Semi-Supervised Deep Hashing with a Bipartite Graph

Xinyu Yan, Lijun Zhang, Wu-Jun Li
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
{yanxy, zhanglj @lamda.nju.edu.cn, liwujun@nju.edu.cn

Abstract

Recently, deep learning has been successfully applied to the problem of hashing, yielding remarkable performance compared to traditional methods with hand-crafted features. However, most of existing deep hashing methods are designed for the supervised scenario and require a large number of labeled data. In this paper, we propose a novel semi-supervised hashing method for image retrieval, named Deep Hashing with a Bipartite Graph (BGDH), to simultaneously learn embeddings, features and hash codes. More specifically, we construct a bipartite graph to discover the underlying structure of data, based on which an embedding is generated for each instance. Then, we feed raw pixels as well as embeddings to a deep neural network, and concatenate the resulting features to determine the hash code. Compared to existing methods, BGDH is a universal framework that is able to utilize various types of graphs and losses. Furthermore, we propose an inductive variant of BGDH to support out-of-sample extensions. Experimental results on real datasets show that our BGDH outperforms state-of-the-art hashing methods.

1 Introduction

With the explosion in the volume of image data, it has been raised as a big challenge about how to index and organize these data efficiently and accurately. Approximate Nearest Neighbor (ANN) search [Indyk and Motwani, 1998] has become a popular way to retrieve content from images with both computational efficiency and search quality. Among existing ANN search methods, hashing is advantageous due to its fast query speed and low memory complexity [Gionis *et al.*, 1999; Gong *et al.*, 2013]. It aims to transform high-dimensional images into a set of short binary codes while maintaining similarity of the original data.

Generally speaking, hashing methods can be divided into two categories: unsupervised and supervised. Unsupervised methods utilize some kinds of distance metrics to learn a hash function from unlabeled data. Methods in this category include data-independent ones like Locally Sensitive Hashing (LSH) [Gionis *et al.*, 1999] and data-dependent ones like Iterative Quantization (ITQ) [Gong *et al.*, 2013], Spectral Hashing (SH) [Weiss *et al.*, 2009], Anchor Graph Hashing (AGH) [Liu *et al.*, 2011]. On the other hand, to deal with more complicated semantic similarity, supervised hashing methods are proposed to exploit label information to improve the hashing quality. Representative supervised methods include Latent Factor Hashing (LFH) [Zhang *et al.*, 2014], Fast Supervised Hashing (FastH) [Lin *et al.*, 2014], Supervised Discrete Hashing (SDH) [Shen *et al.*, 2015]. However, labeling large-scale image dataset is inefficient and time-consuming. As a result, Semi-Supervised Hashing (SSH) [Wang *et al.*, 2012] has been developed to make use of labeled data as well as the abundant unlabeled data.

In traditional hashing methods, images are represented by hand-crafted features such as GIST [Oliva and Torralba, 2001], and the choice of features requires heavy manual interventions. Motivated from the great success of deep neural networks in image analysis [Krizhevsky *et al.*, 2012], recently some deep hashing methods have been proposed to learn features and hash codes simultaneously [Li *et al.*, 2016; Zhu *et al.*, 2016; Liu *et al.*, 2016]. Although those deep hashing methods yield better performance compared with the traditional methods, they usually need a large number of labeled instances as training data. To address this limitation, a semi-supervised deep hashing, named SSDH, have been developed [Zhang *et al.*, 2016]. SSDH is fundamentally built upon graph-based semi-supervised learning [Zhou *et al.*, 2004] and the loss function contains a graph regularization term which involves both the labeled and unlabeled data. In theory, SSDH needs to construct a nearest neighbor graph of all the data. Unfortunately, this step takes $(\ ^2)$ time, where is the number of instances, and thus intractable for large scale data.

In this paper, we propose a novel semi-supervised hashing method, named Deep Hashing with a Bipartite Graph (BGDH), which performs graph embedding, feature learning and hash code learning in a unified framework. First, we construct a bipartite graph to capture the information hidden in the labeled and unlabeled data. The bipartite graph could be a semantic graph that describes relationships between images and concepts, an anchor graph that describes similarities between images and landmarks [Liu *et al.*, 2010], or a traditional nearest neighbor graph. Then, inspired by the recent work on graph embedding [Yang *et al.*, 2016], we learn an embedding for each instance to predict the neighborhood context in the graph. Finally, we feed both raw pixels and embeddings to a deep neural network, and concatenate the corresponding hidden layers when producing binary codes. BGDH is a general learning framework in the sense that any loss function of hashing and any type of graph can be incorporated.

