, H.-F. Yang, J.-H. Hsiao, and C.-S. Chen, “Deep learning of binary hash codes for fast image retrieval,” in *Proc. CVPR*, Boston, MA, USA, 2015, pp. 27–35.

[20] H. Lai, Y. Pan, Y. Liu, and S. Yan, “Simultaneous feature learning and hash coding with deep neural networks,” in *Proc. CVPR*, Boston, MA, USA, 2015, pp. 3270–3278.

[21] L. Zheng *et al.*, "Query-adaptive late fusion for image search and person re-identification," in *Proc. CVPR*, Boston, MA, USA, 2015, pp. 1741–1750.

[22] J. Song, T. He, L. Gao, X. Xu, and H. T. Shen, "Deep region hashing for efficient large-scale instance search from images," in *Proc. AAAI*, 2018, pp. 2156–2162.

[23] J. Song, "Binary generative adversarial networks for image retrieval," in *Proc. AAAI*, 2018.

[24] T.-T. Do, A.-D. Doan, and N.-M. Cheung, "Learning to hash with binary deep neural network," in *Proc. ECCV*, Amsterdam, The Netherlands, 2016, pp. 219–234.

[25] H.-F. Yang, K. Lin, and C.-S. Chen, "Supervised learning of semantics-preserving hash via deep convolutional neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 2, pp. 437–451, Feb. 2018.

[26] L. Gao, J. Song, F. Zou, D. Zhang, and J. Shao, "Scalable multimedia retrieval by deep learning hashing with relative similarity learning," in *Proc. ACM MM*, Brisbane, QLD, Australia, 2015, pp. 903–906.

[27] Y. Gu, C. Ma, and J. Yang, "Supervised recurrent hashing for large scale video retrieval," in *Proc. ACM MM*, Amsterdam, The Netherlands, 2016, pp. 272–276.

[28] K. Zhou *et al.*, "Deep self-taught hashing for image retrieval," in *Proc. ACM MM*, Brisbane, QLD, Australia, 2015, pp. 1215–1218.

[29] X. Liu, J. He, B. Lang, and S.-F. Chang, "Hash bit selection: A unified solution for selection problems in hashing," in *Proc. CVPR*, Portland, OR, USA, 2013, pp. 1570–1577.

[30] J. Song *et al.*, "A distance-computation-free search scheme for binary code databases," *IEEE Trans. Multimedia*, vol. 18, no. 3, pp. 484–495, Mar. 2016.

[31] W. Kong, W.-J. Li, and M. Guo, "Manhattan hashing for large-scale image retrieval," in *Proc. ACM SIGIR*, Portland, OR, USA, 2012, pp. 45–54.

[32] D. Zhang, F. Wang, and L. Si, "Composite hashing with multiple information sources," in *Proc. ACM SIGIR*, Beijing, China, 2011, pp. 225–234.

[33] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *Proc. CVPR*, Boston, MA, USA, 2015, pp. 2475–2483.

[34] F. Zhao, Y. Huang, L. Wang, and T. Tan, "Deep semantic ranking based hashing for multi-label image retrieval," in *Proc. CVPR*, Boston, MA, USA, 2015, pp. 1556–1564.

[35] J. Song, H. Zhang, X. Li, L. Gao, M. Wang, and R. Hong, "Self-supervised video hashing with hierarchical binary auto-encoder," *IEEE Trans. Image Process.*, vol. 27, no. 7, pp. 3210–3221, Jul. 2018.

[36] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 769–790, Apr. 2018.

[37] J. Song *et al.*, "Optimized graph learning using partial tags and multiple features for image and video annotation," *IEEE Trans. Image Process.*, vol. 25, no. 11, pp. 4999–5011, Nov. 2016.

[38] L. Gao *et al.*, "Learning in high-dimensional multimedia data: The state of the art," *Multimedia Syst.*, vol. 23, no. 3, pp. 303–313, 2017.

[39] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML*, Lille, France, 2015, pp. 448–456.

[40] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object retrieval with large vocabularies and fast spatial matching," in *Proc. CVPR*, Minneapolis, MN, USA, 2007, pp. 1–8.

[41] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *Proc. ECCV*, Marseille, France, 2008, pp. 304–317.

[42] S. Romberg, L. G. Pueyo, R. Lienhart, and R. van Zwol, "Scalable logo recognition in real-world images," in *Proc. ICMR*, Trento, Italy, 2011, p. 25.

