Two-Stream Deep Hashing With Class-Specific Centers for Supervised Image Search

Cheng Deng®, Member, IEEE, Erkun Yang®, Tongliang Liu®, Dacheng Tao®, Fellow, IEEE

Abstract—Hashing has been widely used for large-scale approximate nearest neighbor search due to its storage and search efficiency. Recent supervised hashing research has shown that deep learning-based methods can significantly outperform nondeep methods. Most existing supervised deep hashing methods exploit supervisory signals to generate similar and dissimilar image pairs for training. However, natural images can have large intraclass and small interclass variations, which may degrade the accuracy of hash codes. To address this problem, we propose a novel two-stream ConvNet architecture, which learns hash codes with class-specific representation centers. Our basic idea is that if we can learn a unified binary representation for each class as a center and encourage hash codes of images to be close to the corresponding centers, the intraclass variation will be greatly reduced. Accordingly, we design a neural network that leverages label information and outputs a unified binary representation for each class. Moreover, we also design an image network to learn hash codes from images and force these hash codes to be close to the corresponding class-specific centers. These two neural networks are then seamlessly incorporated to create a unified, end-to-end trainable framework. Extensive experiments on three popular benchmarks corroborate that our proposed method outperforms current state-of-the-art methods.

Index Terms—Deep learning, hashing, nearest neighbor search, two-stream ConvNet.

I. INTRODUCTION

SIMILARITY search is a fundamental research topic in numerous computer vision and multimedia applications, including object recognition [1], information retrieval [2]–[19], image classification [20], [21], motion estimation [22], and image understanding [23], [24]. For big data applications, exact nearest neighbor search, which aims to find the object in a reference database with the smallest distance to a query object, is commonly believed to be very costly. Experiments show that exact methods can rarely outperform the brute-force linear scan method when dimensionality is high [25]. However, approximate nearest neighbor search (ANN-search), which attempts to find the nearest neighbor with a high probability under a sublinear or even constant time complexity, has been actively studied and successfully applied to many machine learning and data mining problems [19], [26]–[28].

Representative ANN-search solutions can be divided into tree-based [29], [30] and hash-based methods [17], [31]–[35]. The traditional tree-based methods usually suffer from the curse of dimensionality, and their performances have been theoretically shown to degrade to the level of the linear scan in many cases [36], while hash-based methods can efficiently identify these nearest neighbors using an efficient Hamming distance computation. Moreover, in real-world applications [7], [28], visual data are usually represented by high-dimensional features, e.g., scale invariant feature transform (SIFT)-based bag-of-words features [37] and deep features. Due to search and storage efficiency factors, the focus has shifted to hash-based methods for ANN-search.

As the pioneering method of hashing, locality-sensitive hashing (LSH) [31], which aims to use random projections to construct randomized hash functions, sets out the paradigm for the locality-sensitive hashing technique and guarantees that similar data points will be mapped to similar hash codes in Hamming space. Since the random projections in LSH-like methods are typically independent of training data, they are also known as data-independent methods. Conversely, the data-dependent methods [38]–[41], also known as learning to hashing (L2H) methods, aim to learn hash functions from training data. In real applications [42], [43], the data-dependent methods can generally achieve more promising results than data-independent methods with shorter binary codes, which has led to their increased popularity in recent years.

Depending on whether instance labels are used, data-dependent methods can be further divided into two categories: unsupervised hashing and supervised hashing. Unsupervised hashing tries to generate hash codes using unlabeled data, and usually learns hash functions by preserving the distance across the original data space and Hamming space and exploring the intrinsic properties of expected hash codes (such as balance and independence), while supervised hashing aims to learn hash functions that can maintain the semantic

Digital Object Identifier 10.1109/TNNLS.2019.2929068
similarity constructed by supervisory information. In this paper, we focus primarily on supervised hashing, as this approach usually achieves more compelling results than its unsupervised counterpart.

Traditional supervised hashing methods, which rely on nondeep architectures, treat feature learning and hash code quantization as two separate processes and cannot simultaneously optimize these two aspects, representing a missed opportunity to learn hash codes more effectively. Recent deep learning-based hashing methods [28], [44], which combine feature learning and hash code quantization, have shown that deep networks can contribute to more effective hash code learning. Deep hashing (DH) methods usually learn hash codes by first constructing pairwise [43] or triplet [18] similarity relationships, and then learn hash functions to generate hash codes from images that are consistent with these relationships. However, in real scenarios, as illustrated in Fig. 1, natural images have large intraclass and small interclass variations; accordingly, the hash codes learned from images may be inconsistent with their high-level semantic labels, which may impede the hash function learning and degrade the learned hash codes.

To address the above-mentioned problem, we design a novel DH method dubbed two-stream DH (TSDH), which constructs a two-stream ConvNet architecture and learns hash codes with class-specific centers to minimize the intraclass variation. More specifically, we design a label network, which can learn a unified binary representation from labels for each class. We also design an image network to learn hash codes from images. The unified binary representations from the label network are then regarded as class-specific centers. Compared with the existing methods, our method can explicitly reduce the intraclass variation by forcing hash codes from the image network to be close to the corresponding class-specific centers. In addition, the adopted pairwise loss function can reduce the interclass variation. Therefore, our method can simultaneously minimize the intraclass variation and maximize the interclass variation effectively. We also point out that the learned class-specific centers can inherently exploit the multi-label dependence and deal with multi-label applications efficiently and effectively.

Accordingly, the main contributions of this work can be summarized as follows.

1) We design a two-stream ConvNet architecture consisting of a label network and an image network, which can learn hash codes from labels and images, respectively. By integrating these two neural networks into a unified framework, we can optimize them simultaneously and leverage information from both image content and semantic labels. The learning processes for image and label streams can thus benefit from each other, yielding more promising results.

