Binary Optimized Hashing

Qi Dai[1], Jianguo Li[2], Jingdong Wang[3], Yu-Gang Jiang[1]
[1]School of Computer Science, Shanghai Key Lab of Intelligent Information Processing,
Fudan University, Shanghai, China
[2]Intel Labs China, Beijing, China
[3]Microsoft Research Asia, Beijing, China
{daiqi, ygj}@fudan.edu.cn, jianguo.li@intel.com, jingdw@microsoft.com

ABSTRACT

This paper studies the problem of learning to hash, which is essentially a mixed integer optimization problem, containing both the binary hash code output and the (continuous) parameters forming the hash functions. Different from existing relaxation methods in hashing, which have no theoretical guarantees for the error bound of the relaxations, we propose *binary optimized hashing* (BOH), in which we prove that if the loss function is Lipschitz continuous, the binary optimization problem can be relaxed to a bound-constrained continuous optimization problem. Then we introduce a surrogate objective function, which only depends on unbinarized hash functions and does not need the slack variables transforming unbinarized hash functions to discrete functions, to approximate the relaxed objective function. We show that the approximation error is bounded and the bound is small when the problem is optimized. We apply the proposed approach to learn hash codes from either handcraft feature inputs or raw image inputs. Extensive experiments are carried out on three benchmarks, demonstrating that our approach outperforms state-of-the-arts with a significant margin on search accuracies.

CCS Concepts

•**Computing methodologies** → **Visual content-based indexing and retrieval; Image representations;** *Neural networks;*

Keywords

Hashing; Binary Optimized Hashing; Image Retrieval; CNN

1. INTRODUCTION

Hashing has been attracting broad attention due to the explosive growth of various kinds of data like documents, images and videos. It aims at encoding input data to a set of compact binary codes, while preserving some notion of similarity of the original data in the Hamming space. With the compact hash codes, one can easily realize highly efficient search applications with the Hamming distance under significantly lower storage cost.

The early exploration on hashing began with the well-known locality sensitive hashing (LSH) [4], which imposes random projection on data to generate hash codes. LSH is a data-independent hashing method, so that its results are usually not satisfying. Realizing the limitation of the data-independent hashing methods, many recent hashing techniques try to exploit various machine learning paradigms to learn effective hash functions from a given dataset, ranging from unsupervised to supervised or even semi-supervised settings [32, 31]. Recently, as deep learning has demonstrated strong results on many visual recognition tasks [11], several researches also proposed to learn hash codes from input images in an end-to-end manner with deep learning [34, 15], which led to significant improvements over traditional handcraft feature based hashing methods. All these data-dependent methods are generally called learning to hash.

Learning to hash is essentially a mixed integer nonlinear optimization problem with binary code outputs and continuous parameters for hash functions. The problem is difficult to be solved directly due to its NP-hard nature. Various relaxations have been introduced, such as (1) continuous relaxation which discards the sign function and discrete constraints to solve one continuous problem and then thresholds the continuous outputs to produce hash codes; (2) two-step schemes which inference hash-bit first and train hash functions with given hash-bits [18, 17, 27]. Although some methods in this category were claimed to be optimal with the relaxed objective functions, there are no theoretical guarantees for the error-bound of the relaxations.

Inspired by the works on discrete optimization [3, 22], this paper proposes an optimal hashing method, namely *binary optimized hashing* (BOH). We relax the binary variable to a bounded continuous variable, and show that the binary program is equivalent to a bound-constrained continuous optimization problem with an additional penalty function if the objective function satisfies certain conditions. We further introduce a surrogate objective function with respect to unbinarize hash functions only, to approximate the relaxed problem. We show that the approximation error is bounded and the bound is small when the problem is optimized. We apply the proposed approach to learn hash codes directly from either handcraft feature inputs or raw image inputs in a unified learning paradigm.

Extensive experiments show that coupling with neural networks, BOH can achieve superior performance over the

MM '16, October 15-19, 2016, Amsterdam, Netherlands

DOI: http://dx.doi.org/10.1145/2964284.2964331

state-of-the-art methods. In details, coupling with multi-layer perceptron (MLP), BOH gives superior results over compared methods for handcraft feature inputs; while coupling with convolutional neural networks (CNN), BOH outperforms all state-of-the-arts with a significant margin for raw image inputs. The main contributions of this paper are summarized as below:

(1) We relax the binary optimization problem to a bound-constrained continuous optimization problem. We further prove that the two problems are equivalent if the loss functions satisfy certain conditions. Hence we derive a surrogate objective function with respect only to hash functions, so that we can learn hash codes in a unified framework.

(2) We prove the feasibility of using two popular loss functions: triplet ranking loss and contrastive loss. We apply BOH to learn hash codes from both the handcraft feature inputs and raw image inputs.

(3) We demonstrate superior performance of the proposed approach on popular benchmarks. Our approach outperforms state-of-the-art approaches on both handcraft feature inputs and raw image inputs with clear margins.

This paper is organized as follows. We introduce related works in Section 2 and present details of the BOH in Section 3. Section 4 discusses experimental settings and results. Conclusions are drawn in Section 5.

2. RELATED WORKS

In this section we discuss related works in the following three aspects.

Learning to hash can be roughly divided into the following three categories: unsupervised, supervised and semi-supervised. Unsupervised methods try to preserve the inherent properties within data. Such properties include data distributions (e.g., spectral hashing [33], kernelized locality sensitive hashing [13, 14]), manifold structures (e.g., graph hashing [21]) and various similarities in feature space (e.g., angular quantization hashing [5], spherical hashing [8]). Although unsupervised hashing methods work well to preserve distance/similarity in the feature space, many applications also require to preserve semantical similarity in the Hamming space. For this purpose, various supervised hashing methods are proposed, ranging from metric learning to kernel learning (e.g., LDAHash [29], supervised hash with kernels [20], Hamming distance metric learning [24], supervised discrete hashing [27]). In addition, as it is very expensive to annotate large-scale datasets, it is practically needed to learn hash codes with both labeled and unlabeled data (e.g., semi-supervised hashing [30]).

Deep learning based hashing has recently shown very strong performance improvements over the shallow learning counterparts. Semantic Hashing [26] employs a two-stage learning (pretraining and fine-tuning) on deep generative models to abstract input documents to several bits. Deep hashing [2] seeks multiple hierarchical non-linear transformations to learn hash codes. Besides, convolutional neural networks (CNN) provides a way to learn hash codes from raw input in an end-to-end way. Specially, deep semantic ranking hashing [35] presents a new ranking loss, preserving the similarity relationships between multi-label images. Convolutional neural network hashing [34] also uses a two-step learning framework. The first stage performs semantic similarity matrix factorization to obtain hash codes, and the second stage trains a CNN network to obtain hash functions from raw image inputs. Deep neural network hashing [15] uses triplet ranking loss to model the similarity relations between images. A divide-and-encode module is employed to map the image features to approximate hash codes.

Different relaxations are implicitly or explicitly employed in learning to hash due to the NP-hard nature of the mixed integer optimization problem. Most aforementioned methods discard the discrete constraints to solve one continuous problem, and then threshold the continuous outputs to be binary codes. This simplifies the optimization greatly, but usually yields low-quality approximated solutions. Recently, some novel relaxations are proposed. The two-step schemes first produce binary codes with some discrete optimization such as binary quadratic optimization [18], graph cuts [17], semantic matrix factorization [34], discrete cyclic coordinate descent [27], etc. After that, certain learning techniques are employed to learn the hash functions with the desired hash codes. Iterative multi-step schemes are also proposed to optimize binary codes and hash functions alternatively [19, 27].

