



Relaxation-Free Deep Hashing via Policy Gradient

Xin Yuan, Liangliang Ren, Jiwen Lu^(✉), and Jie Zhou

Department of Automation, Tsinghua University, Beijing, China
{yuanx16, renll16}@mails.tsinghua.edu.cn,
{lujiwen, jzhou}@tsinghua.edu.cn

Abstract. In this paper, we propose a simple yet effective relaxation-free method to learn more effective binary codes via policy gradient for scalable image search. While a variety of deep hashing methods have been proposed in recent years, most of them are confronted by the dilemma to obtain optimal binary codes in a truly end-to-end manner with non-smooth sign activations. Unlike existing methods which usually employ a general relaxation framework to adapt to the gradient-based algorithms, our approach formulates the non-smooth part of the hashing network as sampling with a stochastic policy, so that the retrieval performance degradation caused by the relaxation can be avoided. Specifically, our method directly generates the binary codes and maximizes the expectation of rewards for similarity preservation, where the network can be trained directly via policy gradient. Hence, the differentiation challenge for discrete optimization can be naturally addressed, which leads to effective gradients and binary codes. Extensive experimental results on three benchmark datasets validate the effectiveness of the proposed method.

Keywords: Deep hashing · Relaxation-free · Policy gradient

1 Introduction

With the rapid development of information technology, large-scale and high-dimensional image data have been widespread on the Internet. A variety of efforts have been made to deal with the large scale similarity search, which is shown to be useful for many practical applications (*e.g.* computer vision [3, 25, 37], machine learning [9, 27, 39], and data mining [44]). The hashing technique [1, 5, 6, 16, 32, 34, 35, 38] is a popular approach of encoding high-dimensional data as low-dimensional binary codes, which benefits from its computation and storage efficiencies. Learning based hashing [10, 11, 20, 23, 29, 45] which mines the data properties and the semantic affinities shows better performance than data-independent hashing methods [8].

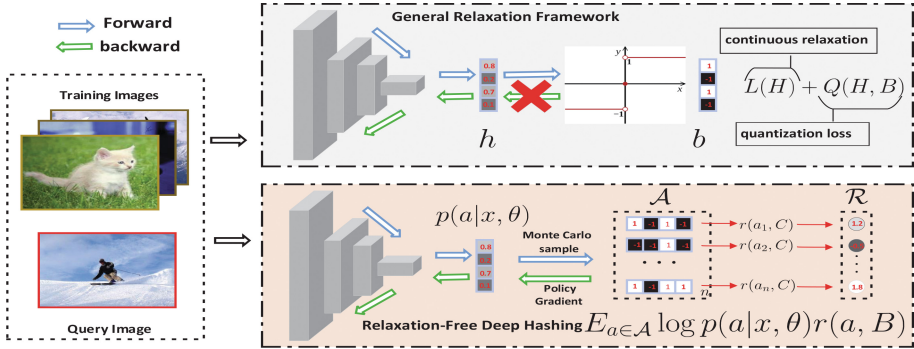


Fig. 1. Illustration of our approach. Unlike most existing learning-based hashing methods (on the top) which solve the differential difficulty by continuous relaxations, our method (on the bottom) modifies the non-smooth part as a stochastic policy, where samples for binary codes are encouraged to earn maximum rewards for similarity preservation. The network is trained via policy gradient directly

Most previous learning-based hashing methods encode data samples with shallow architectures [11, 20, 29], which map similar samples to close in the learned hamming space by learning a single projection matrix. While encouraging performance can be obtained, most of them suffer from the non-linear feature representation, scalability and non-linearity issues. Recently, deep learning based hashing methods [17, 42] have been proposed to learn discriminative feature representations and nonlinear hash mappings, which have shown state-of-the-art performance on various scalable image retrieval datasets. However, the binary constraint of the non-smooth discrete optimization is a challenging problem in these methods, which prevents deep hashing to be learned in a truly end-to-end manner. By continuous relaxation, the non-smooth optimization can be transformed to a continuous one which can be solved by standard gradient methods, leading to the deviation from the optimal binary codes. While many methods have been proposed to control the quantization errors, they still cannot learn exactly binary hash codes in an optimization procedure. Hence this may lead to substantial performance loss due to the sub-optimal of the learned binary codes.

In this paper, we present a relaxation-free deep hashing method via policy gradient (PGDH) for scalable image search. Figure 1 shows the key idea of our proposed method. Specifically, we formulate the non-smooth part of the hashing network as sampling with a stochastic policy, so that the relaxation procedure used in most previous hashing methods can be removed. We directly generate binary codes and maximize the expectation of rewards for similarity preservation, which leads to more effective gradient and binary hash codes and the differentiation issue for discrete optimization can be naturally addressed. Extensive evaluations on three benchmark datasets show that our method significantly improves the state-of-the-arts.

2 Related Work

A variety of learning-based hashing methods have been proposed in recent years, which can be mainly classified into unsupervised hashing and supervised hashing.

Unsupervised hashing methods learn binary codes by exploiting data properties such as distributions and manifold structures. For example, spectral hashing (SH) [40] formulated hashing learning as a graph partitioning problem and approximately solved the problem with the assumption of the uniform data distribution. Anchor graph hashing (AGH) [26] approximated neighborhoods by using a tractable graph based method. Deep hashing (DH) [21] employed a multi-layer neural network to learn hash functions to preserve the nonlinear relationship of samples. Iterative quantization (ITQ) [9] minimized quantization loss by seeking a rotation matrix in an iterative manner. Manifold hashing (MH) [31] learned binary embeddings from cluster centers and mapped data into a low-dimension manifold. Discrete graph hashing (DGH) [24] presented a tractable alternating optimization method for similarity preservation in the discrete code space.

