Discretely Coding Semantic Rank Orders for Supervised Image Hashing

Li Liu[1], Ling Shao[1], Fumin Shen[2], and Mengyang Yu[3]

[1]School of Computing Science, University of East Anglia, UK
[2]Center for Future Media, University of Electronic Science and Technology of China, China
[3]Computer Vision Laboratory, ETH Zurich, Switzerland
{li.liu, ling.shao}@uea.ac.uk, fumin.shen@gmail.com, mengyangyu@gmail.com

Abstract

Learning to hash has been recognized to accomplish highly efficient storage and retrieval for large-scale visual data. Particularly, ranking-based hashing techniques have recently attracted broad research attention because ranking accuracy among the retrieved data is well explored and their objective is more applicable to realistic search tasks. However, directly optimizing discrete hash codes without continuous-relaxations on a nonlinear ranking objective is infeasible by either traditional optimization methods or even recent discrete hashing algorithms. To address this challenging issue, in this paper, we introduce a novel supervised hashing method, dubbed ***D****iscrete* ***Se****mantic* ***R****anking* ***H****ashing (****DSeRH****), which aims to directly embed semantic rank orders into binary codes. In DSeRH, a generalized Adaptive Discrete Minimization (ADM) approach is proposed to discretely optimize binary codes with the quadratic nonlinear ranking objective in an iterative manner and is guaranteed to converge quickly. Additionally, instead of using* $0/1$ *independent labels to form rank orders as in previous works, we generate the listwise rank orders from the high-level semantic word embeddings which can quantitatively capture the intrinsic correlation between different categories. We evaluate our DSeRH, coupled with both linear and deep convolutional neural network (CNN) hash functions, on three image datasets, i.e., CIFAR-10, SUN397 and ImageNet100, and the results manifest that DSeRH can outperform the state-of-the-art ranking-based hashing methods.*

1. Introduction

Hashing or binary coding is becoming increasingly popular due to its ability to help manipulate large-scale data with significant reduction on both memory and computation complexities, and is demonstrably powerful in many visual information retrieval applications. The core idea of hashing is encoding the high-dimensional feature vectors, *e.g.*, of images, documents or videos, into a set of low-dimensional binary strings in a Hamming space, which effectively preserves the original data similarities of interest.

Existing hashing methods can be roughly divided into two groups: unsupervised [2, 16, 44, 24, 23, 35, 15, 18, 19, 4, 3, 30, 9] and supervised (including semi-supervised) [39, 8, 28, 22, 33, 12, 40]. Among unsupervised hashing, the most well-known method is Locality-Sensitive Hashing (LSH) [2], which randomly projects nearby data points from the Euclidean space to a Hamming space with similar binary codes. Unlike LSH, Spectral Hashing (SpH) [44] is a data-dependent method, which aims to learn compact binary codes preserving the data similarity in the original space. Many other unsupervised hashing methods have also been proposed [23, 3, 14, 19, 4, 34, 25, 21, 20, 1, 32] and effectively applied to large-scale data retrieval tasks.

Alternatively, (semi-)supervised hashing methods optimize hash codes by involving the semantic label information in the learning phase, which usually yields a significant performance improvement. Representative methods in this group include the Semi-Supervised Hashing (SSH) [40], Linear Discriminant Analysis Hashing (LDAH) [39], Kernel-Based Supervised Hashing (KSH) [22], Binary Reconstructive Embeddings (BRE) [8], Minimal Loss Hashing (MLH) [28], Supervised Discrete Hashing (SDH) [33] and Evolutionary Compact Embedding (ECE) [17].

Most of the above supervised hashing methods generate binary codes by respecting the piecewise label [33] or pairwise similarity of data points [28, 22]. However, their objectives would be suboptimal for realistic search tasks since the ranking information, which quantitatively indicates semantic nearest neighbors of a specific query, is not fully exploited. Inspired by this, many ranking-based hashing approaches [11, 29, 5, 13, 42, 37, 43, 38, 41, 47, 46, 27, 48] have been developed. Generally, these ranking-based hashing methods try to preserve either the triplets supervision (e.g., Column Generation Hashing (CGH) [11] and Hamming Distance Metric Learning (HDML) [29]) or listwise supervision (e.g., Deep Semantic Ranking Hashing (DSRH) [47] and Ranking Preserving Hashing (RPH) [43]). Regardless of their difference, the core idea of these hashing methods is to minimize the empirical loss over the ranking violation according to the ground-truth rank order supervision.

However, the optimization for current ranking-based hashing methods is particularly challenging as minimizing a *highly nonconvex* ranking loss with *binary constraints* is generally NP-hard. To make the optimization tractable, previous methods (e.g., [43, 37]) first solve a relaxed problem by discarding the discrete constraints, then threshold (quantize) the optimized continuous embedding to form an approximate binary solution. However, this relaxation may produce less effective hash functions and thus low-quality binary codes due to accumulated quantization errors during optimization, especially on long binary codes. Recently, a few methods realize the importance of discrete optimization for hashing [21, 33], however, these methods still cannot handle the ranking loss very well. For example, the discrete cyclic coordinate descent (DCC) [33] method is designed for a particular binary quadratic program (BQP), but is unsuitable for the problem with highly nonlinear ranking loss as in this work.

To address this challenging issue, in this paper, we introduce a novel Discrete Semantic Ranking Hashing (DSeRH) method, which can be jointly optimized in an alternating manner with two associated sub-problems: discretely optimizing binary codes for encoding semantic rank orders and hash function learning. To generate high-quality hash codes in the first sub-problem, we develop a novel binary optimization approach, termed Adaptive Discrete Minimization (ADM), which can effectively optimize discrete binary codes toward the quadratic nonlinear ranking objective without any relaxation. In fact, ADM can be regarded as a generic tool to *effectively* and *efficiently* handle the discrete (thus nonsmooth and nonconvex) hashing problems with a minimization formulation of objective functions. In addition, instead of generating ranking supervision by using traditional 0/1 independent labels as in previous ranking-based methods, we adopt the high-level *semantic word embedding* learned via an NLP word-vector transformation to form the ground-truth rank order list. As such, the reasonable semantic correlation among different categories will be quantitatively measured and captured (see details in Section 2.2). The main contributions of this paper include:

- We propose a novel ranking-based hashing algorithm DSeRH, which can generate high-quality binary codes by well preserving the listwise ranking information computed from high-level semantic word embeddings.
- Different from other ranking-based hashing methods using continuous-relaxation or sign function approximation, we propose the Adaptive Discrete Minimization (ADM) algorithm to directly optimize the discrete hash codes in DSeRH with the highly nonconvex ranking loss. Moreover, ADM is not restricted to the loss in DSeRH, but can also be applied more generally to solve other binary minimization problems.
- In DSeRH, both linear and deep CNN-based hash functions are jointly learned with the binary code optimization. Extensive experiments on three benchmark datasets clearly demonstrate the superiority of the two DSeRH variants over the state-of-the-art ranking-based hashing methods.

2. Discrete Semantic Ranking Hashing

In this section, we will elaborate on our semantic ranking-based supervised hashing method DSeRH by introducing objective function formulation and then defining how to compute semantic rank orders.