3. BINARY OPTIMIZED HASHING

The purpose of learning-to-hash is to seek a set of hash functions $H(\boldsymbol{x}) = \{h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), ..., h_d(\boldsymbol{x})\}$, which map the feature $\boldsymbol{x}$ to the Hamming space $\{0,1\}^d$, obtaining a sequence of binary codes $\boldsymbol{b} = H(\boldsymbol{x})$, so that the similarity in the Hamming space, defined from the Hamming distance $\|\boldsymbol{b}_i - \boldsymbol{b}_j\|_1$, is able to preserve some notion of similarity evaluated in the original feature space. In general, the objective function can be written as follows,

$$\begin{aligned} \min \quad & L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_N) \\ s.t. \quad & \boldsymbol{b}_i \in \{0,1\}^d \\ & \boldsymbol{b}_i = H(\boldsymbol{x}_i), i = 1, 2, \cdots N, \end{aligned} \tag{1}$$

where N is the number of training samples. Usually the loss function $L(\cdot)$ is defined as the sum of loss functions over cliques $\mathcal{C} = \{c\}$ with c being a set of points, $L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_N) = \sum_{c \in \mathcal{C}} \ell(\boldsymbol{b}_{i_1}, \ldots, \boldsymbol{b}_{i_{|c|}})$, $(i_1, \ldots, i_{|c|}) = c)$. There are various forms to formulate the loss function. In this paper, we will study two widely-used forms: triplet ranking loss, in which c is a triplet (i, j, k) and contrastive loss where c is a point pair (i, j).

The problem (1) is a mixed-integer optimization problem. Due to its NP-hard nature, the problem is solved with various relaxations, such as continuous relaxation, two-step schemes, and iteratively multi-step schemes as mentioned in related works. Our solution starts from relaxing the binary variable into a bounded continuous variable and introducing a penalty function. We first only consider the binary variables in the problem (1) and show that the 0-1 program is equivalent to a continuous optimization problem if the objective function $L(\cdot)$ satisfies certain condition. Then we take into consideration the hash function and relax the whole problem into a continuous problem, which subsequently is transformed to a surrogate objective function with respect to only the hash functions.

3.1 Equivalent Continuous Problem

Let us consider the following generic 0-1 program,

$$\begin{aligned} \min \quad & f(\boldsymbol{y}) \\ s.t. \quad & \boldsymbol{y} \in \{0,1\}^d, \end{aligned} \tag{2}$$

and a transformed optimization problem,

$$\begin{aligned} \min \quad & f(\boldsymbol{y}) + \lambda\phi(\boldsymbol{y}) \\ s.t. \quad & \boldsymbol{y} \in [0,1]^d, \end{aligned} \tag{3}$$

where $\phi(\boldsymbol{y}) : \mathbb{R}^d \to \mathbb{R}$ to $f(\boldsymbol{y})$ is a penalty term and λ is the penalty coefficient. It is shown in [3, 22] that under certain conditions, the above two problems are equivalent.

Lemma 1: *Let $\|\cdot\|$ be a suitably chosen norm. Suppose the following hypotheses hold:*
a) $f(\boldsymbol{y})$ is bounded when $\boldsymbol{y} \in [0,1]^k$, and there exists an open set $A \supset \{0,1\}^d$ and real number $\alpha, \eta > 0$, such that $\forall \boldsymbol{y}_1, \boldsymbol{y}_2 \in A$, $f(\cdot)$ satisfies the following Hölder condition:

$$|f(\boldsymbol{y}_1) - f(\boldsymbol{y}_2)| \leqslant \eta \|\boldsymbol{y}_1 - \boldsymbol{y}_2\|^\alpha. \tag{4}$$

b) It is possible to find $\phi : \mathbb{R}^d \to \mathbb{R}$, such that:

(1) ϕ is continuous on $[0,1]^d$;
(2) $\phi(\boldsymbol{y}) = 0, \forall \boldsymbol{y} \in \{0,1\}^k$; and $\phi(\boldsymbol{y}) > 0, \forall \boldsymbol{y} \in (0,1)^k$
(3) $\forall \boldsymbol{z} \in \{0,1\}^k$, there exists a neighborhood $S(\boldsymbol{z})$ of $\boldsymbol{z}$, and a real $\epsilon(\boldsymbol{z}) > 0$, such that:

$$\phi(\boldsymbol{y}) \geqslant \epsilon(\boldsymbol{z})\|\boldsymbol{y} - \boldsymbol{z}\|^\alpha, \forall \boldsymbol{y} \in S(\boldsymbol{z}) \cap (0,1)^k. \tag{5}$$

Then, a real value λ_0 exists, such that $\forall \lambda > \lambda_0$, the two problems (2) and (3) are equivalent.

When $\alpha = 1$, the condition (a) is also called *Lipschitz condition*, and $f(\cdot)$ is called *Lipschitz continuous*.

Lemma 2: *If $f_1(\cdot)$ and $f_2(\cdot)$ are* Lipschitz continuous, *then $f_1(\cdot) + f_2(\cdot)$ is also* Lipschitz continuous.

In [22], the equivalence between (2) and (3) has been proved for various forms of function ϕ. This paper considers a simple penalty form

$$\phi(\boldsymbol{y}) = \boldsymbol{y}^T(\boldsymbol{e} - \boldsymbol{y}), \tag{6}$$

where $\boldsymbol{e} = [1, 1, \ldots, 1]^T$. This leads to the following equivalent problem,

$$\begin{aligned} \min \quad & f(\boldsymbol{y}) + \lambda \boldsymbol{y}^T(\boldsymbol{e} - \boldsymbol{y}) \\ s.t. \quad & \boldsymbol{y} \in [0,1]^d \end{aligned} \tag{7}$$

Lemma 3: *Let f satisfy condition (a) of **Lemma 1** with $\alpha = 1$, i.e. $f(\boldsymbol{y})$ is bounded when $\boldsymbol{y} \in [0,1]^d$ and also Lipschitz continuous on an open set $A \supset \{0,1\}^d$. Then, there exists $\lambda_0 \in \mathbb{R}$ such that for every $\lambda \geq \lambda_0$ problems (2) and (7) are equivalent.*

We take two popular loss functions, triplet ranking loss [15, 16, 24] and contrastive loss as examples, and show that they are *Lipschitz continuous* below.

3.1.1 Triplet Ranking Loss is Lipschitz Continuous

The triplet loss function is given as

$$\begin{aligned} f(\boldsymbol{y}) &= \ell(\boldsymbol{b}_i, \boldsymbol{b}_j, \boldsymbol{b}_k) \\ &= \max(0, 1 - (\|\boldsymbol{b}_i - \boldsymbol{b}_k\|_2^2 - \|\boldsymbol{b}_i - \boldsymbol{b}_j\|_2^2)), \end{aligned} \tag{8}$$

where (i,k) is a similar pair and (i,j) is a dissimilar pair and $\boldsymbol{y} = [\boldsymbol{b}_i^T, \boldsymbol{b}_j^T, \boldsymbol{b}_k^T]^T$. In the following, we show that $f(\boldsymbol{y})$ is Lipschitz continuous, from which the overall objective function, the summation of Lipschitz continuous functions, $L(\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_N) = \sum_{(i,j,k)\in\mathcal{C}} \ell(\boldsymbol{b}_i, \boldsymbol{b}_j, \boldsymbol{b}_k)$ is obviously Lipschitz continuous according to **Lemma 2**.