2) The learned hash codes from the label network are used as class-specific centers for the hash codes learned from the image network, which can greatly reduce the intraclass variation of hash codes learned from the image network.

3) By learning from labels, these class-specific centers can inherently exploit the multi-label dependence and deal with multi-label applications efficiently and effectively. Experiments on three popular benchmark data sets demonstrate that TSDH can outperform current state-of-the-art methods.

The rest of this paper is organized as follows. We review the relevant literature in Section II. We present our novel TSDH for ANN-search in Section III and its optimization in Section IV. Section V details the experiments, after which concluding remarks are presented in Section VI.

II. RELATED WORK

A variety of hashing methods have been developed over the past decade, including traditional hashing methods and the more recent deep learning-based methods. In this section, we briefly review some works related to our proposed hashing method.

A. Traditional Unsupervised Hashing Methods

Unsupervised hashing methods map original data into Hamming space without any label information. The fundamental rationale behind these methods is to preserve the distance across the original space and Hamming space. Quantization loss, bit balance, and independence properties are often used to achieve this goal.

Various unsupervised hashing methods have been proposed in recent years [38]–[41], [45], [46]. Spectral hashing (SH) [38] interprets the hash code learning as a particular spectral graph partition and tries to learn binary codes with balanced and uncorrelated properties and make the neighbors in the input space to have small Hamming distances. Iterative quantization (ITQ) [40] first uses the principal component analysis (PCA) to map the data to a low-dimensional space, then exploits a simple and efficient alternating minimization scheme to find a rotation matrix, which maps the data to binary codes with the minimum quantization error. Discrete graph hashing (DGH) [45] casts the graph hashing problem...
into a discrete optimization framework and explicitly deals with the discrete constraints that can directly output binary codes. Spherical hashing (SpH) [46] minimizes the spherical distance between the original real-valued features and the learned binary codes; the distance metric used in this approach is also called spherical Hamming distance. Anchor graph hashing (AGH) [39] utilizes anchor graphs to obtain tractable low-rank adjacency matrices and approximate the data structure, which can automatically discover the inherent neighborhood structure in the training data. Stochastic generative hashing (SGH) [41] utilizes a generative mechanism to learn hash functions through the minimum description length principle. The hash codes are optimized to maximally compress the data set as well as to regenerate the inputs. A stochastic distributional gradient-based learning algorithm is also developed to avoid the optimization difficulty caused by binary constraints.

Although current traditional unsupervised hashing methods have made great progress, they usually depend on the predefined features and cannot simultaneously optimize the feature and the hash code learning processes, resulting in a missed opportunity to learn more effective hash codes.

B. Traditional Supervised Hashing Methods

Traditional supervised hashing aims to take full advantage of the supervisory information of labeled data to learn more efficient binary codes, showing higher search performance for ANN-search than the unsupervised approaches. The fundamental idea behind supervised hashing is to learn hash functions that can map data points to Hamming space, where the semantic similarity can be preserved.

Recently, a variety of supervised hashing methods have been proposed [32], [34], [47]–[50], which greatly progress the hash learning field. Kernel-based supervised hashing (KSH) [34] learns hash functions to map data points to compact codes whose Hamming distances are minimized for similar pairs and maximized for dissimilar pairs by using inner products to approximate the Hamming distance. Adaptive hashing (AH) [47] learns hash functions in an online manner, and designs an online learning algorithm based on stochastic gradient descent (SGD) to iteratively update the hash functions with streaming data. Minimal loss hashing (MLH) [48] adopts structure prediction with latent variables and a hinge-like loss function to learn similarity-preserving hash functions, allowing it to project high-dimensional data onto binary codes. Fast supervised hashing (FastHash) [49] adopts boosted decision trees to achieve nonlinearity in hashing and fits the learned binary codes, an efficient GraphCut-based block search method is also proposed to solve large-scale inference. Column sampling-based discrete supervised hashing (COSDISH) [50] learns hash codes iteratively. In each iteration, several columns are sampled from the predefined semantic similarity matrix; the hash code can then be decomposed into two parts, which can be alternately optimized in a discrete way. Fast supervised discrete hashing (FSDH) [32] uses a simple yet effective regression from the class labels of training examples to the corresponding hash code to accelerate the learning process. FSDH also has a closed-form solution, and thus only requires a single hash code-solving step rather than iterative steps to solve the problem.

Similar to traditional unsupervised hashing methods, traditional supervised hashing methods, which are based on shallow architectures, also suffer from the separation between the processes of feature extraction and the hash code learning.

C. Deep Learning-Based Hashing Methods

Recently, deep learning has revolutionized computer vision, machine learning, and other related areas. Deep learning-based hashing methods [43], [51]–[65] have also been proposed, showing that deep networks can help to facilitate more effective hash code learning. Most deep learning-based methods adopt neural networks to integrate feature and hash code learning into a unified, end-to-end trainable framework. The powerful representation ability of neural networks and the simultaneous optimization of feature and hash code learning allow deep learning-based methods to achieve more compelling results.

For unsupervised DH, due to the lack of semantic labels, hash functions are usually learned directly from the original data. A pioneering method, semantic hashing [66] utilizes restricted Boltzmann machines (RBMs) as an autoencoder network to generate efficient hash codes. Reconstruction loss is adopted to optimize this algorithm. DH [51] develops a neural network to seek multiple hierarchical nonlinear transformations to map data points to Hamming space. Quantization loss between the original real-valued feature and the corresponding binary code is adopted. DeepBit [67] considers original images and their corresponding rotated images as similar pairs and tries to learn hash codes to preserve this similarity. However, since the rotated images are generated from the original images, DeepBit cannot utilize the semantic relationships between different images. Semantic structure-based unsupervised DH (SSDH) [64] tries to learn semantic structures without labels. It takes full advantage of the semantic information in deep features and learns semantic structures based on the pairwise cosine distances for different data points and a Gaussian estimation for the distance histogram. The resulting semantic structure is then used to guide the hash code learning process.