Supervised hashing methods learn binary codes by exploiting the label information of samples, which have shown superior performance than unsupervised approaches. For example, kernelized supervised hashing (KSH) [25] utilized the equivalence between code inner products and Hamming distances, which aims to keep the inner product of hash codes consistent with the pairwise supervision. Fast supervised hashing [19] employed boosted decision trees to iteratively perform alternative optimization on a subset of binary codes. Supervised discrete hashing (SDH) [30] formulated the discrete optimization objective by introducing an auxiliary variable and used a kernel based hashing function to learn binary codes. The supervised extension of deep hashing [21] learned multi-layer functions by considering the label information of samples. Recent advances in deep learning [12, 15, 33] show that deep convolutional networks learn robust and powerful feature representations for complex data, which has gained great successes in many computer vision applications. Hence, it is natural to leverage deep learning to obtain compact binary codes. For example, CNNH [42] adopted a two-stage strategy in which the first stage learned hash codes and the second stage learned a deep network based hash function to obtain the codes. DNNH [17] improved the two-stage CNNH with a simultaneous feature learning and hash coding pipeline so that representations and hash codes can be optimized in a joint learning procedure. DSH [22] improved DNNH by adding a max-margin loss and a quantization loss which jointly preserved pairwise similarity and controlled the quantization error. HashNet [2] gradually approximated the non-smooth sign activation with a smoothed activation by a continuation method.

3 Approach

3.1 Overview of General Relaxation Framework

Given a training set of N points (images) $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$, each sample is represented by either a D -dimensional feature vector or raw pixels. A set of pairwise labels $\mathbf{S} = \{s_{ij}\}$ is provided, where $s_{ij} = 1$ if \mathbf{x}_i and \mathbf{x}_j are similar while $s_{ij} = -1$

if \mathbf{x}_i and \mathbf{x}_j are dissimilar. For supervised hashing, \mathcal{S} can be constructed from semantic labels of data points or the relevance feedback from click-through data. We aim to learn a mapping function $f: \mathbf{x} \mapsto \mathbf{b} \in \{-1, 1\}^K$ from the input space to the Hamming space $\{-1, 1\}^K$, where each data point \mathbf{x} is encoded as a compact K -bit binary hash code. The binary codes $\mathbf{B} = \{\mathbf{b}_i\}_{i=1}^N$ should preserve some notion of similarity in \mathcal{S} . Hence, the hashing learning problem can be generally formulated as follows:

$$\min_{f, \mathbf{B}} \mathcal{L}(\mathbf{B}), \quad s.t. \quad \mathbf{B} \in \{-1, 1\}^{n \times K} \quad (1)$$

where \mathcal{L} is the predefined loss function with similarity preservation.

To directly optimize the problem in Eq. (1) with the discrete constrain on \mathbf{B} , we need to adopt the sign function $\mathbf{b} = \text{sgn}(\mathbf{h})$ as the activation function to convert the continuous representation \mathbf{h} to the binary hash code \mathbf{b} . However, the sign function is non-differentiable at zero and with zero gradient for all nonzero inputs, which makes standard back-propagation infeasible. As a result, it is inappropriate to directly solve the discrete optimization problem by standard gradient-based methods. Most existing hashing methods relax the intractable optimization problem mainly in two ways: (1) continuous relaxation by introducing a quantization function, and (2) approximating the sign function with sigmoid or tanh relaxation [2, 17]. For the first strategy, these methods derive an optimization problem $\mathcal{L}(\mathbf{H})$ from the hashing objective $\mathcal{L}(\mathbf{B})$ by continuous relaxation and control the quantization loss between \mathbf{B} and \mathbf{H} , which is denoted as $Q(\mathbf{B}, \mathbf{H})$. The objective of these methods can be usually reformulated as:

$$\min_{f, \mathbf{H}, \mathbf{B}} \mathcal{L}(\cdot) + Q(\mathbf{B}, \mathbf{H}), \quad s.t. \quad \mathbf{B} \in \{-1, 1\}^{n \times K} \quad (2)$$

where $\mathcal{L}(\cdot)$ indicates $\mathcal{L}(\mathbf{H})$ for continuous optimization [18] or $\mathcal{L}(\mathbf{B})$ for discrete optimization [22]. However, since $Q(\mathbf{B}, \mathbf{H})$ is NP-complete and cannot be minimized to zero, there still exists a gap between \mathbf{B} and \mathbf{H} . Thus a local minimum is usually obtained by such relaxation optimization problems.

For the second strategy, the non-smooth sign function is approximated by continuation method, which leads to a convergence to the original hash learning objective. However, to obtain feasible gradients, such relaxation inevitably becomes more non-smooth and slows down or suppresses the convergence, which makes it difficult to optimize the learning model.

3.2 Relaxation-Free Deep Hashing via Policy Gradient

In this section, we propose a new architecture for deep learning to hash with policy gradient inspired by the REINFORCE algorithm [41]. The architecture of our proposed framework contains: (1) a convolutional network (CNN) for learning deep representations of images, and (2) a fully-connected policy layer with a sigmoid activation function for transforming each feature representation into a K -dimensional vector, where each dimension represents the probability of taking the binary action. The proposed end-to-end learning framework can be

viewed as an *agent* that interacts with an external *environment* (images in our case). The aim of the agent is to get maximum possible similarity preservation with difference minimization, which can be considered as the reward to the agent.

We define a policy as $\pi(\mathbf{x}_i, \theta) = \{\pi_{\mathbf{x}_i, \theta}^{(k)}\}_{k=1:K}$, which is parametrized by network parameter θ with i -th input \mathbf{x}_i . The policy generates a sequence of actions $\mathbf{a}_i = \{a_{i,k}\}_{k=1:K} \sim P_\theta(\mathbf{x}_i)$, where $a_{i,k} = \{0, 1\}$ represents a binary action value. $\pi_{\mathbf{x}_i, \theta}^{(k)}$ only outputs the probability of the hash code +1, which is different from most existing reinforcement learning methods which predict the probability distribution for each possible action (e.g. softmax probability). Hence, the probability distribution in our method can be formulated as follows:

$$p(a_{i,k}) = \begin{cases} \pi_{\mathbf{x}_i, \theta}^{(k)}, & \text{if } a_{i,k} = 1 \\ 1 - \pi_{\mathbf{x}_i, \theta}^{(k)}, & \text{if } a_{i,k} = 0 \end{cases} \quad (3)$$

Having generated *action* \mathbf{a}_i , the agent observes a *reward* $r(\mathbf{a}_i)$ that is related to the similarity preservation. The reward is computed by an evaluation metric by comparing the similarity relationship in the Hamming space with ground-truth similarity function \mathcal{S} .