Lemma 4: *Any continuously differentiable function is locally Lipschitz continuous.*

Lemma 5: *If $f_1(\cdot)$ and $f_2(\cdot)$ are Lipschitz continuous, then $\max(f_1(\cdot), f_2(\cdot))$ is also Lipschitz continuous.*

Theorem 1: *The triplet ranking loss function (8) is* Lipschitz continuous.

Proof: It is obvious that $0 \leqslant f(\boldsymbol{y}) \leqslant d^2 + 1$, i.e., $f(\cdot)$ is bounded.

For the *Lipschitz condition* on an open set A, we separate the max function to two functions $g(\boldsymbol{y})$ and $k(\boldsymbol{y})$, where $g(\boldsymbol{y}) = 0$, and $k(\boldsymbol{y}) = 1 - (\|\boldsymbol{b}_i - \boldsymbol{b}_k\|_2^2 - \|\boldsymbol{b}_i - \boldsymbol{b}_j\|_2^2)$.

Suppose the open set A to be $(-1,2)^k$ ($k = 3d$). It is obvious that function $k(\boldsymbol{y})$ is continuously differentiable on A. According to **Lemma 4**, $k(\boldsymbol{y})$ is *Lipschitz continuous* on A. On the other side, function $g(\boldsymbol{y})$ is a constant function, indicating that $g(\boldsymbol{y})$ is also *Lipschitz continuous* on A. Then two functions are both *Lipschitz continuous*.

According to **Lemma 5**, $\max(g(\boldsymbol{y}), k(\boldsymbol{y}))$ is *Lipschitz continuous*. This completes the proof.

3.1.2 Contrastive Loss is Lipschitz Continuous

Contrastive loss [7] is widely used for metric learning with only pairwise constraints. Suppose we have a pair of images $(\mathcal{I}, \mathcal{I}_c)$, and $(\boldsymbol{b}, \boldsymbol{b}_c)$ are the corresponding hash codes. $\boldsymbol{y} = [\boldsymbol{b}; \boldsymbol{b}_c]$. The contrastive loss function f is defined as:

$$f(\boldsymbol{y}) = l_c\|\boldsymbol{b} - \boldsymbol{b}_c\|_2^2 + (1 - l_c)\max(0, m - \|\boldsymbol{b} - \boldsymbol{b}_c\|_2^2) \tag{9}$$

where $l_c = 1$ indicates $\mathcal{I}$ and $\mathcal{I}_c$ are similar, otherwise 0. This loss function is indeed similar to LDAHash [29].

Theorem 2: *The contrastive loss function (9) is* Lipschitz continuous.

Proof: According to **Lemma 4** and previous proof, we can easily prove the two parts $u(\boldsymbol{y}) = l_c\|\boldsymbol{b} - \boldsymbol{b}_c\|_2^2$ and $v(\boldsymbol{y}) = (1 - l_c)\max(0, m - \|\boldsymbol{b} - \boldsymbol{b}_c\|_2^2)$ are *Lipschitz continuous* on the open set $A = (-1,2)^k$ separately.

According to **Lemma 2**, the linear combination of $u(\boldsymbol{y})$ and $v(\boldsymbol{y})$ is also *Lipschitz continuous*. This completes the proof.

3.2 Surrogate Objective Function

We derive hash function $h(\boldsymbol{x}) \in \{0,1\}$ from its unbinarized version $\tilde{h}(\boldsymbol{x})$, ($\tilde{h}(\boldsymbol{x}) \in [0,1]$ in this paper): $h(\boldsymbol{x}) = \tilde{h}(\boldsymbol{x}) + \epsilon$, where $|\epsilon| = \min(\tilde{h}(\boldsymbol{x}), 1 - \tilde{h}(\boldsymbol{x}))$. The hash function optimization is equivalent to the quantization loss problem,

$$\begin{aligned} \min \quad & \|h(\boldsymbol{x}) - \tilde{h}(\boldsymbol{x})\|_2^2 \\ s.t. \quad & h(\boldsymbol{x}) \in \{0,1\}. \end{aligned} \tag{10}$$

Considering the hash function as a constraint of the problem (1) and combining the optimization problem (10) into (1), we have

$$\begin{aligned} \min \quad & L(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_N) + \rho \sum_{i=1}^{N} \|\boldsymbol{b}_i - \tilde{H}(\boldsymbol{x}_i)\|_2^2 \\ s.t. \quad & \boldsymbol{b}_i \in \{0,1\}^d, i = 1, 2, \cdots N, \end{aligned} \tag{11}$$

where $\tilde{H}(\boldsymbol{x}_i) = [\tilde{h}_1(\boldsymbol{x}_i), \tilde{h}_2(\boldsymbol{x}_i), \ldots, \tilde{h}_d(\boldsymbol{x}_i)]$. It can be proved that a real value ρ_0 exists, such that $\rho > \rho_0$, the problems (11) and (1) are equivalent.

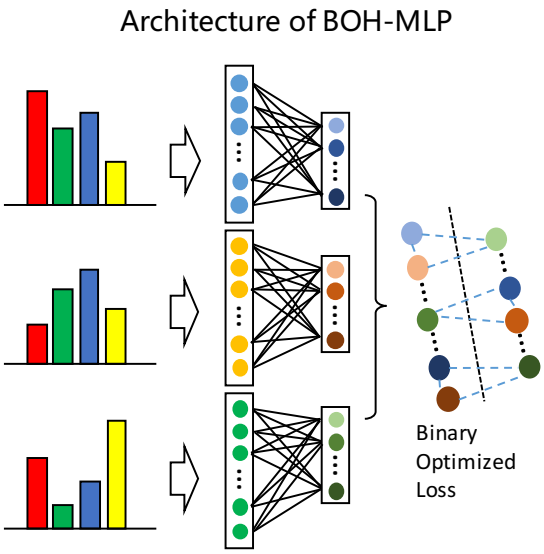

Figure 1: Architecture of BOH-MLP, in which we adopt MLP as hash functions, and use triplet ranking loss for training, with handcraft features as inputs.

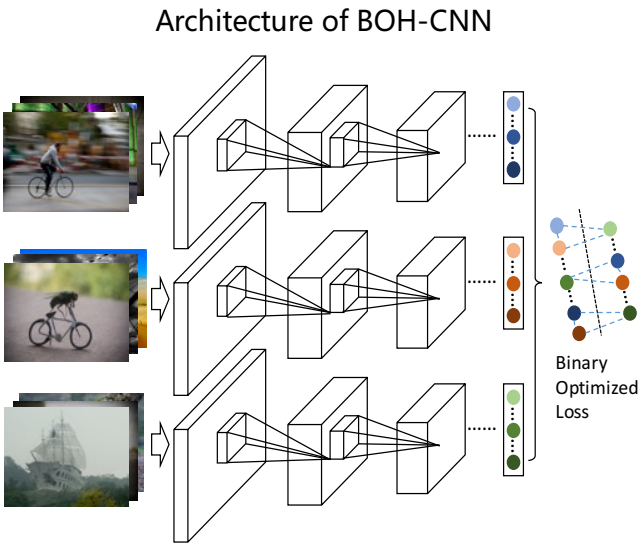

Figure 2: Architecture of BOH-CNN, in which we adopt CNN as hash functions. Similar to Figure 1, triplet ranking loss is used here, but with raw image inputs.

