This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Accelerate Learning of Deep Hashing With Gradient Attention

Long-Kai Huang Jianda Chen Sinno Jialin Pan
Nanyang Technological University, Singapore
lhuang018@e.ntu.edu.sg jianda001@e.ntu.edu.sg sinnopan@ntu.edu.sg

Abstract

Recent years have witnessed the success of learning to hash in fast large-scale image retrieval. As deep learning has shown its superior performance on many computer vision applications, recent designs of learning-based hashing models have been moving from shallow ones to deep architectures. However, based on our analysis, we find that gradient descent based algorithms used in deep hashing models would potentially cause hash codes of a pair of training instances to be updated towards the directions of each other simultaneously during optimization. In the worst case, the paired hash codes switch their directions after update, and consequently, their corresponding distance in the Hamming space remain unchanged. This makes the overall learning process highly inefficient. To address this issue, we propose a new deep hashing model integrated with a novel gradient attention mechanism. Extensive experimental results on three benchmark datasets show that our proposed algorithm is able to accelerate the learning process and obtain competitive retrieval performance compared with state-of-the-art deep hashing models.

1. Introduction

With the explosive growth of visual content on the Internet in recent years, learning to hash techniques have attracted more and more attention in large-scale image retrieval. By mapping the raw data to binary codes while preserving data similarity information in the Hamming space, hashing techniques facilitate the storage of large-scale images data and enable efficient approximate nearest neighbors search. Among various hashing techniques [29, 28], supervised hashing has proven to be effective to generate domain relevant and compact codes and achieve satisfactory performance. With the development of deep learning, most recently developed supervised hashing models are build on top of deep neural networks to provide an end-to-end solution for image retrieval [14, 35, 3, 5, 2, 24]. In this paper, we focus on deep learning based supervised hashing models.

The overall objective of hashing is to learn similarity-preserving binary codes to represent images. In most supervised hashing methods, supervision is provided in the form of pairwise similarities, i.e. each training "example" is in the form of a pair of instances with a label to denote whether the paired instances are similar or dissimilar. Note that though in some hashing methods [14, 34], supervision is provided in the form of triplet ranking or list ranking, the ranking information still depends on pairwise similarities. Therefore, in most deep hashing models, the loss function or the objective function is designed based on the predictions on pairwise similarities.

In deep hashing models, similar to other deep learning models, a commonly used optimization algorithm is gradient descent. Basically, given a labeled pair of instances, based on the prediction error on their similarity, the hash codes of pairs of instances are first updated based on gradient descent, and then the decomposed error would be back-propagated to update other model parameters via gradient descent. However, based on our analysis in Section 3, we find that optimizing a deep hashing model by gradient descent possibly leads to a "dilemma" in optimization, where the hash codes of the pairs of instances may be always updated towards the directions of each other simultaneously during optimization. In the worst case, the paired hash codes may switch their directions after update, and their inner product or Hamming distance remains the same. When the update of some training pairs falls into this dilemma, their corresponding loss will not be decreased, which slows down the learning process.

As the aforementioned dilemma is caused by backward passing the gradients of both hash codes of a pair of training instances, a solution to overcome this dilemma is to selectively ignore or downgrade the gradient of one hash code of the pair during back-propagation. In this way, the optimization focuses on optimizing the desired code of one instance of each pair only. As a result the hash codes of the paired instances do not move towards the directions of each other simultaneously. Now, the remaining issue is which one of a paired hash codes should be chosen for update. This issue becomes much more complicated when taking all training pairs in a mini-batch into consideration.

Inspired by the idea of learning to learn [1, 9], in this paper, we cast the design of the hash code update selection criteria as a learning problem. To be specific, we propose a gradient attention mechanism to generate attentions for the gradients of hash codes of each pair of training instances by a neural network. With the gradient attention mechanism, the aforementioned dilemma can be reduced and thus the learning process can be accelerated.

In summary, our contributions are threefold:

- Firstly, we present the failure of gradient descent based algorithms in learning a deep hashing model: the learning process may get stuck as some pairs of hash codes tend to switch their directions of each other during optimization.
- Secondly, we propose a gradient attention mechanism which is integrated in a deep hashing architecture to address the aforementioned learning issue, and thus accelerate the learning process.
- Thirdly, we apply our proposed method on different loss functions and verify its performance with extensive experiments on three large-scale image datasets.

2. Preliminaries: Deep Hashing Models

To train a deep hashing model for similarity-based image retrieval, we are given N instances $\{\mathbf{x}_i\}_{i=1}^N$ as a training set, together with side information on pairwise similarities $\{s_{ij}\}$'s, where $s_{ij}=1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar and otherwise $s_{ij}=0$. The goal of a deep hashing model $M(\theta;\mathbf{x})$ is to map input instances $\{\mathbf{x}_i\}$'s into K-dimensional feature vectors $\{\mathbf{z}_i\}$'s, and then binarize them to obtain corresponding binary codes $\{\mathbf{b}_i\in\{\pm1\}^K\}$'s, which preserve pairwise similarities between instances. Here θ denotes the parameters of the model. In general, the binarization is done by using a sign function $\mathbf{b}=sgn(\mathbf{z})$. However, due to the ill-posed gradient of the sign function, each binary code is often relaxed to a continuous code $\mathbf{h}\in[-1,1]^K$ during training by replacing the sign function with the hyperbolic tangent function $tanh(\cdot)$. Besides, we denote by h^k and b^k the k-th entry of $\mathbf{h}$ and the k-th bit of $\mathbf{b}$, respectively.