We adopt a minibatch-based strategy for learning and sample a minibatch of points from the whole training set in each iteration. For each mini-batch with m training samples, we aim to utilize the global information by maximizing the preserved information between each binary code $\mathbf{b}_i = 2 * (\mathbf{a}_i - 0.5)$ and the codebook $\mathbf{C} = \{\hat{\mathbf{b}}_j\}_{j=1}^n$ of all the training points in the Hamming space. For a pair of binary codes \mathbf{b}_i and $\hat{\mathbf{b}}_j$, we represent the Hamming distance $dist_H(\cdot, \cdot)$ by inner product $\langle \cdot, \cdot \rangle$ as: $dist_H(\mathbf{b}_i, \hat{\mathbf{b}}_j) = \frac{1}{2}(K - \langle \mathbf{b}_i, \hat{\mathbf{b}}_j \rangle)$. The weighted reward of learning to effective hash codes can be written as follows:

$$r(\mathbf{a}_i) = -\frac{1}{2} \sum_{j=1}^n \hat{s}_{ij} (K - \mathbf{b}_i^T \hat{\mathbf{b}}_j) \\ \text{s.t. } \mathbf{b}_i, \hat{\mathbf{b}}_j \in \{-1, +1\}^K \quad (4)$$

where

$$\hat{s}_{ij} = \begin{cases} \beta, & \text{if } s_{ij} = 1 \\ \beta - 1, & \text{otherwise} \end{cases} \quad (5)$$

is the weighted similarity measurement to compensate the imbalance of positive and negative pairs. The parameter β allows different weights on the positive and negative pairs. Note that the codebook \mathbf{C} is updated slower than the learning model θ during the training process, which will be discussed later.

The goal of training is to minimize the negative expected reward of the minibatch:

$$\mathcal{L}(\theta) = - \sum_i \mathbb{E}_{\mathbf{a}_i \sim P_\theta(\mathbf{x}_i)} [r(\mathbf{a}_i)] \quad (6)$$

Note that in our framework the description of the environment consists of images, which is not determined by the previous states or actions. Strictly speaking, this formulation is not a full reinforcement learning framework where a state transition is clearly defined. Here we only focus on the optimization under the guidance of the rewards related to similarity preservation and improving performance of hash learning.

Policy Gradient with REINFORCE: In our proposed hash learning method, the expected reward r is non-differentiable. In order to compute $\nabla L(\theta)$ directly, we use the REINFORCE algorithm, which computes the expected gradient of the non-differentiable reward function as follows:

$$\nabla_{\theta} \mathcal{L}(\theta) = - \sum_i \mathbb{E}_{\mathbf{a}_i \in \mathcal{A}_i} [r(\mathbf{a}_i) \nabla_{\theta} \log(P_{\theta}(\mathbf{a}_i | \mathbf{x}_i))] \quad (7)$$

where \mathcal{A}_i is the set of all possible actions for i -th input data in the minibatch. The expected gradient can be approximated using Monte Carlo sample. We represent a T -samples Monte Carlo on \mathbf{a}_i as:

$$\mathcal{A}_i = \{\mathbf{a}_i^1, \mathbf{a}_i^2, \dots, \mathbf{a}_i^T\} = MC^{P_{\theta}(\mathbf{a}_i | \mathbf{x}_i)}(T) \quad (8)$$

For training examples in a minibatch, the expected policy gradient can be computed as:

$$\nabla_{\theta} \mathcal{L}(\theta) \approx - \frac{1}{T} \sum_i \sum_t [r(\mathbf{a}_i^t) \nabla_{\theta} \log(P_{\theta}(\mathbf{a}_i^t | \mathbf{x}_i))] \quad (9)$$

where the log probability in Eq. (9) can be calculated by the binary cross entropy over the Bernoulli distribution in Eq. (3).

REINFORCE with a Baseline: The above gradient estimator is simple but suffers from high variance because of the difficulty of credit assignment. To reduce the variance of the gradient estimation, we again approximate the expected gradient with widely used Baseline method in policy gradient. For each training minibatch:

$$\nabla_{\theta} \mathcal{L}(\theta) \approx - \frac{1}{T} \sum_i \sum_t [(r(\mathbf{a}_i^t) - r') \nabla_{\theta} \log(P_{\theta}(\mathbf{a}_i^t | \mathbf{x}_i))] \quad (10)$$

where the baseline r' should be the value which is independent on the action. Adding such a baseline term will not change the expectation of the gradient¹ but can reduce the variance of the gradient estimation. Here we choose average of all

¹ $\sum_i \mathbb{E}_{\mathbf{a}_i \in \mathcal{A}_i} [r' \nabla_{\theta} \log(P_{\theta}(\mathbf{a}_i^t | \mathbf{x}_i))] = \sum_i r' \nabla_{\theta} \sum_{\mathbf{a}_i} P_{\theta}(\mathbf{a}_i^t | \mathbf{x}_i) = \sum_i r' \nabla_{\theta} 1 = 0.$

Algorithm 1. PGDH

Input: Training set: $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$, pairwise labels: $\mathcal{S} = \{s_{ij}\}$ and codebook update interval $R > 1$.

Output: Learning model θ and codebook \mathbf{C}

```

1: Initialize  $p_\theta$  and  $\mathbf{C}$ ;
2: for  $iter = 1, 2, \dots, M$  do
3:   Sample random minibatch from  $\mathbf{X}$ ;
4:   Compute the action probability by feeding minibatch to the model;
5:   Compute the rewards for MC samples of the minibatch according to Eq. (4)
6:   Compute policy gradient according to Eq. (10);
7:   Update the model  $\theta$  according to Eq. (11);
8:   if  $iter \% R = 0$  then
9:     Update codebook  $\mathbf{C}$ ;
10:  end if
11: end for
12: return model  $\theta$  and codebook  $\mathbf{C}$ ;

```

rewards in each mini-batch as the baseline. The binary codes that preserve more similarity information with the codebook \mathbf{C} than the baseline will get positive rewards, while those that with less similarity information will be penalized by negative rewards. We then update the network’s parameters as:

$$\theta \leftarrow \theta - \lambda \nabla_{\theta} \mathcal{L}(\theta) \quad (11)$$

where λ denotes the learning rate.