Many supervised hashing methods also incorporate with deep learning techniques. Convolutional neural networks-based hashing (CNNH) [52] decomposes the hash function learning into two stages. In the first stage, a pairwise similarity matrix is constructed and decomposed into the product of an approximate hash code matrix. In the second stage, CNNH simultaneously learns feature representation and hash functions by predicting the learned hash codes as well as the discrete class labels of images. Network-in-network hashing (NINH) [53] proposes a deep architecture with three blocks: 1) a subnetwork to learn intermediate image features; 2) a divide-and-encode module that divides the intermediate image features into multiple branches, each encoded into one hash bit; and 3) a triplet loss to preserve the semantic information in Hamming space. Deep pairwise-supervised
hashing (DSH) [59] also adopts a pairwise loss function; moreover, the regularized term to decrease the quantization error between real-valued outputs and binary codes. Deep supervised hashing (DSH) [59] exploits a regularization term to decrease the quantization error between real-valued outputs and binary codes. Deep Cauchy hashing [60] adopts Cauchy distribution to continue to optimize data pairs in a relatively small Hamming ball, which are difficult to optimize effectively using traditional loss functions. A quantization loss based on the Cauchy distribution is also proposed to reduce the discrepancy between continuous features and binary codes.

All the above supervised DH methods are based on networks and loss functions designed for image data. However, due to the great variations among natural images, hash codes learned from image may be degraded and inconsistent with semantic labels. In this work, instead of only learning hash codes from images contents, we design another network that can learn binary representation as class-specific centers. Compared with other works, our method can reduce the intraclass variations and learn more effective hash codes.

III. FORMULATION

A. Notation and Problem Definition

In this paper, boldface uppercase letters (such as $A$) are used to denote matrices. $A_{ij}$ denotes the $i$th column of $A$. $A_{ij}$ denotes the $(i,j)$th element of $A$. $||A||_F$ and $A^\top$ are used to denote the Frobenius norm and the transpose of the matrix $A$, respectively. The $\odot$ symbol is used to denote the Hadamard product and $\text{sign}(\cdot)$ represents the elementwise signum function as

$$\text{sign}(x) = \begin{cases} 
1 & \text{if } x \geq 0 \\
-1 & \text{if } x < 0.
\end{cases} \quad (1)$$

The goal of the proposed TSDH is to learn hash functions, which can map images into hash codes in Hamming space. Here, hash codes indicates fixed length binary vectors, whose elements are either $-1$ or $+1$. Hamming space represents the set of all binary vectors. Assume that we have $m$ images in a given data set denoted by $X = \{x_i\}_{i=1}^m$. Each image $x_i$ is associated with a label vector $I_i$. The goal of unsupervised hashing is to learn binary hash codes for database points and a hash function that can be used to generate hash codes for query images. We use $B = \{b_i\}_{i=1}^m$ to denote the learned hash codes for $X$, and $b_i \in \{-1,+1\}^k$ corresponds to the hash code for the point $x_i$, where $k$ denotes the hash code length. For each vector $I_i$, we learn a class-specific center, which denotes the representation center for data in the corresponding class.

For convenience, Table I briefly outlines some important notations used in this paper.

B. Two-Stream Architecture for Hashing

1) Image Stream Architecture: For the image stream, we design a neural network that maps images to binary codes in Hamming space. More specifically, we adopt a convolutional neural network modified from VGG16 [68] as hash functions. There are five convolutional layers and three fully connected layers. The convolutional layers and the first two fully connected layers are the same as those in the VGG16 models, while the last fully connected layer is replaced by a fully connected layer with $k$ hidden units to incorporate the hash code learning process into this CNN model. It should be noted that we adopt the VGG16 network only for illustrative purposes; any other networks can be easily integrated into the framework.

2) Label Stream Architecture: For the label stream, we design a multi-layer perceptron (MLP) to learn hash codes from labels. More specifically, the adopted MLP consists of two fully connected layers, denoted as "full1–full2." The detailed configurations are given in Table II, where the number of units in the corresponding layer is denoted. Rectified linear unit (ReLU) is adopted as the activation function for the first layer; for the second layer, we adopt the identity function as the activation function.

C. Two-Stream Loss Function

Since hash codes are learned from two streams, we here denote hash codes learned from the image stream as...
$U = \{u_1, u_2, \ldots, u_m\}$, where $u_i \in \{-1, 1\}^k$, and hash codes learned from the label stream as $v = \{v_1, v_2, \ldots, v_m\}$, where $v_j \in \{-1, 1\}^k$.

In the customary setting for supervised hashing [34], pairwise labels are usually given as supervised information. Accordingly, in this paper, we can construct pairwise labels based on the category labels: label 1 specifies similar pairs collected in set $\mathcal{V}$, and label -1 designates dissimilar pairs collected in set $\mathcal{C}$. Such pairs can be acquired by considering signal pairs with at least category label in common as similar pairs, and others as dissimilar pairs. To explicitly record the pairwise relationships among the data set, we define a pairwise similarity matrix $S \in \mathbb{R}^{m \times m}$ as

$$S_{ij} = \begin{cases} 1, & (x_i, x_j) \in \mathcal{V}, \\ -1, & (x_i, x_j) \in \mathcal{C}. \end{cases}$$

(2)

In order to maintain the semantic similarity between different data points, we try to map semantically similar data points into hash codes with the minimal Hamming distance, i.e., 0, and semantically dissimilar data points into hash codes with maximal Hamming distance, i.e., the number of hash bits $k$. However, in practice, directly optimizing the Hamming distances is nontrivial due to the complex mathematical formula. Therefore, in this paper, as indicated in [34], we adopt the inner products of hash codes as a surrogate for Hamming distances. More specifically, we try to minimize the $l_2$ loss between the pairwise similarity and the inner product between hash codes. The basic objective function can be formulated as follows:

$$\min_{U} J(U) = \sum_{i=1}^{m} \sum_{j=1}^{m} \left( \frac{1}{k} u_i^T u_j - S_{ij} \right)^2$$

s.t. $U \in \{-1, 1\}^{m \times k}$

$$u_i = h_1(x_i), \quad \forall i \in \{1, 2, \ldots, m\}$$

(3)

where $h_1(\cdot)$ represents the hash function for images.