The objective for hashing is generally designed based on the predicted pairwise similarities in the form of

$$\min \sum_{(i,j)\in\mathcal{P}} \ell(\hat{s}_{ij}), \tag{1}$$

where $\ell(\cdot)$ is the loss function, $\mathcal{P}=\{(i,j)\}$ is the set indexes of training pairs, and $\hat{s}_{ij}$ is the predicted similarity of a training instance pair (i,j). The predicted similarity is calculated either by inner product between binary codes [2, 3, 35], i.e. $\hat{s}_{ij}=\langle\mathbf{b}_i,\mathbf{b}_j\rangle$ or by the relaxed Hamming distance between binary codes [5, 14], i.e. $\hat{s}_{ij}=\|\mathbf{b}_i-\mathbf{b}_j\|^2$. Since $\|\mathbf{b}_i-\mathbf{b}_j\|^2=\frac{1}{2}(K-\langle\mathbf{b}_i,\mathbf{b}_j\rangle)$, the Hamming distance and inner product are interchangeable for binary codes. In this paper, given two binary codes, we adopt their inner product $\hat{s}_{ij}=\langle\mathbf{b}_i,\mathbf{b}_j\rangle$ and the inner product between their continuous relaxations $\hat{s}_{ij}=\langle\mathbf{h}_i,\mathbf{h}_j\rangle$ to measure their similarities.

Similar to other deep learning models, a deep hashing model is generally optimized by mini-batch stochastic gradient descent (SGD) as $\theta\leftarrow\theta-\alpha\frac{\partial\ell}{\partial\theta}$. The gradient of the model parameter θ at different layers is computed by the backward propagation. In deep hashing, the partial derivative $\partial\ell/\partial\hat{s}_{ij}$ is first computed. Then the partial derivative of the k-th bit of $\mathbf{h}_i$, denoted by h_i^k, is computed as

$$\frac{\partial\ell}{\partial h_i^k}=\sum_j\frac{\partial\ell}{\partial\hat{s}_{ij}}\frac{\partial\hat{s}_{ij}}{\partial h_i^k}. \tag{2}$$

Finally, the gradient $\partial\ell/\partial\theta$ is computed as

$$\frac{\partial\ell}{\partial\theta}=\sum_i\frac{\partial\ell}{\partial\mathbf{h}_i}\frac{\partial\mathbf{h}_i}{\partial\theta}, \tag{3}$$

where $\partial\ell/\partial\mathbf{h}_i=[\partial\ell/\partial h_i^1,\partial\ell/\partial h_i^2,...,\partial\ell/\partial h_i^K]^\top$.

The goal of optimization is to update the model parameters θ. However, it is not clear how the intermediate variables, such as $\mathbf{h}$, change after update. To analyze the update behaviors of $\mathbf{h}$, we present the following lemma.

Lemma 1. *Given a composite function $g(y)$ where $y=f(\mathbf{x})$ is a scaler and $\mathbf{x}$ is a vector. If $\mathbf{x}$ is updated by gradient descent, i.e. $\mathbf{x}^+=\mathbf{x}-\alpha\frac{\partial g}{\partial\mathbf{x}}$ where α is a step size, then y approximately changes along the negative of its gradient, i.e. $y^+=f(\mathbf{x}^+)\approx y-\hat{\alpha}\frac{\partial g}{\partial y}$, where $\hat{\alpha}$ is some positive scaler.*

Proof. By applying first-order Taylor expansion of $f(x^+)$ on x, we have

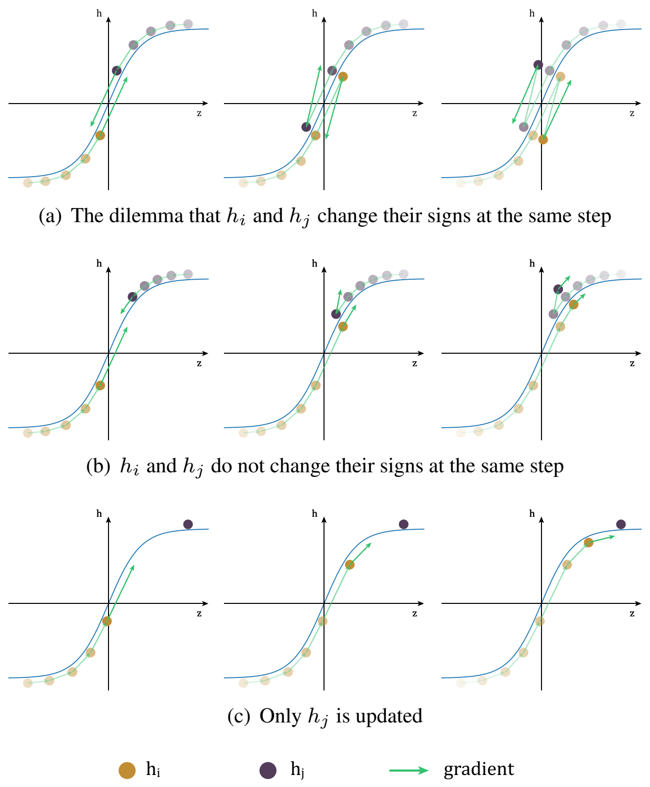
$$\begin{aligned} y^+-y=f(x^+)-f(x)&\approx\left\langle\frac{\partial y}{\partial x},x^+-x\right\rangle\\ &=-\alpha\left\langle\frac{\partial y}{\partial x},\frac{\partial g}{\partial x}\right\rangle=-\alpha\left\langle\frac{\partial y}{\partial x},\frac{\partial g}{\partial y}\frac{\partial y}{\partial x}\right\rangle\\ &=-\alpha\left\|\frac{\partial y}{\partial x}\right\|^2\frac{\partial g}{\partial y}=-\hat{\alpha}\frac{\partial g}{\partial y}, \end{aligned}$$

where $\hat{\alpha}\doteq\alpha\left\|\frac{\partial y}{\partial x}\right\|^2$. This completes the proof. □

By substituting θ, any bit of $\mathbf{h}$ and $tanh(M(\theta))$ into the variables $\mathbf{x}$, y and $f(\mathbf{x})$ in Lemma 1, respectively, we find that after θ is updated by gradient descent, the continuous code $\mathbf{h}$ change along its negative gradient. This allows us to study the change of $\mathbf{h}$ by its gradient instead of drilling down to the gradient of model parameters.