During the learning process, the codebook \mathbf{C} is updated slower than the model for the training stability and performance improvement. We can formulate the codebook update as:

$$\hat{\mathbf{b}}_j = 2 * (\hat{\mathbf{a}}_j - 0.5), \hat{\mathbf{a}}_i \sim P(\mathbf{x}_j | \theta^-) \quad (12)$$

This strategy is motivated by [28] which introduces a target network θ^- with slower updating rate than the online network θ to gain more stable performance.

In summary, **Algorithm 1** shows full details of the proposed method.

3.3 Out-of-Sample Extensions

Having completed the learning procedure, we only generate the optimized hash codes for the training points by maximizing the expectation of rewards. How to perform out-of-sample extensions to generate hash codes for the points which are not in the training dataset remains unclear. To address this, we perform the out-of-sample extensions in two ways: Deterministic and Stochastic.

Deterministic Generation: Denote a data point which is not in the training dataset as \mathbf{x}_q , we feed it to our proposed architecture and get a vector with K values $\pi_{\mathbf{x}_q, \theta}$, each represents the probability of the binary action 1 (sigmoid activation ranges from 0 to 1). We can directly obtain the binary codes in the deterministic way:

$$b_q^k = \begin{cases} +1, & \text{if } \pi_{\mathbf{x}_q, \theta}^{(k)} > 0.5 \\ -1, & \text{otherwise} \end{cases} \quad (13)$$

Stochastic Generation: Having obtained the probability vector, we can write the stochastic code generation function as:

$$b_q^k = \begin{cases} +1, & \text{with probability } \pi_{\mathbf{x}_q, \theta}^{(k)} \\ -1, & \text{with probability } 1 - \pi_{\mathbf{x}_q, \theta}^{(k)} \end{cases} \quad (14)$$

The stochastic way seems more appealing than the deterministic one but in practice the performance differs slightly after the learning model converges. In our experiments, we report the performance directly using deterministic generation and we also conduct investigation on the two ways to generate hash codes.

4 Experiments

4.1 Datasets and Experimental Settings

We conduct extensive empirical evaluation on three public widely used benchmark datasets: CIFAR-10 [14], NUS-WIDE [43] and ImageNet [4]. **CIFAR10** contains 60,000 manually single-labeled color images belonging to 10 classes (6000 images per class). Following the same setting in [36], we construct the query set by randomly sampling 1,000 images with 100 images per category and use the remaining 59,000 images to form the database. Then we uniformly select 500 images per class to form the training set from the database. **NUS-WIDE**² is a public Web image dataset of 269,648 images collected from Flickr. This is a multi-label dataset, namely, each image is associated with one or multiple labels from a given 81 concepts. We follow the settings in [42, 46] and use the subset of 195,834 images that are associated with the 21 most frequent concepts, where each concept consists of at least 5,000 images. We randomly sample 2,100 images with 100 images per category to form the test set and use the remaining images as the database. We uniformly sample 500 images per category out of the database to form a training set. **ImageNet** is a large dataset for visual recognition which contains over 1.2M images in the training set and 50K images in the validation

² <http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm>.

set covering 1,000 categories. Following the same setting in [2], we randomly select 100 categories, use all the images of these categories in the training set as the database and all the images in the validation set as the queries. To train hashing methods, we randomly select 100 images per category from the database as the training points.

Following the same evaluation protocol as previous work [22], the similarity information, which is constructed from image labels, is used for ground truth evaluation and constructing the pairwise similarity matrix for training. For both single and multiple labeled dataset, we define the ground truth semantic neighbors as images sharing at least one label. Note that by constructing the training data in this way, all three datasets exhibit the data imbalance problem because of the imbalance of positive and negative pairs, which can be used to evaluate the effects of our weighted rewards controlled by β .

We evaluate the retrieval performance of generated binary codes with the following metrics: mean average precision (MAP), precision-recall (P-R) curve, precision at top retrieved samples (P@N), and Hamming lookup precision within a Hamming radius $r = 2$ (HLP@2). We choose to evaluate the performance over binary codes with lengths of 16, 32, 48, and 64 bits. Note that for the ImageNet dataset we calculate the MAP@1000 as each category has only 1,300 images, and for NUS-WIDE we adopt MAP@5000.

In our implementation of PGDH, we utilize the AlexNet network structure and implement it in the Pytorch framework. We initialize first seven layers of PGDH by copying the parameters of convolutional layers *conv1* – *conv5* and fully-connected layers *fc6* – *fc7* in the pre-trained model on ImageNet and fine-tuned these layers. We also initialize the final policy layer with the Gaussian distribution and train this layer from scratch. In the training phase, we use Adam [13] with the initial learning rate as 0.005 and set the batch size as 128. For parameter tuning, we evenly split the training set into ten parts to cross validate the parameters. We fix the Monte Carlo samples T as 10 in each iteration and codebook update interval R as 5.

4.2 Results and Analysis

Comparison with the State-of-the-Arts: We compare the proposed PGDH with twelve state-of-the-art hashing methods, including unsupervised methods: LSH [8], SH [40], ITQ [9], supervised methods: KSH [25], CCA-ITQ [9], FastH [19], SDH [30], and supervised deep methods: CNNH [42], DNNH [17], DPSH [18], DSH [22], HashNet [2]. We report their results by running the source codes provided by their respective authors to train the models by ourselves, except for DNNH due to the inaccessibility of the source code. For conventional hashing methods, we use *DeCAF7* [7] features as input. For deep hashing methods, we directly use raw images as input and resize images to fit the adopted network. Note that we adopt the AlexNet architecture for all deep hashing for fair comparison.