However, (3) only considers hash codes generated from images. In practice, images in the same class have large variances, while those in different classes have small variances, which may cause the hash codes learned from (3) to be inconsistent with semantic labels and, thus, compromise the performance. To address this problem, in addition to the hash function $h_1(\cdot)$ for images, we propose another hash function $h_2(\cdot)$ as introduced in III-B2. Note that hash codes learned from $h_2(\cdot)$, for instances, in the same class will always be the same. Thus, we can learn a class-specific binary representation from $h_2(\cdot)$ for each class. These binary representations are regarded as representation centers for the hash codes of images from the same class. Forcing the hash codes from $h_1(\cdot)$ to be close to the corresponding class-specific centers can greatly reduce the intraclass variations and yield more promising search results. Accordingly, we integrate $h_1(\cdot)$ and $h_2(\cdot)$ into a unified framework and optimize them simultaneously. The objective function can be formulated as

$$\min_{U, V} J(U, V) = \sum_{i=1}^{m} \sum_{j=1}^{m} \left( \frac{1}{k} u_i^T v_j - S_{ij} \right)^2$$

s.t. $U \in \{-1, 1\}^{m \times k}$, $V \in \{-1, 1\}^{m \times k}$

$$u_i = h_1(x_i), \quad v_j = h_2(l_j), \quad \forall i, j \in \{1, 2, \ldots, m\}.$$  (4)

To incorporate this objective function with the neural networks in the image stream and label stream, we can set $h_1(x_i) = \text{sign}(F_1(x_i; \Theta_1))$ and $h_2(l_j) = \text{sign}(F_2(l_j; \Theta_2))$, where $F_1(x_i; \Theta_1) \in \mathbb{R}^k$ denotes the output of the image network for $x_i$, and $F_2(l_j; \Theta_2) \in \mathbb{R}^k$ denotes the corresponding output of the label network; $\Theta_1$ and $\Theta_2$ are the parameters.

However, a problem still exists here, namely, that the binary outputs of $\text{sign}(F_1(x_i; \Theta_1))$ and $\text{sign}(F_2(l_j; \Theta_2))$ make it infeasible to optimize the objective function with the traditional gradient descent based optimization methods. Accordingly, in this paper, we use $\tanh(\cdot)$ to approximate the $\text{sign}(\cdot)$ function, such that the objective function can be rewritten as

$$\min_{\Theta_1, \Theta_2} J(\Theta_1, \Theta_2) = \sum_{i=1}^{m} \sum_{j=1}^{m} \left( \frac{1}{k} h_1(x_i) \tanh(h_2(l_j) - S_{ij}) \right)^2$$

s.t. $h_1(x_i) = \tanh(F_1(x_i; \Theta_1)), \quad \forall i \in \{1, 2, \ldots, m\}$

$$h_2(l_j) = \tanh(F_2(l_j; \Theta_2)), \quad \forall j \in \{1, 2, \ldots, m\}.$$  (5)

After training (5), we can obtain two kinds of hashing functions: hashing functions from the image stream, which can project data points to Hamming space when the original images are given, and hash functions from the label stream, which can project data points to Hamming space when labels are given. In practice, for data points with available labels, we can hash them by using the hashing functions for the label stream, which usually preserve the semantic similarity more effectively. Moreover, for query points whose labels are usually missing, we can generate their corresponding hash codes by using the original images as inputs through the image stream hashing.

D. Connections

1) Connection to Center Loss: Center loss in [69] aims to minimize the intraclass variations by learning a center for each class and penalizing the distance between the deep representations and their corresponding class centers

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^{m} \|f(x_i) - c_{y_i}\|^2_2$$

(6)

where $f(x_i)$ is the learned representations for the input point $x_i$, $c_{y_i}$ is the center for the $y_i$th class, and $m$ is the total number of instances. By minimizing 6, the distances between data representations and their corresponding class centers can be minimized. This mechanism has shown promising results in reducing the intraclass variation in face recognition tasks. We note here that the center loss is similar to our two-stream learning in the single label scenario, where hash codes learned from the label stream can be regarded as the common class centers, and the hash codes learned from the image stream...
are forced to be close to these common centers. Class centers in [69] are calculated and updated from the representations in the corresponding class as follows:

\[ e_j = e_j - \Delta e_j, \]
\[ \Delta e_j = \sum_{i=1}^{m} \delta(y_i = j) \cdot (e_j - f(x_i)) \]
\[ \frac{\partial J}{\partial e_j} = \frac{\sum_{i=1}^{m} \delta(y_i = j)}{1 + \sum_{i=1}^{m} \delta(y_i = j)} \]

where \( \delta(\text{condition}) \) equals 1 if the condition is satisfied, and 0 otherwise. However, these class centers are designed for face recognition, where data points only have a single label; if we directly apply the center loss to multi-class scenarios, the number of class centers will increase exponentially with the number of classes, not to mention the fact that this approach cannot exploit the multi-class dependence. Accordingly, the center loss is not suitable for multi-label scenarios. By contrast, our method learns the class centers from a label network and can learn similar centers for similar labels; thus, our method is inherently able to exploit the multi-label dependence and deal with multi-label applications efficiently and effectively.

