Strongly Constrained Discrete Hashing

Yong Chen, Zhibao Tian, Hui Zhang, Jun Wang, and Dell Zhang, *Senior Member, IEEE*

***Abstract*—Learning to hash is a fundamental technique widely used in large-scale image retrieval. Most existing methods for learning to hash address the involved discrete optimization problem by the continuous relaxation of the binary constraint, which usually leads to large quantization errors and consequently suboptimal binary codes. A few discrete hashing methods have emerged recently. However, they either completely ignore some useful constraints (specifically the balance and decorrelation of hash bits) or just turn those constraints into regularizers that would make the optimization easier but less accurate. In this paper, we propose a novel supervised hashing method named Strongly Constrained Discrete Hashing (SCDH) which overcomes such limitations. It can learn the binary codes for all examples in the training set, and meanwhile obtain a hash function for unseen samples with the above mentioned constraints preserved. Although the model of SCDH is fairly sophisticated, we are able to find *closed-form* solutions to all of its optimization sub-problems and thus design an efficient algorithm that converges quickly. In addition, we extend SCDH to a kernelized version SCDH$_K$. Our experiments on three large benchmark datasets have demonstrated that not only can SCDH and SCDH$_K$ achieve substantially higher MAP scores than state-of-the-art baselines, but they train much faster than those that are also supervised as well.**

***Index Terms*—Learning to hash, image retrieval, discrete optimization.**

I. Introduction

In this era of big data, there are more and more data that need to be stored, indexed, and processed automatically. Learning to hash, as a promising technique to represent data as compact binary codes for economical storage and efficient computation, has attracted much attention from many researchers as well as practitioners [2], [1], [3], [5], [6], [7], [8], [34]. To facilitate approximate nearest neighbors search, the binary codes should try to maintain the semantic similarity between any pair of samples in the data. The methods for learning to hash that preserve pairwise similarities in the learned binary Hamming space have already been shown to deliver impressive results in a number of applications, particularly large-scale image retrieval. Nevertheless, how to further improve the effectiveness and efficiency of such methods is still an important and challenging research problem today.

Generally speaking, there are two kinds of learning to hash methods: unsupervised and supervised. The latter usually works better than the former (due to the exploitation of the label information) but can be considerably slower (due to the more sophisticated optimization algorithm needed).

In the unsupervised category, Spectral Hashing (SH) [9] first constructs the pairwise similarity matrix of the unlabeled data with a predefined kernel function and then solves the semantic hashing problem via spectral decomposition, which is quite inefficient when the dataset is large. Self-Taught Hashing (STH) [13] learns compact binary codes via relaxed SH from the unlabeled training documents and trains SVM classifiers to predict the binary codes for the testing documents. Similar to SH, the time-consuming spectral decomposition would make it impractical for large-scale real-world applications. Hashing with Graphs (AGH) [10] converts the pairwise similarity matrix into low-rank adjacency matrices by utilizing anchor graphs, which makes the corresponding optimization problem computationally feasible on large-scale image collections. However, the performance of image retrieval using the produced binary codes is sensitive to the selection of anchors which is sometimes tricky. Scalable Graph Hashing (SGH) [12] utilizes feature transformation to approximate the whole pairwise similarity matrix efficiently and develops a sequential bit-by-bit learning algorithm, but the bit-wise optimization can be slow on large datasets particularly when the code length is long. All the above unsupervised learning to hash methods either make the continuous relaxation of the binary constraint or adopt just one of the discrete constraints to simplify the corresponding optimization problem, which leaves much room for improvement. Discrete Graph Hashing (DGH) [11] could be considered as an extension of AGH, which casts the graph-based hashing into a sophisticated discrete optimization framework. Although DGH also utilizes strong constraints as our proposed approach, due to the limitations of its optimization algorithm it often underperforms as we will show later in the experiments.

Semi-Supervised Hashing (SSH) [14] utilizes the class labels of data items to infer their semantic similarities/dissimilarities and then learn binary codes from them. Its eigen-decomposition based solution can be quite fast when the amount of available labeled data is not very large. Minimal Loss Hashing (MLH) [15] models the semantic relationships among data items in the structural SVM framework in order to learn similarity-preserving binary codes. Supervised Hashing with Kernels (KSH) [16] exploits kernel-based hash functions to learn the binary codes which could represent complex nonlinear data. Fast Supervised Hashing with Decision Trees (FastH) [18] leverages the advantages of nonlinear functions over linear ones, and uses boosted decision trees to generate better binary codes. All the aforementioned supervised methods for learning to hash, along with other similar work [20], [19], [21], [22], [24], [25], face the same problem: when the labeled dataset is big, the corresponding pairwise similarity matrix would be

Yong Chen is with the Key Lab of Machine Perception, School of EECS, Peking University, Beijing 100871, China.

Zhibao Tian and Hui Zhang are with the Department of Computer Science and Engineering, Beihang University, Beijing 100191, China.

Jun Wang is with the Department of Computer Science, University College London, London WC1E 6BT, U.K.

Dell Zhang is the corresponding author. He is with the Department of Computer Science and Information Systems, Birkbeck, University of London, London WC1E 7HX, and also with Blue Prism AI Labs, London WC2B 6NH, U.K. (e-mail: dell.z@ieee.org)

huge and therefore those methods will run very slowly and sometimes cannot finish in a reasonable time. Column Sampling Based Discrete Supervised Hashing (COSDISH) [26] is a fast algorithm using random partial labeled samples which can learn the binary codes for a large dataset with millions of images in just a dozen of seconds. However, the supervision information is not utilized to the fullest in this method, which restricts its effectiveness. Fast Scalable Supervised Hashing (FSSH) [44] combines pairwise and pointwise supervision signals in its discrete optimization algorithm which performs quite well for image retrieval, but the construction of the pairwise similarity matrix is quite space-inefficient and time-consuming. Notice that such supervised hashing methods usually have to ignore or drop some useful constraints (e.g., the balance and decorrelation of hash bits) in order to make the corresponding optimization problems easier to solve. Although recently, there emerges Discrete Proximal Linearized Minimization (DPLM) [46] and Binary Deep Neural Network (BDNN) [52] both of which start to take the balance and decorrelation constraints into account, they simply convert those constraints into parts of the objective function and thus make the optimization easier but less accurate.

There also exist some pairwise similarity based deep learning approaches to hashing [21], [42], [40], [43], [41]. Such deep-learning methods could achieve competitive performance, but they all would require massive training data and computational resources (with GPUs or TPUs), which makes them fairly expensive. In this paper, we focus on fast shallow models for learning to hash, which are cheap to run and practical for most large-scale real-world applications. How to reduce the cost of deep-learning based hashing is an open research problem, and we leave it for future work.

To unleash the full potential of supervised learning to hash, we propose a novel method named "Strongly Constrained Discrete Hashing (SCDH)". Its main characteristics are summarized as follows.

- SCDH is a supervised discrete hashing method with complex constraints that not only require the hashing model yield binary codes but also enforce the balance and decorrelation of hash bits. Although the balance and decorrelation constraints have been shown to be crucially important for unsupervised learning to hash, they are typically absent in the existing supervised learning to hash methods because of the difficulty arising from discrete optimization.
- To address the tricky discrete optimization problem of SCDH, we introduce an auxiliary variable and decompose the original problem into several subproblems each of which has a *closed-form* solution. This makes the learning algorithm converge in just a small number of iterations (usually fewer than 10). Furthermore, we extend SCDH to a kernelized version dubbed "SCDH$_K$" which could realize non-linear retrieval functions for complex datasets.
- Extensive experiments on three large-scale image datasets have confirmed that our proposed methods could substantially outperform many state-of-the-art competitors with higher retrieval accuracies and meanwhile lower time costs. For example, on the NUS-WIDE dataset with 180k+ images, SCDH/SCDH$_K$ could be trained within a couple of minutes using a commodity PC and achieve superior performance to the alternatives.

Table I: The notations adopted in this paper.