Graph-based methods are usually transductive, because they can only handle instances that are already appeared in the graph. Since embeddings of instances are learnt from the graph, the basic BGDH is also transductive. To address the out-of-sample problem, we further propose an inductive variant of BGDH, in which the embeddings are defined as a parametric function of the raw features. In this way, we can produce hash codes for new instances that have not seen during training. To demonstrate the effectiveness of our approach, we conduct extensive experiments on two large-scale datasets: CIFAR-10 and NUS-WIDE. Experimental results show that BGDH outperforms other methods and achieves the state-of-the-art performance in image retrieval.

Finally, we emphasize that although both BGDH and SSDH are semi-supervised deep hashing methods, the proposed BGDH differs from SSDH in the following two aspects:

a. While SSDH is built upon graph regularization, our BGDH relies on graph embedding.
b. SSDH uses graphs to exploit the unlabeled data, in contract BGDH makes use of bipartite graphs, which can be constructed more efficiently since building an anchor graph only costs ( ) time.

2 Notations and Problem Definitions

In this section, we introduce notations and problem definitions.

2.1 Notations

We use script letters like $\mathcal{X}$ to denote sets, boldface lowercase letters like $\mathbf{e}$ to denote vectors and boldface uppercase letters like $\mathbf{E}$ to denote matrices. We denote the element at the -th row and -th column of $\mathbf{E}$ by $_{ij}$. $\mathbf{E}^T$ is the transpose of $\mathbf{E}$. $\| \cdot \|_2$ denotes the Euclidean norm of a vector and $\| \cdot \|_F$ denotes the Frobenius norm of a matrix. $\mathrm{sgn}(\cdot)$ is an element-wise sign function and $(\cdot)$ is the sigmoid function. $[\mathbf{u}; \mathbf{v}]$ denotes the concatenation of two vectors $\mathbf{u}$ and $\mathbf{v}$.

2.2 Problem Definitions

Given a set of instances/images $\mathcal{X} = \{\mathbf{x}_i \;_{i=1}$ where $\mathbf{x}_i$ is the feature vector of the -th instance. Without loss of generality, we assume the first instances $\{\mathbf{x}_1 \quad \mathbf{x}$ are labeled and the rest are unlabeled. We assume the supervised information is given in term of pairwise labels, though our method can also support other kinds of labels. Specifically, for the first instances, we have a set of pairwise labels $\mathcal{S} = \{ \;_{ij}$ with $_{ij} \in \{0 \;\, 1$, where $_{ij} = 1$ means that $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar, $_{ij} = 0$ means that $\mathbf{x}_i$ and $\mathbf{x}_j$ are dissimilar. In addition, a bipartite graph $\mathcal{G} = (\mathcal{X} \;\; \mathcal{O} \;\; \mathcal{E})$ between instances and objects are given, where $\mathcal{O}$ is a set of objects such as concepts or landmarks, and $\mathcal{E}$ is the set of edges.

The goal of our semi-supervised hashing method is to learn a mapping function $\mathcal{H} : \mathbf{x}_i \rightarrow \{-1 \;\, 1 \;^c$, which encodes each point $\mathbf{x}_i$ into a -dimensional binary code $\mathbf{b}_i = \mathcal{H}(\mathbf{x}_i) = [\;_1(\mathbf{x}_i) \;\; _2(\mathbf{x}_i) \cdots \;\; _c(\mathbf{x}_i)]^T \in \{-1 \;\, 1 \;^c$. The binary codes $\mathcal{B} = \{\mathbf{b}_i \;_{i=1}$ should preserve the semantic similarity and structure similarity in the Hamming space.

3 Semi-Supervised Deep Hashing with a Bipartite Graph

In this section, we first present the details of our semi-supervised Deep Hashing with a Bipartite Graph (BGDH), then introduce an inductive variant and finally discuss the learning procedure.

