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 capture the information hidden in 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 $O(n^2)$ time, where $n$ 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 embed-
ding 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 been 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.

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

\[ L_s + \lambda L_g \] (1)

where \( L_s \) is a supervised loss designed to preserve the similarity between pairwise instances, and \( L_g \) is an unsupervised loss of predicting the graph context. In the following, we first introduce \( L_g \) which aims to learn embeddings from the bipartite graph \( G \), then formulate \( L_s \) 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 \( G = (X', O, E) \) 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 \( X' \). In this case, \( O \) contains \( m \) landmarks and the construction of \( G \) takes \( O(n \log n) \) time [Liu et al., 2010].
- A nearest neighbor graph. In this case, \( O = X' \) and the construction of \( G \) takes \( O(n^2) \) time.
- A semantic graph constructed from external data. In this case, \( O \) may contain \( m \) concepts, styles, or owners.

In the following, we briefly introduce one way to construct an anchor graph. We first randomly sample \( m \) instances from \( X' \) as landmarks, denoted by \( O = \{o_1, \ldots, o_m\} \). Then, we put an edge between \( x_i \) and \( o_j \) if \( o_j \) is among \( k \) nearest landmarks of \( x_i \), or if the distance between them is smaller than some threshold \( \epsilon \). Let \( A \in \mathbb{R}^{n \times m} \) be the similarity matrix of \( G \), where \( A_{ij} \) denotes the weight of the edge between \( x_i \) and \( o_j \). The value of \( A_{ij} \) may be binary, that is, \( A_{ij} = 1 \) if there is an edge, otherwise, 0. If a real value is preferred, \( A_{ij} \) can be set according to the heat kernel: \( A_{ij} = e^{-\|x_i - o_j\|^2/\rho} \), where \( \rho > 0 \) 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 concept of an instance can be simply defined as its neighbors in the graph, or generated by sophisticated methods such as random
Algorithm 1: Context generation based on random walk

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

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

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 $x_i$ is represented by $\phi(x_i; \theta_x)$, where $\theta_x$ 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 $e_i$ is denoted by $\psi(e_i; \theta_e)$, where $\theta_e$ 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.

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

```
L^2 = \frac{1}{t} \sum_{j=1}^{t} L(e_i^T w_{e_j}, \gamma_j)
```

where $L(\cdot, \cdot)$ is a loss that measures the discrepancy between $e_i^T w_{e_j}$ and $\gamma_j$. In machine learning, the following losses are commonly used.

- The square loss
  
  $L(e_i^T w_{e_j}, \gamma_j) = (\gamma_j - e_i^T w_{e_j})^2$

- The logistic loss
  
  $L(e_i^T w_{e_j}, \gamma_j) = \log(1 + e^{-\gamma_i e_i^T w_{e_j}})$

To constrain the solution space, we may further impose sparse constraints or nonnegative constraints [Lee and Seung, 1999].
choose the loss function of deep pairwise-supervised hashing (DPSSH) [Li et al., 2016], and \( \mathcal{L}_s \) is given by

\[
\mathcal{L}_s = - \sum_{s_{ij} \in S} (s_{ij} \Theta_{ij} \log(1 + e^{\Theta_{ij}})) + \eta \sum_{i=1}^t \| \mathbf{b}_i - \mathbf{u}_j \|^2_2,
\]

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

\[
\mathcal{L}_s = - \sum_{s_{ij} \in S} (s_{ij} \Theta_{ij} \log(1 + e^{\Theta_{ij}})) + \eta \sum_{i=1}^t \| \mathbf{b}_i - (\mathbf{M}^T \phi(\mathbf{x}_i; \theta_x); \psi(\mathbf{e}_i; \theta_e) + \mathbf{v}) \|^2_2.
\]