Symbol	Explanation
N	a scalar
$\boldsymbol{v}$	a vector
$\boldsymbol{M}$	a matrix
v_i	a scalar: the (i)-th element of vector $\boldsymbol{v}$
$\boldsymbol{m_i}$	a vector: the (i)-th column of matrix $\boldsymbol{M}$
m_{ij}	a scalar: the (ij)-th element of matrix $\boldsymbol{M}$
$\mathbf{0}_N$	an $N \times 1$ vector with all 0 elements
$\mathbf{1}_N$	an $N \times 1$ vector with all 1 elements
$\boldsymbol{I}_N$	an $N \times N$ identity matrix
$\boldsymbol{O}$	a matrix with all 0 elements
$\boldsymbol{M}^T$	the transpose of matrix $\boldsymbol{M}$
$\boldsymbol{M}^{-1}$	the inverse of square matrix $\boldsymbol{M}$
$tr(\boldsymbol{M})$	the trace of square matrix $\boldsymbol{M}$: $\sum_i m_{ii}$
$\|\|\boldsymbol{v}\|\|_2$	the l_2 norm of vector $\boldsymbol{v}$: $\sqrt{\sum_i v_i^2}$
$\|\|\boldsymbol{M}\|\|_F$	the Frobenius norm of $\boldsymbol{M}$: $\sqrt{\sum_{ij} m_{ij}^2}$
$\mathrm{sgn}(\cdot)$	the element-wise sign function

II. Related Work

From the perspective of optimization, the development of learning to hashing techniques could be roughly divided into three stages as follows.

Stage I: Hashing with spectral relaxation. The spectral hashing (SH) [9] method, to the best of our knowledge, is probably the first to propose the balance and decorrelation constraints, in addition to the apparent binary constraint, for the task of learning to hash. The balance constraints require each bit to fire 50% of the time, while the decorrelation constraints require the bits to be uncorrelated. However, such a formulation implies an NP-hard mixed-integer optimization problem. To overcome this obstacle, SH chooses to relax the binary constraint into a continuous one during the learning of hash functions. Similarly, Self-Taught Hashing (STH) [13], Semi-Supervised Hashing (SSH) [14], Hashing with Graphs (AGH) [10] and Supervised Hashing with Kernels (KSH) [16] all belong to this family of spectral-based hashing methods in which the binary constraint is relaxed. This technique of continuous relaxation would greatly reduce the difficulty of optimization, but the solution could be suboptimal, i.e., the binary codes resulting from thresholding the continuous codes are likely to be inferior to those obtained by optimizing with the original binary constraint intact [17], [52]. Hence, to avoid such negative effects for hashing, our proposed SCDH/SCDH$_K$ keeps the binary constraint discrete rather than relax them.

Stage II: Discrete hashing with the binary constraint only. To make the discrete hashing problem tractable, Supervised Discrete Hashing (SDH) [17] does not make continuous relaxations but discard the "balance" and "decorrelation" constraints (employed in the above mentioned spectral-based hashing approaches), and thus develops the "discrete cyclic coordinate descent" (DCC) algorithm. Fast Supervised Discrete Hashing (FSDH) [45] enhances SDH using an exchangeable regression trick that leads to a closed-form solution for efficient binary codes. Fast Scalable Supervised Hashing (FSSH) [44] differs from FSDH mainly in the utilization of both pointwise and pairwise labeled information; it can achieve better retrieval performance than FSDH that only leverages pointwise supervision. Column Sampling Based Discrete Supervised Hashing (COSDISH) [26] realizes binary hashing to handle large-scale image datasets by randomly sampling columns during the iterative learning process. To sum up, such kind of methods can avoid continuous relaxation and generate binary codes directly via discrete optimization algorithms. However, they have all ignored the desirable balance and decorrelation properties of hash bits, which would hurt the effectiveness of hashing. Compared with those methods, our SCDH/SCDH$_K$ can also produce binary codes directly, and meanwhile try to satisfy the balance and decorrelation constraints.

Stage III: Discrete hashing with the other constraints too. As pointed out in SH [9], the balance and decorrelation constraints really help to maximize the compactness of binary codes. Recently, Discrete Proximal Linearized Minimization (DPLM) [46] and Binary Deep Neural Network (BDNN) [52] have been proposed to bring all those constraints (binary, balance and decorrelaiton) together to achieve *strongly constrained discrete hashing*. However, what those methods actually do is to move the balance and decorrelation properties from the constraints to the objective function, i.e., treat them not as constraints but as regularizes instead. Although this is a popular trick for solving hard optimization problems approximately, it usually requires many iterations for the corresponding algorithms to converge. In contrast, our SCDH/SCDH$_K$ attempts to find *closed-form* solutions to the strongly constrained optimization problem while maintaining both balance and decorrelation as constraints. Being able to get *closed-form* solutions makes our algorithm much faster than the aforementioned iterative algorithms.

III. Problem Statement

Let $\mathcal{D} = \{(\boldsymbol{x_i}, \boldsymbol{l_i})\}_{i=1}^N$ be a set of images, where $\boldsymbol{x_i} \in \mathbb{R}^M$ denotes the (i)-th image represented by an M-dimensional vector, and $\boldsymbol{l_i} \in \{0,1\}^C$ is its corresponding label vector, i.e., if image $\boldsymbol{x_i}$ belongs to the c-th class ($c \in \{1, 2, \cdots, C\}$), then the c-th element of $\boldsymbol{l_i}$ is 1; otherwise, it is 0. N and C are the number of images and the number of classes in the dataset respectively. As in Ref. [39], the similarity s_{ij} between $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ $(i, j = 1, 2, \cdots, N)$ is calculated as:

$$\begin{aligned} s_{ij} &= 2\cos < \boldsymbol{l_i}, \boldsymbol{l_j} > -1 \\ &= 2\frac{\boldsymbol{l_i}^T\boldsymbol{l_j}}{||\boldsymbol{l_i}||_2 \cdot ||\boldsymbol{l_j}||_2} - 1 \\ &= 2\left(\frac{\boldsymbol{l_i}}{||\boldsymbol{l_i}||_2}\right)^T \left(\frac{\boldsymbol{l_j}}{||\boldsymbol{l_j}||_2}\right) - 1 \ . \end{aligned} \tag{1}$$

If we further set:

$$\boldsymbol{G} = \left[\frac{\boldsymbol{l_1}}{||\boldsymbol{l_1}||_2}, \frac{\boldsymbol{l_2}}{||\boldsymbol{l_2}||_2}, \cdots, \frac{\boldsymbol{l_N}}{||\boldsymbol{l_N}||_2}\right]^T , \tag{2}$$

then the pairwise similarity matrix $\boldsymbol{S} = (s_{ij})_{N\times N}$ could be derived from the label information with:

$$\boldsymbol{S} = 2\boldsymbol{G}\boldsymbol{G}^T - \boldsymbol{1}_N\boldsymbol{1}_N^T \ , \tag{3}$$

where each element s_{ij} would be in the range of $[-1, +1]$. We aim to learn a set of hash functions that can preserve the label-based pairwise similarity in the Hamming space. Specifically, K hash functions $\boldsymbol{H(\cdot)} = [h_1(\cdot), h_2(\cdot), \cdots, h_K(\cdot)]^T$ embed each image $\boldsymbol{x_i}$ into a K-bit binary code, i.e., $\boldsymbol{b_i} = \boldsymbol{H(x_i)} \in \{-1, +1\}^K$, and then the whole dataset could be transformed into $\boldsymbol{B} = [\boldsymbol{b_1}, \boldsymbol{b_2}, \cdots, \boldsymbol{b_N}]^T \in \{-1, +1\}^{N\times K}$. In principle, if $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ share more class labels, then the Hamming distance between their corresponding binary codes $\boldsymbol{b_i}$ and $\boldsymbol{b_j}$ should be smaller. The mathematical notations used in this paper are summarized in Table I.

IV. Proposed Method

Here we describe in detail SCDH, a novel supervised discrete hashing method, in the joint learning framework where binary codes and hash functions are obtained simultaneously.

A. Similarity Preservation

Given a pair of images ($\boldsymbol{x_i}$, $\boldsymbol{x_j}$) each of which is encoded as a K-bit binary vector in the $\{-1, +1\}^K$ space, the value of their dot-product (which is in the range of $[-K, +K]$) should, ideally, be proportional to their semantic similarity s_{ij}. So we make the binary codes preserve pairwise similarities through the following optimization:

$$\min_{\boldsymbol{B}} ||K \cdot \boldsymbol{S} - \boldsymbol{B}\boldsymbol{B}^T||_F^2 \tag{4}$$

$$\text{s.t. } \boldsymbol{B} \in \{-1, +1\}^{N\times K}, \boldsymbol{B}^T\boldsymbol{1}_N = \boldsymbol{0}_K, \boldsymbol{B}^T\boldsymbol{B} = N \cdot \boldsymbol{I}_K \ ,$$

where the constraint $\boldsymbol{B}^T\boldsymbol{1}_N = \boldsymbol{0}_K$ requires the hash bits to be *balanced* (i.e., each bit fires 50% of the time) and the constraint $\boldsymbol{B}^T\boldsymbol{B} = N \cdot \boldsymbol{I}_K$ requires the hash bits to be *uncorrelated*. These two constrains, balance and decorrelation, are known to encourage the generation of compact binary codes [9], [11].