3. The Dilemma of Gradient Descent

To study the limitations of gradient descent based algorithms for deep hashing, we first focus on a simple case



(a) The dilemma that h_i and h_j change their signs at the same step

(b) h_i and h_j do not change their signs at the same step

(c) Only h_j is updated

Figure 1. Possible changes of h_i and h_j during update. A horizontal axis is added to better visualize the location of h_i and h_j

where there is only one pair of training instances (i, j), whose similarity ground truth is "+1", i.e., they are similar to each other. Suppose their current one-bit continuous codes are $h_i = -1$ and $h_j = 1$, respectively. The loss is simply defined as $\ell = 1 - \hat{s}_{ij} = 1 - h_i h_j$. Obviously, the loss is positive and the current codes are not optimal, which need to be updated. The gradients of h_i and h_j are

$$\frac{\partial \ell}{\partial h_i} = \frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_i} = -h_j \text{ and } \frac{\partial \ell}{\partial h_j} = \frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_j} = -h_i,$$

respectively. Based on Lemma 1, the values of h_i and h_j after update, denoted by h_i^+ and h_j^+, respectively, can be expressed as follows:

$$h_i^+ = h_i - \alpha_{ij} \frac{\partial \ell}{\partial h_i} = h_i + \alpha_{ij} h_j,$$
$$h_j^+ = h_j - \alpha_{ji} \frac{\partial \ell}{\partial h_j} = h_j + \alpha_{ji} h_i,$$

where α_{ij} and α_{ji} are some positive scalers. The update rules show that h_i moves towards the direction of h_j, tending to be of the same value as h_j. And same for h_j. Note that when h_i and h_j become the same, the hash codes are optimal as the predicted similarity is correct. However, in the worst case, by assuming that the values of α_{ij} and α_{ji} are large, e.g., $\alpha_{ij} = \alpha_{ji} = 2$, we have $h_i^+ = h_j = 1$ and $h_j^+ = h_i = -1$. It turns out that their inner product $h_i^+ h_j^+ = -1$ remains unchanged after update, and thus the loss after update remains the same. In the next round of update, both h_i and h_j will switch their signs while their corresponding loss does not decrease, and this happens over and over again. We call this phenomenon the "dilemma" of updating deep hashing models by gradient descent. Note that the dilemma also happens on dissimilar pairs when their codes are updated to move away from each other.

In practice, α_{ij} and α_{ji} may be small and the aforementioned case may not happen at one step. However, when $|h_i| < |\alpha_{ij} h_j|$ and $|h_j| < |\alpha_{ji} h_i|$, both codes will change their signs simultaneously as illustrated in Figure 1(a), which still results in an incorrect similarity prediction.

To overcome the aforementioned dilemma and generate optimal codes such that the loss can be decreased fast, a possible solution is to reduce the probability of changing the signs of the paired codes simultaneously. This can be done by reducing the update scale, i.e. $\alpha_{ij} \frac{\partial \ell}{\partial h_i}$ and $\alpha_{ji} \frac{\partial \ell}{\partial h_j}$. In this way, the probability that both codes changes their signs becomes small. In an ideal case, if only one code changes its sign, then the update could achieve the optimal values as shown in Figure 1(b). To reduce the update scale, one can set a small learning rate, which, however, may lead to a slow learning process. Another way is to add a quantization loss, with which an opposite gradient is added to $\frac{\partial \ell}{\partial h_i}$ to reduce its scale. However, since a quantization loss naturally hinders the sign change of codes, its coefficient needs to be carefully controlled in the objective function. Otherwise, the learning process may be sluggish.

Note that in Figure 1(b), h_j first moves from $+1$ towards h_j, and moves backwards to $+1$ after h_i becomes postive. The process that h_j moves towards h_i is unnecessary. A better solution is to selectively update one of the paired codes while keeping the other unchanged when the prediction on their similarity is incorrect. This code update selection strategy could avoid the signs switch problem effectively as shown in Figure 1(c). However, when the predicted similarity is correct, it is better to update both h_i and h_j towards their sign value. Therefore, to update h_i and h_j, the weights of their gradients should be carefully chosen such that only one code is updated when the predicted similarity is not correct, while both codes are updated when the predicted similarity is correct. To implement this idea on top of a deep hashing model, we propose a novel gradient attention mechanism to generate weights on the gradients of hash code pairs.

4. Gradient Attention Network

Our proposed gradient attention mechanism is applicable to any deep hashing model with pairwise loss function. As demonstrated in Section 3, for a specific pair (i, j), if the predicted similarity is not correct and the partial derivatives of h_i^k and h_j^k are both back-propagated, h_i^k and h_j^k may change sign at the same step, and the loss may not decrease after update. To generate appropriate weights of derivatives of h_i^k and h_j^k in back-propagation, we propose to train a gradient attention network to generate attentions on the derivatives of h_i^k and h_j^k. Before the derivatives of h_i^k and h_j^k are passed backward to θ, the attention weights generated by the gradient attention network are applied on them. Then the gradient of θ is computed based on the weighted derivatives of h_i^k and h_j^k, instead of the original ones. Considering all training pairs in a batch, the weighted derivatives of h_i^k is computed as

$$\begin{aligned} \frac{\partial \ell}{\partial h_i^k} &= \sum_j \beta_{ij,i}^k \frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_i^k}, \\ \frac{\partial \ell}{\partial h_j^k} &= \sum_i \beta_{ij,j}^k \frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_j^k}, \end{aligned} \tag{4}$$

where $\beta_{ij,i}^k$ and $\beta_{ij,j}^k$ are a pair of attention weights generated for derivatives of the pair of binary bits h_i^k and h_j^k, respectively. By substituting the weighted derivatives in (4) into (3), we obtain a new gradient, denoted by $g(\varphi)$, where φ is the parameters of gradient attention network, for optimizing the deep hashing model.