(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

\[
\mathcal{L}_s + \lambda \mathcal{L}_g = - \sum_{s_{ij} \in S} (s_{ij} \Theta_{ij} \log(1 + e^{\Theta_{ij}})) + \eta \sum_{i=1}^t \| \mathbf{b}_i - (\mathbf{M}^T \phi(\mathbf{x}_i; \theta_x); \psi(\mathbf{e}_i; \theta_e) + \mathbf{v}) \|^2_2
\]

\[+ \frac{\lambda}{t} \sum_{j=1}^t \ell(\mathbf{e}_j^T \mathbf{w}_j, \gamma_j)\]

where \( \lambda > 0 \) 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 an 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 \( e_i \) becomes a parameterized function of \( x_i \), denoted by \( \xi(x_i; \theta_x) \). The loss function of inductive hashing model can be written as:

\[
\mathcal{L}_s + \lambda \mathcal{L}_g = - \sum_{s_{ij} \in S} (s_{ij} \Theta_{ij} \log(1 + e^{\Theta_{ij}})) + \eta \sum_{i=1}^t \| \mathbf{b}_i - (\mathbf{M}^T \phi(\mathbf{x}_i; \theta_x); \psi(\xi(x_i; \theta_x); \theta_e) + \mathbf{v}) \|^2_2
\]

\[+ \frac{\lambda}{t} \sum_{j=1}^t \ell(\mathbf{e}_j^T \mathbf{w}_j, \gamma_j)\]

We can predict the hash code of any point \( x_i \notin X \) as:

\[
\mathbf{b}_i = \text{sgn}(\mathbf{M}^T \phi(\mathbf{x}_i; \theta_x); \psi(\xi(x_i; \theta_x); \theta_e) + \mathbf{v}).
\]

3.3 Learning

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

First, we sample a batch of labeled instances of which set \( T_1 \) contains indexes. A gradient step is then taken to optimize the supervised loss \( \mathcal{L}_s \). For all \( i \in T_1 \), \( \mathbf{b}_i \) can be directly optimized as follow:

\[
\mathbf{b}_i = \text{sgn}(\mathbf{u}_i) = \text{sgn}(\mathbf{M}^T \phi(\mathbf{x}_i; \theta_x); \psi(\mathbf{e}_i; \theta_e) + \mathbf{v}) \quad (9)
\]

For other parameters in \( \mathcal{L}_s \), i.e., \( \mathbf{M}, \mathbf{v}, \theta_x, \theta_e, \) and \( \{\mathbf{e}_i : i \in T_1\} \), we use back propagation (BP) to optimize them. Derivatives of \( \mathcal{L}_s \) w. r. t. \( \mathbf{u}_i \) are presented as follows:

\[
\frac{\partial \mathcal{L}_s}{\partial \mathbf{u}_i} = \frac{1}{2} \sum_{j: s_{ij} \in S} (a_{ij} - s_{ij}) \mathbf{u}_j + \frac{1}{2} \sum_{j: s_{ij} \in S} (a_{ji} - s_{ji}) \mathbf{u}_j + 2\eta (\mathbf{u}_i - \mathbf{b}_i)
\]

(10)

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

\[
\frac{\partial \mathcal{L}_s}{\partial \mathbf{M}} = [\phi(\mathbf{x}_i; \theta_x); \psi(\mathbf{e}_i; \theta_e)] \left( \frac{\partial \mathcal{L}_s}{\partial \mathbf{u}_i} \right)^T, \quad (11)
\]

\[
\frac{\partial \mathcal{L}_s}{\partial \mathbf{v}} = \frac{\partial \mathcal{L}_s}{\partial \mathbf{u}_i}, \quad (12)
\]

\[
\frac{\partial \mathcal{L}_s}{\partial \phi(\mathbf{x}_i; \theta_x)} = \frac{\partial \mathcal{L}_s}{\partial \psi(\mathbf{e}_i; \theta_e)} = \mathbf{M} \frac{\partial \mathcal{L}_s}{\partial \mathbf{u}_i}. \quad (13)
\]