It is obvious that $\sum_{i=1}^{N} \|\boldsymbol{b}_i - \tilde{H}(\boldsymbol{x}_i)\|_2^2$ is Lipschitz continuous. We have shown $L(\cdot)$ is Lipschitz continuous. Therefore, problem (11) is relaxed to the following equivalent problem:

$$\begin{aligned} \min \quad & L(\tilde{\boldsymbol{b}}_1 + \boldsymbol{\epsilon}_1, \cdots, \tilde{\boldsymbol{b}}_N + \boldsymbol{\epsilon}_N) + \rho \sum_{i=1}^{N} \|\boldsymbol{\epsilon}_i\|_2^2 \\ & + \lambda \sum_{i=1}^{N} \phi(\tilde{\boldsymbol{b}}_i + \boldsymbol{\epsilon}_i) \\ s.t. \quad & \tilde{\boldsymbol{b}}_i + \boldsymbol{\epsilon}_i \in [0,1]^d, i = 1, 2, \cdots N, \end{aligned} \tag{12}$$

where $\tilde{\boldsymbol{b}}_i = \tilde{H}(\boldsymbol{x}_i)$, $\boldsymbol{\epsilon}_i = H(\boldsymbol{x}_i) - \tilde{H}(\boldsymbol{x}_i) = \boldsymbol{b}_i - \tilde{\boldsymbol{b}}_i$ and $\phi(\cdot)$ follows Equation (6).

Suppose $F(\tilde{\boldsymbol{b}}_1, \cdots, \tilde{\boldsymbol{b}}_N) = L(\tilde{\boldsymbol{b}}_1, \cdots, \tilde{\boldsymbol{b}}_N) + \lambda \sum_{i=1}^{N} \phi(\tilde{\boldsymbol{b}}_i)$. As both $L(\cdot)$ and $\phi(\cdot)$ are Lipschitz continuous, according to **Lemma 1**, we have

$$\begin{aligned} & |F(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_N) - F(\tilde{\boldsymbol{b}}_1, \cdots, \tilde{\boldsymbol{b}}_N)| \\ = & |F(\tilde{\boldsymbol{b}}_1 + \boldsymbol{\epsilon}_1, \cdots, \tilde{\boldsymbol{b}}_N + \boldsymbol{\epsilon}_N) - F(\tilde{\boldsymbol{b}}_1, \cdots, \tilde{\boldsymbol{b}}_N)| \\ \leqslant & \eta \sum_{i=1}^{N} \|\boldsymbol{\epsilon}_i\|^1. \end{aligned} \tag{13}$$

This means the approximation error $|F(\boldsymbol{B}) - F(\tilde{\boldsymbol{B}})|$ is bounded, where $\boldsymbol{B} = [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N]$. Therefore, we could transform problem (12) to an approximate problem

$$\begin{aligned} \min \quad & L(\tilde{\boldsymbol{b}}_1, \cdots, \tilde{\boldsymbol{b}}_N) + \rho \sum_{i=1}^{N} \|\boldsymbol{\epsilon}_i\|_2^2 + \lambda \sum_{i=1}^{N} \phi(\tilde{\boldsymbol{b}}_i) \\ s.t. \quad & \tilde{\boldsymbol{b}}_i \in [0,1]^d, i = 1, 2, \cdots N. \end{aligned} \tag{14}$$

which can be approximately optimized with two subsequent problems. First, we optimize unbinarized hash codes $\tilde{\boldsymbol{b}}_i$ with

$$\begin{aligned} \min \quad & L(\tilde{\boldsymbol{b}}_1, \cdots, \tilde{\boldsymbol{b}}_N) + \lambda \sum_{i=1}^{N} \phi(\tilde{\boldsymbol{b}}_i), \\ s.t. \quad & \tilde{\boldsymbol{b}}_i \in [0,1]^d. \end{aligned} \tag{15}$$

Then we optimize $\boldsymbol{\epsilon}_i$ with

$$\begin{aligned} \min \quad & \rho \sum_{i=1}^{N} \|\boldsymbol{\epsilon}_i\|_2^2 \\ s.t. \quad & \tilde{\boldsymbol{b}}_i + \boldsymbol{\epsilon}_i \in \{0,1\}^d, i = 1, 2, \cdots N. \end{aligned} \tag{16}$$

The second sub-problem is very simple with solution $\epsilon_{ij} = -\tilde{b}_{ij}$ if $\tilde{b}_{ij} \leq 0.5$ and otherwise $\epsilon_{ij} = 1 - \tilde{b}_{ij}$, from which we can easily obtain the binarized hash codes $\boldsymbol{b}_i$. Equation (13) proves that $\tilde{\boldsymbol{b}}_i$ is a good approximation of $\boldsymbol{b}_i$ with bounded error. Along with the optimization, $\tilde{\boldsymbol{b}}_i$ becomes closer and closer to $\boldsymbol{b}_i$. We will verify this point in experiments (ref. Section 4.2.3).

Considering the second term in the final objective function (15), we have (drop the subscript i for clarity)

$$\begin{aligned} \phi(\tilde{\boldsymbol{b}}) &= \tilde{\boldsymbol{b}}^T (1 - \tilde{\boldsymbol{b}}) \\ &\geqslant (\min(\tilde{\boldsymbol{b}}, 1 - \tilde{\boldsymbol{b}}))^2 \\ &= \|\boldsymbol{\epsilon}\|_2^2, \end{aligned} \tag{17}$$

which indicates that $\|\boldsymbol{\epsilon}\|_2^2$ is small when $\tilde{\boldsymbol{B}}$ reaches the optimal solution of the problem (15) and we choose an unbinarized hash fucntion whose range is $[0, 1]$ implying ϵ is small. This means that the error bound in Equation (13) is small. In other words, the solution of the problem (14) is an accurate approximation of that of the problem (12).

Finally, the surrogate objective function of learning the hash functions is rewritten as follow,

$$\begin{aligned} \min \quad & L(\tilde{H}(\boldsymbol{x})) + \lambda \tilde{H}(\boldsymbol{x})^T (\boldsymbol{e} - \tilde{H}(\boldsymbol{x})), \\ s.t. \quad & \tilde{H}(\boldsymbol{x}) \in [0,1]^d. \end{aligned} \tag{18}$$

3.3 Model Details

3.3.1 Hash Functions

We formulate the unbinarized hash function $\tilde{\boldsymbol{b}} = \tilde{h}(\boldsymbol{x})$ using a neural network with the output based on a sigmoid

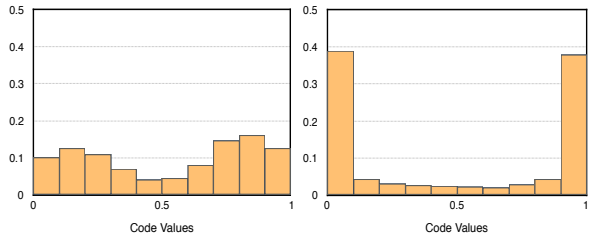

Figure 3: Histogram of output values by CNN trained with only the triplet ranking loss and the unified BOH loss. The vertical axis represents the frequency ratio of samples falling to certain intervals in the range [0,1]. Left: triplet ranking loss. Right: unified BOH loss.

Table 1: Result comparison on the effectiveness of the unified BOH loss, using 48 bits output and triplet ranking loss. The table above shows the results of mAP, and the table below shows the results of P@R⩽2.