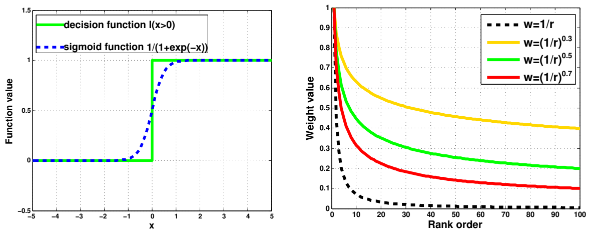
First let us define some notation. Suppose that the data matrix is $\mathbf{X} = \{\mathbf{x}_i\} \in \mathbb{R}^{d\times n}$, where d is the dimension of the features and n is the size of the training data. Due to our supervised framework, we also introduce the ground-truth matrix $\mathbf{Y} = \{\mathbf{y}_i\} \in \{0,1\}^{C\times n}$, where $\mathbf{y}_i$ is the label vector, $i = 1,\ldots,n$. We set $y_{ci} = 1$ if $\mathbf{x}_i$ belongs to class c and 0 otherwise. Additionally, for large-scale image hashing, previous works [43, 41] with listwise supervision are always computationally very complex and cause heavy memory loads during the code learning, as the length of the preserved rank orders is the size of the full training set, n. To effectively reduce the costs, in this paper, we randomly select k anchor points from $\mathbf{X}$ as $\widehat{\mathbf{X}} = \{\widehat{\mathbf{x}}_j\} \in \mathbb{R}^{d\times k}$ to form the ranking lists, where $\widehat{\mathbf{X}} \subseteq \mathbf{X}$ and also denote $\widehat{\mathbf{Y}} = \{\widehat{\mathbf{y}}_j\} \in \{0,1\}^{C\times k}$. $\mathbf{L}_{ind} \in \{0,1\}^{n\times k}$ is the indicator matrix which indexes the location of selected anchor points in $\mathbf{X}$, where each column has only one 1 corresponding to the location of $\widehat{\mathbf{x}}_j$, otherwise 0. For each training data $\mathbf{x}_i$ in $\mathbf{X}$, we can form a ranking list vector over k selected anchor points from $\widehat{\mathbf{X}}$ written as:

$$\{\mathbf{r}_i^j\}_{j=1}^k = [\mathbf{r}_i^1,\ldots,\mathbf{r}_i^k] \in \{1,\ldots,k\}^{1\times k}, \qquad (1)$$

where $\mathbf{r}_i^j$ denotes the rank order of the j-th anchor point in terms of the i-th training data.

2.1. Formulation of DSeRH

Our goal is to learn a set of binary codes $\mathbf{B} = \{\mathbf{b}_i\} \in \{-1,+1\}^{m\times n}$ for $\mathbf{X}$, where m is the length of the binary codes. Similarly, denote by $\widehat{\mathbf{B}} = \{\widehat{\mathbf{b}}_j\} \in \{-1,+1\}^{m\times k}$ the binary codes for the anchor set $\widehat{\mathbf{X}}$ and we have $\widehat{\mathbf{B}} = \mathbf{B}\mathbf{L}_{ind}$.



(a) (b)

Figure 1. (a) Sigmoid approximation for decision function $I(\cdot)$; (b) Comparison of our controlled weight penalty with previous fast-decay penalty design.

Note that, for m-bit binary vectors $\mathbf{a}, \mathbf{b}$, the Hamming distance $||\mathbf{a} - \mathbf{b}||_\mathrm{H} = \frac{1}{4}||\mathbf{a} - \mathbf{b}||^2 = \frac{1}{4}(2m - 2\mathbf{a}^\top\mathbf{b})$. To preserve the ranking information in codes, we define the rank position of anchor point $\widehat{\mathbf{x}}_j$ for the training data $\mathbf{x}_i$ as

$$\begin{aligned}\Lambda(\mathbf{x}_i, \widehat{\mathbf{x}}_j) &= \sum_{l=1}^{k} I(||\mathbf{b}_i - \widehat{\mathbf{b}}_j||_\mathrm{H} \geq ||\mathbf{b}_i - \widehat{\mathbf{b}}_l||_\mathrm{H}) \\ &= \sum_{l=1}^{k} I(\mathbf{b}_i^\top(\widehat{\mathbf{b}}_l - \widehat{\mathbf{b}}_j) \geq 0) \qquad (2) \\ &\approx \sum_{l=1}^{k} g(\mathbf{b}_i^\top(\widehat{\mathbf{b}}_l - \widehat{\mathbf{b}}_j)),\end{aligned}$$

where $I(\cdot)$ is the decision function that outputs 1 if the inside statement is true and 0 otherwise. Similar to previous work [37, 43, 38], we use the sigmoid function $g(x) = \frac{1}{1+\exp(-x)}$ to well simulate $I(\cdot)$ (as shown in Fig. 1(a)), where $\exp(\cdot)$ is the element-wise exponential operation.

Since the returned samples with high ranks should be more relevant/important to the query/user in retrieval tasks, we introduce a weight to penalize the incorrectly ranked samples at the top of the ranking list more than those with low ranking orders in the Hamming space. Thus, the semantic rank loss for anchor point $\widehat{\mathbf{x}}_j$ with the training data $\mathbf{x}_i$ can be defined as

$$\begin{aligned}L(\mathbf{x}_i, \widehat{\mathbf{x}}_j) &= \mathbf{w}_i^j \left(\sum_{l=1}^{k} g(\mathbf{b}_i^\top(\widehat{\mathbf{b}}_l - \widehat{\mathbf{b}}_j)) - \mathbf{r}_i^j \right)^2, \\ &\text{s.t. } \mathbf{b}_i \in \{-1, +1\}^{m\times 1}, \qquad (3)\end{aligned}$$

where $\mathbf{w}_i^j = (1/\mathbf{r}_i^j)^\tau$ is the weight penalty. The parameter $\tau \in (0, 1)$ effectively suppresses the fast weight decay and takes a reasonable adjustment for both top-ranked and low-ranked data (depicted in Fig. 1(b)). It is noted that a previous work [27] adopts $\mathbf{w}_i^j = 1/\mathbf{r}_i^j$ as the weight. However this will produce high weights for top-ranked data and near-zero weights for low-ranked data as shown in Fig. 1(b).

Different from most previous ranking-based hashing methods [37, 43, 38, 41, 27] which simplify the NP-hard binary code learning to a relaxed continuous-value embedding problem, we keep the binary constraints during the optimization of DSeRH. In this paper, we propose to jointly optimize the binary codes and hash functions with the following objective:

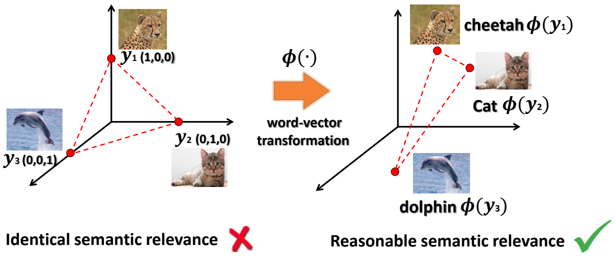


Figure 2. Independent 0/1 label vector v.s. word embedding.