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 \( \{e_i, w_c : (i, c) \in T_2\} \) will be updated where \( T_2 \) denotes the set containing indexes of sampled instances and contexts. The above operations are repeated for \( T_1 \) and \( T_2 \) 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 \( \{e_i\} \). For the inductive variant, we will update parameters \( \theta_x \) instead of embeddings \( \{e_i\} \).
Algorithm 2 Learning algorithm for BGDH

1: **Input:** A bipartite graph \( \mathcal{G} = (\mathcal{X}, \mathcal{E}) \), images \( \mathcal{X} = \{x_i\}_{i=1}^n \), pairwise labels \( \mathcal{S} = \{s_{ij}\} \), parameters \( \eta, \lambda \), batch iterations \( T_1, T_2 \) and sizes \( N_1, N_2 \)

2: **Output:** Binary codes \( \mathcal{B} = \{b_i\}_{i=1}^n \)

3: **Initialization:** Initialize \( \theta_x \) with the pre-trained CNN-F model on ImageNet; Initialize each entry of \( \mathbf{M}, \mathbf{v}, \) and \( \theta_e \) by randomly sampling from a Gaussian distribution with mean 0 and variance 0.01.

4: **REPEAT**

5: for \( i \leftarrow 1 \) to \( T_1 \) do

6: Randomly sample \( N_1 \) labeled images, and let \( \mathcal{I}_1 \) be the set containing indexes of sampled instances

7: Calculate \( \phi(x_i; \theta_x) \) and \( \psi(e_i; \theta_e) \) for all \( i \in \mathcal{I}_1 \) by forward propagation

8: Compute \( \mathbf{u}_i = \mathbf{M}^T[\phi(x_i; \theta_x); \psi(e_i; \theta_e)] + \mathbf{v} \) and the binary code of \( x_i \) with \( \mathbf{b}_i = \text{sgn}(\mathbf{u}_i) \) for all \( i \in \mathcal{I}_1 \)

9: Compute the derivative of \( \mathcal{L}_{\theta} \) w. r. t. \( \{\mathbf{u}_i : i \in \mathcal{I}_1\} \)

10: Update parameters \( \mathbf{M}, \mathbf{v}, \theta_x, \theta_e, \) and \( \{e_i : i \in \mathcal{I}_1\} \) by back propagation

11: **end for**

12: for \( i \leftarrow 1 \) to \( T_2 \) do

13: Randomly generate a batch of triples by invoking Algorithm 1 \( N_2 \) times, and let \( \mathcal{I}_2 \) be the set containing indexes of sampled instances and contexts

14: Compute the derivative of \( \mathcal{L}_{\theta} \) w. r. t. \( \{\mathbf{e}_i, \mathbf{w}_e : (i, c) \in \mathcal{I}_2\} \)

15: Update parameters \( \{\mathbf{e}_i, \mathbf{w}_e : (i, c) \in \mathcal{I}_2\} \)

16: **end for**

17: **UNTIL** stopping

18: Calculate \( \mathbf{b}_i = \text{sgn}(\mathbf{M}^T[\phi(x_i; \theta_x); \psi(e_i; \theta_e)] + \mathbf{v}) \) for \( i = 1, \ldots, n \)

4 Experiments

In this section, we present experimental results. All the experiments are performed on a NVIDIA K80 GPU server with MatConvNet [Vedaldi and Lenc, 2014].

4.1 Datasets and Setting

We conduct experiments on two widely used benchmark datasets: CIFAR-10 and NUS-WIDE. The CIFAR-10 dataset\(^1\) consists of 60,000 images from 10 classes (6000 images per class). It is a single-label dataset. The NUS-WIDE dataset\(^2\) contains 209,648 images from Flickr. Following the settings in [Lai et al., 2015], we use 19,5834 images belonging to the 21 most frequent classes and each class consists at least 5000 images. Additionally, two images are defined as a ground-truth neighbor when they share at least one common label.