The objective function in (4) is quite common in supervised hashing with pairwise similarities preserved, but there exist two computational challenges: (1) how to construct the $N \times N$ pairwise similarity matrix $\boldsymbol{S}$ efficiently; (2) how to solve the strongly constrained discrete optimization problem efficiently. In response to the first challenge, we represent $\boldsymbol{S}$ using the low-rank matrix $\boldsymbol{G}$ (in general $C \ll N$) as shown in (3), which would significantly reduce the storage cost and also greatly accelerate the subsequent computation in the hashing process. With regard to the second challenge, most existing hashing methods (e.g., SH [9] and STH [13]) relax the discrete constraint $\boldsymbol{B} \in \{-1, +1\}^{N\times K}$ to a continuous one $\boldsymbol{B} \in \mathbb{R}^{N\times K}$, which simplifies the optimization but meanwhile hurts the retrieval performance. Our solution, however, can afford to retain the discrete constraints with the help of an auxiliary variable, as explained below.

B. Joint Learning

Let $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N]^T$. For fast image retrieval, we use linear hash functions $\boldsymbol{P} \in \mathbb{R}^{M\times K}$ to produce the binary codes:

$$\boldsymbol{B} = \text{sgn}(\boldsymbol{X}\boldsymbol{P}) \ . \tag{5}$$

The hash functions $\boldsymbol{P}$ can be learned simultaneously with the binary codes $\boldsymbol{B}$ by expanding (4) to the following:

$$\min_{\boldsymbol{B},\boldsymbol{P}} ||K\cdot\boldsymbol{S}-\boldsymbol{B}\boldsymbol{B}^T||_F^2+\lambda||\mathrm{sgn}(\boldsymbol{X}\boldsymbol{P})-\boldsymbol{B}||_F^2+\beta||\boldsymbol{P}||_F^2 \quad (6)$$
$$\text{s.t. } \boldsymbol{B}\in\{-1,+1\}^{N\times K}, \boldsymbol{B}^T\boldsymbol{1}_N=\boldsymbol{0}_K, \boldsymbol{B}^T\boldsymbol{B}=N\cdot\boldsymbol{I}_K\ ,$$

where λ is a positive parameter to weigh the relative importance of binary codes and hash functions, while β is a non-negative smoothing factor to prevent overfitting or irreversibility.

The sign function $\mathrm{sgn}(\cdot)$ is not differentiable, which makes the optimization problem (6) difficult to solve directly. Therefore, we replace $\mathrm{sgn}(\boldsymbol{XP})$ with just $\boldsymbol{XP}$. That is to say, we require each element of $\boldsymbol{XP}$ itself rather than its sign to be as close as possible to the corresponding element of $\boldsymbol{B}$ (which is either $+1$ or -1). Moreover, to make this discrete optimization problem easier, we introduce an auxiliary variable $\boldsymbol{Z}$ as an alias of $\boldsymbol{B}$ (i.e., $\boldsymbol{Z}=\boldsymbol{B}$) and rewrite (6) as:

$$\min_{\boldsymbol{B},\boldsymbol{P},\boldsymbol{Z}} ||K\cdot\boldsymbol{S}-\boldsymbol{B}\boldsymbol{Z}^T||_F^2+\lambda||\boldsymbol{X}\boldsymbol{P}-\boldsymbol{B}||_F^2+\beta||\boldsymbol{P}||_F^2 \quad (7)$$
$$\text{s.t. } \begin{cases} \boldsymbol{B}\in\{-1,+1\}^{N\times K}, \\ \boldsymbol{Z}=\boldsymbol{B}, \boldsymbol{Z}^T\boldsymbol{1}_N=\boldsymbol{0}_K, \boldsymbol{Z}^T\boldsymbol{Z}=N\cdot\boldsymbol{I}_K \end{cases} .$$

C. The Complete Optimization Problem

Finally, we go further to drop the constraint $\boldsymbol{Z}=\boldsymbol{B}$ and make the relaxation that $\boldsymbol{Z}$ is a real-valued continuous variable approximating the discrete variable $\boldsymbol{B}$. In other words, $\boldsymbol{B}$ and $\boldsymbol{Z}$ are no longer required to be strictly identical but they should be similar to each other. Thus, the overall objective function that takes all the above considerations into account can be extended from (7) as follows:

$$\begin{aligned}\min_{\boldsymbol{B},\boldsymbol{P},\boldsymbol{Z}} \mathcal{O}(\boldsymbol{P},\boldsymbol{B},\boldsymbol{Z}) &= ||K\cdot\boldsymbol{S}-\boldsymbol{B}\boldsymbol{Z}^T||_F^2+ \\ &\lambda||\boldsymbol{X}\boldsymbol{P}-\boldsymbol{B}||_F^2+\alpha||\boldsymbol{B}-\boldsymbol{Z}||_F^2+\beta||\boldsymbol{P}||_F^2\end{aligned} \quad (8)$$
$$\text{s.t. } \begin{cases} \boldsymbol{B}\in\{-1,+1\}^{N\times K}, \\ \boldsymbol{Z}\in\mathbb{R}^{N\times K}, \boldsymbol{Z}^T\boldsymbol{1}_N=\boldsymbol{0}_K, \boldsymbol{Z}^T\boldsymbol{Z}=N\cdot\boldsymbol{I}_K. \end{cases} ,$$

where the additional parameter α controls how closely $\boldsymbol{Z}$ approximates $\boldsymbol{B}$.

With the above joint learning framework, the binary codes for training data and the hash functions for out-of-sample data (e.g., testing samples, new query samples) can be obtained simultaneously. Given a set of out-of-sample images $\boldsymbol{X}_{oos}$, we can encode them into binary codes using the hash functions:

$$\boldsymbol{B}_{oos}=\mathrm{sgn}(\boldsymbol{X}_{oos}\boldsymbol{P}), \quad (9)$$

which is essentially a linear transformation and therefore can be computed very efficiently.

D. Kernelization

As demonstrated in KLSH [35], [36], KSH [16] and FastH [18], nonlinear hash functions can often perform much better than linear ones because of their ability of fitting complex patterns in the data. SCDH can also be extended to nonlinear hashing through kernel functions. Given a nonlinear mapping $\boldsymbol{\Phi}:\boldsymbol{x}\in\mathbb{R}^M\mapsto\boldsymbol{\Phi}(\boldsymbol{x})\in\mathbb{R}^D$ (D could be infinite), the entire image collection could be mapped into $\boldsymbol{\Phi}(\boldsymbol{X})\equiv[\boldsymbol{\Phi}(\boldsymbol{x}_1),\boldsymbol{\Phi}(\boldsymbol{x}_2),\cdots,\boldsymbol{\Phi}(\boldsymbol{x}_N)]^T\in\mathbb{R}^{N\times D}$. Let us randomly select Q anchors (i.e., a subset of images) from the image dataset, and denote them by $\boldsymbol{y}_1,\boldsymbol{y}_2,\cdots,\boldsymbol{y}_Q$; then we can view $\boldsymbol{\Phi}(\boldsymbol{y}_1),\boldsymbol{\Phi}(\boldsymbol{y}_2),\cdots,\boldsymbol{\Phi}(\boldsymbol{y}_Q)$ as a set of base vectors that can be used to represent any vector in $\mathbb{R}^D$. This is a popular trick for handling big data and it usually works well in practice. Thus, we have:

$$\begin{aligned}\boldsymbol{\Phi}(\boldsymbol{P}) &= \boldsymbol{\Phi}([\boldsymbol{p}_1,\boldsymbol{p}_2,\cdots,\boldsymbol{p}_K]) \\ &= [\boldsymbol{\Phi}(\boldsymbol{y}_1),\boldsymbol{\Phi}(\boldsymbol{y}_2),\cdots,\boldsymbol{\Phi}(\boldsymbol{y}_Q)]\boldsymbol{A},\end{aligned} \quad (10)$$