$$\begin{aligned}\min_{\mathbf{B}, h(\cdot)} &\sum_{j=1}^{k}\sum_{i=1}^{n} \mathbf{w}_i^j \left(\sum_{l=1}^{k} g(\mathbf{b}_i^\top(\widehat{\mathbf{b}}_l - \widehat{\mathbf{b}}_j)) - \mathbf{r}_i^j \right)^2 \\ &+ \lambda||h(\mathbf{x}_i) - \mathbf{b}_i||_2^2, \quad \text{s.t. } \mathbf{b}_i \in \{-1, +1\}^{m\times 1}, \quad (4)\end{aligned}$$

where $h(\cdot)$ is the hash coding function and λ is the balance parameter. The regularization term can well fit the quantization error between the binary codes $\mathbf{b}_i$ and continuous hash functions $h(\mathbf{x}_i)$. With a sufficiently large λ, it will obtain the resulting hash functions $h(\mathbf{x}_i) \approx \mathbf{b}_i$. In practice, we can tolerate small differences between $\mathbf{b}_i$ and $h(\mathbf{x}_i)$. Thus, $h(\mathbf{x}_i)$ can be regarded as a surrogate of $\mathbf{b}_i$ and achieve out-of-sample coding with $\mathbf{b} = \mathrm{sign}(h(\mathbf{x}))$. We then rewrite (4) into the following more compact matrix form

$$\begin{aligned}\min_{\mathbf{B}, h(\cdot)} &\sum_{j=1}^{k} \left|\left| \mathbf{W}^j \odot \left(g(\mathbf{B}^\top(\widehat{\mathbf{B}} - \widehat{\mathbf{B}}\mathbf{z}_j\mathbf{1}^\top))\mathbf{1} - \mathbf{R}^j \right) \right|\right|_2^2 \quad (5) \\ &+ \lambda||h(\mathbf{X}) - \mathbf{B}||_\mathrm{F}^2, \quad \text{s.t. } \mathbf{B} \in \{-1, +1\}^{m\times n},\end{aligned}$$

where $||\cdot||_\mathrm{F}$ denotes the Frobenius norm, $\mathbf{W}^j = [\sqrt{\mathbf{w}_1^j}, \ldots, \sqrt{\mathbf{w}_n^j}]^\top \in \mathbb{R}^{n\times 1}$ is the weight penalty vector for $\widehat{\mathbf{x}}_j$, $\mathbf{R}^j = [\mathbf{r}_1^j, \ldots, \mathbf{r}_n^j]^\top \in \mathbb{R}^{n\times 1}$ are ground-truth rank orders for $\widehat{\mathbf{x}}_j$, $j = 1, \ldots, k$, $\mathbf{1} = [1, \ldots, 1]^\top \in \mathbb{R}^{k\times 1}$, and $\mathbf{z}_j \in \{0, 1\}^{k\times 1}$ has only one 1 at the j-th entry and 0 otherwise and "$\odot$" indicates the element-wise multiplication. Before going to the optimization of problem (5), we describe how to compute semantic rank order $\mathbf{r}$.

2.2. Semantic Rank Order

Most of the previous work defines ground-truth rank orders in either an unsupervised manner (i.e., measuring the ℓ_2 Euclidean distance) or a supervised one (i.e., counting the number of the shared semantic tags) [41, 43, 47]. However, for both of above schemes, it is very difficult to capture the meaningful correlations between categories. Particularly, for supervised ranking-based hashing methods, using

0/1 label vectors (e.g., $\mathbf{y}_i$) may implicitly make the labels independent to each other and cause unreasonably identical high-level semantic relevance (i.e., *cat* and *dolphin* will have the same relevance strength to *cheetah* as shown in Fig. 2 left). Such rank orders cannot truly reflect the intrinsic similarities/dissimilarities between different categories.

In order to cope with this problem, in this paper, the ground-truth rank orders are obtained by first embedding the independent labels into latent semantic representations. Specifically, we transfer the semantics of each category into a word embedding space [26] via the NLP word-vector toolbox[1]. Therefore, the semantic correlation among different categories can be quantitatively measured and captured. For instance, in the word embedding space, the *cat* and *cheetah* will be close to each other but far away from *dolphin* as in Fig. 2 right. Thus, the rank order can be obtained as:

$$\begin{cases} \mathbf{r}_i^u < \mathbf{r}_i^v, \text{ if } \widehat{\mathbf{y}}_u = \widehat{\mathbf{y}}_v \;\&\; \|\mathbf{x}_i - \widehat{\mathbf{x}}_u\|_2^2 < \|\mathbf{x}_i - \widehat{\mathbf{x}}_v\|_2^2, \\ \mathbf{r}_i^u > \mathbf{r}_i^v, \text{ if } \widehat{\mathbf{y}}_u = \widehat{\mathbf{y}}_v \;\&\; \|\mathbf{x}_i - \widehat{\mathbf{x}}_u\|_2^2 > \|\mathbf{x}_i - \widehat{\mathbf{x}}_v\|_2^2, \\ \mathbf{r}_i^u < \mathbf{r}_i^v, \text{ if } \widehat{\mathbf{y}}_u \neq \widehat{\mathbf{y}}_v \;\&\; \mathrm{sim}(\phi(\mathbf{y}_i), \phi(\widehat{\mathbf{y}}_u)) > \mathrm{sim}(\phi(\mathbf{y}_i), \phi(\widehat{\mathbf{y}}_v)), \\ \mathbf{r}_i^u > \mathbf{r}_i^v, \text{ if } \widehat{\mathbf{y}}_u \neq \widehat{\mathbf{y}}_v \;\&\; \mathrm{sim}(\phi(\mathbf{y}_i), \phi(\widehat{\mathbf{y}}_u)) < \mathrm{sim}(\phi(\mathbf{y}_i), \phi(\widehat{\mathbf{y}}_v)), \end{cases} \tag{6}$$

where $\phi(\cdot)$ is the word-vector transformation and $\mathrm{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|}$ is the cosine similarity, $i = 1, \ldots, n$ and $u, v = 1, \ldots, k$. Besides, another advantage of our ranking scheme is that Eq. (6) can handle the ranking of *single-labeled data* which is basically intractable for previous supervised ranking hashing techniques [43, 47, 41]. In the next section, we will elaborate on the optimization of our DSeRH method.

3. Alternating Optimization with ADM

The joint problem of learning both hash codes $\mathbf{B}$ and hash function $h(\cdot)$ in (5) with the discrete constraint is highly non-convex and non-smooth. To effectively solve this problem, we alternatingly optimize (5) with each of its two associated sub-problems: binary code optimization and hash function learning.

3.1. Binary Code Optimization

By fixing $h(\mathbf{X})$, problem (5) w.r.t. $\mathbf{B}$ is rewritten as

$$\min_{\mathbf{B}} f(\mathbf{B}) := \sum_{j=1}^{k} \|\mathbf{W}^j \odot \big(g(\mathbf{B}^\top \mathbf{B} \mathbf{D}_j)\mathbf{1}\big) - \mathbf{R}^j)\|_2^2 \tag{7}$$
$$+ \lambda \|h(\mathbf{X}) - \mathbf{B}\|_{\mathrm{F}}^2 \quad \text{s.t. } \mathbf{B} \in \{-1, +1\}^{m \times n},$$

where $\mathbf{D}_j = \mathbf{L}_{ind}(\mathbf{I} - \mathbf{z}_j \mathbf{1}^\top)$ and $\mathbf{I}$ is the identity matrix. Considering the discrete constraints, the above problem is NP-hard. To solve this problem, in this paper, we introduce a novel iterative optimization algorithm, termed Adaptive Discrete Minimization (ADM), which can be regarded as a generalized solution to effectively solve binary discrete optimization of the hashing problem.