4.1. Architecture of Gradient Attention Network

For the gradient attention network, we feed the factors that may affect the change of h_i^k in gradient descent as input. Considering a single pair (i, j), the change of h_i^k is affected by its original value and derivative regarding to the loss of this pair only, i.e. $\frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_i^k}$. Considering all training pairs, the change of h_i^k is affected by its original value and its derivative regarding to loss of all training pairs, i.e. $\frac{\partial \ell}{\partial h_i^k}$ in (2). Therefore, the input is h_i^k, $\frac{\partial \ell}{\partial \hat{s}_{ij}} \frac{\partial \hat{s}_{ij}}{\partial h_i^k}$ and $\frac{\partial \ell}{\partial h_i^k}$.

To obtain the attention weights for the k-th bit of a pair of codes, the attention network first generates a feature value $y_{ij,i}^k$ for each side of the pair, and then normalizes it via a softmax function as

$$\beta_{ij,i}^k = \frac{\exp(y_{ij,i}^k)}{\exp(y_{ij,i}^k) + \exp(y_{ij,j}^k)} \tag{5}$$

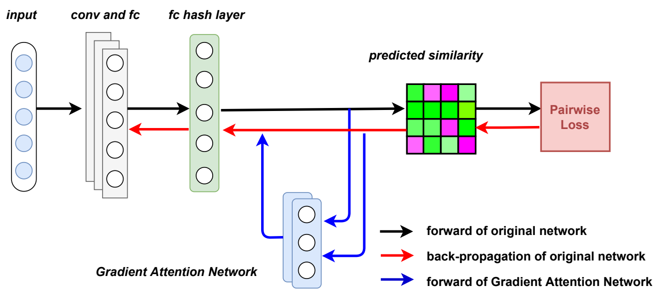
and

$$\beta_{ij,j}^k = \frac{\exp(y_{ij,j}^k)}{\exp(y_{ij,i}^k) + \exp(y_{ij,j}^k)} . \tag{6}$$

For the architecture of gradient attention network, it contains two fully connection layers with 100 hidden units. The overall architecture of the deep hashing model together with the gradient attention network is shown in Figure. 2.

4.2. Loss Function of Gradient Attention Network

In this section, we introduce the loss function for training the gradient attention network. Note that the gradient attention mechanism is proposed to reduce the probability


input
conv and fc
fc hash layer
predicted similarity
Pairwise Loss
Gradient Attention Network
forward of original network
back-propagation of original network
forward of Gradient Attention Network
Figure 2. The network structure of the proposed deep hash algorithms.

that a code pair change their bits at the same step and thus accelerate the learning process. Therefore, the quality of the attention weights can be measured by the decrease of loss in an update. With the gradient attention network, the parameters θ of hashing model is updated by

$$\theta^+ = \theta - \alpha g(\varphi), \tag{7}$$

where $g(\varphi)$ is the new gradient generated by the gradient attention network and φ is the parameter of the gradient attention network. The loss after update is $\ell(\theta^+) = \ell(\theta - \alpha g(\varphi))$ and the decrease of loss is $\ell(\theta) - \ell(\theta^+)$. To obtain a better gradient attention network, we should maximize the decrease of loss. By excluding the constant term $\ell(\theta)$ of the gradient attention network φ, we obtain the primary objective for optimizing the gradient attention network as

$$\min_{\varphi} \ell(\theta - \alpha g(\varphi)).$$

Note that the update of θ in (7) is by the simplest form of SGD. In practice, θ can be updated by any first-order optimizer as long as θ^+ is a function of $g(\varphi)$. And φ can be updated by any optimizer. The algorithm of our proposed algorithm, Gradient Attention deep Hashing (GAH), is summarized in Algorithm 1.

5. Related Works

Learning based hashing methods have been studied for years. Based on the information used in learning, they are categorized into unsupervised-based [30, 21, 10, 19], supervised-based [20, 22, 17, 23], and semi-supervised-based [26, 27] hashing models. Recently deep learning based hashing approaches demonstrate the superiority over the shallow approaches taking handcrafted features as input. Deep hashing was first proposed in [32] known as CNNH, which learns hash codes and a neural network based hashing model in two separated stage. In order to learn feature representation and hashing codes simultaneously, DHHN [14] was proposed. Later, [16, 35, 3, 2, 5, 33] were proposed with various loss functions to preserve similarity information and different quantization techniques to tackle the issue of continuous relaxation of binary codes.