Method	mAP		
	CIFAR-10	SVHN	NUS-WIDE
MLP	0.400	0.799	0.602
MLP+Regularization	0.352	0.800	0.603
BOH-MLP	0.420	0.818	0.623
CNN	0.627	0.914	0.831
CNN+Regularization	0.521	0.908	0.801
BOH-CNN	0.657	0.927	0.855

Method	P@R⩽2		
	CIFAR-10	SVHN	NUS-WIDE
MLP	0.421	0.774	0.600
MLP+Regularization	0.424	0.795	0.601
BOH-MLP	0.434	0.823	0.615
CNN	0.563	0.918	0.796
CNN+Regularization	0.220	0.710	0.327
BOH-CNN	0.666	0.924	0.839

function $s(t) = \frac{1}{1+e^{-t}}$ ($\in [0,1]$), which ensures that the error bound given in the Equation (13) is not large, as well as that the output satisfies the border constraint $[0,1]$. The input of the neural network could be either handcraft features or raw images, which leads to different architectures.

For the handcraft feature inputs, we take multi-layer perceptron (MLP) as hash functions, denoted as BOH-MLP. Figure 1 illustrates an architecture example with triplet ranking BOH loss. The inputs of BOH-MLP are dimension-fixed handcraft feature vectors, followed by several fully-connection (FC) layers, where the last FC layer is of size d, indicating the number of output hash bits. A sigmoid layer is connected to the last FC layer, ensuring the output values are in the range $[0,1]$.

For the raw image inputs, we take CNN as hash functions, and learn hash codes in an end-to-end manner, denoted as BOH-CNN. Figure 2 illustrates an architecture example also with triplet ranking BOH loss. We do not train CNN hash functions from scratch, but take parameters from the pre-trained CNN models (i.e. on ImageNet etc), and fine-tune the network parameters with stochastic gradient descent (SGD).

As it is trivial to get gradient from triplet ranking loss and contrastive loss, we omit the derivations here.

Table 2: The mAP (table above) and P@R⩽2 (table below) of BOH with contrastive loss (contra. in short) and triplet ranking loss, using 48 bits output.

Method	mAP		
	CIFAR-10	SVHN	NUS-WIDE
BOH-MLP Contra.	0.383	0.799	0.599
BOH-MLP Triplet	0.420	0.818	0.623
BOH-CNN Contra.	0.608	0.892	0.651
BOH-CNN Triplet	0.657	0.927	0.855

Method	P@R⩽2		
	CIFAR-10	SVHN	NUS-WIDE
BOH-MLP Contra.	0.422	0.779	0.591
BOH-MLP Triplet	0.434	0.823	0.615
BOH-CNN Contra.	0.370	0.893	0.349
BOH-CNN Triplet	0.666	0.924	0.839

3.3.2 *Additional Regularization Term*

In addition to the aforementioned two kinds of losses, we introduce one regularization term

$$\psi(\boldsymbol{y}) = \|\boldsymbol{b} + \boldsymbol{b}^- - \boldsymbol{e}\|_2^2, \tag{19}$$

where $\boldsymbol{y} = [\boldsymbol{b}; \boldsymbol{b}^-]$, $\boldsymbol{b}$ and $\boldsymbol{b}^-$ are hash codes of dissimilar images, and $\boldsymbol{e}$ is a vector with all elements being 1. The motivation behind this term is that dissimilar images should have opposite hash codes, i.e., they should have a larger Hamming distance.

It is easy to prove that $\psi(\boldsymbol{y})$ is Lipschitz continuous. We therefore define a unified loss by combining triplet ranking loss or contrastive loss $f(\cdot)$ with the regularization term

$$f(\boldsymbol{y}) + \lambda \boldsymbol{y}^T(\boldsymbol{e} - \boldsymbol{y}) + \beta\psi(\boldsymbol{y}), \tag{20}$$

where β is a hyper parameter, which controls the strength of regularization.

According to **Lemma 2**, the unified BOH loss is also Lipschitz continuous. We optimize problem (18) with this unified loss function.

3.3.3 *Complexity Analysis*

The training complexity of BOH-MLP is the same as MLP based hashing method, such as HDML [24]. Besides, BOH-CNN also has the same training complexity as the CNN baseline like DNNH [15]. For the testing part, the computation complexity of two-layer BOH-MLP is $O((d_i + d_o) \times d_h)$, while d_i, d_h and d_o are the dimension of input features, hidden neuron node size, and number of output hash bits. The testing complexity of BOH-CNN depends on the CNN structure used. However, it is generally with same order as DNNH [15].

4. EXPERIMENTS

4.1 Datasets and Experimental Settings

We conduct comprehensive experiments on three datasets, CIFAR-10 [10], SVHN [23] and NUS-WIDE [1], and compare our proposed method with several state-of-the-art hashing methods.

CIFAR-10 consists of 60K images of size 32×32, which are categorized into 10 classes, with 6K images per class. Each image has a unique label. We follow the settings of [34, 15], if there are no additional explanations. 1K images

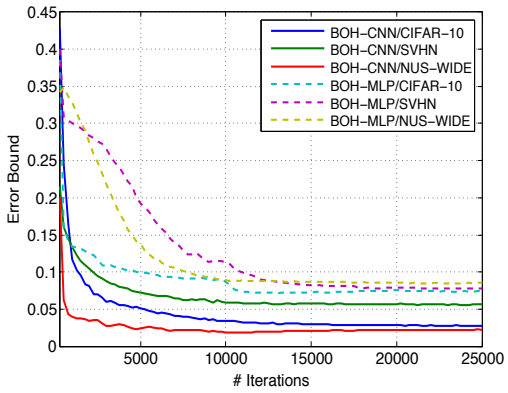

Figure 4: Average error bound vs. number of iterations. Here the triplet ranking loss function is used, with 48 bits hash codes.

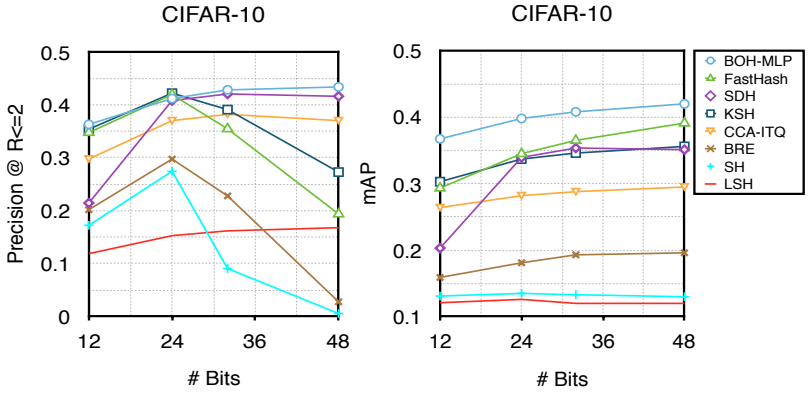

Figure 5: The mAP (right) and P@R⩽2 (left) of the compared methods on CIFAR-10 with the number of hash bits ranging from 12 to 48. The number of used training samples is 5,000.

(100 images per class) are randomly sampled to form the test query set. For the unsupervised methods, the training set contains the rest 59K images, while for the supervised methods, we randomly sample 5K images (500 images per class) for training. One exception is that in Sec. 4.3, we add one more additional training setting.

SVHN is a real-world image dataset which contains over 600K small 32×32 digits images. These images are split into three subset, named train set, test set and extra set. Following the settings in [15], we randomly select 1K images (100 images per class) from the whole set as test queries. For the unsupervised methods, rest of the images form the training set, while for the supervised methods, we randomly sample 5K images (500 images per class) for training. When testing, we use all the remaining images (over 600K) as query database.

NUS-WIDE contains about 270K Flickr images associated with 81 tags. Unlike CIFAR-10 and SVHN, each image in NUS-WIDE may have multiple tags. We follow the settings of [21, 15]. We only use 21 most frequent tags, where each tag has at least 5K images. We randomly sample 2,100 images (100 images per tag) to form the test query set. For the unsupervised methods, all the remaining images are used as training data. For the supervised methods, we randomly sample 10,500 images (500 per tag) for training.