Table 1. The comparison of the retrieval performance among all compared hashing methods in terms of mean average precision (MAP %) on the three image datasets for different number of bits of 16, 32, 48, and 64

| Methods | CIFAR-10 (%) | | | | NUS-WIDE (%) | | | | ImageNet (%) | | | |
|-------------|--------------|-------------|-------------|-------------|--------------|-------------|-------------|-------------|--------------|-------------|-------------|-------------|
| | 16 | 32 | 48 | 64 | 16 | 32 | 48 | 64 | 16 | 32 | 48 | 64 |
| LSH [8] | 12.9 | 15.2 | 16.9 | 17.8 | 40.3 | 49.2 | 49.3 | 55.1 | 10.1 | 23.5 | 30.1 | 34.9 |
| SH [40] | 12.2 | 13.5 | 12.1 | 12.6 | 47.9 | 49.1 | 49.8 | 51.5 | 20.8 | 32.7 | 39.5 | 42.0 |
| ITQ [9] | 21.3 | 23.4 | 23.8 | 25.3 | 56.7 | 60.3 | 62.2 | 62.6 | 32.5 | 46.2 | 51.3 | 55.6 |
| CCA-ITQ [9] | 31.4 | 36.1 | 36.6 | 37.9 | 50.9 | 54.4 | 56.8 | 67.6 | 26.6 | 43.6 | 54.8 | 58.0 |
| KSH [25] | 35.6 | 40.8 | 53.1 | 44.1 | 40.6 | 40.8 | 38.7 | 39.8 | 16.0 | 28.8 | 34.2 | 39.4 |
| FastH [19] | 45.3 | 46.1 | 48.7 | 50.3 | 51.9 | 61.0 | 64.7 | 65.2 | 22.8 | 44.7 | 51.7 | 55.6 |
| SDH [30] | 40.2 | 42.0 | 44.9 | 45.6 | 53.4 | 61.8 | 63.1 | 64.5 | 29.9 | 45.1 | 54.9 | 59.3 |
| CNNH [42] | 48.8 | 51.2 | 53.4 | 53.6 | 61.2 | 62.3 | 62.1 | 63.7 | 28.8 | 44.7 | 52.8 | 55.6 |
| DNNH [17] | 55.5 | 55.8 | 58.1 | 62.3 | 68.1 | 71.3 | 71.8 | 72.0 | 29.7 | 46.3 | 54.0 | 56.6 |
| DPSH [18] | 64.6 | 66.1 | 67.7 | 68.6 | 71.5 | 72.6 | 73.8 | 75.3 | 32.6 | 54.6 | 61.7 | 65.4 |
| DSH [22] | 68.9 | 69.1 | 70.3 | 71.6 | 71.8 | 72.3 | 74.2 | 75.6 | 34.8 | 55.0 | 62.9 | 66.5 |
| HashNet [2] | 70.3 | 71.1 | 71.6 | 73.9 | 73.3 | 75.2 | 76.2 | 77.6 | 50.6 | 62.9 | 66.3 | 68.4 |
| PGDH | 73.6 | 74.1 | 74.7 | 76.2 | 76.1 | 78.0 | 78.6 | 79.2 | 51.8 | 65.3 | 70.7 | 71.6 |

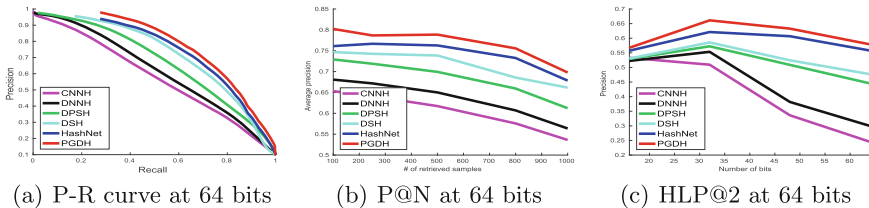


Fig. 2. The experimental results of PGDH and comparison methods on the CIFAR-10 dataset under three evaluation metrics

Table 1 shows the overall retrieval performance of different hashing methods in terms of MAP at different code lengths. We can observe that our proposed PGDH outperforms all compared methods. Compared with the best competitor in deep learning based hashing methods, PGDH consistently outperforms by around 3%. The significant performance improvement attributes to the effective binary codes obtained via policy gradient instead of the general relaxation framework. Note that our PGDH also utilizes the weighted rewards function to attack the data imbalance problem which is ignored by many existing methods. Also, we see that the recently proposed HashNet boosts the performance of other deep learning methods (e.g. DSH and DPSH) because HashNet tackles the optimization difficulty by continuation method and the data imbalance problem

by weighted maximum likelihood. Compared with the best conventional hashing methods, PGDH also boosts the performance by a large improvement. Note that the deep hashing methods sustainably outperform the conventional hash learning methods on both datasets by a large margin even though the conventional ones utilize the CNN features, which suggests the end-to-end learning scheme is advantageous.

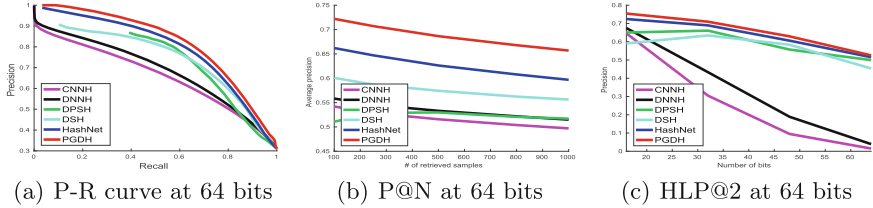


Fig. 3. The experimental results of PGDH and comparison methods on the NUSWIDE dataset under three evaluation metrics

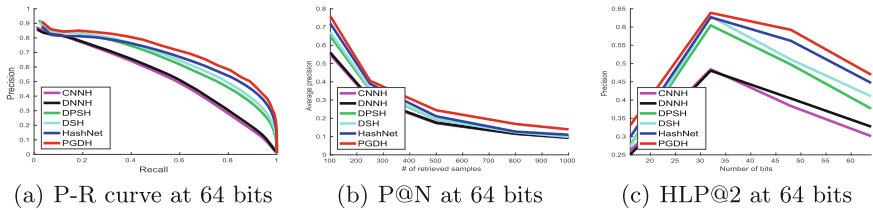


Fig. 4. The experimental results of PGDH and comparison methods on the ImageNet dataset under three evaluation metrics

The performance on CIFAR-10, NUS-WIDE and ImageNet datasets in terms of Precision-Recall (PR) curves for 64-bit binary codes are shown in Figs. 2(a), 3(a) and 4(a). Here we only show the results in terms of PR curves on the deep learning based hashing methods to evaluate the effectiveness of the hashing learning. The results show that PGDH outperforms all the compared methods by large margins. PGDH achieves much higher precision at the same recall level than compared methods which suggests that effective hash codes are learnt via policy gradient. This attribute is appreciated in practical precision-first image retrieval system where high probability of finding true neighbors is more important.