3.1 The Proposed BGDH Framework

The end-to-end deep learning architecture of our BGDH is shown in Figure 1, which includes three main components: graph embedding, feature learning, and hash code learning. Similar to other semi-supervised learning methods [Yang *et al.*, 2016], the loss function of BGDH can be expressed as

$$\mathcal{L} \;\; + \;\; \mathcal{L}_g \tag{1}$$

where $\mathcal{L}$ is a supervised loss designed to preserve the similarity between pairwise instances, and $\mathcal{L}_g$ is an unsupervised loss of predicting the graph context. In the following, we first introduce $\mathcal{L}_g$ which aims to learn embeddings from the bipartite graph $\mathcal{G}$, then formulate $\mathcal{L}$ which is used to learn both features and binary codes from hidden layers of deep networks.

Graph embedding

We propose to use a bipartite graph $\mathcal{G} = (\mathcal{X} \;\; \mathcal{O} \;\; \mathcal{E})$ to capture the information hidden in the unlabeled data. It can be constructed in different ways as stated below.

- An anchor graph constructed from the dataset $\mathcal{X}$. In this case, $\mathcal{O}$ contains landmarks and the construction of $\mathcal{G}$ takes ( ) time [Liu *et al.*, 2010].
- A nearest neighbor graph. In this case, $\mathcal{O} = \mathcal{X}$ and the construction of $\mathcal{G}$ takes $(\;^2)$ time.
- A semantic graph constructed from external data. In this case, $\mathcal{O}$ may contain concepts, styles, or owners.

In the following, we briefly introduce one way to construct an anchor graph. We first randomly sample instances from $\mathcal{X}$ as landmarks, denoted by $\mathcal{O} = \{\mathbf{o}_1 \quad \mathbf{o}$. Then, we put an edge between $\mathbf{x}_i$ and $\mathbf{o}_j$ if $\mathbf{o}_j$ is among nearest landmarks of $\mathbf{x}_i$, or if the distance between them is smaller than some threshold . Let $\mathbf{A} \in \mathbb{R} \;^\times$ be the similarity matrix of $\mathcal{G}$, where $_{ij}$ denotes the weight of the edge between $\mathbf{x}_i$ and $\mathbf{o}_j$. The value of $_{ij}$ may be binary, that is, $_{ij} = 1$ if there is an edge, otherwise 0. If a real value is preferred, $_{ij}$ can be set according to the heat kernel: $_{ij} = \;^{-\|\mathbf{x}_i - \mathbf{o}_j\|_2^2/\rho}$, where 0 is a parameter.

The goal of graph embedding is to learn an embedding for each instance that predicts the context in the graph. Given an instance and its context, the objective of graph embedding is usually formulated as minimizing certain loss of predicting the context using the embedding of an instance as input feature [Weston *et al.*, 2012; Mikolov *et al.*, 2013]. The context of an instance can be simply defined as its neighbors in the graph, or generated by sophisticated methods such as random