Algorithm 1 Adaptive Discrete Minimization (ADM)

```
Input: Ranking list $\{\mathbf{R}^j\}_{j=1}^k$; initial hash function $h(\cdot)$, $\mathbf{B} \in \{-1,+1\}^{m \times n}$ and selection ratio $\varphi^{(1)}$; maximum iteration number $T$; parameter $\lambda$.
Output: Optimized binary codes $\mathbf{B}$.
  for $t = 1, \ldots, T$ do
    Generate $S^{(t)}$ with selection ratio $\varphi^{(t)}$;
    $\mathbf{B}^{(t+1)} = \mathcal{F}(\mathrm{sign}(-\nabla f(\mathbf{B}^{(t)})), \mathbf{B}^{(t)}, S^{(t)})$;
    if $f(\mathbf{B}^{(t+1)}) > f(\mathbf{B}^{(t)})$ then
      repeat
        $\varphi^{(t)} \leftarrow 0.5\,\varphi^{(t)}$ and generate $S^{(t)}$ with $\varphi^{(t)}$;
        $\mathbf{B}^{(t+1)} = \mathcal{F}(\mathrm{sign}(-\nabla f(\mathbf{B}^{(t)})), \mathbf{B}^{(t)}, S^{(t)})$;
      until $f(\mathbf{B}^{(t+1)}) \le f(\mathbf{B}^{(t)})$
    end if
    if $\varphi^{(t)} < \frac{1}{mn}$ then
      Exit and Return the optimized $\mathbf{B} = \mathbf{B}^{(t+1)}$;
    end if
    Update $\varphi^{(t+1)} \leftarrow \min(1.2\,\varphi^{(t)}, 1)$;
  end for
```

Given $\mathbf{B}^{(t)}$ in the t-th iteration, $\mathbf{B}^{(t+1)}$ is calculated as follows. Define the tangent hyperplane on $\mathbf{B}^{(t)}$: $\hat{f}_t(\mathbf{B}) = f(\mathbf{B}^{(t)}) + \langle \nabla f(\mathbf{B}^{(t)}), \mathbf{B} - \mathbf{B}^{(t)} \rangle$. Then $\hat{f}_t(\mathbf{B})$ is the *linearization* of $f(\mathbf{B})$ on $\mathbf{B}^{(t)}$ and the gradient $-\nabla f(\mathbf{B}^{(t)})$ is the descending direction of f. Then we use the following update rule:

$$\mathbf{B}^{(t+1)} = \mathcal{F}(\mathrm{sign}(-\nabla f(\mathbf{B}^{(t)})), \mathbf{B}^{(t)}, S^{(t)}). \tag{8}$$

Here $\mathcal{F}$ is an element-wise function:

$$\Big(\mathcal{F}(\mathbf{M}, \mathbf{N}, S^{(t)})\Big)_{p,q} = \begin{cases} \mathbf{M}_{p,q} & \text{if } (p,q) \in S^{(t)}, \\ \mathbf{N}_{p,q} & \text{otherwise}, \end{cases} \tag{9}$$

where (p, q) is the matrix coordinate. The set $S^{(t)}$ is generated by randomly selecting items from the set $\{(p,q) | \mathbf{M}_{p,q} \neq \mathbf{N}_{p,q}, \mathbf{M}_{p,q} \neq 0\}$, where $\mathbf{M} = \mathrm{sign}(-\nabla f(\mathbf{B}^{(t)}))$ and $\mathbf{N} = \mathbf{B}^{(t)}$ for our problem. Denoting $|S|$ as the cardinal number of S, we set a selection ratio $\varphi^{(t)} = \frac{|S^{(t)}|}{mn} \in (0, 1)$, which can be adaptively adjusted as discussed later.

We straightforwardly summarize our ADM as: in the $(t+1)$-th iteration, the bits of $\mathbf{B}^{(t)}$ at positions $\{(p,q)\} \in S^{(t)}$ are replaced with $-(\nabla f(\mathbf{B}^{(t)}))_{p,q}$. Based on the definition of S, any selected $(p,q) \in S^{(t)}$ must guarantee $\mathrm{sign}(-\nabla f(\mathbf{B}^{(t)}))_{p,q} \neq \mathbf{B}^{(t)}$ and $\nabla f(\mathbf{B}^{(t)})_{p,q} \neq 0$ to ensure effective updating.

To achieve above ADM, we need to compute the gradient of f w.r.t. $\mathbf{B}$. Let us denote by $f(\mathbf{B})$ the objective of (7) and

[1] https://code.google.com/archive/p/word2vec/. The model is trained from the first billion characters from Wikipedia and each word can be interpreted into a 1000-d semantic word-vector.

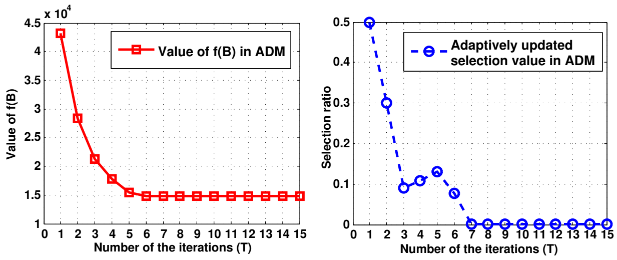


Figure 3. An illustration of convergence of ADM (left) and the adaptively updated selection ratio φ during iterations (right).

note that $\nabla g(x) = g(x)(1 - g(x))$, we then obtain

$$\nabla f(\mathbf{B}) = -2\mathbf{B}\left(\frac{\partial f}{\partial(-\mathbf{B}^\top\mathbf{B})}\right) + 2\lambda(\mathbf{B} - h(\mathbf{X})) \tag{10}$$

$$= 4\sum_{j=1}^{k}\mathbf{B}\Big(\Big(\Big(\widetilde{\mathbf{W}}^j \odot \Big(g(\mathbf{B}^T\mathbf{B}\mathbf{D}_j)\mathbf{1} - \mathbf{R}^j\Big)\Big)\mathbf{1}^\top\Big)$$

$$\odot\, g(\mathbf{B}^\top\mathbf{B}\mathbf{D}_j) \odot (\tilde{\mathbf{1}} - g(\mathbf{B}^T\mathbf{B}\mathbf{D}_j))\Big)\mathbf{D}_j^\top + 2\lambda(\mathbf{B} - h(\mathbf{X})),$$

where $\widetilde{\mathbf{W}}^j = [\mathbf{w}_1^j, \dots, \mathbf{w}_n^j]^\top$ and $\tilde{\mathbf{1}} = \{1\}^{n\times k}$. The algorithm of ADM is illustrated in Algorithm 1 with the updating selection ratio $\varphi^{(t)}$ during each iteration:

$$\varphi^{(t+1)} \leftarrow \begin{cases} 0.5\,\varphi^{(t)} & \text{if } f(\mathbf{B}^{(t+1)}) > f(\mathbf{B}^{(t)}), \\ \min(1.2\,\varphi^{(t)}, 1) & \text{otherwise.} \end{cases} \tag{11}$$

If $\varphi^{(t)} < \frac{1}{mn}$ (i.e, $S^{(t)} = \emptyset$), $\mathbf{B}^{(t+1)} = \mathcal{F}(\text{sign}(-\nabla f(\mathbf{B}^{(t)})), \mathbf{B}^{(t)}, \emptyset) = \mathbf{B}^{(t)}$ which indicates the convergence criterion of Algorithm 1 and $\mathbf{B}^{(t+1)}$ is a local optimum of DSeRH. Fig. 3 illustrates the convergence of ADM with the adaptively updated φ during iterations.