2) Connection to Multi-Modal Hashing: Multi-modal hashing methods usually learn hash functions by taking the advantage of the information from multiple modalities, such as images and texts. In our method, we consider both semantic labels and images as input signals. If we treat labels and images as two different modalities, our method can be generally categorized as a multi-modal hashing framework. However, unlike traditional multi-modal hashing, which usually treats different modalities equally, our method treats images and labels in different ways. Since the label stream can learn hash codes that are highly consistent with labels, it is used to guide the image stream learning and to generate hash codes for database points. Moreover, since the image stream can exploit the image information and deal with images without labels, we optimize the image stream by encouraging the learned codes it produces to be close to the class-specific centers as a means of reducing the intraclass variation. After optimization, the image stream can be used to generate hash codes for queries, the labels for which are usually missing.

IV. OPTIMIZATION

We simultaneously optimize the two networks to learn the parameter \( \Theta_1 \) for the image stream hashing and the parameter \( \Theta_2 \) for the label stream hashing. An overview of the whole learning algorithm is presented in Algorithm 1, while the detailed derivation is introduced in the remaining content.

A. Learning \( \Theta_1 \) for the Image Network

We learn the CNN parameter \( \Theta_1 \) for the image stream hashing by using a minibatch SGD method with back-propagation (BP) algorithm. More specifically, in each iteration, we first sample a minibatch size of data points from the training set and then conduct our optimization algorithm based on the sampled data. For the sake of simplicity, we define

\[ \tilde{u}_i = F_1(x_i, \Theta_1) \]
\[ \tilde{v}_j = F_2(l_j, \Theta_2) \]

We then compute the gradient of the objective function with regards to the output \( \tilde{u}_i \) as follows:

\[ \frac{\partial J}{\partial \tilde{u}_i} = \sum_{j=1}^{m} \tanh(\tilde{u}_j) \times \left( \frac{1}{k} \tanh(\tilde{u}_i)^T \tanh(\tilde{v}_j) - S_{ij} \right) (1 - \tanh(\tilde{u}_j))^2. \]

Subsequently, we can compute \( \frac{\partial J}{\partial \Theta_1} \) with \( \frac{\partial J}{\partial \tilde{u}_i} \) by using the chain rule, and the BP algorithm can be used to update \( \Theta_1 \).

B. Learning \( \Theta_2 \) for the Label Network

The CNN parameter \( \Theta_2 \) for the label stream hashing is also learned by using a minibatch SGD with BP algorithm. The gradient of the objective function with regards to the output \( \tilde{v}_j \) can be computed as follows:

\[ \frac{\partial J}{\partial \tilde{v}_j} = \sum_{i=1}^{m} \tanh(\tilde{u}_i) \times \left( \frac{1}{k} \tanh(\tilde{v}_j)^T \tanh(\tilde{u}_i) - S_{ij} \right) (1 - \tanh(\tilde{v}_j))^2. \]

We can then compute \( \frac{\partial J}{\partial \Theta_2} \) with \( \frac{\partial J}{\partial \tilde{v}_j} \) by using the chain rule, and the BP algorithm can be used to update \( \Theta_2 \).
set are used as the database.

A retrieval set, from which we randomly select 5000 images as a test set; the remaining images are used as a training set.

Fig. 3. Examples randomly sampled from the CIFAR10, FLICKR25K, and NUSWIDE data sets.

C. Out of Sample Extension

For a query point $x_i$ without labels, we can generate its hash code by forward-propagating it through the image stream hashing network as follows:

$$b_i = \text{sign}(\tanh(F_1(x_i; \Theta_1))).$$

V. EXPERIMENTS

We evaluate our method on three popular benchmark data sets, FLICKR25K, NUSWIDE, and CIFAR10, and provide extensive evaluations to demonstrate its performance. Some examples from these data sets are illustrated in Fig. 3. In this section, we first introduce the data sets and then present and discuss our experimental results together with comparative evaluations with other state-of-the-art methods.

A. Data Sets

FLICKR25K contains 25000 images collected from the Flickr website. Each image is manually annotated with at least one of the 24 unique labels provided. We randomly select 2000 images as a test set; the remaining images are used as a retrieval set, from which we randomly select 5000 images as a training set.

NUSWIDE contains 269648 images, each of which are annotated with multiple labels referring to 81 concepts. The subset containing the ten most popular concepts is used here. We randomly select 5000 images as a test set; the remaining images are used as a retrieval set, and 5000 images are randomly selected from the retrieval set for use as a training set.

CIFAR10 is a popular image data set containing 60000 images in ten classes. For each class, we randomly select 1000 images as queries and 500 as training images, resulting in a query set containing 10000 images and a training set made up of 5000 images. All images except for the query set are used as the database.

B. Evaluation

To evaluate the performance of our proposed method, we adopt four evaluation criteria: mean of average precision (MAP), topN-precision, recall@K, and precision-recall. The first three criteria are based on Hamming ranking, which ranks data points based on their Hamming distances to the query; for its part, precision-recall is based on hash lookup, as it constructs a lookup table and returns points within a given Hamming radius.

1) MAP is one of the most widely used criteria for evaluating retrieval accuracy. Given a query and a list of $R$ ranked retrieval results, the AP for this query is defined as

$$\text{AP}(x_q) = \frac{1}{N} \sum_{r=1}^{R} P(r) \delta(r)$$

where $N$ is the number of ground-truth relevant instances in the database, and $P(r)$ denotes the precision for the top $r$ retrieved instances. $\delta(r) = 1$ when the $r$th retrieval instance is similar with the query, otherwise $\delta(r) = 0$. MAP is, thus, defined as the average of APs for all queries

$$\text{MAP} = \frac{1}{Q} \sum_{q=1}^{Q} \text{AP}(x_q)$$

where $Q$ is the number of queries. For all three data sets, we set $R$ as the number of database points. Two data points are considered neighbors if they share the same label (for CIFAR10) or share at least one common label (for multi-label data sets FLICKR25K and NUSWIDE).

2) TopN-precision is defined as the average ratio of similar instances among the top $W$ retrieved instances for all queries in terms of Hamming distance. In our experiments, $W$ is set to 1000.