For BOH-MLP, we use 512-dimensional GIST feature [25] for CIFAR-10/SVHN, and the provided 500-dimensional bag-of-words feature for NUS-WIDE. We design BOH-MLP with two hidden FC layers for all our experiments. The number of hidden neurons are 250 and 100 for the two layers, respectively. For BOH-CNN, we use raw image inputs directly. Different network structures are used for different datasets. For NUS-WIDE and CIFAR-10, we take the VGG-16 model [28] as our network structure. For SVHN, a smaller network structure is used, which contains three convolutional layers and two FC layers. The optimal values for hyper parameters β and λ in Equation (20) are determined on the training set in a cross-validation manner. Both BOH-MLP and BOH-CNN are implemented using Caffe [9].

We adopt the following four widely used metrics to evaluate the results: (1) mean average precision (mAP), (2) mean precision within Hamming radius 2 (in short P@R⩽2), (3) precision-recall curves, and (4) precision of top returned neighbors. In particular, on NUS-WIDE, we compute the mAP on top-5000 returned results. For P@R⩽2, if a query has no neighbors within radius 2, we treat the query with zero precision.

4.2 Design Evaluations

In this section we evaluate several design factors which affect the performance of our method.

4.2.1 Effectiveness of the Unified BOH Loss

One may be interested in knowing how much the unified BOH loss can contribute to the performance. To answer this question, we carry out experiments for the triplet ranking loss with the following three kinds of settings: 1) MLP or CNN trained with triplet ranking loss; 2) MLP or CNN trained with triplet ranking loss plus the regularization term from Equation 19; 3) MLP or CNN trained with the unified BOH loss (triplet ranking loss under BOH relaxation) from Equation 20. We set the number of hash bits $d = 48$.

Table 1 lists the comparison results. On CIFAR-10, the unified BOH loss reaches 2% and 3% gains on mAP with MLP and CNN hash function respectively. On SVHN, the performance gains are 1.9% and 1.3% for MLP and CNN respectively. And on NUS-WIDE, 2.1% and 2.4% improvements are observed. This shows that the improvements are consistently large over both CNN (strong) and MLP (weak) baseline. It is interesting to note that pure MLP or CNN with regularization term even yields noticeable accuracy drops to the counter part with the same settings. For instance, the mAP drops 5% and 10% on CIFAR-10 from the model trained with triplet ranking loss only. In terms of P@R⩽2 metric, the accuracy drops are even larger. This is due to the fact that MLP or CNN training with triplet loss does not guarantee optimal hash value outputs. We further present a visual illustration in Figure 3, which shows the histogram distribution of the hash values by CNN trained with triplet loss and unified BOH loss. It is obvious that BOH produces values mostly closer to 0 and 1, indicating the optimization nature of the method.

The results indicate that merely using regularization term does not bring performance improvement. On the contrary, it even hurts the performance seriously. Therefore, we conclude that the unified BOH loss as a whole is the most effective way for learning hash codes.

Table 3: The mAP (table above) and P@R⩽2 (table below) of HDML and BOH-HDML, with 48 bits output.

Method	mAP		
	CIFAR-10	SVHN	NUS-WIDE
HDML	0.273	0.679	0.544
BOH-HDML	0.306	0.714	0.582

Method	P@R⩽2		
	CIFAR-10	SVHN	NUS-WIDE
HDML	0.348	0.798	0.523
BOH-HDML	0.369	0.807	0.575

4.2.2 Triplet Ranking Loss vs. Contrastive Loss

In this experiment, we compare the performance of triplet ranking loss and contrastive loss under the unified BOH framework. Both BOH-MLP and BOH-CNN are evaluated, with the number of output hash bits $d = 48$.

Table 2 lists the comparison results on the three datasets. It shows that triplet ranking loss performs much better than contrastive loss on all metrics for these datasets. Specially, triplet ranking loss yields 1.9∼20.4% performance gain over contrastive loss on the mAP metric. For the P@R⩽2 metric on CNN based solutions, the differences are 29.6%, 49% and 3.5% on CIFAR-10, NUS-WIDE and SVHN respectively, where the results are much lower on the former two datasets. The reasons are two-fold. First, contrastive loss yields quite a few query cases that have no returned results within Hamming distance radius 2, and we treat these as zero precision so the P@R⩽2 accuracy is much lower. Second, the triplet ranking loss is naturally good for ranking so it has better top-ranked retrieval results. Based on these results, we use the triplet ranking loss in the following studies.

4.2.3 Convergence of BOH

It has been shown in Sec. 3.2 that we get an approximate problem (14) by optimizing the surrogate objective function. The approximation error is bounded by $\|\boldsymbol{\epsilon}\|^1$, as in (13). The optimization will yield the approximation error converged to a small value. In order to verify such approximation, we evaluate the error bound in this experiment.

Figure 4 illustrates the average error bound vs. number of iterations. Suppose the hash code is d bits, the average error bound is defined as $\|\boldsymbol{\epsilon}\|^1/d$ over all evaluating samples. We use triplet ranking loss here, and set the number of hash bits $d = 48$. We compute the average error bound every 250 iterations. It could be observed that the average error bound decreases gradually when training is performed. The error bound becomes stable and keeps at a low value after certain number of iterations, revealing that the obtained solution is very close to the optimal binarized solution. Besides, it also demonstrates that BOH-CNN could achieve much lower error bound than BOH-MLP.

4.2.4 Generalization to Existing Hashing Methods

The proposed BOH loss is fairly generic. It could be applied to some existing hashing methods, especially for those with triplet ranking loss. Here, we consider extending our BOH loss to HDML [24], which defines the loss as

$$L(\boldsymbol{w}) = \sum l_{triplet}(\boldsymbol{b}(\boldsymbol{x};\boldsymbol{w}), \boldsymbol{b}(\boldsymbol{x}^+;\boldsymbol{w}), \boldsymbol{b}(\boldsymbol{x}^-;\boldsymbol{w})) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2,$$

where $\{\boldsymbol{x}^+, \boldsymbol{x}, \boldsymbol{x}^-\}$ is a triplet, $\boldsymbol{b}(\boldsymbol{x};\boldsymbol{w})$ is the hash function with parameter $\boldsymbol{w}$, and $l_{triplet}(\cdot)$ is triplet ranking loss. HDML adopts perceptrons as hash functions, and optimizes it with triplet ranking loss plus one matrix norm regularization term. In this study, we replace the regularized triplet ranking loss with the proposed BOH loss, and use one-layer perceptron with sigmoid activation as hash functions. We evaluate the performance on three datasets with 48 hash bits. The results are shown in Table 3. It could be observed that we gain substantial improvement with BOH loss, which demonstrates the generalization capability of the proposed method.

Table 4: mAP and P@R⩽2 of various methods on CIFAR-10 with 48 bits codes. The "# training" column indicates the number of training samples used.

Method	# training	mAP	P@R⩽2
LSH	5000	0.120	0.168
SH	5000	0.130	0.017
BRE	5000	0.196	0.028
CCA-ITQ	5000	0.295	0.370
KSH	5000	0.356	0.273
FastHash	5000	0.391	0.194
SDH	5000	0.351	0.416
BOH-MLP	5000	**0.420**	**0.434**
FastHash	59000	0.581	0.162
SDH	59000	0.509	0.529
BOH-MLP	59000	**0.626**	**0.598**

4.3 Result Comparison on Handcraft Features

We first compare BOH-MLP with existing state-of-the-art methods, including LSH [4], SH [33], BRE [12], CCA-ITQ [6], KSH [20], FastHash [17] and SDH [27]. We conduct experiments on CIFAR-10. Besides the regular 5K training samples, we also use one additional 59K training samples for SDH, FastHash and the proposed BOH-MLP. Note that for fair comparison, we use handcraft features as inputs for all the methods.