where $\boldsymbol{A}\in\mathbb{R}^{Q\times K}$. Accordingly, Eq. (5) is extended to:

$$\begin{aligned}\boldsymbol{B} &= \mathrm{sgn}(\boldsymbol{\Phi}(\boldsymbol{X})\boldsymbol{\Phi}(\boldsymbol{P})) \\ &= \mathrm{sgn}([\boldsymbol{\Phi}(\boldsymbol{x}_1),\boldsymbol{\Phi}(\boldsymbol{x}_2),\cdots,\boldsymbol{\Phi}(\boldsymbol{x}_N)]^T \\ &\qquad [\boldsymbol{\Phi}(\boldsymbol{y}_1),\boldsymbol{\Phi}(\boldsymbol{y}_2),\cdots,\boldsymbol{\Phi}(\boldsymbol{y}_Q)]\boldsymbol{A}) \\ &= \mathrm{sgn}\left((\boldsymbol{\Phi}(\boldsymbol{x}_i)^T\boldsymbol{\Phi}(\boldsymbol{y}_j))_{N\times Q}\boldsymbol{A}\right)\ .\end{aligned} \quad (11)$$

Let $\mathcal{K}:\mathbb{R}^D\times\mathbb{R}^D\mapsto\mathbb{R}$ denote the kernel function corresponding to the nonlinear mapping Φ and $\boldsymbol{\mathcal{K}_Q}\equiv(\boldsymbol{\Phi}(\boldsymbol{x}_i)^T\boldsymbol{\Phi}(\boldsymbol{y}_j))_{N\times Q}$ the kernel matrix. Similar to (8), the kernelized version of SCDH is formulated as:

$$\begin{aligned}\min_{\boldsymbol{B},\boldsymbol{A},\boldsymbol{Z}} &||K\cdot\boldsymbol{S}-\boldsymbol{B}\boldsymbol{Z}^T||_F^2+\lambda||\boldsymbol{\mathcal{K}_Q}\boldsymbol{A}-\boldsymbol{B}||_F^2 \\ &+\alpha||\boldsymbol{B}-\boldsymbol{Z}||_F^2+\beta||\boldsymbol{A}||_F^2\end{aligned} \quad (12)$$
$$\text{s.t. } \begin{cases} \boldsymbol{B}\in\{-1,+1\}^{N\times K}, \\ \boldsymbol{Z}\in\mathbb{R}^{N\times K}, \boldsymbol{Z}^T\boldsymbol{1}_N=\boldsymbol{0}_K, \boldsymbol{Z}^T\boldsymbol{Z}=N\cdot\boldsymbol{I}_K. \end{cases} ,$$

which is called SCDH$_K$ for short.

After the kernel function $\mathcal{K}$ being chosen and the matrix $\boldsymbol{A}$ being learned, an out-of-sample image $\boldsymbol{x}_{oos}$ can be encoded:

$$\begin{aligned}\boldsymbol{b}_{oos} &= \mathrm{sgn}\left(\left[(\boldsymbol{\Phi}(\boldsymbol{x}_{oos})^T\boldsymbol{\Phi}(\boldsymbol{y}_j))_{1\times Q}\boldsymbol{A}\right]^T\right) \\ &= \mathrm{sgn}\left(\left[(\mathcal{K}(\boldsymbol{x}_{oos},\boldsymbol{y}_j))_{1\times Q}\boldsymbol{A}\right]^T\right)\ .\end{aligned} \quad (13)$$

The vector-matrix multiplication involved in the above equation should be quite computationally efficient, as usually $Q\ll N$.

V. Optimization

In this section, we explain how the intricate optimization problem of the above SCDH method can be solved efficiently. The solution to SCDH$_K$ is very similar, so it is omitted here.

The optimization problem (8) has three variables to be optimized: $\boldsymbol{P}$, $\boldsymbol{B}$ and $\boldsymbol{Z}$. Our algorithm is to update each variable while holding the other two fixed (i.e., alternating minimization), and iterate this process until convergence.

A. Update $\boldsymbol{P}$ with $\boldsymbol{B}$ and $\boldsymbol{Z}$ Fixed

When $\boldsymbol{B}$ and $\boldsymbol{Z}$ are fixed, the objective function of $\boldsymbol{P}$ is given by:

$$\min_{\boldsymbol{P}} \mathcal{O}(\boldsymbol{P})=\lambda||\boldsymbol{X}\boldsymbol{P}-\boldsymbol{B}||_F^2+\beta||\boldsymbol{P}||_F^2, \quad (14)$$

which is in fact a least-squares problem with L$_2$ regularization. Setting $\frac{\partial\mathcal{O}(\boldsymbol{P})}{\partial\boldsymbol{P}}=\boldsymbol{O}$, we get the closed-form solution:

$$\boldsymbol{P}=\left(\boldsymbol{X}^T\boldsymbol{X}+\frac{\beta}{\lambda}\boldsymbol{I}_M\right)^{-1}\boldsymbol{X}^T\boldsymbol{B}. \quad (15)$$

B. Update $\boldsymbol{B}$ with $\boldsymbol{Z}$ and $\boldsymbol{P}$ Fixed

When $\boldsymbol{P}$ and $\boldsymbol{Z}$ are fixed, the objective function of $\boldsymbol{B}$ is simplified into:

$$\begin{aligned} \min_{\boldsymbol{B}} \mathcal{O}(\boldsymbol{B}) = ||K \cdot \boldsymbol{S} - \boldsymbol{B}\boldsymbol{Z}^T||_F^2 + \\ \lambda||\boldsymbol{X}\boldsymbol{P} - \boldsymbol{B}||_F^2 + \alpha||\boldsymbol{B} - \boldsymbol{Z}||_F^2 \\ \text{s.t. } \boldsymbol{B} \in \{-1, +1\}^{N \times K} . \end{aligned} \tag{16}$$

This is equivalent to the optimization problem:

$$\begin{aligned} \max_{\boldsymbol{B}} \ tr(\boldsymbol{B}^T\{K \cdot \boldsymbol{S}\boldsymbol{Z} + \lambda \boldsymbol{X}\boldsymbol{P} + \alpha \boldsymbol{Z}\}) \\ \text{s.t. } \boldsymbol{B} \in \{-1, +1\}^{N \times K} \end{aligned} \tag{17}$$

which can be solved by applying the following theorem.

Theorem 1. *Given a matrix $\boldsymbol{C} \in \mathbb{R}^{N \times K}$, the optimization problem*

$$\max_{\boldsymbol{B}} \ tr(\boldsymbol{B}\boldsymbol{C}^T) \quad \text{s.t. } \boldsymbol{B} \in \{-1, +1\}^{N \times K}, \tag{18}$$

has the closed-form solution $\boldsymbol{B} = \mathrm{sgn}(\boldsymbol{C})$.

Proof. According to the definition of the trace function,

$$tr(\boldsymbol{B}\boldsymbol{C}^T) = \sum\nolimits_{i,j} b_{ij} c_{ij} \ . \tag{19}$$

So the optimization problem (18) is the same as:

$$\max_{b_{ij}} \ b_{ij} c_{ij} \quad \text{s.t. } b_{ij} \in \{-1, +1\} \ , \tag{20}$$

for each b_{ij} with $i \in \{1, 2, \cdots, N\}$, $j \in \{1, 2, \cdots, K\}$. Obviously, to achieve the maximum, each $b_{ij}c_{ij}$ needs to be positive, i.e., $b_{ij} = \mathrm{sgn}(c_{ij})$. Q.E.D. □

Thus, the closed-form solution of (16) is given by:

$$\boldsymbol{B} = \mathrm{sgn}(K \cdot \boldsymbol{S}\boldsymbol{Z} + \lambda \boldsymbol{X}\boldsymbol{P} + \alpha \boldsymbol{Z}) \ . \tag{21}$$