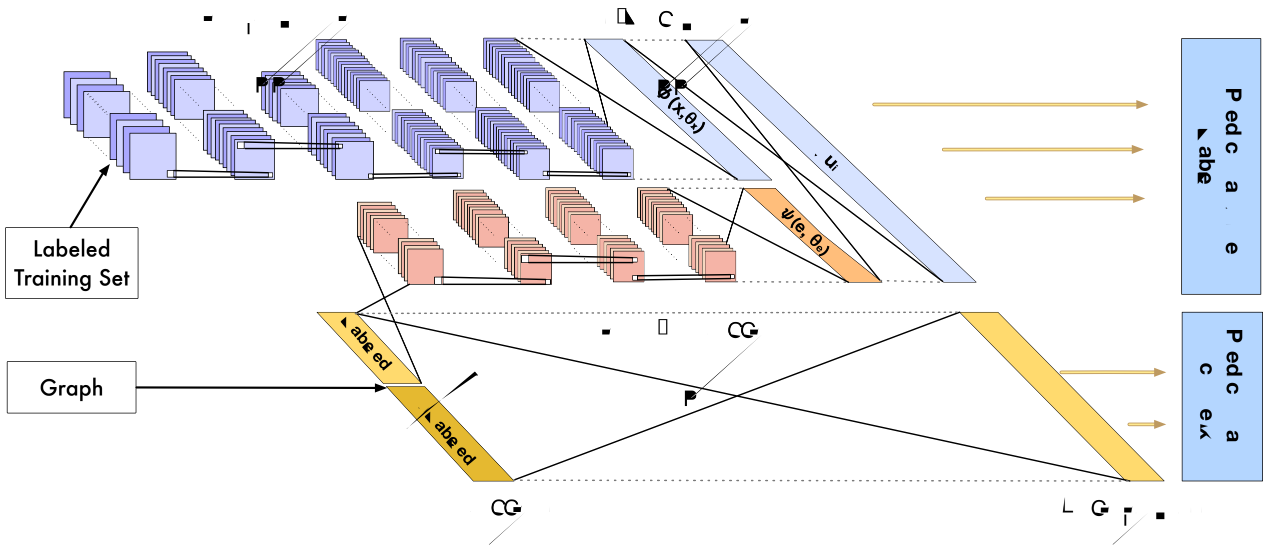


Figure 1: The end-to-end deep learning architecture of BGDH.

walk [Perozzi *et al.*, 2014]. Following previous studies [Yang *et al.*, 2016], we present a simple procedure for context generation in Algorithm 1, where the parameter is the length of the random walk and $\in (0\ 1)$ determines the ratio of positive contexts.

By invoking Algorithm 1 times, we obtain a set of triples $\{(\ \ \ )\}_{=1}$ where $= 1$ indicates node is a context of node , and $= -1$ indicates is not a context. Let $\mathbf{e}_{i_j}$ be the embedding of node , and $\mathbf{w}_{c_j}$ be the parameters for predicting node as a context. We define the objective function of graph embedding

$$\mathcal{L}_g = \frac{1}{\ }\sum_{=1} (\mathbf{e}_{i_j}^\top \mathbf{w}_{c_j}\ \ ) \tag{2}$$

where $(\cdot\ \cdot)$ is a loss that measures the discrepancy between $\mathbf{e}_{i_j}^\top \mathbf{w}_{c_j}$ and . In machine learning, the following losses are commonly used.

- The square loss

$$(\mathbf{e}_{i_j}^\top \mathbf{w}_{c_j}\ \ ) = \left(\ - \mathbf{e}_{i_j}^\top \mathbf{w}_{c_j}\right)^2$$

- The logistic loss

$$(\mathbf{e}_{i_j}^\top \mathbf{w}_{c_j}\ \ ) = \log\left(1 + {}^{-\gamma_j \mathbf{e}_{i_j}^\top \mathbf{w}_{c_j}}\right)$$

To constrain the solution space, we may further impose sparse constraints or nonnegative constraints [Lee and Seung, 1999].

Feature learning and hash code learning

We utilize deep neural network model to learn features from the raw pixels and embeddings of labeled instances, and then combine them together to learn the hash codes. BGDH contains a CNN model to learn features from raw image pixels, and the model has seven layers as those of CNN-F [Chatfield *et al.*, 2014] while other networks like AlexNet [Krizhevsky and Hinton, 2009] can be used too. The configuration of the network is presented in Table 1, and a detailed explanation can be found in [Li *et al.*, 2016]. The output of the last feature learning layer (full7) of labeled instance $\mathbf{x}_i$ is represented by $(\mathbf{x}_i;\ )$, where denotes all the parameters in the seven layers of feature learning part. In contrast with existing supervised hashing methods, we also learn features from embeddings of labeled instances. The output associated with embedding $\mathbf{e}_i$ is denoted by $(\mathbf{e}_i;\ _e)$, where $_e$ contains all the parameters in hidden layers. In this paper, we only add one fully-connected layer for embeddings, of which the size is determined by the dimension of embeddings.