The performance on the three datasets in terms of the average precision with respect to different numbers of top retrieved results(P@N) of deep learning methods for 64-bit binary codes are shown in Figs. 2(b), 3(b) and 4(b). Note that the maximum of N is set to 1,000 here for the consistency on all the three datasets. From the result figures, we can see that PGDH consistently provides

superior precision than the compared hashing methods for the same amount of retrieved samples. This stands for that more semantic neighbors are retrieved, which is desirable in practical use.

The performance in terms of Hamming lookup precision within Hamming radius 2 (HLP@2) for deep learning based hashing methods at different bit lengths on three datasets are shown in Figs.2(c), 3(c) and 4(c). This evaluation metric measures the precision of the retrieved results falling into the buckets within the Hamming radius 2. The results validate the compactness of the binary codes learnt by PGDH. We also observe that the best performance is achieved at a moderate length of binary codes. This is because that longer binary code makes the data distribution in Hamming space sparse and fewer samples fall within the set Hamming ball.

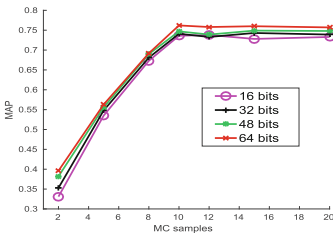


Fig. 5. Effects of the number of Monte Carlo samples in terms of MAP with 16, 32, 48 and 64-bit binary codes on the CIFAR-10 dataset

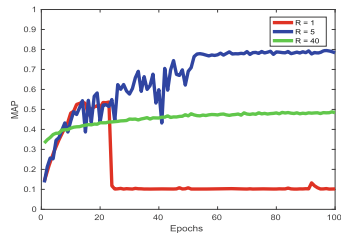


Fig. 6. Effects of the frequency of code-book update by setting R as 1, 5 and 40 in terms of MAP with 64-bit binary codes on CIFAR-10 (Color figure online)

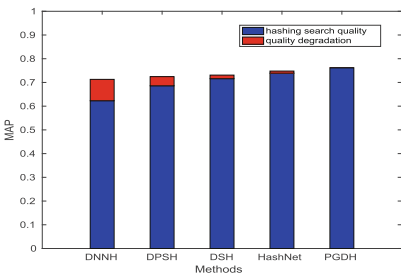


Fig. 7. Loss of search quality in MAP (by red bars) due to conversion from continuous features to 64-bit binary codes on the CIFAR-10 dataset (Color figure online)

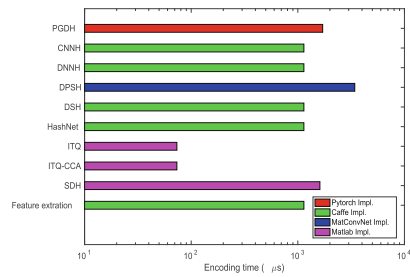


Fig. 8. Time cost to encode one new-coming image of different hashing methods on the CIFAR-10 dataset with 64-bit binary codes

Investigation on Samples: We study the effects of the number of Monte Carlo samples in the optimization procedure by changing the parameter T in PGDH. Note that it costs more time to train a minibatch of data as T increases. We report the performance results of different T values selected from $\{2, 5, 8, 10, 12, 15, 20\}$ in Fig. 5 in terms of MAP on the CIFAR-10 dataset. The results show that when T is small, the search quality degrades because efficient gradients cannot be obtained without enough MC samples. We also observe that the performance exhibits saturation when we keep enlarging T . For a tradeoff of the search quality and the training efficiency, we choose to fix T as 10 during training.

Investigation on Codebook Update: We study the effects of the frequency of codebook update during training by changing the interval parameter R in PGDH. Figure 6 shows MAP performance evolution of the first 60 epochs during training with respect to R on the CIFAR-10 dataset with length of binary codes set as 64 bits. The network is hard to optimize and MAP exhibits a very low value during training (red curve) when we update the codebook \mathcal{C} every iteration ($R = 1$). When we update the codebook \mathcal{C} once a epoch ($R = 40$), the network can be trained steadily but MAP raises up very slowly (green curve). We also observe that the best performance (blue curve) is achieved at a moderate value of $R = 5$.

Table 2. MAP (%) for different code generation schemes related to different training epochs on the CIFAR-10 dataset with 64-bit binary codes

| Training Epochs | 1 | 5 | 10 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
|-----------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Deterministic | 24.51 | 47.18 | 66.56 | 72.13 | 74.73 | 74.78 | 74.77 | 75.17 | 75.50 | 75.54 |
| Stochastic | 10.10 | 18.18 | 58.32 | 73.54 | 74.18 | 74.93 | 75.12 | 75.18 | 74.90 | 75.21 |

Table 3. MAP (%) for different β on the three datasets with 64-bits binary codes

| β | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 |
|----------|-------|-------|-------|-------|-------|-------|--------------|--------------|-------|
| CIFAR-10 | 10.12 | 18.38 | 20.08 | 49.43 | 73.65 | 70.32 | 75.23 | 75.12 | 34.12 |
| NUS-WIDE | 31.32 | 43.65 | 54.13 | 66.12 | 77.95 | 76.12 | 77.32 | 79.18 | 78.80 |
| ImageNet | 1.14 | 1.14 | 33.12 | 43.64 | 69.65 | 68.69 | 70.32 | 70.11 | 70.03 |

Deterministic vs. Stochastic: We investigate the deterministic and stochastic generation during the testing phase. Table 2 shows the MAP performance of the 64-bit codes generated by these two ways at different epochs on the CIFAR-10 dataset. We can observe that the performance differs a lot during the first decades of epochs. This is because that the stochastic way generates binary

codes by sampling in an uncertain manner, which will influence results if the model doesn't converge. We also observe that the MAP differs slightly when the learning model converges as the epochs increase. Although the stochastic way seems more appealing in PGDH, it will take more time for code generation during testing because of the sampling operation in practice.