]

Algorithm 1 Deep Hashing with Gradient Attention (GAH)

```
1: repeat
2:     Sample a mini-batch of pairs from training set
3:     Evaluate loss $\ell(\theta)$ and backward propagate to obtain
       the derivative $\partial\ell/\partial\hat{s}_{ij}$
4:     Take $h_i^k$, $\frac{\partial\ell}{\partial\hat{s}_{ij}}\frac{\partial\hat{s}_{ij}}{\partial h_i^k}$ and $\frac{\partial\ell}{h_i^k}$ as input to gradient atten-
       tion network to obtain attention weights $y_{ij,i}^k$
5:     Compute $\beta_{ij,i}^k$ and $\beta_{ij,j}^k$ by (5) and (6)
6:     Compute the new derivative $\frac{\partial\ell}{h_i^k} = \sum_j \beta_{ij,i}^k \frac{\partial\ell}{\partial\hat{s}_{ij}}\frac{\partial\hat{s}_{ij}}{\partial h_i^k}$,
       and back-propagate it to obtain gradient of $\theta$ as $g(\varphi)$.

7:     Update $\theta$ by $\theta^+ = \theta - \alpha g(\varphi)$
8:     Evaluate the loss after update as $\ell(\theta - \alpha g(\varphi))$
9:     Update $\varphi$ using back-propagation
10:    $\theta \leftarrow \theta^+$
11: until a fixed number of iterations
```

In our work, we train an extra model to change the gradient in back-propagation, which is related to learning to optimize. Learning to optimize algorithms [1, 15, 31] are proposed fairly recently to learn to generate gradient descent for general objective function optimization. In [1], the gradient generator is RNN, which is updated by gradient descent. In [15], learning to optimize is done based on reinforcement learning. In [31], the authors proposed a hierarchical RNN architecture which is scalable for large-scale problems. Unlike existing learning to optimize algorithms, our proposed gradient attention network is specially designed for deep hashing model. Besides, our gradient attention network only generates the gradient of an intermediate variable and the gradient of original network parameters are obtained by the back-propagation algorithm, while the learning to optimize algorithms generate gradients for each parameter and does not depend on back-propagation.

6. Experiments

6.1. Datasets and Settings

To evaluate the performance of the proposed algorithm GAH against state-of-the-art hashing methods, we conduct experiments on three benchmark datasets: CIFAR-10 [25], NUS-WIDE [6], ImageNet [7].

CIFAR-10 consists of 10 classes, with each class containing $6,000$ 32×32 color images. We follow [2, 35, 14, 32] to construct the training set by randomly sampling 500 images per class and the query set by randomly sampling 100 images per class. The remaining images are used as database for retrieval.

NUS-WIDE contains nearly $270,000$ images collected from Flickr. Most of images are associated with one or multiple labels from a given 81 concepts. After removing images that are associated no labels and that are not available, there are about $180,000$ images. We randomly select $10,000$ images as training set and $5,000$ images as query set, and use the remaining as database.

ImageNet is a large scale image benchmark for visual recognition challenge and is widely used to evaluate performance of deep learning model. In the dataset, there are more than 1.2 million images, each with one class label from $1,000$ categories. We follow [3, 5] to randomly select 100 categories and use images in training set to form the database. From the database, we randomly sample 130 images per category to train the hashing model. The image in validation set are used as queries.

As a common protocol in existing hashing methods, the ground truth similarity information are defined according to the class labels. If two instances share at least one label, they are considered as similar and assigned similarity as $s_{ij} = 1$; otherwise, their similarity $s_{ij} = 0$. Note that the data imbalance is observed in the dataset after constructing the similarity label in this way. For CIFAR-10 and ImageNet, the ratio between dissimilar and similar pairs roughly equals to the number of categories. For NUS-WIDE, the ratio is around 5.

The performance of hashing model is evaluated based on three metrics: mean average precision (MAP), Precision-Recall curves, and Hamming lookup Precision curves within Hamming radius 2. Specially, we follow [3, 35] to evaluate MAP on top $5,000$ returned samples on CIFAR-10 and NUS-WIDE datasets and on top $1,000$ returned samples on ImageNet.

We compare the performance of the proposed GAH with nine classical or state-of-the-art hashing methods: two are unsupervised methods LSH [4] and ITQ [10]; two are supervised shallow methods ITQ-CCA [10] and KSH [20]; five are deep learning based methods CNNH [32], DNNH [14], DPSH [16], HashNet [3] and GH [24].

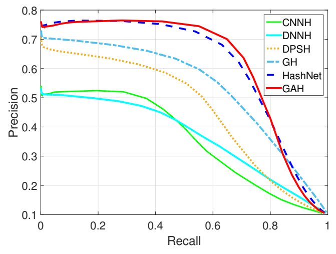
6.2. Implementation details

We implement our model with PyTorch. The deep hashing model part in the proposed GAH utilizes same structure as AlexNet [13], except that the last fully connected layer *fc8* is replaced by a fully connected hash layer with output K-dimension features, followed by an activation function $tanh(\cdot)$. The optimizer for hashing model of GAH is SGD with 0.0005 weight decay, 0.9 momentum and step decaying learning rate. As for the gradient attention network of GAH, it consists of two fully connected layers with 100 hidden units and it is optimized by Adam [12]. The mini-batch size of images is fixed to 128. For shallow hashing models, we feed them DeCAF$_7$ features [8], i.e. the $fc7$ output of pretrained Alexnet, as input.

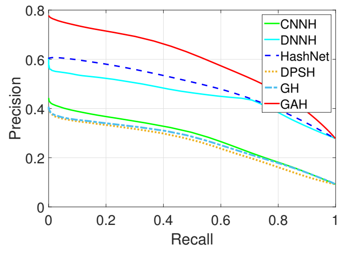
For the loss function to train the hashing model, we choose the Weighted Maximum Likelihood (WML) estima-

Table 1. Mean Average Precision (MAP) over three datasets with different bit lengths.