Algorithm 1 Context generation based on random walk

```
1:  Input: A bipartite graph $\mathcal{G} = (\mathcal{X}\ \mathcal{O}\ \mathcal{E})$, parameters  and
2:  Uniformly sample an instance  from $\mathcal{X}$
3:  Uniformly sample a random variable  from $[0\ 1]$
4:  if  then
5:      $\leftarrow +1$
6:      Uniformly sample a random walk sequence  of length  started from
7:      Uniformly sample a context  from  except
8:  else
9:      $\leftarrow -1$
10:     Uniformly sample context  from $\mathcal{X}$
11: end if
12: return (  )
```

We concatenate $(\mathbf{x}_i;\ )$ and $(\mathbf{e}_i;\ _e)$ as a new feature for instance , then send it to a hash code learning layer as:

$$\mathbf{u}_i = \mathbf{M}^T[\ (\mathbf{x}_i;\ );\ \ (\mathbf{e}_i;\ _e)] + \mathbf{v} \tag{3}$$

where $\mathbf{M} \in \mathbb{R}^{(4096+d)\times c}$ denotes a weight matrix, is the dimension of embedding, and $\mathbf{v} \in \mathbb{R}^{c\times 1}$ is a bias vector. Note that any supervised loss function of hashing can be used in our framework to learn parameters $\mathbf{M}$ and $\mathbf{v}$. In this paper, we

Table 1: Configuration of the feature learning network

Layer	Configuration
conv1	filter 64×11×11, stride 4×4, pad 0, LRN, pool 2×2
conv2	filter 256×5×5, stride 1×1, pad 2, LRN, pool 2×2
conv3	filter 256×3×3, stride 1×1, pad 1
conv4	filter 256×3×3, stride 1×1, pad 1
conv5	filter 256×3×3, stride 1×1, pad 1, pool 2×2
full6	4096
full7	4096

choose the loss function of deep pairwise-supervised hashing (DPSH) [Li *et al.*, 2016], and $\mathcal{L}$ is given by

$$\mathcal{L} = -\sum_{ij\in\mathcal{S}} ({}_{ij}\Theta_{ij} - \log(1 + {}^{\Theta_{ij}})) + \sum_{i=1} \| \mathbf{b}_i - \mathbf{u}_i \|_2^2 \tag{4}$$

where $\Theta_{ij} = \frac{1}{2}\mathbf{u}_i^T\mathbf{u}_j$ and 0 is a regularization parameter. By substituting Eq. (3) into Eq. (4), we obtain the final loss function of the supervised part:

$$\mathcal{L} = -\sum_{ij\in\mathcal{S}} ({}_{ij}\Theta_{ij} - \log(1 + {}^{\Theta_{ij}})) + \sum_{i=1} \left\|\mathbf{b}_i - (\mathbf{M}^T[(\mathbf{x}_i;); (\mathbf{e}_i; {}_e)] + \mathbf{v})\right\|_2^2 \tag{5}$$

The objective of BGDH

We combine the supervised part and unsupervised part to form the transductive version of BGDH. From (2) and (5), the loss function of BGDH is

$$\mathcal{L} + \mathcal{L}_g = -\sum_{ij\in\mathcal{S}} ({}_{ij}\Theta_{ij} - \log(1 + {}^{\Theta_{ij}})) + \sum_{i=1} \left\|\mathbf{b}_i - (\mathbf{M}^T[(\mathbf{x}_i;); (\mathbf{e}_i; {}_e)] + \mathbf{v})\right\|_2^2 + - \sum_{j=1} (\mathbf{e}_{i_j}^\top \mathbf{w}_{c_j} \; {}_j) \tag{6}$$

where 0 is a constant weighting factor. The first two terms are the loss of predicting pairwise labels and the third one is the loss of predicting context. As a result, our BGDH can simultaneously learn embeddings, features, and hash codes. During the training phase, semantic similarity can affect graph embeddings, at the same time structure of data also influence the prediction of pairwise labels.

3.2 An Inductive Variant

Note that the basic BGDH is transductive, because the embeddings of instances are learnt from the graph. Since the hash code of an instance depends on both the raw pixels and its embedding, we need to design a way to infer the embedding of a unseen instance. To this end, we insert hidden layers to connect the raw pixels and embedding [Yang *et al.*, 2016], and in this way, the embedding $\mathbf{e}_i$ becomes a parameterized function of $\mathbf{x}_i$, denoted by $(\mathbf{x}_i;)$. The loss function of inductive hashing model can be written as:

$$\mathcal{L} + \mathcal{L}_g = -\sum_{ij\in\mathcal{S}} ({}_{ij}\Theta_{ij} - \log(1 + {}^{\Theta_{ij}})) + \sum_{i=1} \left\|\mathbf{b}_i - (\mathbf{M}^T[(\mathbf{x}_i;); ((\mathbf{x}_i;); {}_e)] + \mathbf{v})\right\|_2^2 + - \sum_{j=1} (\mathbf{e}_{i_j}^\top \mathbf{w}_{c_j} \; {}_j) \tag{7}$$