Investigation on Weighted Rewards: We investigate the effect of weighted rewards on dealing with the imbalance problem. The weight is controlled by the parameter β in Eq. (5). The algorithm merely utilizes the positive pairs to learn hash codes when we set β to a large value. Setting β close to 0, the algorithm merely utilizes the negative pairs to learn hash codes. With the definition of semantic similarity and the datasets, the imbalance problem substantially deteriorates the performance of hashing methods. Table 3 shows the variation of performance in terms of MAP with respect to β on three datasets with the length of binary codes set as 64 bits. The retrieval performance ascends when setting $\beta > 0.5$, which shows the effect of introducing weighted rewards in our method.

Comparison of Search Quality Degradation: A crucial superiority of PGDH over the comparison methods lies in that PGDH directly learns effective compact binary codes via policy gradient, while comparison methods relax the discrete objective to adopt to the gradient-based algorithm. Intuitively, searching with binary codes using Hamming distance is evidently inferior to searching with continuous features using Euclidean distance, due to substantial information loss by relaxation. The search quality loss in terms of MAP due to binarization is shown in Fig. 7. Note that since PGDH directly outputs the binary codes, we only show the absolute MAP value for PGDH. From the result figure, we see that DNNH (9 % degradation), DPSH (3.85 % degradation), DSH (1.56 % degradation) and HashNet (0.9 % degradation) suffer from MAP loss while our PGDH can even break the bottleneck of the search quality with continuous features obtained by the compared methods. In other words, PGDH can learn more effective binary codes which are more accurate than all other methods.

Comparison of Encoding Time: The time to generate the binary code for a new-coming sample is an important factor to evaluate retrieval system in practical use. In this part, we compare the encoding time of our PGDH with (1) five deep learning based hashing methods, CNNH, DNNH, DPSH, DSH and HashNet, and (2) three conventional hashing methods, ITQ, ITQ-CCA, SDH, including the unsupervised and supervised hashing with linear and nonlinear hashing functions. For deep hashing methods, which directly take the raw images as input, we report the encoding time on GPUs. For conventional hashing methods, we take into consideration both the time cost for deep feature extraction on GPUs and the time cost for hashing encoding on CPUs. Figure 8 shows the comparison of the encoding time of involved hashing methods in logarithmic

scale on the CIFAR-10 dataset with 64-bit binary codes. Our computing platform is equipped with a 4.0 GHz Intel CPU, 32 GB RAM, and NVIDIA GTX 1080Ti. Although HashNet and DSH are faster than our PGDH because of the higher computational efficiency in Caffe implementation, we can easily convert the trained Pytorch model into a Caffe version during the test phase to realize the encoding acceleration while keeping the retrieval performance.

5 Conclusion

In this paper, we have proposed a new relaxation-free framework for deep hashing via policy gradient. We modified the non-smooth part of the hashing network for sampling as a stochastic policy to address the back-propagation difficulty. We directly generated binary codes through the network and maximized the expectation of the rewards related to the similarity preservation. We trained the proposed network via policy gradient, which naturally avoids the differentiation difficulty for discrete optimization, leading to more effective binary codes. We have conducted extensive experiments to validate the superiority of the proposed PGDH through comparison with the state-of-the-art hashing methods.

Acknowledgements. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFA0700802, in part by the National Natural Science Foundation of China under Grant 61672306, Grant U1713214, Grant 61572271, and in part by the Shenzhen Fundamental Research Fund (Subject Arrangement) under Grant JCYJ20170412170602564.

References

1. Çakir, F., He, K., Bargal, S.A., Sclaroff, S.: MIHash: online hashing with mutual information. In: ICCV, pp. 437–445 (2017). <https://doi.org/10.1109/ICCV.2017.55>
2. Cao, Z., Long, M., Wang, J., Yu, P.S.: HashNet: deep learning to hash by continuation. In: ICCV, pp. 5609–5618 (2017). <https://doi.org/10.1109/ICCV.2017.598>
3. Dean, T.L., Ruzon, M.A., Segal, M., Shlens, J., Vijayanarasimhan, S., Yagnik, J.: Fast, accurate detection of 100, 000 object classes on a single machine. In: CVPR, pp. 1814–1821 (2013). <https://doi.org/10.1109/CVPR.2013.237>
4. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Li, F.: ImageNet: a large-scale hierarchical image database. In: CVPR, pp. 248–255 (2009). <https://doi.org/10.1109/CVPRW.2009.5206848>
5. Do, T.-T., Doan, A.-D., Cheung, N.-M.: Learning to hash with binary deep neural network. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9909, pp. 219–234. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46454-1_14
6. Do, T., Tan, D.L., Pham, T.T., Cheung, N.: Simultaneous feature aggregating and hashing for large-scale image search. In: CVPR, pp. 4217–4226 (2017). <http://doi.ieeecomputersociety.org/10.1109/CVPR.2017.449>