Theorem 1. *Let $\{\mathbf{B}^{(t)}\}$ be generated by the updating rule of Algorithm 1, then $\{f(\mathbf{B}^{(t)})\}$ is monotonically non-increasing, i.e., $f(\mathbf{B}^{(t+1)}) \leq f(\mathbf{B}^{(t)})$, and $\{\mathbf{B}^{(t)}\}$ converges.*

Please refer to our *supplementary document* for the proof of Theorem 1. The main computational complexity of ADM comes from calculating $\nabla f(\mathbf{B})$ of each iteration in $\mathcal{O}((nmk)^2)$. In practice, our algorithm usually converges in $T = 5 \sim 10$ iterations. It is noted that although n is a relatively large value in our task, the most expensive operation in Eq. (10) is the matrix product, ensuring the complexity of ADM is still acceptable. Additionally, during ADM optimization, the maximum storage complexity is $\mathcal{O}(nm)$ due to the property of matrix product, which leads to the light and efficient computation in practice.

Remark. Although the fundamental idea of ADM has small similarities to the one in DGH [21], the majorization-minimization scheme used in [21] formulates a surrogate of the loss function with its linearization at each iteration. The surrogate is guaranteed to be an upper bound of the minimizing loss due to its concavity. However, this optimization scheme is not applied to our problem, since the objective in Eq. (7) is highly non-concave and non-convex. Consequently, the condition $\hat{f}_t(\mathbf{B}) > f(\mathbf{B})$ used in [21] does not always hold for our objective. Besides, directly using the algorithm in [21] is also ineffective for our minimization problem since $f(\mathbf{B}) = f(-\mathbf{B})$ if $f(\mathbf{B})$ is an *even-order* function on $\mathbf{B}$. That is, updating with $\mathbf{B}^{(t+1)} = -\nabla f(\mathbf{B}^{(t)})$ in each step may conversely increase the loss. To remedy this problem, we design our ADM updating rules by setting a selection ratio to determine which subset of bits in B will be updated. The recently proposed DCC algorithm [33] is also inapplicable for (7), since DCC-optimization is only designed for a special binary quadratic program (BQP) and cannot handle the quadratically nonlinear ranking loss like ours.

Algorithm 2 Discrete Semantic Ranking Hashing (DSeRH)

```
Input: Training image data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$; code length $m$; number of anchor points $k$; parameter $\tau$.
Output: Binary codes $\mathbf{B}$ and hash function $h(\cdot)$.
Randomly select $k$ samples $\{\widehat{\mathbf{x}}_j, \widehat{\mathbf{y}}_j\}_{j=1}^k$ from the training data and get indicator matrix $\mathbf{L}_{ind} \in \{0,1\}^{n\times k}$.
Compute the ranking list $\{\mathbf{R}^j\}_{j=1}^k \in \{1,\dots,k\}^{n\times 1}$ and penalty weights $\{\mathbf{W}^j\}_{j=1}^k \in \mathbb{R}^{n\times 1}$.
Randomly initialize $\mathbf{B}$ with a $\{-1,+1\}^{m\times n}$ matrix.
repeat
    B-step: Update $\mathbf{B}$ with ADM in Algorithm 1.
    $h(\cdot)$-step: Update $h(\cdot)$ with a linear hash function as Eq. (13) or a deep hash function via VGG-19 CNN.
until convergence
return $\mathbf{b} = \text{sign}(h(\mathbf{x})), \forall\mathbf{x}$
```

3.2. Hash Function Learning

To compute the hash function $h(\cdot)$, we fix $\mathbf{B}$ and the problem (5) shrinks to

$$\min_{h(\cdot)} ||h(\mathbf{X}) - \mathbf{B}||_{\text{F}}^2. \tag{12}$$

Learning the linear hash function: We denote $h(\mathbf{X}) = \mathbf{P}\mathbf{X}$, where $\mathbf{P} \in \mathbb{R}^{m\times d}$ is the linear projection. Thus we can easily obtain the optimal $\mathbf{P}$ via linear regression as

$$\mathbf{P} = \mathbf{B}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top). \tag{13}$$

Learning the deep hash function: Inspired by recent success on deep hashing (e.g., [47, 46, 10]), in this paper, we also learn a deep hash function for directly mapping the raw input images into high-quality hash codes. Specifically, the pre-trained VGG-19 CNN model [36] on ImageNet [31] is fine-tuned with a *Euclidean loss* layer with the supervision from $\mathbf{B}$ obtained by Algorithm 1.

DSeRH is optimized by alternately updating $\mathbf{B}$ and $h(\cdot)$ until convergence. The overall algorithm of DSeRH is sum-

Table 1. Retrieval result comparison (MAP, precision of top 100 samples, NDCG of top rank 50 with Hamming distance ranking and training/test time cost) on **CIFAR-10** dataset using 512-d GIST features. The best performances are displayed in bold with **blue** color.

Methods	CIFAR-10											
	MAP				**Precision@top100**				**NDCG@rank50**			
# Bits	$m = 16$	$m = 32$	$m = 64$	$m = 128$	$m = 16$	$m = 32$	$m = 64$	$m = 128$	$m = 16$	$m = 32$	$m = 64$	$m = 128$
LSH [2]	0.1199	0.1382	0.1447	0.1496	0.1401	0.1637	0.1954	0.2166	0.1021	0.1063	0.1080	0.1114
ITQ [3]	0.1610	0.1724	0.1793	0.1902	0.2011	0.2648	0.3053	0.3216	0.2050	0.2223	0.2374	0.2395
SpH [44]	0.1488	0.1462	0.1450	0.1413	0.1787	0.2011	0.2389	0.2527	0.1411	0.1546	0.1570	0.1603
HDML [29]	0.2840	0.3313	0.3550	0.3671	0.3420	0.3932	0.4114	0.4210	0.2313	0.2504	0.2631	0.2700
RSH† [41]	0.2154	0.2854	0.3112	0.3216	0.2420	0.3153	0.3464	0.3630	0.1919	0.2270	0.2325	0.2381
CGH [11]	0.3336	0.3696	0.4034	0.4201	0.3770	0.4315	0.4441	0.4570	0.2512	0.2644	0.2698	0.2685
RPH† [43]	0.3296	0.3512	0.3877	0.4004	0.3510	0.4017	0.4255	0.4341	0.2564	0.2611	0.2706	0.2813
Top-RSBC [37]	0.3651	0.4081	0.4195	0.4441	0.3920	0.4496	0.4568	0.4773	**0.2688**	0.2745	0.2801	0.2793
DSeRH-L†	**0.4014**	**0.4373**	**0.4508**	**0.4819**	**0.4554**	**0.4732**	**0.5011**	**0.5353**	0.2612	**0.2902**	**0.3088**	**0.3175**

The "†" indicates the ground-truth ranking list computed via Eq. (6) for this method.