We can predict the hash code of any point $\mathbf{x} \in \mathcal{X}$ as:

$$\mathbf{b} = \mathrm{sgn}(\mathbf{M}^T[(\mathbf{x} ;); ((\mathbf{x} ;); {}_e)] + \mathbf{v}) \tag{8}$$

3.3 Learning

In the transductive version of BGDH, the optimization variables include $\mathbf{M}$, $\mathbf{v}$, $\{\mathbf{b}_i$, , ${}_e$, $\{\mathbf{e}_i$ and $\{\mathbf{w}_c$. We adopt stochastic gradient descent (SGD) [Bottou, 2010] to train our model.

First, we sample a batch of labeled instances of which set $\mathcal{I}_1$ contains indexes. A gradient step is then taken to optimize the supervised loss $\mathcal{L}$. For all $\in \mathcal{I}_1$, $\mathbf{b}_i$ can be directly optimized as follow:

$$\mathbf{b}_i = \mathrm{sgn}(\mathbf{u}_i) = \mathrm{sgn}\left(\mathbf{M}^T[(\mathbf{x}_i;); (\mathbf{e}_i; {}_e)] + \mathbf{v}\right) \tag{9}$$

For other parameters in $\mathcal{L}$, i.e., $\mathbf{M}$, $\mathbf{v}$, , ${}_e$, and $\{\mathbf{e}_i : \in \mathcal{I}_1$, we use back propagation (BP) to optimize them. Derivatives of $\mathcal{L}$ w. r. t. $\mathbf{u}_i$ are presented as follows:

$$\frac{\mathcal{L}}{\mathbf{u}_i} = \frac{1}{2}\sum_{j: ij\in\mathcal{S}} ({}_{ij} - {}_{ij})\mathbf{u}_j + \frac{1}{2}\sum_{j: ji\in\mathcal{S}} ({}_{ji} - {}_{ji})\mathbf{u}_j + 2 (\mathbf{u}_i - \mathbf{b}_i) \tag{10}$$

where ${}_{ij} = (\frac{1}{2}\mathbf{u}_i^T\mathbf{u}_j)$. We can then update other parameters according to

$$\frac{\mathcal{L}}{\mathbf{M}} = [(\mathbf{x}_i;); (\mathbf{e}_i; {}_e)] \left(\frac{\mathcal{L}}{\mathbf{u}_i}\right)^T \tag{11}$$

$$\frac{\mathcal{L}}{\mathbf{v}} = \frac{\mathcal{L}}{\mathbf{u}_i} \tag{12}$$

$$\frac{\mathcal{L}}{ (\mathbf{x}_i;)} = \frac{\mathcal{L}}{ (\mathbf{e}_i; {}_e)} = \mathbf{M}\frac{\mathcal{L}}{\mathbf{u}_i} \tag{13}$$

Then, we perform a gradient step to optimize the unsupervised graph embedding loss $\mathcal{L}_g$ calculated by sampling triples generated in Algorithm 1. In this case parameters $\{\mathbf{e}_i \; \mathbf{w}_c :$ $(\;) \in \mathcal{I}_2$ will be updated where $\mathcal{I}_2$ denotes the set containing indexes of sampled instances and contexts. The above procedures are repeated for ${}_1$ and ${}_2$ iterations respectively.

The whole learning algorithm of BGDH is summarized in Algorithm 2. Notice that before training jointly, we first train unsupervised part for a number of iterations to learn the initialization embeddings $\{\mathbf{e}_i$. For the inductive variant, we will update parameters instead of embeddings $\{\mathbf{e}_i$.