7. Donahue, J., et al.: DeCAF: a deep convolutional activation feature for generic visual recognition. In: ICML, pp. 647–655 (2014)
8. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: VLDB, pp. 518–529 (1999)
9. Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F.: Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI* **35**(12), 2916–2929 (2013). <https://doi.org/10.1109/TPAMI.2012.193>
10. Gui, L., Wang, Y., Hebert, M.: Few-shot hash learning for image retrieval. In: ICCVW, pp. 1228–1237 (2017). <http://doi.ieeecomputersociety.org/10.1109/ICCVW.2017.148>
11. He, K., Wen, F., Sun, J.: K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In: CVPR, pp. 2938–2945 (2013). <https://doi.org/10.1109/CVPR.2013.378>
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016). <https://doi.org/10.1109/CVPR.2016.90>
13. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR abs/1412.6980 (2014). <http://arxiv.org/abs/1412.6980>
14. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report (2009)
15. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS, pp. 1106–1114 (2012). <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks>
16. Kulis, B., Jain, P., Grauman, K.: Fast similarity search for learned metrics. *TPAMI* **31**(12), 2143–2157 (2009). <https://doi.org/10.1109/TPAMI.2009.151>
17. Lai, H., Pan, Y., Liu, Y., Yan, S.: Simultaneous feature learning and hash coding with deep neural networks. In: CVPR, pp. 3270–3278 (2015). <https://doi.org/10.1109/CVPR.2015.7298947>
18. Li, W., Wang, S., Kang, W.: Feature learning based deep supervised hashing with pairwise labels. In: IJCAI, pp. 1711–1717 (2016). <http://www.ijcai.org/Abstract/16/245>
19. Lin, G., Shen, C., Shi, Q., van den Hengel, A., Suter, D.: Fast supervised hashing with decision trees for high-dimensional data. In: CVPR, pp. 1971–1978 (2014). <https://doi.org/10.1109/CVPR.2014.253>
20. Lin, Z., Ding, G., Hu, M., Wang, J.: Semantics-preserving hashing for cross-view retrieval. In: CVPR, pp. 3864–3872 (2015). <https://doi.org/10.1109/CVPR.2015.7299011>
21. Liong, V.E., Lu, J., Wang, G., Moulin, P., Zhou, J.: Deep hashing for compact binary codes learning. In: CVPR, pp. 2475–2483 (2015). <https://doi.org/10.1109/CVPR.2015.7298862>
22. Liu, H., Wang, R., Shan, S., Chen, X.: Deep supervised hashing for fast image retrieval. In: CVPR, June 2016
23. Liu, H., Ji, R., Wu, Y., Huang, F., Zhang, B.: Cross-modality binary code learning via fusion similarity hashing. In: CVPR, pp. 6345–6353 (2017). <https://doi.org/10.1109/CVPR.2017.672>
24. Liu, W., Mu, C., Kumar, S., Chang, S.: Discrete graph hashing. In: NIPS, pp. 3419–3427 (2014)
25. Liu, W., Wang, J., Ji, R., Jiang, Y., Chang, S.: Supervised hashing with kernels. In: CVPR, pp. 2074–2081 (2012). <https://doi.org/10.1109/CVPR.2012.6247912>
26. Liu, W., Wang, J., Kumar, S., Chang, S.: Hashing with graphs. In: ICML (2011)
27. Liu, W., Wang, J., Mu, Y., Kumar, S., Chang, S.: Compact hyperplane hashing with bilinear functions. In: ICML (2012)

28. Mnih, V., et al.: Human-level control through deep reinforcement learning. *Nature* **518**(7540), 529–533 (2015). <https://doi.org/10.1038/nature14236>
29. Norouzi, M., Fleet, D.J.: Minimal loss hashing for compact binary codes. In: *ICML*, pp. 353–360 (2011)
30. Shen, F., Shen, C., Liu, W., Shen, H.T.: Supervised discrete hashing. In: *CVPR*, pp. 37–45 (2015). <https://doi.org/10.1109/CVPR.2015.7298598>
31. Shen, F., Shen, C., Shi, Q., van den Hengel, A., Tang, Z.: Inductive hashing on manifolds. In: *CVPR*, pp. 1562–1569 (2013). <https://doi.org/10.1109/CVPR.2013.205>
32. Shen, Y., Liu, L., Shao, L., Song, J.: Deep binaries: encoding semantic-rich cues for efficient textual-visual cross retrieval. In: *ICCV*, pp. 4117–4126 (2017). <http://doi.ieeecomputersociety.org/10.1109/ICCV.2017.441>
33. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. *CoRR* abs/1409.1556 (2014). <http://arxiv.org/abs/1409.1556>
34. Song, J.: Binary generative adversarial networks for image retrieval. *CoRR* abs/1708.04150 (2017). <http://arxiv.org/abs/1708.04150>
35. Wang, J., Zhang, T., Song, J., Sebe, N., Shen, H.T.: A survey on learning to hash. *CoRR* abs/1606.00185 (2016). <http://arxiv.org/abs/1606.00185>
36. Wang, J., Kumar, S., Chang, S.: Semi-supervised hashing for large-scale search. *TPAMI* **34**(12), 2393–2406 (2012). <https://doi.org/10.1109/TPAMI.2012.48>
37. Wang, J., Liu, W., Kumar, S., Chang, S.: Learning to hash for indexing big data - a survey. *Proc. IEEE* **104**(1), 34–57 (2016). <https://doi.org/10.1109/JPROC.2015.2487976>
38. Wang, Q., Si, L., Zhang, D.: Learning to hash with partial tags: exploring correlation between tags and hashing bits for large scale image retrieval. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) *ECCV 2014*. LNCS, vol. 8691, pp. 378–392. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10578-9_25
39. Weinberger, K.Q., Dasgupta, A., Langford, J., Smola, A.J., Attenberg, J.: Feature hashing for large scale multitask learning. In: *ICML*, pp. 1113–1120 (2009). <https://doi.org/10.1145/1553374.1553516>
40. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: *NIPS*, pp. 1753–1760 (2008)
41. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **8**, 229–256 (1992). <https://doi.org/10.1007/BF00992696>
42. Xia, R., Pan, Y., Lai, H., Liu, C., Yan, S.: Supervised hashing for image retrieval via image representation learning. In: *AAAI* (2014). <http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8137>
43. Xiao, J., Hays, J., Ehinger, K.A., Oliva, A., Torralba, A.: SUN database: large-scale scene recognition from abbey to zoo. In: *CVPR*, pp. 3485–3492 (2010). <https://doi.org/10.1109/CVPR.2010.5539970>
44. Zhang, D., Li, W.: Large-scale supervised multimodal hashing with semantic correlation maximization. In: *AAAI*, pp. 2177–2183 (2014). <http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8382>
45. Zhang, R., Lin, L., Zhang, R., Zuo, W., Zhang, L.: Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *IEEE Trans. Image Process.* **24**(12), 4766–4779 (2015). <https://doi.org/10.1109/TIP.2015.2467315>
46. Zhu, H., Long, M., Wang, J., Cao, Y.: Deep hashing network for efficient similarity retrieval. In: *AAAI*, pp. 2415–2421 (2016). <http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12039>