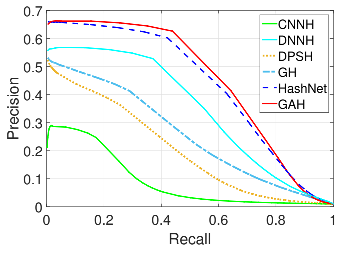
Method	CIFAR-10				NUS-WIDE				ImageNet			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
GAH	**0.6985**	**0.7531**	**0.7605**	**0.7607**	**0.7235**	**0.7576**	**0.7662**	**0.7733**	**0.5034**	**0.6231**	**0.6563**	**0.6821**
HashNet [3]	0.6896	0.7446	0.7537	0.7566	0.7218	0.7561	0.7585	0.7592	0.4633	0.5991	0.6417	0.6563
GH [24]	0.5927	0.6640	0.6518	0.6763	0.6840	0.7425	0.7430	0.7459	0.3383	0.4878	0.4833	0.4893
DPSH [16]	0.4355	0.5780	0.6246	0.6331	0.6822	0.7152	0.7263	0.7356	0.2022	0.3584	0.4214	0.4492
DNNH [14]	0.4876	0.5182	0.5025	0.4982	0.5532	0.5827	0.5976	0.6081	0.4373	0.5565	0.5560	0.5813
CNNH [32]	0.4886	0.5023	0.5251	0.5216	0.5228	0.5436	0.5418	0.5530	0.2204	0.2720	0.3010	0.2941
KSH [20]	0.4138	0.4998	0.5519	0.5668	0.6135	0.6731	0.6960	0.7229	0.4112	0.5211	0.5731	0.5984
ITQ-CCA [10]	0.2040	0.1582	0.1359	0.1288	0.5675	0.5324	0.4905	0.4665	0.2481	0.3954	0.4923	0.5640
ITQ [10]	0.2559	0.2672	0.2786	0.3017	0.6200	0.6475	0.6591	0.6657	0.2267	0.3254	0.3827	0.4179
LSH [4]	0.1716	0.1858	0.2094	0.2350	0.3632	0.4192	0.4663	0.4980	0.0671	0.1303	0.1901	0.2406



(a) CIFAR-10

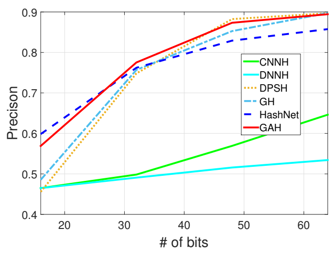


(b) NUS-WIDE

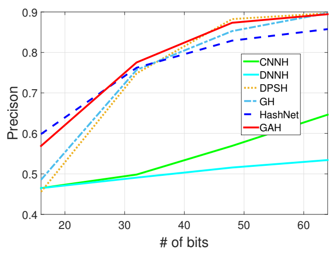


(c) ImageNet

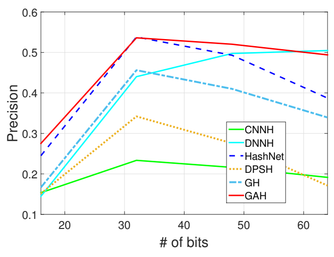
Figure 3. Precision-recall curve @ 64 bits over three datasets.



(a) CIFAR-10



(b) NUS-WIDE



(c) ImageNet

Figure 4. Precision within Hamming radius 2 over three datasets.

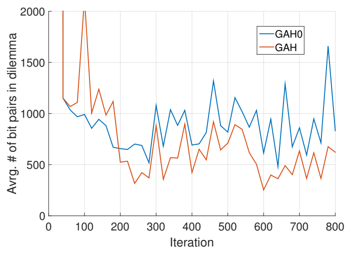
tion of the hash code which is defined as

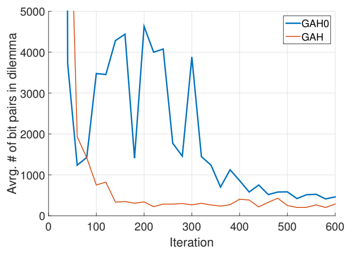
$$\min_{\theta} \ell(\theta) \\ = \sum_{(i,j)\in\mathcal{P}} w_{ij}(\log(1+\exp(\gamma\langle \mathbf{h}_i,\mathbf{h}_j\rangle))-\gamma s_{ij}\langle \mathbf{h}_i,\mathbf{h}_j\rangle), \quad (8)$$

where $\mathcal{P} = \{(i,j)\}$ is the set indexes of training pairs; γ is the parameter in the adaptive sigmoid function to formulate a conditional probability of the predicted similarity given two hash codes; and w_{ij} is the weights to address the imbalance issue between similar and dissimilar training pairs, which is defined as

$$w_{ij} = \begin{cases} |\mathcal{S}_0|/|\mathcal{S}_1|, & \text{if } s_{ij}=1, \\ 1, & \text{if } s_{ij}=0, \end{cases}$$

where $\mathcal{S}_0 = \{s_{ij} \in \mathcal{S} : s_{ij}=0\}$ and $\mathcal{S}_1 = \{s_{ij} \in \mathcal{S} : s_{ij}=1\}$. The above loss function has been widely used in deep hashing methods [16, 35, 3, 2], together with different types of quantization losses. In our model, the quantization loss is not necessary, because our model does not require a quantization loss to reduce the probability of codes signs switch during training. Our loss function naturally pushes h_i^k and h_j^k to its binarized value $\mathbf{b}_i$ and $\mathbf{b}_j$ when $h_i^k h_j^k$ has the same sign as $(s_{ij}-0.5)$. Note that the original GH [24] is proposed with a classifer on hash code and is trained by minimizing the Cross Entropy loss. For fair comparison, we assume only similarity label is available during training and thus its loss function is changed to the one in (8) with a quantization loss defined in [24].