[43] K. Lin, J. Lu, C.-S. Chen, and J. Zhou, "Learning compact binary descriptors with unsupervised deep neural networks," in *Proc. CVPR*, Las Vegas, NV, USA, 2016, pp. 1183–1192.

[44] F. Shen, W. Liu, S. Zhang, Y. Yang, and H. T. Shen, "Learning binary codes for maximum inner product search," in *Proc. ICCV*, Santiago, Chile, 2015, pp. 4148–4156.

[45] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," in *Proc. CVPR*, Las Vegas, NV, USA, 2016, pp. 2064–2072.

[46] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, Sep. 2014.

Yu Liu received the B.S. and M.S. degrees in computer science and technology from the Wuhan Institute of Technology, Wuhan, China, in 2008 and 2012, respectively, and the Ph.D. degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, in 2017.

He is currently a Post-Doctoral Researcher with the Department of Software Engineering, HUST. His current research interests include machine learning, large-scale multimedia search, and big data and storage.

Jingkuan Song received the Ph.D. degree in information technology from the University of Queensland, Brisbane, QLD, Australia, in 2014.

He is a Professor with the University of Electronic Science and Technology of China, Chengdu, China. He joined Columbia University, New York, NY, USA, as a Post-Doctoral Research Scientist from 2016 to 2017, and the University of Trento, Trento, Italy, as a Research Fellow from 2014 to 2016. His current research interest includes large-scale multimedia retrieval, image/video segmentation and image/video annotation using hashing, graph learning, and deep learning techniques.

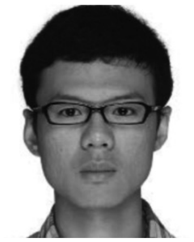
Dr. Song is a Guest Editor of the IEEE TRANSACTIONS ON MULTIMEDIA, *World Wide Web Journal* and he is a PC Member of MM'18, AAAI Conference on Artificial Intelligence'18, and IEEE Conference on Computer Vision and Pattern Recognition'18.

Ke Zhou received the B.E., M.E., and Ph.D. degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1996, 1999, and 2003, respectively.

He is a Full Professor with the School of Computer Science and Technology, HUST. He has over 50 publications in journals and international conferences, including performance evaluation, the Conference on File and Storage Technologies, Mass Storage System and Technologies, the ACM International Conference on Multimedia, the ACM SIGOPS International System and Storage Conference, the IEEE International Conference on High Performance Computing and Communication System, and the IEEE International Conference on Communications. His current research interests include computer architecture, cloud storage, and parallel I/O and storage security

Lingyu Yan received the B.S. degree in mechanical engineering, the M.S. degree in software engineering, and the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2008, 2010, and 2014, respectively.

She is currently a Lecturer with the School of Computer Science, Hubei University of Technology, Wuhan. Her current research interests include image retrieval and machine learning.



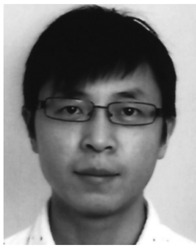
Li Liu received the B.Eng. degree in electronic information engineering from Xi'an Jiaotong University, Xi'an, China, in 2011, and the Ph.D. degree from the Department of Electronic and Electrical Engineering, University of Sheffield, Sheffield, U.K., in 2014.

He is currently a Lead Scientist with the Inception Institute of Artificial Intelligence, Abu Dhabi, UAE. His current research interests include computer vision, machine learning, multimedia, and data mining.

Fuhao Zou received the B.E. degree in computer science from Huazhong Normal University, Wuhan, China, in 1998, and the M.S. and Ph.D. degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, in 2003 and 2006, respectively.

He is currently an Associate Professor with the College of Computer Science and Technology, HUST. His current research interests include machine learning, multimedia understanding and analysis, big data analysis, semantic-based storage, and cloud storage.

Dr. Zou is a Senior Member of CCF and a member of ACM.



Ling Shao (M'09–SM'10) received the B.Eng. degree in electronic and information engineering from the University of Science and Technology of China, Hefei, China, and the M.Sc. and Ph.D. degrees from the University of Oxford, Oxford, U.K.

He is currently the Chief Scientist of JD Artificial Intelligence Research, Beijing, China, and a Professor of computer vision and machine learning with the University of East Anglia, Norwich, U.K. His current research interests include computer vision, deep learning/machine learning, multimedia, and image/video processing.

Dr. Shao is an Associate Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING, the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM FOR VIDEO TECHNOLOGY, and several other journals. He is a fellow of the British Computer Society and the Institution of Engineering and Technology.