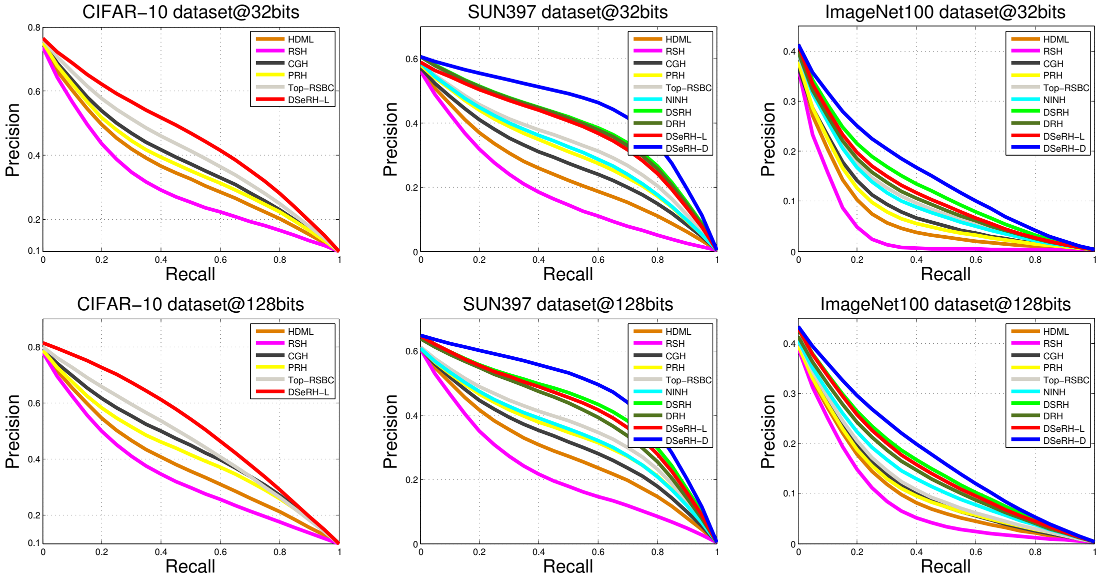


Figure 4. Comparison of precision-recall curves on three datasets with 32 bits and 128 bits codes, respectively.

marized in Algorithm 2. In fact, either a local or global minimum is achieved in each sub-problem, ensuring the overall objective in Eq. (5) is lower-bounded. Thus, the convergence of the alternating optimization in Algorithm 2 is guaranteed. Such a alternating scheme is also widely adopted by recent hashing methods, e.g., [33, 21, 3].

To avoid confusion in later experiments, we further denote by DSeRH-L and DSeRH-D the linear model and deep CNN used as the hash function, respectively. In the next section, we will extensively evaluate DSeRH and compare it with the state-of-the-art ranking-based methods.

4. Experiments

In this section, we evaluate our methods on three realistic large-scale datasets: CIFAR-10 [7], SUN397 [45] and ImageNet100 [31], for image similarity retrieval tasks. **CIFAR-10** consists of 60,000 tiny images with 32 × 32 resolution distributed evenly over 10 classes. The dataset is split into a query set of 1000 samples and the remaining samples are used as the training set as well as the retrieval gallery set. The **SUN397** dataset has 108,754 images from 397 well-sampled categories. In this experiment, we only select 100 images from each of 34 largest categories (at least 550 images per category) to construct our query set. The remaining images from these 34 categories are the training set. **ImageNet100** is a subset of [31] containing the most frequent 100 object categories and each category has no less than 1,600 images. We further choose 100 and 500 images from each category, respectively, to form the query set and the training set. All of the images excluding query sets for both SUN397 and ImageNet100 datasets are also treated as the retrieval gallery. We further extract a 512-d GIST feature vector from the images in the CIFAR-10 dataset. For both SUN397 and Imagenet100, each image in them is represented by a 4096-d deep feature computed from the same pre-trained VGG-19 network as used in ours.

4.1. Compared Methods and Evaluation Settings

In our experiments, we compare the proposed DSeRH-L and DSeRH-D with 11 hashing methods including 3 con-

Table 2. Retrieval result comparison (MAP, precision of top 100 samples, NDCG of top rank 50 with Hamming distance ranking and training/test time cost) on **SUN397** dataset using 4096-d CNN features from VGG19 *fc7*. The best performances among non-deep techniques and deep techniques are displayed in bold with **blue** and **black** colors, respectively. (Better to view in color)

Methods	**SUN397**											
	MAP				**Precision@top100**				**NDCG@rank50**			
# Bits	$m = 16$	$m = 32$	$m = 64$	$m = 128$	$m = 16$	$m = 32$	$m = 64$	$m = 128$	$m = 16$	$m = 32$	$m = 64$	$m = 128$
LSH [2]	0.0151	0.0184	0.0203	0.0254	0.0198	0.0254	0.0328	0.0701	0.1510	0.1633	0.1704	0.1778
ITQ [3]	0.1297	0.1630	0.1792	0.1830	0.2244	0.2617	0.3288	0.3346	0.2323	0.2454	0.2496	0.2504
SpH [44]	0.0701	0.0832	0.1014	0.1139	0.1697	0.2014	0.2390	0.2762	0.2114	0.2196	0.2263	0.2312
HDML [29]	0.2217	0.2534	0.2790	0.2901	0.3955	0.4267	0.4514	0.4736	0.2440	0.2713	0.2887	0.2865
RSH† [41]	0.1691	0.2052	0.2202	0.2211	0.3186	0.3552	0.3817	0.3926	0.2325	0.2456	0.2517	0.2600
CGH [11]	0.2527	0.2878	0.3106	0.3311	0.4011	0.4302	0.4417	0.4823	0.2714	0.2919	0.3052	0.3173
RPH† [43]	0.2749	0.3100	0.3392	0.3428	0.4171	0.4330	0.4619	0.4880	0.2904	0.3217	0.3304	0.3350
Top-RSBC [37]	0.3226	0.3404	0.3628	0.3710	**0.4330**	0.4516	0.4882	0.5074	0.3006	0.3123	0.3294	0.3361
DSeRH-L†	**0.3407**	**0.3806**	**0.3913**	**0.4028**	0.4302	**0.4843**	**0.5124**	**0.5281**	**0.3101**	**0.3308**	**0.3512**	**0.3600**
NINH [10]	0.3105	0.3259	0.3431	0.3567	0.4264	0.4408	0.4740	0.4885	0.2866	0.3060	0.3211	0.3306
DSRH† [47]	0.3501	0.3918	0.4052	0.4107	0.4412	0.4928	0.5230	0.5311	0.3108	0.3343	0.3565	0.3644
DRH [46]	0.3480	0.3825	0.3955	0.4008	0.4381	0.4900	0.5145	0.5230	0.3002	0.3225	0.3523	0.3610
DSeRH-D†	**0.3883**	**0.4337**	**0.4450**	**0.4511**	**0.4726**	**0.5373**	**0.5547**	**0.5600**	**0.3401**	**0.3622**	**0.3720**	**0.3814**

The "†" indicates the ground-truth ranking list computed via Eq. (6) for this method. The "$*$" indicates the time cost computed via GPU, otherwise denotes the CPU time.