C. Update $\boldsymbol{Z}$ with $\boldsymbol{P}$ and $\boldsymbol{B}$ Fixed

When $\boldsymbol{P}$ and $\boldsymbol{B}$ are fixed, the objective function of $\boldsymbol{Z}$ is written as:

$$\begin{aligned} \min_{\boldsymbol{Z}} \mathcal{O}(\boldsymbol{Z}) = ||K \cdot \boldsymbol{S} - \boldsymbol{B}\boldsymbol{Z}^T||_F^2 + \alpha||\boldsymbol{B} - \boldsymbol{Z}||_F^2 \\ \text{s.t. } \boldsymbol{Z} \in \mathbb{R}^{N \times K}, \boldsymbol{Z}^T\boldsymbol{1}_N = \boldsymbol{0}_K, \boldsymbol{Z}^T\boldsymbol{Z} = N \cdot \boldsymbol{I}_K \ . \end{aligned} \tag{22}$$

It can be further reduced to:

$$\begin{aligned} \max_{\boldsymbol{Z}} \ tr(\boldsymbol{Z}^T\{K \cdot \boldsymbol{S}\boldsymbol{B} + \alpha \boldsymbol{B}\}) \\ \text{s.t. } \boldsymbol{Z} \in \mathbb{R}^{N \times K}, \boldsymbol{Z}^T\boldsymbol{1}_N = \boldsymbol{0}_K, \boldsymbol{Z}^T\boldsymbol{Z} = N \cdot \boldsymbol{I}_K \ . \end{aligned} \tag{23}$$

Let $\boldsymbol{E} = K \cdot \boldsymbol{S}\boldsymbol{B} + \alpha \boldsymbol{B}$, and then we can get the closed-form solution through the following theorem.

Theorem 2. *The optimization problem*

$$\max_{\boldsymbol{Z}} \ tr(\boldsymbol{Z}^T\boldsymbol{E}) \quad \text{s.t. } \boldsymbol{Z}^T\boldsymbol{1}_N = \boldsymbol{0}_K, \boldsymbol{Z}^T\boldsymbol{Z} = N \cdot \boldsymbol{I}_K \ , \tag{24}$$

has the closed-form solution:

$$\boldsymbol{Z} = \sqrt{N}[\boldsymbol{U}, \bar{\boldsymbol{U}}][\boldsymbol{V}, \bar{\boldsymbol{V}}]^T \ . \tag{25}$$

The matrices

$$\boldsymbol{U} = [\boldsymbol{u_1}, \boldsymbol{u_2}, \cdots, \boldsymbol{u}_{K'}] \text{ and } \boldsymbol{V} = [\boldsymbol{v_1}, \boldsymbol{v_2}, \cdots, \boldsymbol{v}_{K'}]$$

are obtained via the Singular Value Decomposition (SVD) of $\boldsymbol{J}\boldsymbol{E}$ with $\boldsymbol{J} = \boldsymbol{I}_N - \frac{1}{N}\boldsymbol{1}_N\boldsymbol{1}_N^T$, i.e.,

$$\boldsymbol{J}\boldsymbol{E} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \sum\nolimits_{k=1}^{K'} \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k^T \ . \tag{26}$$

Note that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{K'} > 0$. Then, the matrices $\bar{\boldsymbol{U}} \in \mathbb{R}^{N \times (K-K')}$ and $\bar{\boldsymbol{V}} \in \mathbb{R}^{K \times (K-K')}$ are obtained via the Gram-Schmidt process such that $\bar{\boldsymbol{U}}^T\bar{\boldsymbol{U}} = \boldsymbol{I}_{K-K'}$, $[\boldsymbol{U}, \boldsymbol{1}_N]^T\bar{\boldsymbol{U}} = \boldsymbol{O}$, $\bar{\boldsymbol{V}}^T\bar{\boldsymbol{V}} = \boldsymbol{I}_{K-K'}$, and $\boldsymbol{V}^T\bar{\boldsymbol{V}} = \boldsymbol{O}$. If $K' = K$, $\bar{\boldsymbol{U}}$ and $\bar{\boldsymbol{V}}$ will be empty.

Proof. Please refer to Ref. [11]. □

D. Computational Complexity

The learning algorithm for SCDH is built on top of the above three subproblems of optimization and formally specified in Algorithm 1. In each iteration, three closed-form solutions — Eqs. (15), (21) and (25) — need to be computed for the three corresponding subproblems respectively.

Regarding the $\boldsymbol{P}$-subproblem, the main computational operations are the multiplications of $\boldsymbol{X}^T\boldsymbol{X}$ and the inverse of a $M \times M$ square matrix whose time complexities are $O(NM^2)$ and $O(M^3)$ respectively. The whole time complexity of this subproblem is $O(NM^2 + M^3 + NMK + KM^2)$, where M is the number of original features and K is the length of hash codes. Usually $M, K \ll N$ (the number of samples in the dataset), which makes the time complexity of this subproblem linear w.r.t. N.

Regarding the $\boldsymbol{B}$-subproblem, the most time-consuming part is the computation of $\boldsymbol{S}\boldsymbol{Z}$. However, due to the fact that $\boldsymbol{S} = 2\boldsymbol{G}\boldsymbol{G}^T - \boldsymbol{1}_N\boldsymbol{1}_N^T$, the time complexity could be reduced from $O(KN^2)$ to $O(CKN)$, where C is the number of class labels ($C \ll N$). Thus, the whole time complexity of this subproblem is $O(CKN + KMN)$, which is linear w.r.t. N.

Regarding the $\boldsymbol{Z}$-subproblem, the main step requiring intensive computation is the SVD for a $N \times K$ matrix whose time complexity is $O(NK^2)$. It is easy to see that the other operations would require less expenditure of time than this. Therefore, we can conclude that the whole time complexity of this subproblem is also linear w.r.t. N.

Algorithm 1: SCDH

Input: Data matrix $\boldsymbol{X}$, label matrix $\boldsymbol{G}$, length of hash codes K, hyperparameters α, β and λ, max iterations $maxIter$, precision ε.

Output: Binary codes $\boldsymbol{B}$, auxiliary variable $\boldsymbol{Z}$ and hash functions $\boldsymbol{P}$.

```
Randomly initialize P, B, and Z;
while not convergent do
    /* Convergence: the number of
       iterations is bigger than maxIter,
       or the error is less than ε. */
    Optimize P according to Eq. (15);
    Optimize B according to Eq. (21);
    Optimize Z according to Eq. (25);
end
Return P, B, and Z.
```

In summary, the total computational complexity of the entire SCDH algorithm is linear w.r.t. N for each iteration. Moreover, in practice, the algorithm usually needs only a few (< 10) iterations to reach convergence (see Fig. 1). Hence, the proposed SCDH method is indeed highly efficient.

VI. Experiments

We have used several large-scale image datasets to evaluate SCDH's retrieval performance on a PC with Intel(R) Xeon(R) CPU E5-2650 v4 @2.20GHz and 64GB RAM.

A. Datasets

Caltech256 contains 30,607 images belonging to 256 categories [27]. Each image is represented by a 1,024-dimension CNN feature vector associated with one category label. We randomly select 26,000 samples for training and 3,000 samples for testing (i.e., Train:Test=26,000:3,000).

Cifar10 includes 60,000 color images (of size 32×32) that are divided evenly into 10 classes (each of which holds 6,000 samples) [28]. We choose 5,400 samples from each class as the training set and the remaining as the testing set (i.e., Train:Test=54,000:6,000). For each image, a 512-dimension GIST feature vector is extracted as its representation.

NUS-WIDE is a real-world web database originally containing 269,648 images each associated with multiple textual tags [37]. Following the protocol in Ref. [38], we focus on 186,577 images that cover the top-10 most frequent semantic concepts. In our experiments, we take 1% of the dataset as the testing set and the remaining as the training set (i.e., Train:Test=184,711:1,866). Each image is converted into a 500-dimensional bag-of-visual-word features. This is a relatively larger and more challenging dataset for image retrieval.

B. Evaluation

In the retrieval experiments, those images sharing at least one class label or tag with the query image would be considered as relevant results. Mean Average Precision (MAP) is a very popular metric for evaluating the retrieval performance of learning to hash methods [31], [32], [30], [18], [26], [24]. For all our experiments on the above mentioned image datasets, we would also employ MAP as the measure of effectiveness. Besides, the running time of each method in the experiments would also be recorded to assess its efficiency.