3) Recall@K counts the percentage of true neighbors from the top $K$ retrieved instances among all the ground-truth instances in terms of Hamming distance. In our experiments, $K$ is set to 1000.

4) Precision-recall reveals the precision at different recall levels and is a good indicator of overall performance for different algorithms. Typically, the area under the precision-recall curve is computed, with a larger value indicating better performance.

C. Baseline Methods

The proposed method is compared with six state-of-the-art nonDH methods [LSH, SH, ITQ, KSH, ITQ-Canonical Correlation Analysis (CCA), and supervised discrete hashing (SDH)] and three recently proposed deep learning-based hashing methods [deep hashing network (DHN), DPSH, and HashNet].

1) LSH [70] maps a high-dimensional data to a low-dimensional binary code via a set of appropriately chosen random projection functions.

2) DSH [42], which can be regarded as an extension of LSH, explores the geometric structure of training data,
TABLE III

<table>
<thead>
<tr>
<th>method</th>
<th>FLICKR25K</th>
<th>NUSWIDE</th>
<th>CIFAR10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td>LSH</td>
<td>0.6091</td>
<td>0.6105</td>
<td>0.6033</td>
</tr>
<tr>
<td>SH</td>
<td>0.6119</td>
<td>0.6315</td>
<td>0.6381</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.6492</td>
<td>0.6518</td>
<td>0.6546</td>
</tr>
<tr>
<td>DSH</td>
<td>0.6574</td>
<td>0.6521</td>
<td>0.6618</td>
</tr>
<tr>
<td>KSH</td>
<td>0.6452</td>
<td>0.6547</td>
<td>0.6551</td>
</tr>
<tr>
<td>ITQ-CCA</td>
<td>0.6362</td>
<td>0.6283</td>
<td>0.6253</td>
</tr>
<tr>
<td>DHN</td>
<td>0.5841</td>
<td>0.5945</td>
<td>0.6060</td>
</tr>
<tr>
<td>DPSH</td>
<td>0.8099</td>
<td>0.8277</td>
<td>0.8290</td>
</tr>
<tr>
<td>HashNet</td>
<td>0.8160</td>
<td>0.8346</td>
<td>0.8431</td>
</tr>
<tr>
<td>TSDH</td>
<td>0.8245</td>
<td>0.8650</td>
<td>0.8703</td>
</tr>
</tbody>
</table>

Fig. 4. Precision-recall curves on FLICKR25K, NUSWIDE, and CIFAR10 with code length 64. (a) FLICKR25K. (b) NUSWIDE. (c) CIFAR10.

Fig. 5. TopN-precision curves on FLICKR25K, NUSWIDE, and CIFAR10 with code length 64. (a) FLICKR25K. (b) NUSWIDE. (c) CIFAR10.

avoiding the purely random selection of projections and using the projective functions that best agree with the data distributions.

3) SH [38] treats the problem of finding the best code for a given data set as graph partitioning problem and obtains a spectral method by relaxing this partitioning problem.

4) ITQ [40] first performs PCA to map original data to a low-dimensional space, then learns an orthogonal rotation matrix to minimize the quantization error.

5) KSH [34] approximates the Hamming distance with inner products and learns hash functions by minimizing the Hamming distance for similar pairs and maximizing this distance for dissimilar pairs.

6) ITQ-CCA [40] extends the original ITQ to supervised settings, which adopts the CCA as a supervised dimensionality reduction process and then combines the low-dimensional features with ITQ to learn hash codes with minimal quantization error under supervised settings.

7) DHN [71] formulates the supervised hashing problem in a Bayesian learning framework. A deep network architecture is proposed to optimize the pairwise cross-entropy loss and pairwise quantization loss simultaneously.
DENG et al.: TWO-STREAM DH WITH CLASS-SPECIFIC CENTERS FOR SUPERVISED IMAGE SEARCH

Fig. 6. Recall@K curves on FLICKR25K, NUSWIDE, and CIFAR10 with code length 64. (a) FLICKR25K. (b) NUSWIDE. (c) CIFAR10.

**TABLE IV**
CLUSTERING COMPARISON IN TERMS OF PURITY. THE BEST ACCURACY IS SHOWN IN BOLDFACE.

<table>
<thead>
<tr>
<th>method</th>
<th>CIFAR10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>LSH</td>
<td>0.2206</td>
</tr>
<tr>
<td>SH</td>
<td>0.3084</td>
</tr>
<tr>
<td>ITQ</td>
<td>0.3262</td>
</tr>
<tr>
<td>DSH</td>
<td>0.2702</td>
</tr>
<tr>
<td>KSH</td>
<td>0.6321</td>
</tr>
<tr>
<td>ITQ-CCA</td>
<td>0.4959</td>
</tr>
<tr>
<td>DHN</td>
<td>0.7231</td>
</tr>
<tr>
<td>DPSH</td>
<td>0.7359</td>
</tr>
<tr>
<td>HashNet</td>
<td>0.7386</td>
</tr>
<tr>
<td>TSDH*</td>
<td>0.7521</td>
</tr>
<tr>
<td>TSDH</td>
<td><strong>0.7704</strong></td>
</tr>
</tbody>
</table>

8) DPSH [43] adopts a novel DH framework that can perform feature learning and hash code learning simultaneously for applications with pairwise labels. The framework can be optimized by end-to-end training and generates hash codes directly from raw images.

9) HashNet [72] uses a continuation method to address the imbalanced similarity data problem and gradually evolve a tanh function to approximate the sign function. Moreover, a novel weighted pairwise cross-entropy loss function is also proposed. Finally, a multistage pretraining algorithm is adopted to effectively train the continuation DH network.