Algorithm 2 Learning algorithm for BGDH

```
1:  Input: A bipartite graph $\mathcal{G} = (\mathcal{X}\ \mathcal{O}\ \mathcal{E})$, images $\mathcal{X} =$
    $\{\mathbf{x}_i\ _{i=1}$, pairwise labels $\mathcal{S} = \{$ parameters , ,
    batch iterations $_1$, $_2$ and sizes $_1$, $_2$
2:  Output: Binary codes $\mathcal{B} = \{\mathbf{b}_i\ _{i=1}$
3:  Initialization: Initialize with the pre-trained CNN-F
    model on ImageNet; Initialize each entry of $\mathbf{M}$, $\mathbf{v}$, and $_e$
    by randomly sampling from a Gaussian distribution with
    mean $0$ and variance $0\ 01$.
4:  REPEAT
5:  for $\leftarrow 1$ to $_1$ do
6:      Randomly sample $_1$ labeled images, and let $\mathcal{I}_1$ be
        the set containing indexes of sampled instances
7:      Calculate $(\mathbf{x}_i;\ )$ and $(\mathbf{e}_i;\ _e)$ for all $\in \mathcal{I}_1$ by
        forward propagation
8:      Compute $\mathbf{u}_i = \mathbf{M}^T[\ (\mathbf{x}_i;\ );\ (\mathbf{e}_i;\ _e)] + \mathbf{v}$ and the
        binary code of $\mathbf{x}_i$ with $\mathbf{b}_i = \mathrm{sgn}(\mathbf{u}_i)$ for all $\in \mathcal{I}_1$
9:      Compute the derivative of $\mathcal{L}$ w. r. t. $\{\mathbf{u}_i :\ \in \mathcal{I}_1$
10:     Update parameters $\mathbf{M}$, $\mathbf{v}$, , $_e$, and $\{\mathbf{e}_i :\ \in \mathcal{I}_1$
        by back propagation
11: end for
12: for $\leftarrow 1$ to $_2$ do
13:     Randomly generate a batch of triples by invoking Al-
        gorithm 1 $_2$ times, and let $\mathcal{I}_2$ be the set containing in-
        dexes of sampled instances and contexts
14:     the r t. L$_g$ w fe 2sgn(4sgn( 2 1g10n s
                                          end for
                                      11:
```

Table 2: Accuracy in terms of MAP. The best MAPs for each category are shown in boldface. Training size for supervised method is 5000 for CIFAR-10 and 10500 for NUS-WIDE.

Method	CIFAR-10 (MAP)				NUS-WIDE (MAP)			
	12-bits	24-bits	32-bits	48-bits	12-bits	24-bits	32-bits	48-bits
BGDH-T	**0.805**	**0.824**	**0.826**	**0.833**	**0.803**	**0.818**	**0.822**	**0.828**
BGDH-I	0.803	0.818	0.822	0.829	0.801	0.815	0.816	0.825
SSDH	0.801	0.813	0.812	0.814	0.773	0.779	0.778	0.778
DSH	0.604	0.746	0.781	0.810	0.751	0.765	0.767	0.773
DHN	0.692	0.703	0.726	0.735	0.751	0.785	0.792	0.799
DPSH	0.684	0.734	0.750	0.767	0.788	0.809	0.817	0.823
COSDISH	0.522	0.590	0.599	0.615	0.691	0.749	0.745	0.765
SDH	0.525	0.671	0.686	0.696	0.752	0.745	0.744	0.730
FastH	0.291	0.351	0.367	0.390	0.622	0.660	0.670	0.687
LFH	0.335	0.433	0.509	0.515	0.749	0.751	0.775	0.780
ITQ	0.163	0.170	0.173	0.176	0.447	0.465	0.468	0.473
LSH	0.152	0.167	0.170	0.200	0.367	0.394	0.413	0.416
IsoH	0.158	0.162	0.166	0.169	0.436	0.454	0.461	0.465
SpH	0.141	0.153	0.154	0.158	0.399	0.437	0.454	0.465

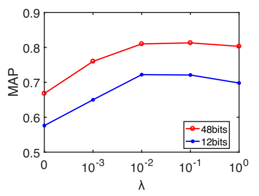
Table 3: Accuracy in terms of MAP. The best MAPs for each category are shown in boldface. Training size for supervised method is 2500 for CIFAR-10 and 5000 for NUS-WIDE.