Regarding the baseline methods for comparison, we have chosen the most representative as well as the currently most competitive ones: LSH[1][2], PCAH[2][31], ITQ (rotation after PCA for binary codes)[3][3], DGH[4][11], SGH[5][12], CCA-ITQ[6][3], SDH[7][17], HC-SDH[4][23], FSDH[8][45], FastH[9][18], COSDISH[5][26], FSSH[10][44]. These 12 competitors in our experiments come from two groups: the first 5 are unsupervised methods which usually run fast but may yield inferior results; whereas the other 7 are supervised methods which often produce high MAPs for image retrieval though their training speed could be slow. Among them, HC-SDH has just been evaluated on Cifar10 due to some of its limitations (e.g., $K \geq C$ and single-label only); the other baseline methods can successfully run on at least two of the three image datasets mentioned earlier. There also exist many other learning to hash methods such as SH [9] and KSH [16] which perform well on small datasets but cannot scale to big datasets and therefore have to be excluded from the experiments.

C. Settings

All the baseline methods except DGH and HC-SDH have already been implemented in MATLAB with their source codes provided by the corresponding authors. To ensure a fair comparison (especially for the speed), we have also implemented DGH and HC-SDH as well as our proposed approaches SCDH/SCDH$_K$ in MATLAB. The inputs (i.e., the data and label matrices) to all the different methods are identical. The initialization of each baseline method is carried out in exactly the same way as described in its original paper. The hyperparameters of each method have been tuned on different datasets to get the best validation performance, in accordance with the authors' proposals.

For our proposed method SCDH, we set $maxIter = 10$ and $\varepsilon = 10^{-10}$ in Algorithm 1. Note that our method usually converges in fewer than 10 iterations in the experiments (see Fig. 1). For the hyperparameters α, β and λ, we have tuned them by grid search with each of them ranging from 10^{-9} to 10^{9}. SCDH would be able to achieve good MAP scores using most of the parameter values within the above range, and the finally chosen combination is ($\alpha = 0.1$, $\beta = 1$, $\lambda = 10$). This set of hyperparameters would be directly adopted by the kernelized version of our method SCDH$_K$ in our experiments. SCDH$_K$ would employ the Gaussian (RBF) kernel $\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \exp(-||\boldsymbol{x} - \boldsymbol{y}||_2^2/(2\sigma^2))$ with $\sigma = 0.4$ and make use of 2,000 anchors (see Section VI-F for further details).

D. Results

Table II shows the MAP scores and the training time costs[11] of our proposed approaches as well as the baseline methods on three different image datasets. The code length has been set to 8, 16, 32, and 64 bits.

Unsurprisingly, the experimental results indicate that the supervised hashing methods outperform the unsupervised ones, though they are usually slower. This is consistent with our intuition and previous research findings that exploiting the label information can in general enhance the effectiveness of hashing.

Among the supervised hashing methods, our proposed SCDH$_K$ always delivers the best performance, while most

[1] http://www.cad.zju.edu.cn/home/dengcai/Data/DSH.html
[2] http://www.cad.zju.edu.cn/home/dengcai/Data/DimensionReduction.html
[3] https://goo.gl/AGuu86
[4] Our own implementation of this algorithm in MATLAB.
[5] http://cs.nju.edu.cn/lwj/L2H.html
[6] https://github.com/jfeng10/ITQ-image-retrieval
[7] https://github.com/bd622/DiscretHashing
[8] https://tongliang-liu.github.io/publications.html
[9] https://bitbucket.org/chhshen/fasthash/src/master/
[10] https://lcbwlx.wixsite.com/fssh
[11] Compared with the training time costs, the testing time costs can usually be ignored, especially on large datasets.

Table II: The MAP scores and training time costs (in seconds) of different hashing methods. The best results are in bold, the second-best are underlined, and "—" means that the method was unable to finish in a reasonable time.

(a) Caltech256 {Train:Test = 26,000:3,000}

Methods /Length	8 bits		16 bits		32 bits		64 bits	
	MAP score	training time	MAP score	training time	MAP score	training time	MAP score	training time
LSH	0.0191	0.007	0.0378	0.002	0.0919	0.006	0.1657	0.004
PCAH	0.0655	0.597	0.1218	0.840	0.1867	1.363	0.2427	1.655
ITQ	0.0842	1.238	0.1578	2.543	0.2434	4.785	0.3236	6.687
DGH	0.0445	97.027	0.1012	41.100	0.1268	42.509	0.2148	58.049
SGH	0.0657	3.693	0.1349	4.491	0.2156	4.940	0.2912	5.328
CCA-ITQ	0.1013	3.327	0.2175	6.465	0.3033	9.052	0.3858	16.722
SDH	0.1535	13.758	0.2789	22.985	0.3498	38.381	0.4102	64.244
FSDH	0.1453	4.978	0.2679	7.217	0.3526	12.136	0.4381	21.123
FastH	0.1847	282.138	0.3872	333.757	0.5303	550.108	0.6501	839.958
COSDISH	0.1130	5.775	0.2322	9.212	0.4045	19.888	0.5699	65.364
FSSH	0.1449	20.844	0.4449	21.221	0.5851	21.323	0.6338	21.898
SCDH	0.1969	8.467	0.3527	4.365	0.4998	7.26	0.6220	5.804
SCDH_K	**0.3242**	16.287	**0.5467**	11.537	**0.6538**	19.075	**0.7076**	14.945

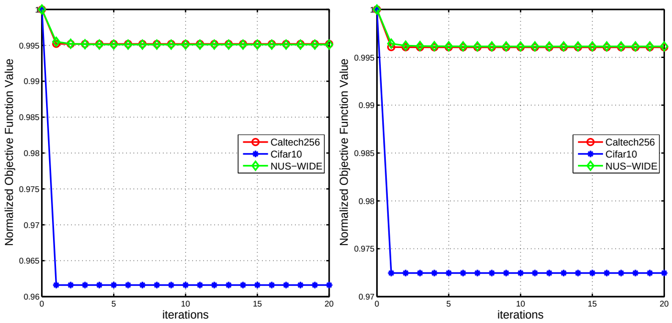
(b) Cifar10 {Train:Test = 54,000:6,000}

Methods /Length	8 bits		16 bits		32 bits		64 bits	
	MAP score	training time	MAP score	training time	MAP score	training time	MAP score	training time
LSH	0.1186	0.001	0.1230	0.001	0.1408	0.012	0.1480	0.002
PCAH	0.1311	0.373	0.1315	0.385	0.1276	0.438	0.1234	0.727
ITQ	0.1517	1.257	0.1615	2.177	0.1674	5.459	0.1737	8.881
DGH	0.1213	102.784	0.1236	121.674	0.1238	167.458	0.1230	127.135
SGH	0.1388	2.483	0.1474	3.159	0.1442	4.707	0.1430	10.047
CCA-ITQ	0.2087	2.689	0.2267	5.662	0.2713	7.730	0.2879	12.658
SDH	0.2576	11.322	0.2868	23.149	0.3280	28.491	0.3372	49.517
FSDH	0.2356	4.742	0.2932	8.164	0.3295	9.750	0.3417	11.006
HC-SDH	n/a	n/a	0.5219	4.058	0.5352	4.209	0.5355	4.253
FastH	0.4568	552.276	0.5463	806.598	0.6163	1258.380	0.6670	2495.000
COSDISH	0.2915	7.383	0.3626	11.647	0.4717	33.977	0.5091	136.961
FSSH	0.6037	100.913	0.6280	101.786	0.6738	102.850	0.6988	108.956
SCDH	0.4999	4.560	0.5544	9.263	0.6116	9.901	0.6376	12.186
SCDH_K	**0.6353**	11.426	**0.6773**	15.499	**0.7023**	16.692	**0.7114**	26.673