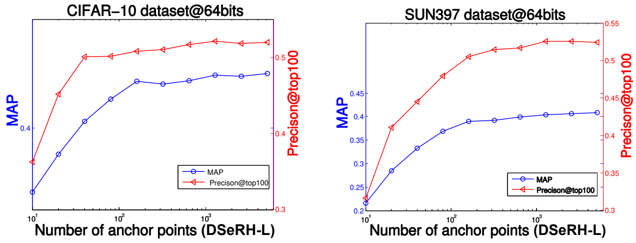


Figure 5. Number of anchor points k on both CIFAR-10 and SUN397 using DSeRH-L.

ventional non-ranking driven methods: LSH, ITQ and SpH; 5 ranking-based hashing methods: CGH, HDML, Top-RSBC with SGD, RPH and Ranking Supervised Hashing (RSH); 2 deep hashing methods with ranking loss: Deep Semantic Ranking Hashing (DSRH) and Deep Ranking Hashing (DRH); and a general deep hashing method Network in Network Hashing (NINH). Besides LSH, SpH and ITQ, other compared methods are all supervised. We implement RPH, Top-RSBC and DRH ourselves due to unavailable codes, and use the publicly provided codes for other compared methods. *It is noteworthy that RPH, RSH and DSRH are originally designed to generate the ranking lists by counting the number of commonly shared semantic tags for multi-label data. However, the single-label datasets (i.e., CIFAR-10, SUN397 and ImageNet100) used here make these three methods inapplicable. Thus, in our experiments, we use the same rules as our Eq. (6) to generate ranking information for RPH, RSH and DSRH.* Following the same experimental setting, all parameters used in the above methods are well tuned by cross-validation on the training set to guarantee a fair comparison with our proposed methods.

For our DSeRH-L/DSeRH-D, the number of the anchor points k is set as 200, the maximum iteration T of ADM is fixed at 10 and the initial ratio φ in ADM is 0.5. The balance parameter λ for each dataset is selected as the value of the range $[10^{-5}, 10^2]$ which yields the best performance by cross-validation. The same procedure is also applied to select τ from $(0, 1)$. For DSeRH-D, the raw images are used as the input instead of the extracted features $\mathbf{X}$. Particularly, we apply the Caffe [6] deep framework on a pre-trained VGG-19 net with the initial fine-tune learning rate $\alpha = 0.001$ ($0.3\alpha \rightarrow \alpha$ with the step size of 3K iterations) and the mini-batch size being 64 images. DSeRH-D is well trained on a workstation configured with two K80 GPUs. Note that we only evaluate DSeRH-L on the CIFAR-10 dataset with GIST features since tiny images are not suitable for the CNN feature learning used in DSeRH-D.

All methods are evaluated with different code lengths (i.e., 16, 32, 64 and 128) under four possible evaluation metrics: Mean Average Precision (MAP), precision@top100, Normalized Discounted Cumulative Gain (NDCG@rank50), and precision-recall curve. Considering the uncertainty of anchor selection, all provided results using our methods are the average of 5 runs.

4.2. Experimental Results

Result comparison: In Tables 1, 2 and 3, we first illustrate the retrieval results of different methods with MAP, Precision@top100 and NDGC@rank50 on the CIFAR-10, SUN397 and ImageNet100 datasets, respectively. Generally, results on the CIFAR-10 dataset are better than those on the other two datasets since fewer categories and small intra-class variation make the data more discriminative for large-scale searching. Furthermore, the tendencies of MAP and Precision@top100 are always consistent when varying the code length. However, the NDCG@rank50 presents a different inclination. In terms of MAP and Precision@top100 on all three datasets, unsupervised methods LSH and SpH achieve lower accuracies than other ranking-supervised methods, while ITQ can lead to competitive performance with RSH. Top-RSBC always produces superior

Table 3. Retrieval result comparison (MAP, precision of top 100 samples, NDCG of top rank 50 with Hamming distance ranking and training/test time cost) on **ImageNet100** dataset using 4096-d CNN features from VGG19 *fc7*. The best performances among non-deep techniques and deep techniques are displayed in bold with **blue** and **black** colors, respectively. (Better to view in color)

Methods	**ImageNet100**											
	MAP				**Precision@top100**				**NDCG@rank50**			
# Bits	$m = 16$	$m = 32$	$m = 64$	$m = 128$	$m = 16$	$m = 32$	$m = 64$	$m = 128$	$m = 16$	$m = 32$	$m = 64$	$m = 128$
LSH [2]	0.0072	0.0153	0.0198	0.0249	0.0101	0.0234	0.0308	0.0901	0.0891	0.1030	0.1154	0.1220
ITQ [3]	0.0612	0.0762	0.1051	0.1184	0.1473	0.2264	0.3083	0.3577	0.1653	0.1826	0.1920	0.2031
SpH [44]	0.0215	0.0383	0.0601	0.0803	0.0975	0.1246	0.2233	0.3203	0.1443	0.1736	0.1892	0.1873
HDML [29]	0.0533	0.0821	0.0991	0.1114	0.1362	0.2357	0.2952	0.3339	0.1716	0.2150	0.2243	0.2279
RSH† [41]	0.0473	0.0682	0.0765	0.0898	0.1344	0.2024	0.2471	0.3041	0.1882	0.1994	0.2193	0.2210
CGH [11]	0.0836	0.0911	0.1103	0.1198	0.1528	0.2241	0.3100	0.3458	**0.2124**	0.2217	0.2403	0.2508
RPH† [43]	0.0603	0.0855	0.1009	0.1096	0.1388	0.2484	0.3047	0.3415	0.2025	0.2234	0.2355	0.2487
Top-RSBC [37]	0.1019	0.1153	0.1248	0.1270	0.1819	0.2538	0.3339	0.3678	0.1914	0.2252	0.2379	0.2512
DSeRH-L†	**0.1115**	**0.1298**	**0.1403**	**0.1485**	**0.2073**	**0.2788**	**0.3530**	**0.3892**	0.2023	**0.2359**	**0.2552**	**0.2708**
NINH [10]	0.1034	0.1112	0.1232	0.1287	0.1754	0.2561	0.3209	0.3588	0.1901	0.2145	0.2306	0.2427
DSRH† [47]	**0.1179**	0.1315	0.1387	0.1497	0.2115	0.2779	0.3451	0.3901	0.2210	0.2421	0.2504	0.2751
DRH [46]	0.0914	0.1201	0.1343	0.1432	0.1801	0.2655	0.3350	0.3731	0.2031	0.2217	0.2452	0.2660
DSeRH-D†	0.1146	**0.1453**	**0.1612**	**0.1700**	**0.2312**	**0.2904**	**0.3734**	**0.4145**	**0.2347**	**0.2689**	**0.2805**	**0.2911**

The "†" indicates the ground-truth ranking list computed via Eq. (6) for this method. The "$*$" indicates the time cost computed via GPU, otherwise denotes the CPU time.

performance to HDML, RPH and CGH since Top-RSBC penalizes the mistakes at the top of the ranking list, however, other mentioned methods treat the mistakes equally. Our DSeRH-L can essentially outperform all compared methods along different code lengths on all datasets. In terms of NDCG@rank50, our proposed methods achieve competitive results with RPH and Top-RSBC but have significantly better performance than other compared methods, since the DSeRH-L objective is designed with a more reasonable weighted penalty for rank preserving in our formulation (see in Fig. 1(b)).