(a) CIFAR-10


(b) NUS-WIDE

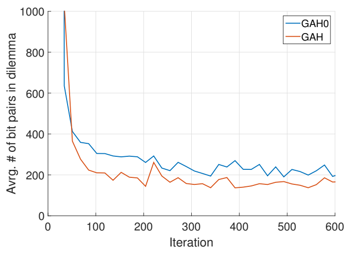
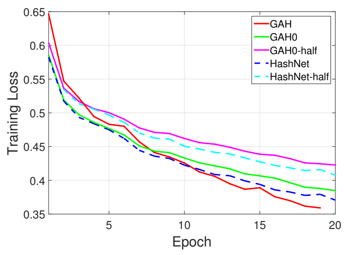
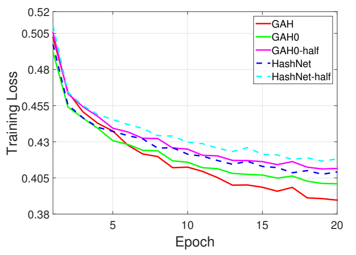

(c) ImageNet

Figure 5. Comparison between methods with and without gradient attention mechanism: number of bit pairs fallen into the dilemma


(a) CIFAR-10


(b) NUS-WIDE

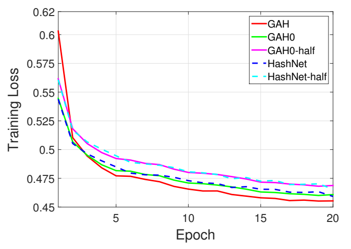

(c) ImageNet

Figure 6. Comparison between methods with and without gradient attention mechanism: training loss through training process

6.3. Results and Analysis

The mean average precision of different hashing methods is shown in Table 1. From Table 1, it is observed that our proposed GAH substantially outperforms all comparison methods. Compared to deep learning based hashing methods, GAH outperforms the best competitor, HashNet, by roughly 0.1% to 3.9%. Besides, GAH outperforms GH by abouth 1.5% to 19.3%. Note that the loss function and settings of HashNet, GH and GAH are nearly the same, except that HashNet is integrated with learning by continuation technique to learn exactly binary code, GH is integrated with discrete optimization technique and an additional quantization loss term, while GAH is integrated with a gradient attention network. Therefore, the superiority of GAH in MAP to HashNet and GH verifies the contribution of gradient attention mechanism to generate binary code with high quality. When compared to shallow hashing methods with deep feature as input, GAH achieves boost by a large margin. It outperform the best competitor KSH, by at least 5.1% over all datasets. This verifies the advance of end-to-end deep learning based hashing methods, which simultaneously learn feature representation and binarization.

The retrieval performance in terms of precision-recall curve is shown in Figure 3. GAH outperforms comparison methods. In particular, GAH achieves much higher precision at lower recall levels. This is desirable for precision-first retrieval, which is widely implemented in practical systems. To evaluate the performance of hashing methods in efficient binary code retrieval using Hamming lookup, which costs $\mathcal{O}(1)$ time for a query, we test the precision within Hamming radius 2 on all methods. The results are shown in Figure 4. The proposed GAH algorithm still outperforms other methods when code length is small, indicating its effectiveness in large scale retrieval with compact code. However, when the code length is large, its performance may be weaker than DNNH, DPSH or GH.

6.4. The study of Gradient Attention Mechanism

In this section, we first evaluate the positive effect of gradient attention mechanism on reducing the number of bit pairs in "dilemma" that a bit pair switch their signs simultaneously when their predicted similarity is not correct. As a baseline, we train the hashing model in GAH without integrating gradient attention network, and denote it by GAH0. The result of average number of bit pairs fallen in the dilemma every 20 training iterations is shown in Figure 5. Note that all dropout layers are disabled in this experiment. The result demonstrates that after a certain number of iterations, the gradient attention mechanism contributes to reducing the number of bit pairs that change their signs in the same step. Compared to GAH0, GAH reduces the number of bit pairs in the dilemma by about $13.1\%$to 61.9% on CIFAR-10 after 200 iterations. On NUS-WIDE and ImageNet, after 100 iterations, the reduction percentage is at least 26.1% and 10.8%, respectively.

Besides, we test the the decrease of training loss of the hashing model, i.e. $\ell(\theta)$, to evaluate the gradient atten-

Table 2. Comparison of different gradient weights assignment strategies on ImageNet.

Baselines	GAH	B1	B2	B3	B4
MAP	**0.6821**	0.6483	0.6536	0.5670	0.6577

tion mechanism in accelerating the learning process. HashNet [3] and GAH0 are used as baselines. Besides, if gradient attention mechanism does not work, it will output same weights for a pair and this is similar to reduce the gradient to its half. To avoid attributing the learning acceleration to smaller learning rate, GAH0 and HashNet with half learning rate are used as baselines as well, and they are denoted by GAH0-half and HashNet-half. From the result shown in Figure 6, we observe that the training loss of the proposed method is large at the beginning when the gradient attention network is not well-trained. However, its loss decreases faster than comparison methods, finally reaches the lowest loss among all the methods. These results verifies that the gradient attention mechanism contributes to accelerating the learning process.

To furture verify the effcet of gradient attention meachnism, we compare it with four different baselines to generate weights on gradients of hash code pairs: B1) assigning weight 1 randomly to one gradient; B2) assigning random weights to two gradients; B3) assigning weight 1 to larger gradient; B4) assigning weight 0 to smaller gradient. The experiments are run on ImageNet Dataset with 64-bit codes. The result is shown in Table 2. The performance of B1 and B2 is close to the model updated by original gradient, because the expected gradient direction of B1 and B2 is same as the original gradient. Due to larger variance of weighted gradient, the performance of B1 is slightly worse than B2. For B4, its performance is better than B1, B2 and slightly better than HashNet, which shows that the hash code quality can be improved if we correctly choose one of the paired codes and update it while keeping the other unchanged. But it is not the best strategy to always choose the code with larger gradient to update. The gradient attention mechanism is able to generate better weights than B4 and thus its performance is the best. This verifies the necessary of gradient attention mechanism to improve the learning of hashing model and generate high-quality binary codes.