(c) NUS-WIDE {Train:Test = 18,4711:1,866}

Methods /Length	8 bits		16 bits		32 bits		64 bits	
	MAP score	training time	MAP score	training time	MAP score	training time	MAP score	training time
LSH	0.3479	0.200	0.3481	0.706	0.3525	0.102	0.3585	0.327
PCAH	0.3722	1.134	0.3678	1.773	0.3620	2.073	0.3569	2.257
ITQ	0.3754	4.072	0.3795	9.224	0.3836	15.831	0.3836	26.666
DGH	0.3388	745.675	0.3389	1412.493	0.3389	614.298	0.3642	1159.796
SGH	0.3425	11.825	0.3422	16.440	0.3418	57.822	0.3432	189.584
CCA-ITQ	0.3987	8.167	0.4232	11.013	0.432	23.447	0.4571	40.779
SDH	0.4422	35.647	0.4506	43.673	0.4629	82.316	0.4707	209.554
FSDH	0.4483	11.334	0.4640	15.440	0.4727	36.629	0.4953	69.578
FastH	—	—	—	—	—	—	—	—
COSDISH	0.5916	18.424	0.5946	38.527	0.6087	122.010	0.6558	448.665
FSSH	0.6332	1023.477	0.6407	1023.513	0.6528	1023.860	0.6729	1027.045
SCDH	0.6180	9.686	0.6463	17.124	0.6553	38.028	0.6561	96.658
SCDH_K	**0.6511**	59.192	**0.6661**	72.174	**0.6737**	89.394	**0.6792**	155.057

of the time SCDH and FSSH compete for the second spot. SCDH_K's noticeable performance gain over the vanilla SCDH confirms the usefulness of nonlinear hash functions for large and complex datasets. Although FastH sometimes provides slightly higher MAP scores than SCDH, it is much more time-consuming, especially with longer binary codes and larger image collections. In fact, FastH was not able to finish the experiments on NUS-WIDE, the largest dataset, in a reasonable time. SDH, a pointwise hashing method, does not really preform better than the pairwise similarity preservation based methods like SCDH/SCDH_K in terms of MAP scores; it is also much slower than our methods in most cases. Although FSDH, an extension of SDH, exhibits a slightly faster training speed than SCDH/SCDH_K, its retrieval effectiveness is a lot worse. Moreover, HC-SDH which incorporates the balance and decorrelation constraints into SDH by Hadamard operations [23] works significantly better than SDH and FSDH, which confirms the merit of imposing such constraints for hashing. However, HC-SDH's retrieval performance still lags far behind that of our proposed SCDH/SCDH_K.



(a) 32 bits (b) 64 bits

Figure 1: The convergence curves of SCDH.

Similar to SCDH$_K$, some baseline methods (i.e., SGH, FSDH, HC-SDH and FSSH) also use the kernel trick to achieve nonlinear hashing[12]. FSSH evidently reaches the best retrieval performance among the baseline methods, but it is consistently inferior to SCDH$_K$ in terms of both MAP scores and training speeds on all the three datasets, which testifies the effectiveness and efficiency of our proposed methods. Specifically, FSSH utilizes both pairwise and pointwise supervision for hashing, while SCDH$_K$ is based entirely on pairwise similarity preservation. The advantages of SCDH$_K$ against FSSH probably come from the two strong constraints, i.e., the balance and decorrelation of hash bits (see Section VI-G for a more detailed analysis of their usefulness).

E. Convergence Analysis

It is clear from Algorithm 1 for SCDH that the value of the objective function $\mathcal{O}(\boldsymbol{P}, \boldsymbol{B}, \boldsymbol{Z})$ will decrease from iteration to iteration until it is stabilized:

$$\begin{aligned} \mathcal{O}(\boldsymbol{P}^t, \boldsymbol{B}^t, \boldsymbol{Z}^t) &\geq \mathcal{O}(\boldsymbol{P}^{t+1}, \boldsymbol{B}^t, \boldsymbol{Z}^t) \\ &\geq \mathcal{O}(\boldsymbol{P}^{t+1}, \boldsymbol{B}^{t+1}, \boldsymbol{Z}^t) \\ &\geq \mathcal{O}(\boldsymbol{P}^{t+1}, \boldsymbol{B}^{t+1}, \boldsymbol{Z}^{t+1}) \ . \end{aligned} \tag{27}$$

Since during the execution of the algorithm, the objective function can only go down and it cannot go lower than zero, the iterative algorithm for SCDH is theoretically guaranteed to converge.

Let us further investigate how fast the algorithm can converge. Fig. 1 shows the convergence curves of SCDH on all those three large datasets for 32-bit and 64-bit codes[13]. In each sub-graph the x-axis represents the iteration number and the y-axis represents the normalized[14] value of the objective function. It is obvious that the SCDH algorithm converges very quickly within just a few iterations. This is probably attributed to the low-rank representations for the pairwise similarity matrix and the efficient closed-form solutions to the subproblems of optimization.

The convergence of SCDH$_K$ is similar to that of SCDH, so its analysis is omitted here.

[12]For a fair comparison, in our experiments, all those nonlinear hashing methods make use of the Gaussian kernel equipped with 2000 anchors, except that SGH uses 300 anchors only (as that leads to comparable MAP scores but much less training time).

[13]The convergence curves of SCDH for other code lengths show the same trend and therefore are omitted.

[14]To normalize the value of the objective function at each iteration, it is divided by its maximum value (which is always received at the first iteration).

Table III: The MAP scores of SCDH$_K$ with different kernels.

kernels	Cifar10 32 bits	Cifar10 64 bits	NUS-WIDE 32 bits	NUS-WIDE 64 bits
Linear	0.6255	0.6279	0.6436	0.6443
Polynomial (α=1, c=0, d=8)	0.6967	0.7255	0.6524	0.6794
Laplacian (σ=0.4)	0.6655	0.6811	0.6615	0.6726
Sigmoid (γ=0.7, c=0)	0.6670	0.7036	0.6726	0.6854
Gaussian (σ=0.4)	0.7023	0.7114	0.6737	0.6792

F. Hyperparameters of SCDH$_K$

SCDH$_K$ (with the Gaussian kernel) has two essential hyperparameters: the number of randomly selected anchors Q and the kernel bandwidth σ.

Keeping all the other parameters fixed, we vary the number of anchors Q from 100 to 8,000 and plot the retrieval performance of SCDH$_K$ in Fig. 2. It can be observed on all the datasets that along with the increase of Q, SCDH$_K$'s MAP scores would get higher and higher. This is reasonable because a certain number of basis vectors (anchors) would be necessary to represent complex data samples well. However, we can also see that as Q becomes bigger, the performance gain is diminishing and the training time cost is rising. Throughout our experiments, Q is set to 2,000 which enables SCDH$_K$ to beat the state-of-the-art methods while being able to finish within just a couple of minutes on all the datasets.

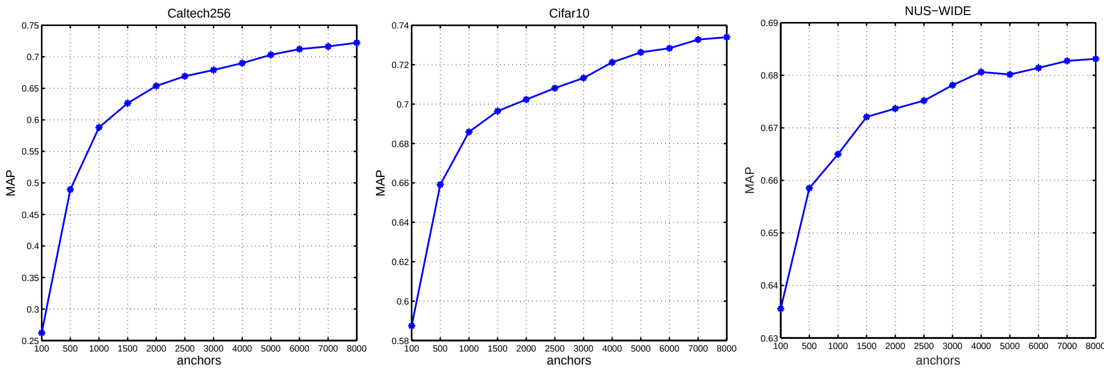
Keeping all the other parameters fixed, we vary the kernel bandwidth σ from 0.01 to 100 and plot the retrieval performance of SCDH$_K$ in Fig. 3. As can be seen clearly, SCDH$_K$ performs well on all the datasets when σ is between 0.3 and 1.0, though its optimal value for each dataset is slightly different from one another. Throughout our experiments, σ is set to 0.4 which can provide decent MAP scores across different datasets.

Furthermore, we explore the possibility of using different kernels other than the default Gaussian kernel in SCDH$_K$. Specifically, the popular kernels including linear, polynomial [47], Laplacian [48], Sigmoid [50] and Gaussian [51] as listed below have been compared empirically.