In our experiments, BGDH-T denotes transductive model while BGDH-I denotes inductive one. We compare our method with several state-of-the-art methods including unsupervised methods: ITQ [Gong et al., 2013], LSH [Gionis et al., 1999], IsoH [Kong and Li, 2012] and SpH [Heo et al., 2012]; supervised methods: LFH [Zhang et al., 2014], FastH [Lin et al., 2014], SDH [Shen et al., 2015] and COSDISH [Kang et al., 2016]; supervised deep hashing methods: DPSH [Li et al., 2016], DHN [Zhu et al., 2016] and DSH [Liu et al., 2016]; semi-supervised deep methods SSDH [Zhang et al., 2016].

For hashing methods using hand-crafted features, we represent each image in CIFAR-10 by a 512-dimensional GIST vector. In NUS-WIDE, an image is represented by a 1134-dimensional feature vector, including 500-D bag-of-words features, 64-D color histogram, 144-D color correlogram, 73-D edge direction histogram, 128-D wavelet texture, 225-D block-wise color moments. For deep methods, we use the raw image pixels as input. The network architectures of these deep models are different from each other, for fair comparison, we adopt the same deep network architecture CNN-F with the same initialization parameters pre-trained on ImageNet [Deng et al., 2009] for all deep hashing methods. The bipartite graph of BGDH is constructed based on hand-crafted features with heat kernel, where the hyper-parameter \( \rho \) is set as 1 for CIFAR-10 and 10 for NUS-WIDE. The hyper-parameter \( \eta \) in BGDH is set as 10 for CIFAR-10 and 100 for NUS-WIDE similar to DPSH [Li et al., 2016]. We simply set \( T_1 = 10, T_2 = 5, \lambda = 0.1 \) in all the experiments.

4.2 Experimental Results

To evaluate the retrieval accuracy, we use the Mean Average Precision (MAP) as the evaluation metric. Following [Lai et al., 2015; Xia et al., 2014], 1000 images (100 images per class) are randomly selected as the query set in CIFAR-10. We use all the rest of images as training set for the unsupervised methods and for building a neighbour graph \( \mathcal{G} \) in BGDH. We randomly select 5000 (500 images per class) images as the training set for supervised methods.

In NUS-WIDE, we randomly sample 2100 query images from 21 most frequent labels following the strategy in [Lai et al., 2015; Xia et al., 2014]. We use all the rest of images as training set for the unsupervised methods. To reduce the time complexity, we only use 5000 landmarks sampled randomly to build a bipartite graph for BGDH. For supervised methods, we randomly select 10500 images from dataset. The MAP values are calculated within the top 5000 returned neighbors.

We present the MAP results in Table 2 where BGDH, SSDH, DSH, DHN, DPSH are deep methods. Except for SSDH which uses triplet labels, all deep methods are trained with pairwise labels. To be fair, parameters of these methods are set according to the suggestions of authors. The results show that in most cases our proposed BGDH method substantially outperforms other baselines including unsupervised methods, supervised methods and semi-supervised methods.

In real applications, the number of labeled instances is usually far less than that of unlabeled instances. To further verify the effectiveness of leveraging both pairwise labels and unlabeled data, we reduce the number of labeled instances while keeping the number of unlabeled instances remain the same. For supervised learning, in CIFAR-10, we randomly select 2500 (250 images per class) images and in NUS-WIDE, we randomly select 5000 images. The MAP results with this experiment setting are listed in Table 3. Note that the results of unlabeled unsupervised methods are the same as those in Table 2. We observe that our BGDH

\(^1\)https://www.cs.toronto.edu/~kriz/cifar.html

\(^2\)http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm
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 $\lambda$ which controls the tradeoff between supervised loss and unsupervised loss. Figure 2 displays the impacts of $\lambda$ 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 $\lambda$ that BGDH performs well. Thus, to a large extent, BGDH is insensitive to $\lambda$ and the parameter selection is not a crucial problem in our algorithm. Additionally, by comparing the MAP of $\lambda = 0$ and $\lambda = 1$, we verify the importance of graph embedding.

### 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


[Chatfield et al., 2014] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the de-