Among all these baseline methods, LSH is a data-independent hashing method, SH and ITQ are two data-dependent unsupervised hashing methods, KSH, ITQ-CCA, and SDH are three no-deep supervised hashing methods, and DHN, DPSH, and HashNet are three deep learning-based hashing methods. All the codes for these methods have been kindly provided by the authors. All algorithms for LSH, SH, ITQ, KSH, ITQ-CCA, and SDH are implemented with MATLAB, while those for DHN and HashNet are implemented with Caffe [73], which is based on C++; for DPSH, we adopt its PyTorch [74] version, which is implemented with Python and achieves higher MAP results for all the numbers of hash code than its MatconvNet [75] version.

**D. Implementation Details**

As shown in Fig. 2, the hybrid deep architecture comprises an image network and a label network. For the image network, we adopt the VGG16 architecture [68] and replace the last fully connected layer with a new fully connected layer with \( k \) units. Parameters for the new fully connected layer are learned from scratch, while parameters for the preceding layers are fine-tuned from the model pretrained on ImageNet [76]. For the label network, we design an MLP, the detailed configurations of which are given in Table II. All the parameters for this label network are learned from scratch. We employ the standard stochastic gradient descent algorithm with 0.9 momentum for optimization, minibatch size is set to 24, and the learning rate is fixed to \( 10^{-3} \).

For a fair comparison, we select the VGG16 network as the base feature learning network for all deep learning-based hashing methods; for other baseline methods, we adopt the deep features extracted from the last fully connected layer from the VGG16 network pretrained on ImageNet. Since VGG16 accepts images of size 224 \( \times \) 224 as inputs, we resize all images from all three data sets to be 224 \( \times \) 224 before inputting them into the VGG16 network. All baseline methods and our method are evaluated in a machine with four Titan 1080 Ti GPUs.

**E. Results and Discussion**

We first present the MAP values for TSDH and all baseline methods across different hash code lengths on the three data sets as a global evaluation. We then draw precision-recall and TopN-precision curves a hash code length of 32 as a more comprehensive comparison.

Table III presents the MAP results for TSDH and all baseline methods on FLICKR25K, NUSWIDE, and CIFAR10, with hash code numbers varying from 16 to 64. By comparing LSH with SH, ITQ, and DSH, we can see that data-dependent hashing methods outperform data-independent hashing method in most cases; this may be because data-dependent methods can learn hash functions that are more suitable for the data structures used here. By comparing KSH and ITQ-CCA to SH, ITQ, and DSH, we can understand that supervised hashing methods can usually achieve more promising results than unsupervised ones due to their exploitation of supervised signals.
Moreover, by comparing DHN, DPSH, and HashNet with other baseline methods, we can see that deep learning-based hashing methods can usually outperform nonDH methods; this may be due to the simultaneous learning of representations and hash codes.

From the MAP results, we can see that TSDH consistently obtains the best results across different hash bit lengths for all three data sets. Specifically, compared to one of the best nonDH methods, i.e., KSH, we achieve absolute improvements of 20.67%, 16.18%, and 31.53% in the average MAP for different bits on FLICKR25K, NUSWIDE, and CIFAR10, respectively. Compared to the state-of-the-art DH method HashNet, we achieve absolute improvements of 2.45%, 7.91%, and 48.31% in average MAP for different bits on the three data sets, respectively. Note that DPSH and HashNet are both DH methods, which adopt the same deep architecture as TSDH, and also use a similar pairwise loss function; the key difference here is that the pairwise loss function for both DPSH and HashNet are used for the image modality only and, thus, cannot fully exploit the information from semantic labels. The MAP results show that our method consistently outperforms these two methods, which adequately demonstrates the efficacy of the proposed two-stream framework.

To illustrate the hash lookup results, we plot the precision-recall curves for all methods with 64 at a hash bit length of 64 on the three data sets in Fig. 4. From these results, we can again observe that TSDH consistently achieves the best performance, which verifies the superiority of our proposed method. Fig. 5 shows the TopN-precision curves for all methods on each of the three data sets, while Fig. 6 shows the recall@K curves for all methods in all cases. Consistent with the previous results, we can see that TSDH achieves the best results among all approaches in most situations. Since MAP values, TopN-precision curves, and recall@K curves are all based on Hamming ranking, an overview of the above-mentioned analysis reveals that TSDH can achieve superior performance for Hamming ranking-based evaluations.

To better demonstrate the effectiveness of the learned hash codes, we use k-means to cluster the hash codes of CIFAR10 test data for all the methods. Then, we use purity to measure the cluster performance. The results are presented in Table IV, where TSDH* is a variant of TSDH without the class-specific center part, which will be introduced in detail in Section V-G. From the results, we can get that the hash codes learned from TSDH can be clustered more effectively, which can also help to support the utility of our method.

The convergence curves of the loss function for a hash bit length of 16 on FLICKR25K, NUSWIDE, and CIFAR10 are shown in Fig. 7. From these results, we can see that our method converges within about 2000 iterations for all the three data sets. The results for other hash bit lengths are similar to those for the 16 hash bit length. Here, we omit them to save space.

To investigate the effect of hash code length, we further evaluate the proposed method with hash code length varying from [80, 96, 112, 128], and present the results in Fig. 8. From the figure, we can get that the map values first increase and then remain stable. This is reasonable, since that short hash codes do not have enough representation capacity, so the map values first increase and when the bit-length is long enough, the map values will not change significantly.

**F. Training and Encoding Time**

In this section, we compare the training and encoding time for all methods. Note that the encoding time for shallow-architecture-based methods includes the feature extracting time. The results for different phases are presented in Table V. In terms of training, we can see that deep learning-based methods usually need a longer training phase. In addition, the training time for TSDH and other deep learning-based methods are similar. In terms of encoding, which is usually more important for modern search systems, we can see that the encoding time for all the methods is comparable.
TABLE VI
MAP RESULTS FOR TSDH* AND TSDH. THE BEST ACCURACY IS SHOWN IN BOLDFACE.