Table 4 shows the mAP and P@R⩽2 results with 48 bits output. We see that BOH-MLP achieves promising results on both training settings. For instance, when the number of training samples is 5K, the mAP of BOH-MLP exceeds FastHash by 2.9% and SDH by 6.9%. In addition, it is not surprising that when we use more training samples (59K), the performance of all the methods becomes better. In particular, BOH-MLP achieves the best performance on mAP (0.626) and P@R⩽2 (0.598), with improvements of 20.6% and 16.4% respectively over 5K training set. FastHash also achieves substantial improvement on mAP (19% improvement). However, its performance on P@R⩽2 is very poor (0.162). The poor P@R⩽2 of FastHash is consistent with the observations in [27]. Besides, SDH obtains approximately 16% improvement on mAP and 11% improvement on P@R⩽2. Moreover, by observing Table 1 and 4 (5K training samples), we can see that MLP achieves similar performance to FastHash (40% vs. 39.1%). The high-bar baseline of MLP (without BOH loss) is majorly due to the triplet ranking loss we used. However, when BOH loss is applied, the BOH-MLP achieves even better performance (42%). This improvement is substantial and verifies the superiority of the proposed method.

Figure 5 further shows the performance over different output hash bits. BOH-MLP shows stable improvements on

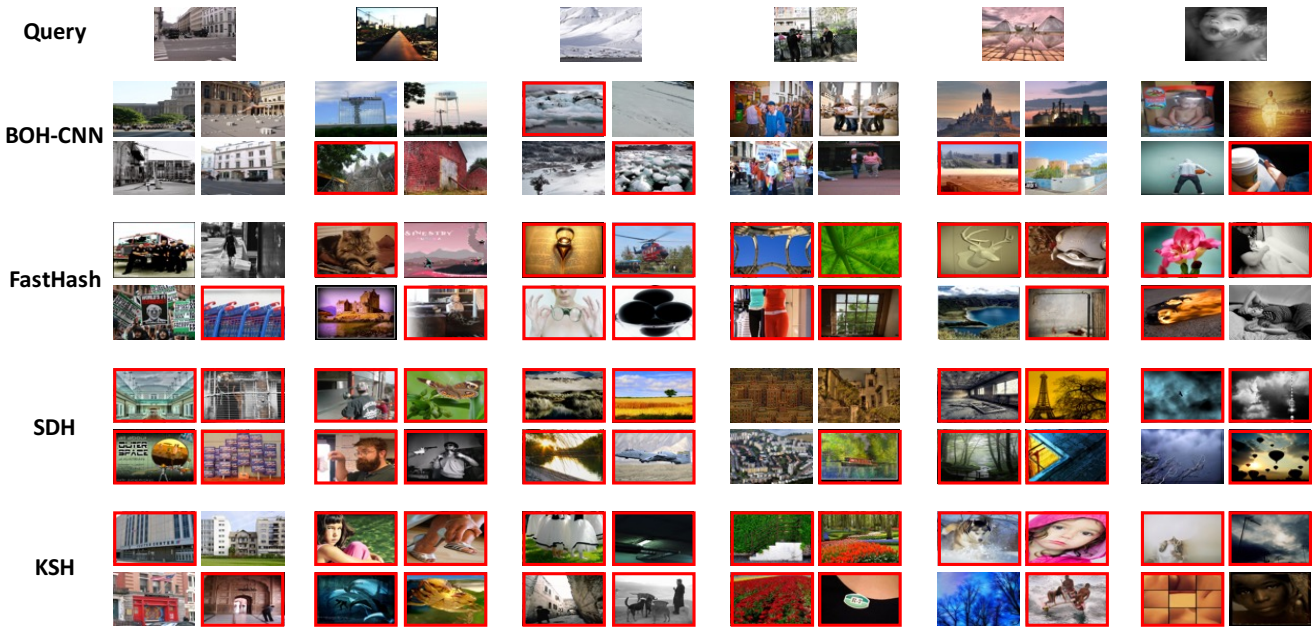

Figure 6: Top returned neighbors of BOH-CNN, FastHash, SDH and KSH, with 48 bits on NUS-WIDE. The images surrounded by red bounding boxes are negative neighbors.

Table 5: The mAP of BOH and state-of-the-art methods on CIFAR-10 and NUS-WIDE. For NUS-WIDE, we compute mAP on top-5000 returned samples. To ensure a fair comparison, we use 5K training samples for CIFAR-10.

Method	CIFAR-10				SVHN				NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
LSH [4]	0.121	0.126	0.120	0.120	0.110	0.122	0.120	0.128	0.403	0.421	0.426	0.441
SH [33]	0.131	0.135	0.133	0.130	0.140	0.138	0.141	0.140	0.433	0.426	0.426	0.423
BRE [12]	0.159	0.181	0.193	0.196	0.165	0.206	0.230	0.237	0.485	0.525	0.530	0.544
CCA-ITQ [6]	0.264	0.282	0.288	0.295	0.428	0.488	0.489	0.509	0.435	0.435	0.435	0.435
KSH [20]	0.303	0.337	0.346	0.356	0.469	0.539	0.563	0.581	0.556	0.572	0.581	0.588
SDH [27]	0.203	0.340	0.354	0.351	0.317	0.746	0.761	0.781	0.530	0.546	0.536	0.582
FastHash [17]	0.293	0.345	0.365	0.391	0.689	0.768	0.783	0.802	0.496	0.568	0.596	0.613
BOH-MLP	**0.367**	**0.398**	**0.408**	**0.420**	**0.791**	**0.803**	**0.806**	**0.818**	**0.574**	**0.605**	**0.613**	**0.623**
CNNH [34]	0.465	0.521	0.521	0.532	0.897	0.903	0.904	0.896	0.623	0.630	0.629	0.625
DNNH [15]	0.552	0.566	0.558	0.581	0.899	0.914	0.925	0.923	0.674	0.697	0.713	0.715
BOH-CNN	**0.620**	**0.633**	**0.644**	**0.657**	**0.904**	**0.923**	**0.926**	**0.927**	**0.786**	**0.834**	**0.837**	**0.855**

both mAP metric and P@R$\leqslant$2 metric, while some other methods even show accuracy drops on the P@R$\leqslant$2 metric when the number of output bits surpass a threshold.

4.4 Result Comparison on Raw Image Inputs

We then compare BOH to state-of-the-art methods on raw image inputs, which are usually based on end-to-end deep learning framework. The compared methods are CNNH [34] and DNNH [15].

Table 5 shows the comparison results of mAP on the three datasets. The bottom part of the table lists the results with raw image inputs, while the top part of the table still keeps some results with the handcraft features. The results clearly show the power of end-to-end deep learning over shallow learning. Figure 7 further illustrates P@R$\leqslant$2 over different hash bits, precision-recall curve and precision of top returned images at 48 bits. Besides, Figure 6 shows some retrieval results of several methods, with 48 bits on NUS-WIDE.