6.5. GAH with unsupservised loss function

To verify the performance of GAH on a different pairwise objective function, we perform experiment on CIFAR-10 dataset in unsupervised setting. We follow [24, 11] to minimize the cosine similarity difference between the features encoded in Euclidean space and the hash code in Hamming space. Denote the feature in Eucldiean space for the

Table 3. MAP @5000 on unsupservised CIFAR-10.

Method	16 bits	32 bits	48 bits	64 bits
GAH	**0.4796**	**0.4986**	**0.5057**	**0.5165**
GH [24]	0.4114	0.4590	0.4942	0.5034
DeepBit [18]	0.1811	0.2171	0.2391	0.2468
ITQ [10]	0.3461	0.3845	0.4080	0.4364
LSH [4]	0.2022	0.2216	0.2331	0.2992

i-th image by $\mathbf{z}_i$, the loss function is defined as

$$\ell(\theta) = \frac{1}{N^2} \sum_{(i,j)\in\mathcal{P}} \left\| \frac{1}{K}\langle \mathbf{h}_i, \mathbf{h}_j \rangle - cos(\mathbf{z}_i, \mathbf{z}_j) \right\|^2, \qquad (9)$$

where $cos(\cdot,\cdot)$ means the cosine distance. To compare the performance, we apply the same loss function (9) to GH [24] with a quantization loss term [24] and run another three unsupervised hash methods, including DeepBit [18], ITQ [10] and LSH [4] on CIFAR-10 dataset. Note that in this experiment, we sample $1,000$ images per class as query set and the remaining $50,000$ images are used as training set and database. The network structure of deep hashing models follows the setting in [24, 18], which is based on VGG16. As for shallow learning based hashing methods, we feed them the 4096-d feature extracted by VGG16.

The results are displayed in Table 3. With a different objective function, the proposed method GAH is still able to outperform GH and other competitors. The boost of GAH over the best competitor GH is more than 1.1% when the code length is 48 or 64. For small code length (i.e. 16 and 32 bits), GAH outperforms GH by a large margin as about 4%. These results verify that the gradient attention mechanism is applicable to different pairwise loss functions.

7. Conclusion

In this paper, we present the dilemma that pairs of binary codes may switch their signs or directions frequently, but their pairwise similarities in Hamming space remain unchanged. This leads to an inefficient learning process. To address this issue, we propose to integrate the hashing model with a novel gradient attention mechanism to generate appropriate weights to gradients of hash code pairs. Empirical studies on three benchmark datasets with both supervised objective and unsupervised objective verifies the effectiveness of the gradient attention mechanism on accelerating learning for deep hashing.

Acknowledgement

This work is funded by NTU Nanyang Assistant Professorship (NAP) grant M4081532.020, MOE AcRF Tier1 grant 2016-T1-001-159, and Microsoft Research Asia.

References

[1] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[2] Yue Cao, Mingsheng Long, Bin Liu, Jianmin Wang, and MOE KLiss. Deep cauchy hashing for hamming space retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1229–1237, 2018.

[3] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and S Yu Philip. Hashnet: Deep learning to hash by continuation. In *ICCV*, pages 5609–5618, 2017.

[4] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *ACM Symposium on Theory of Computing*, pages 380–388, 2002.

[5] Zhixiang Chena, Xin Yuana, Jiwen Lua, Qi Tiand, and Jie Zhoua. Deep hashing via discrepancy minimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6838–6847, 2018.

[6] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, page 48. ACM, 2009.

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

[8] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of International Conference on Machine Learning*, pages 1126–1135, 2017.

[10] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 817–824, 2011.

[11] Mengqiu Hu, Yang Yang, Fumin Shen, Ning Xie, and Heng Tao Shen. Hashing with angular reconstructive embeddings. *IEEE Transactions on Image Processing*, 27(2):545–555, 2018.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, volume 5, 2015.

[13] 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.

[14] 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.

[15] Ke Li and Jitendra Malik. Learning to optimize. *International Conference on Learning Representations*, 2017.

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

[17] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1971–1978, 2014.

[18] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1183–1192, 2016.

[19] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. Discrete graph hashing. In *Neural Information Processing Systems*, pages 3419–3427, 2014.

[20] W. Liu, J. Wang, R. Ji, Y. Jiang, and S-F Chang. Supervised hashing with kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081, 2012.

[21] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *International Conference on Machine Learning*, pages 1–8, 2011.

[22] M. Norouzi and D. Fleet. Minimal loss hashing for compact binary codes. In *International Conference on Machine Learning*, pages 353–360, 2011.

[23] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 37–45, 2015.

[24] Shupeng Su, Chao Zhang, Kai Han, and Yonghong Tian. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *Advances in Neural Information Processing Systems*, pages 806–815, 2018.

[25] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[26] Jun Wang, Ondrej Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3424 –3431, 2010.

[27] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In *International Conference on Machine Learning*, pages 1127–1134, 2010.

[28] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - A survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

[29] Jingdong Wang, Ting Zhang, Nicu Sebe, and Heng Tao Shen. A survey on learning to hash. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):769–790, 2018.

[30] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *Neural Information Processing Systems*, pages 1753 –1760, 2008.

[31] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of International Conference on Machine Learning*, pages 3751–3760, 2017.

[32] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, volume 1, page 2, 2014.

[33] Xin Yuan, Liangliang Ren, Jiwen Lu, and Jie Zhou. Relaxation-free deep hashing via policy gradient. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 134–150, 2018.

[34] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1556–1564, 2015.

[35] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *AAAI*, pages 2415–2421, 2016.