<table>
<thead>
<tr>
<th>Method</th>
<th>FLICKR25K</th>
<th>NUSWIDE</th>
<th>CIFAR10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
<td>32 bits</td>
<td>48 bits</td>
</tr>
<tr>
<td>TSDH*</td>
<td>0.8221</td>
<td>0.8513</td>
<td>0.8497</td>
</tr>
<tr>
<td>TSDH</td>
<td>0.8245</td>
<td>0.8650</td>
<td>0.8703</td>
</tr>
<tr>
<td></td>
<td>0.7599</td>
<td>0.7634</td>
<td>0.7637</td>
</tr>
<tr>
<td></td>
<td>0.7900</td>
<td>0.8150</td>
<td>0.8293</td>
</tr>
<tr>
<td></td>
<td>0.7342</td>
<td>0.7453</td>
<td>0.7456</td>
</tr>
<tr>
<td></td>
<td>0.7924</td>
<td>0.7738</td>
<td>0.8024</td>
</tr>
</tbody>
</table>

G. Ablation Study

In this section, we go deeper to study the efficacy of the proposed two-stream framework. More specifically, we investigate TSDH*, a variant of TSDH with a similar pairwise loss function but only with the image network. The MAP results of TSDH and TSDH* are compared in Table VI, from which we can see that TSDH consistently outperforms TSDH* by margins of 0.24%, 1.37%, 2.06%, and 3.27% for the FLICKR25K data set, 3.01%, 5.16%, 6.56%, and 8.06% for the NUSWIDE data set, and 5.82%, 2.85%, 5.68%, and 9.31% for the CIFAR10 data set at hash bit lengths of 16, 32, 64, and 128, respectively. The results in Table VI clearly demonstrate that through the introduction of the label stream, TSDH can achieve more promising results.

H. Visualization

We first generate the hash codes for the CIFAR10 query set, which has 10 000 examples, with TSDH*, TSDH, and the two best deep learning-based hashing baselines, DPSH and HashNet. The hash codes for TSDH* and TSDH are generated from the image network for that we cannot access to the labels of query images. The t-SNE visualization for these generated hash codes is presented in Fig. 9. From these results, we can observe that compared with DPSH, HashNet, TSDH*, TSDH can generate hash codes with more compact structures and smaller intraclass variations. We can also see that the hash codes from TSDH are more discriminative and can be better separated. Considered together, these results again verify that by adopting a two-stream framework, we can learn more discriminative hash codes, which enables more effective image search.

VI. CONCLUSION

We proposed a DH approach for image search, namely, two-stream DH with class-specific centers. The superiority of the proposed approach lies in the following aspects: 1) a two-stream architecture is designed to fully exploit the information from both image content and semantic labels. These two subnetworks are optimized simultaneously and can benefit from each other in the training process; 2) by forcing the hash codes learned from the image network to be close to the corresponding class-specific centers, the intraclass variation of the hash codes learned from the image network can be greatly reduced; and 3) the learned class-specific centers are inherently capable of exploiting the multi-label dependence and dealing effectively with multi-label scenarios. Experimental results on three benchmark data sets demonstrate that the proposed TSDH surpasses other state-of-the-art methods.

REFERENCES


Erkun Yang received the B.E. degree in electronic and information engineering from Xidian University, Xi’an, China, in 2013, where he is currently pursuing the Ph.D. degree with the School of Electronic Engineering.

His current research interests include computer vision and machine learning.

Tongliang Liu received the B.Eng. degree from the University of Science and Technology of China, Hefei, China, and the Ph.D. degree from the University of Technology Sydney, Ultimo, NSW, Australia.

He is currently a Lecturer with the School of Computer Science and the Faculty of Engineering, and a Core Member with the UBTECH Sydney AI Centre, The University of Sydney, Camperdown, NSW, Australia. He has authored and coauthored more than 60+ research papers including the IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE (TPAMI), the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS (T-NNLS), the IEEE TRANSACTIONS ON IMAGE PROCESSING (T-IP), IJCAI, ICML, CVPR, AAAI, IJCAI, and KDD. His current research interests include statistical learning theory, machine learning, and computer vision.

Mr. Liu was a recipient of the Discovery Early Career Researcher Award (DECRA) from Australian Research Council (ARC) and was shortlisted for the JG Russell Award by Australian Academy of Science (AAS) in 2019.

Dacheng Tao (F’15) is currently a Professor of computer science and ARC Laureate Fellow with the School of Computer Science and the Faculty of Engineering, and the Inaugural Director of the UBTECH Sydney Artificial Intelligence Centre, The University of Sydney. His current research interests include artificial intelligence, machine learning, and computer science.

Mr. Tao is a fellow of the Australian Academy of Science. He was a recipient of the 2018 IEEE ICDM Research Contributions Award.

Cheng Deng (M’11) received the B.E., M.S., and Ph.D. degrees in signal and information processing from Xidian University, Xi’an, China.

He is currently a Full Professor with the School of Electronic Engineering, Xidian University. He has authored and coauthored more than 70 scientific articles at top venues, including IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS (TNNLS), TRANSACTIONS ON IMAGE PROCESSING (TIP), TRANSACTIONS ON CYBERNETICS (TCYB), TRANSACTIONS ON MULTIMEDIA (TMM), TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, ICCV, CVPR, ICML, NIPS, IJCAI, and AAAI. His current research interests include computer vision, pattern recognition, and information hiding.

Dacheng Tao (F’15) is currently a Professor of computer science and ARC Laureate Fellow with the School of Computer Science and the Faculty of Engineering, and the Inaugural Director of the UBTECH Sydney Artificial Intelligence Centre, The University of Sydney. His current research interests include artificial intelligence, machine learning, and computer science.

Mr. Tao is a fellow of the Australian Academy of Science. He was a recipient of the 2018 IEEE ICDM Research Contributions Award.