Method	CIFAR-10 (MAP)				NUS-WIDE (MAP)			
	12-bits	24-bits	32-bits	48-bits	12-bits	24-bits	32-bits	48-bits
BGDH-T	**0.755**	**0.791**	**0.800**	**0.812**	**0.772**	**0.798**	**0.806**	**0.816**
BGDH-I	0.746	0.776	0.787	0.796	0.768	0.794	0.801	0.811
SSDH	0.581	0.589	0.595	0.596	0.743	0.745	0.746	0.749
DSH	0.617	0.707	0.737	0.761	0.749	0.769	0.771	0.786
DHN	0.591	0.646	0.640	0.662	0.741	0.763	0.766	0.773
DPSH	0.576	0.634	0.642	0.668	0.762	0.789	0.791	0.803
COSDISH	0.312	0.348	0.373	0.398	0.648	0.678	0.699	0.713
SDH	0.327	0.357	0.374	0.377	0.574	0.597	0.591	0.595
FastH	0.267	0.298	0.320	0.341	0.604	0.634	0.650	0.667
LFH	0.244	0.288	0.311	0.391	0.611	0.644	0.653	0.669

outperforms all the other methods. Specifically, compared to the best baseline in Table 2, we conclude that when labeled data are insufficient, BGDH is able to leverage unlabeled data to deliver a good result.

4.3 Parameter Selection

In BGDH, there is a hyper-parameter which controls the tradeoff between supervised loss and unsupervised loss. Figure 2 displays the impacts of on the performance of BGDH with the experiment settings being the same as those in Table 3. As can be seen, there is a wide range of that BGDH performs well. Thus, to a large extent, BGDH is insensitive to and the parameter selection is not a crucial problem in our algorithm. Additionally, by comparing the MAP of $= 0$ and $= 1$, we verify the importance of graph embedding.



(a) CIFAR-10

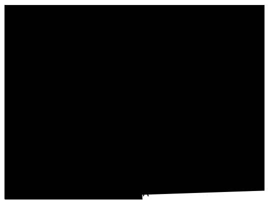

(b) NUS-WIDE

Figure 2: Hyper-parameter Sensitivity

5 Conclusion

In this paper, we propose a novel semi-supervised hashing method, named Deep Hashing with a Bipartite Graph (BGDH). To the best of our knowledge, BGDH is the first method that performs graph embedding, feature learning, and hash code learning simultaneously. BGDH constructs a bipartite graph to discover the underlying structure of data, and is much more efficient than methods based on neighborhood graph. Experimental results demonstrate that BGDH outperforms state-of-the-art methods in image retrieval.

Acknowledgements

This work was partially supported by the NSFC (61603177, 61472182), JiangsuSF (BK20160658), and the Collaborative Innovation Center of Novel Software Technology and Industrialization of Nanjing University.

References

[Bottou, 2010] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of 19th International Conference on Computational Statistics*, pages 177–186. Springer, 2010.

[Chatfield *et al.*, 2014] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proceedings of British Machine Vision Conference*, 2014.

[Deng *et al.*, 2009] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Proceedings of 25th International Conference on Very Large Data Bases*, volume 99, pages 518–529, 1999.

[Gong *et al.*, 2013] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2013.

[Heo *et al.*, 2012] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2957–2964, 2012.

[Indyk and Motwani, 1998] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[Kang *et al.*, 2016] Wang-Cheng Kang, Wu-Jun Li, and Zhi-Hua Zhou. Column sampling based discrete supervised hashing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1230–1236, 2016.

[Kong and Li, 2012] Weihao Kong and Wu-Jun Li. Isotropic hashing. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2012.

[Krizhevsky and Hinton, 2009] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105. 2012.

[Lai *et al.*, 2015] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3270–3278, 2015.

[Lee and Seung, 1999] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[Li *et al.*, 2016] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. Feature learning based deep supervised hashing with pairwise labels. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1711–1717, 2016.

[Lin *et al.*, 2014] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1963–1970, 2014.

[Liu *et al.*, 2010] Wei Liu, Junfeng He, and Shih-Fu Chang. Large graph construction for scalable semi-supervised learning. In *Proceedings of the 27th International Conference on Machine Learning*, pages 679–686, 2010.

[Liu *et al.*, 2011] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1–8, 2011.

[Liu *et al.*, 2016] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. Deep supervised hashing for fast image retrieval. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2064–2072, 2016.

[Mikolov *et al.*, 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[