We also compare our DSeRH-D with three other deep hashing techniques (i.e., NINH, DSRH and DRH) on SUN397 and ImageNet100 datasets and the deep model-based methods can only achieve slightly (less than 5%) better performance than non-deep ones with regard to all three evaluation metrics. The possible reason is that, for non-deep methods, the traditional single-layer linear hash function combined with pre-trained VGG-19 *fc7* features can produce similar results as directly learning hash codes via deep nets. This is because during fine-tuning of deep methods with hashing objectives, a limited number of parameters are updated in early convolutional layers, while most parameters are learned between the last fully connected layer and the supervision layer, similar to the single-layer linear hash function in isolation. Detailed results can be seen in tables. Fig. 4 also presents the precision-recall curves of all ranking-based methods on three datasets, respectively. From all these figures, we can discover that, DSeRH-D outperforms other ranking-based hashing methods by comparing the retrieval precision and the Area Under the Curve (AUC).

Parameter sensitivity analysis: We illustrate the analysis of the anchor number k for our method in Fig. 5. We can observe that the MAP and Precision@top100 curves become approximately stable when k increases on both CIFAR-10 and SUN397 datasets, which indicates that DSeRH-L can lead to relatively robust performance with $k \geq 200$. In Fig. 6, we also compare the results on CIFAR-10 between: 1) using discrete optimization ADM (i.e., proposed DSeRH-L) and 2) the relaxed one by replacing $\mathbf{B}$ in Eq. (5) with $\mathbf{PX}$ and then optimizing over $\mathbf{P}$ via SGD like Top-RSBC [37] and RPH [43]. The performance gaps show the effectiveness of the proposed discrete ranking preserving framework with its solution ADM.

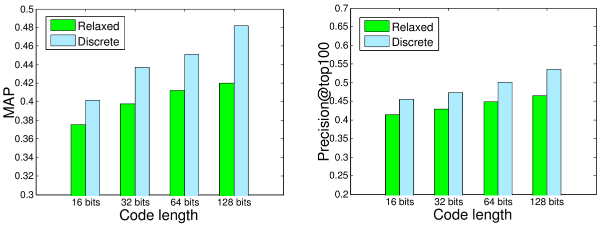


Figure 6. Comparison of discrete optimization and relaxation.

5. Conclusion

This paper proposed a novel Discrete Semantic Ranking Hashing (DSeRH) method to effectively encode the semantic rank order into binary codes. For the key binary code optimization sub-problem, we developed a novel Adaptive Discrete Minimization (ADM) approach, which was able to directly optimize the hashing problem with binary constraints. DSeRH supported hash functions of both linear and deep CNN models. Extensive comparisons with several state-of-the-art ranking-based hashing algorithms validated the efficacy of the proposed DSeRH on large-scale visual retrieval.

References

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

[2] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[3] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *T-PAMI*, 35(12):2916–2929, 2013.

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

[5] T. Ji, X. Liu, C. Deng, L. Huang, and B. Lang. Query-adaptive hash code ranking for fast nearest neighbor search. In *ACM MM*, 2014.

[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM MM*, 2014.

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

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

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

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

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

[12] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*, 2014.

[13] G. Lin, C. Shen, and J. Wu. Optimizing ranking measures for compact binary code learning. In *ECCV*, 2014.

[14] K. Lin, J. Lu, C.-S. Chen, and J. Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *CVPR*, 2016.

[15] Y. Lin, R. Jin, D. Cai, S. Yan, and X. Li. Compressed hashing. In *CVPR*, 2013.

[16] L. Liu and L. Shao. Sequential compact code learning for unsupervised image hashing. *IEEE transactions on neural networks and learning systems*, 27(12):2526–2536, 2016.

[17] L. Liu, L. Shao, and X. Li. Evolutionary compact embedding for large-scale image classification. *Information Sciences*, 316:567–581, 2015.

[18] L. Liu, M. Yu, and L. Shao. Multiview alignment hashing for efficient image search. *IEEE Transactions on image processing*, 24(3):956–966, 2015.

[19] L. Liu, M. Yu, and L. Shao. Latent struture preserving hashing. *IJCV*, 2016.

[20] L. Liu, M. Yu, and L. Shao. Learning short binary codes for large-scale image retrieval. *IEEE Transactions on Image Processing*, 26(3):1289–1299, 2017.

[21] W. Liu, C. Mu, S. Kumar, and S.-F. Chang. Discrete graph hashing. In *NIPS*, 2014.

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

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

[24] J. Lu, V. Erin Liong, and J. Zhou. Simultaneous local binary feature learning and encoding for face recognition. In *ICCV*, 2015.

[25] J. Lu, V. E. Liong, X. Zhou, and J. Zhou. Learning compact binary face descriptor for face recognition. *IEEE T-PAMI*, 37(10):2041–2056, 2015.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[27] V.-A. Nguyen and M. Do. Binary code learning with semantic ranking based supervision. In *ICASSP*, 2016.

[28] M. Norouzi and D. M. Blei. Minimal loss hashing for compact binary codes. In *ICML*, 2011.

[29] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov. Hamming distance metric learning. In *NIPS*, 2012.

[30] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*, 2009.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[32] F. Shen, W. Liu, S. Zhang, Y. Yang, and H. T. Shen. Learning binary codes for maximum inner product search. In *ICCV*, 2015.

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

[34] F. Shen, C. Shen, Q. Shi, A. Van Den Hengel, and Z. Tang. Inductive hashing on manifolds. In *CVPR*, 2013.

[35] F. Shen, C. Shen, Q. Shi, A. van den Hengel, Z. Tang, and H. T. Shen. Hashing on nonlinear manifolds. *IEEE Transactions on Image Processing*, 24(6):1839–1851, 2015.

[36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[37] D. Song, W. Liu, R. Ji, D. A. Meyer, and J. R. Smith. Top rank supervised binary coding for visual search. In *ICCV*, 2015.

[38] D. Song, W. Liu, and D. A. Meyer. Fast structural binary coding. In *IJCAI*, 2016.

[39] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *T-PAMI*, 34(1):66–78, 2012.

[40] J. Wang, S. Kumar, and S. Chang. Semi-supervised hashing for large scale search. *T-PAMI*, 34(12):2393–2406, 2012.

[41] J. Wang, W. Liu, A. X. Sun, and Y.-G. Jiang. Learning hash codes with listwise supervision. In *ICCV*, 2013.

[42] J. Wang, J. Wang, N. Yu, and S. Li. Order preserving hashing for approximate nearest neighbor search. In *ACM MM*, 2013.

[43] Q. Wang, Z. Zhang, and L. Si. Ranking preserving hashing for fast similarity search. In *IJCAI*, 2015.

[44] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, 2008.

[45] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, 2010.

[46] T. Yao, F. Long, T. Mei, and Y. Rui. Deep semantic-preserving and ranking-based hashing for image retrieval. In *IJCAI*, 2016.

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

[48] B. Zhuang, G. Lin, C. Shen, and I. Reid. Fast training of triplet-based deep binary embedding networks. In *CVPR*, 2016.