Our method produces very promising results. Compared with the deep learning based method DNNH, BOH-CNN achieves a fairly significant improvement. For instance, BOH-CNN gains an increase of 7.6%, 0.4% and 14% in terms of mAP at 48-bit outputs on CIFAR-10, SVHN and NUS-WIDE, respectively. Note that DNNH is also based on triplet ranking loss under the same CNN framework. As CNN does not explicitly generate binary codes, DNNH then presents a divide-and-encode module to approximate hash codes from the CNN outputs. The approximation is much weaker than our theoretical guaranteed solution, which is the reason that it produces much worse results.

5. CONCLUSION

This paper proposes *binary optimized hashing* (BOH), in which we prove that if the loss function is Lipschitz continuous, the mixed integral hashing learning problem can be relaxed to a bound-constrained continuous optimization problem. We introduce a surrogate objective function to approximate the relaxed objective function. We show that the

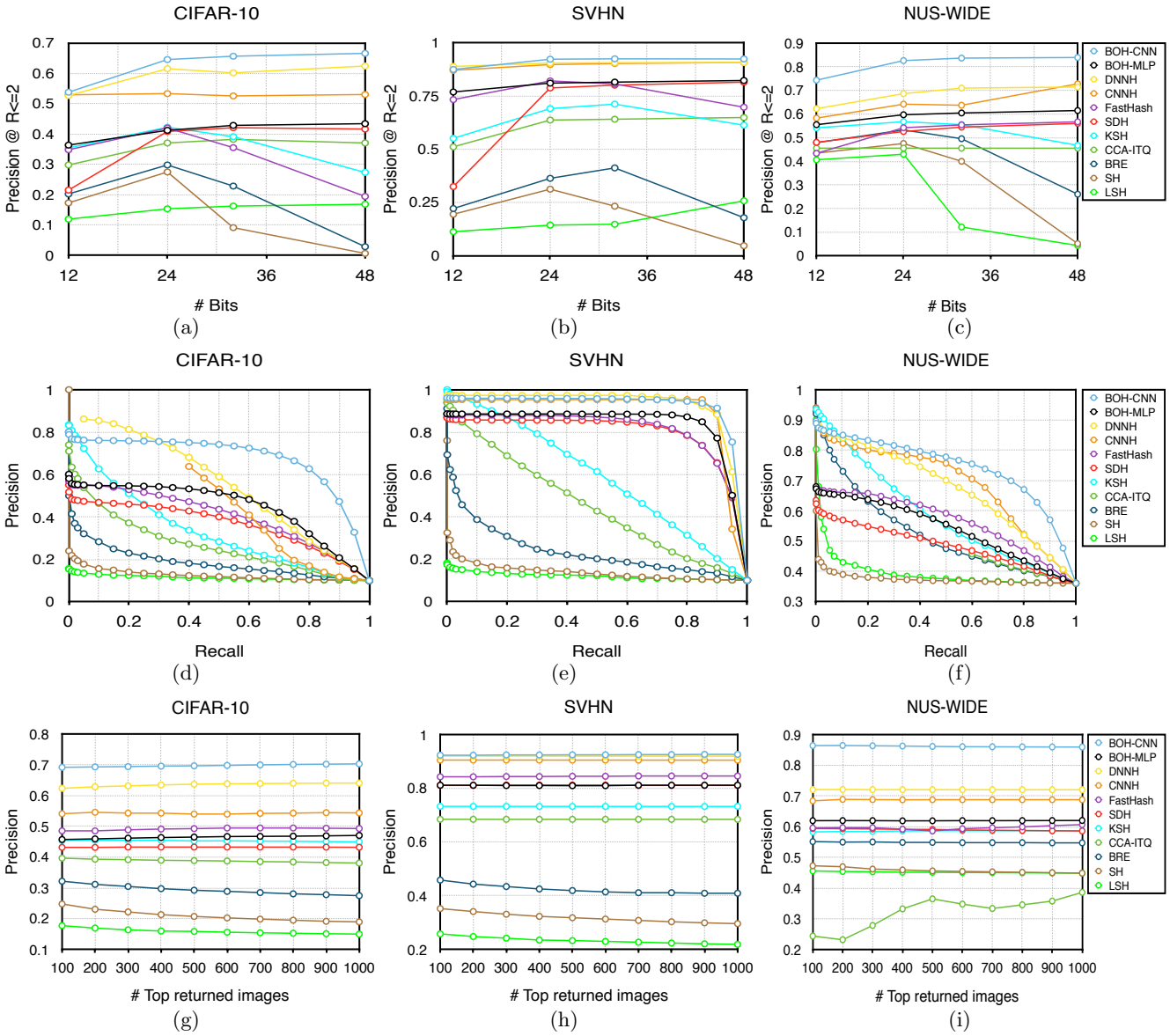

Figure 7: First row: P@R⩽2 over different numbers of bits. Second row: precision-recall curves with 48 bits. Third row: precision of top returned images. First column: on CIFAR-10. Second column: on SVHN. Third column: on NUS-WIDE.

approximation error is bounded, and bound is small when the problem is optimized.

The proposed approach is applied to learn hash codes from either handcraft feature inputs or raw image inputs in an end-to-end manner. Experiments are carried out on three benchmarks, which clearly demonstrate that our approach outperforms the state-of-the-arts (both traditional hashing techniques and deep learning based hashing techniques) with a significant margin on search accuracies. Future works are on the extension of the proposed BOH framework to other loss functions and different learning to hash problems like unsupervised hashing.

Acknowledgments

This work was supported by China's National 863 Program (#2014AA015101) and a grant from NSF China (#U1509206).

6. REFERENCES

[1] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. Nus-wide: a real-world web image database from national university of singapore. In *CIVR*, 2009.

[2] V. Erin Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou. Deep hashing for compact binary codes learning. In *CVPR*, 2015.

[3] F. Giannessi and F. Tardella. Connections between nonlinear programming and discrete optimization. In *Handbook of Combinatorial Optimization*. Springer, 1999.

[4] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[5] Y. Gong, S. Kumar, V. Verma, and S. Lazebnik. Angular quantization-based binary codes for fast similarity search. In *NIPS*, 2012.

[6] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[7] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

[8] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *CVPR*. IEEE, 2012.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[10] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[12] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, 2009.

[13] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, 2009.

[14] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *TPAMI*, 2012.

[15] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, 2015.

[16] X. Li, G. Lin, C. Shen, A. Hengel, and A. Dick. Learning hash functions using column generation. In *ICML*, 2013.

[17] G. Lin, C. Shen, Q. Shi, A. Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*, pages 1963–1970, 2014.

[18] G. Lin, C. Shen, D. Suter, and A. Hengel. A general two-step approach to learning-based hashing. In *ICCV*, pages 2552–2559, 2013.

[19] W. Liu, C. Mu, S. Kumar, and S.-F. Chang. Discrete graph hashing. In *NIPS*, pages 3419–3427, 2014.

[20] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, 2012.

[21] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *ICML*, 2011.

[22] S. Lucidi and F. Rinaldi. Exact penalty functions for nonlinear integer programming problems. *Journal of optimization theory and applications*, 2010.

[23] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, 2011.

[24] M. Norouzi, D. M. Blei, and R. R. Salakhutdinov. Hamming distance metric learning. In *NIPS*, 2012.

[25] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*, 2001.

[26] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[27] F. Shen, C. Shen, W. Liu, and H. T. Shen. Supervised discrete hashing. In *CVPR*, 2015.

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[29] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *IEEE TPAMI*, 2012.

[30] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *TPAMI*, 2012.

[31] J. Wang, W. Liu, S. Kumar, and S.-F. Chang. Learning to hash for indexing big data - a survey. *Proceeding of The IEEE*, 2016.

[32] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014.

[33] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2009.

[34] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, 2014.

[35] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, 2015.