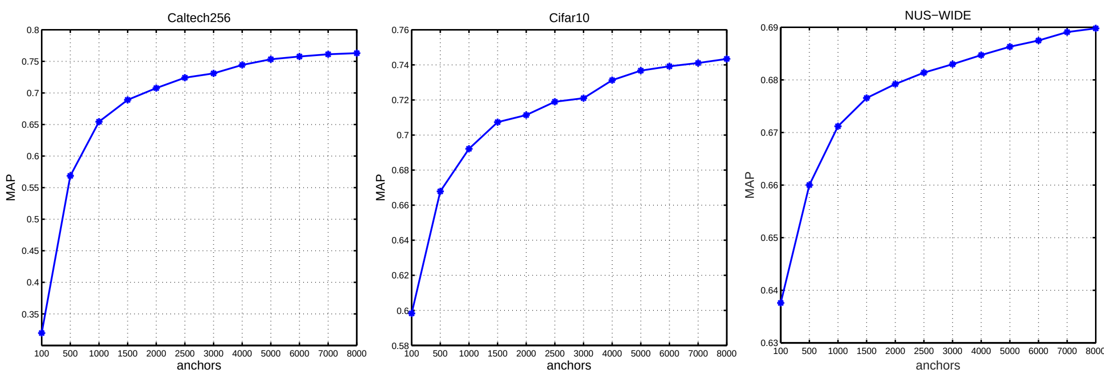
- Linear kernel: $\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y}$;
- Polynomial kernel: $\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \left(\alpha \boldsymbol{x}^T \boldsymbol{y} + c\right)^d$;
- Laplacian kernel: $\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{y}||}{2\sigma}\right)$;
- Sigmoid kernel: $\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \tanh\left(\gamma \boldsymbol{x}^T \boldsymbol{y} + c\right)$;
- Gaussian kernel: $\mathcal{K}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{y}||^2}{2\sigma^2}\right)$.

In our study, the kernel hyperparameters α and c are set to their default values 1 and 0 respectively[15]; the kernel hyperparameters d, σ and γ are tuned for their corresponding kernel functions, as shown in Fig. 4 and Fig. 3. The hyperparameter tuning curves for Laplacian, Sigmoid and Gaussian kernels exhibit similar patterns, while the polynomial kernel looks not

[15]We have also tried using many other values for α and c, but their best results are similar to those using the default values.



(a) 32 bits (b) 32 bits (c) 32 bits



(d) 64 bits (e) 64 bits (f) 64 bits

Figure 2: The MAP scores of SCDH$_K$ (Gaussian kernel) w.r.t. the number of anchors Q.

Table IV: The MAP scores of SCDH$_K$ with the constraints turned on ("✓") or off ("×").

balance	decorrelation	Cifar10 32 bits	Cifar10 64 bits	NUS-WIDE 32 bits	NUS-WIDE 64 bits
×	×	0.6504	0.6795	0.6001	0.6171
✓	×	0.6571	0.6869	0.6351	0.6586
×	✓	0.6607	0.6816	0.6603	0.6751
✓	✓	**0.7023**	**0.7114**	**0.6737**	**0.6792**

Table V: Retrieval performance: SCDH$_K$ vs. DPLM.

Methods/Length		32 bits MAP score	32 bits training time	64 bits MAP score	64 bits training time
Cifar10	DPLM	0.6671	17.903	0.6889	29.377
	SCDH$_K$	**0.7023**	**16.692**	**0.7114**	**26.673**
NUS-WIDE	DPLM	0.6703	95.792	0.6782	170.471
	SCDH$_K$	**0.6737**	**89.394**	**0.6792**	**155.057**

so stable. Accordingly, the best performances that could be achieved by these different kernels are summarized in Table III. It can be seen that (i) all the nonlinear kernels work apparently better than the linear kernel, and (ii) the nonlinear kernels produce somewhat similar performances. Overall, the Gaussian kernel (which has only one hyperparameter σ) seems to be slightly superior to the other kernels in terms of MAP scores. It is the kernel of choice for many nonlinear hashing methods such as KSH [16], SGH [12], FSDH [45], FSSH [44], and also our own SCDH$_K$.

G. Ablation Study

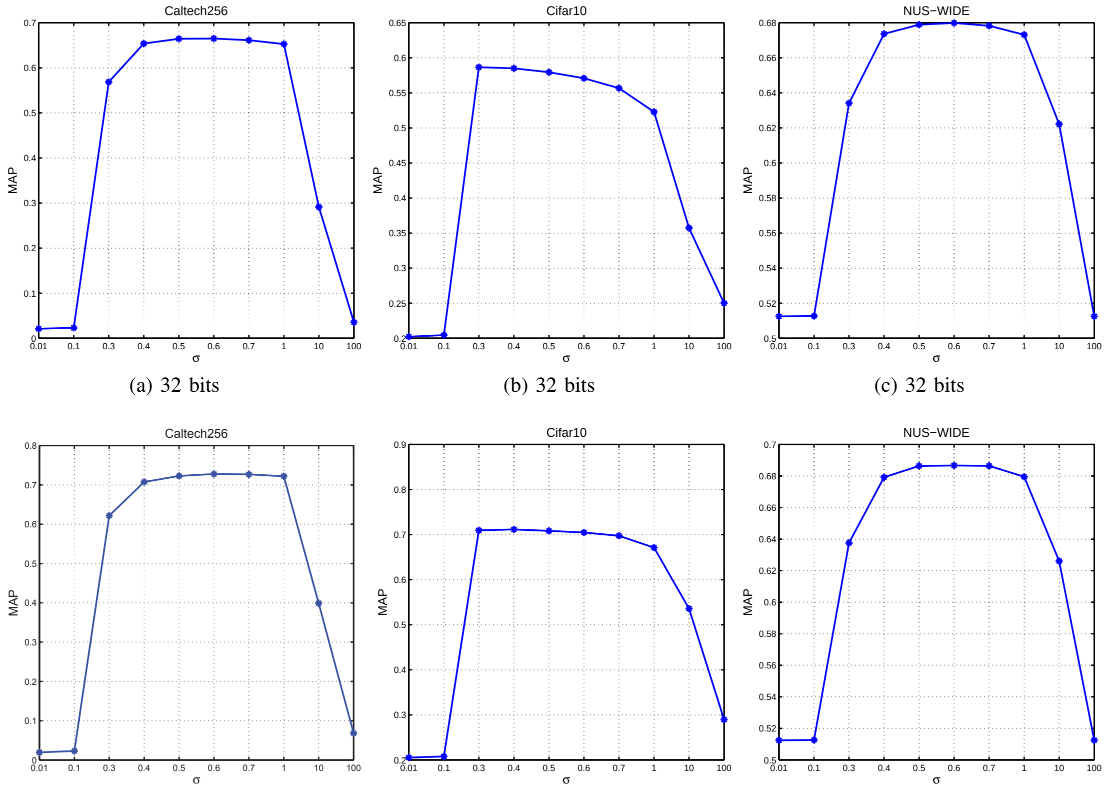
To investigate the contributions of the "balance" constraint ($\boldsymbol{B}^T\mathbf{1}_N = \mathbf{0}_K$) and the "decorrelation" constraint ($\boldsymbol{B}^T\boldsymbol{B} = N \cdot \boldsymbol{I}_K$) to our proposed SCDH$_K$, we conduct ablation study, i.e., we drop either constraint or both from SCDH$_K$ and solve the modified optimization problem. The results on Cifar10 and NUS-WIDE are collected in Table IV, from which some observations can be made.

- Using either constraint would be better than using neither of them, which means that they are both helpful.
- Between these two constraints, "decorrelation" seems to be more important than "balance" in the sense of providing more performance gains.
- Combining these two constraints would make the hashing method benefit from both of them and thus generate the best results.

To summarize, the "balance" and "decorrelation" constraints which have often been ignored due to the optimization difficulty, can indeed make great improvements to hashing for large-scale image retrieval.

H. Constraints vs. Regularizers

Recall that the proposed SCDH$_K$ model (12) has been addressed in Section V with a closed-form solution to each sub-



(a) 32 bits (b) 32 bits (c) 32 bits
(d) 64 bits (e) 64 bits (f) 64 bits

Figure 3: The MAP scores of SCDH$_K$ (Gaussian kernel) w.r.t. the kernel bandwidth σ.

problem of optimization. Actually, it is also possible to tackle this optimization problem by converting the hard "balance" and "decorrelation" constraints into two extra regularizers in the objective function, as in DPLM [46]. To further understand those two different ways of incorporating "balance" and "decorrelation" into hashing, we make empirical comparisons between our SCDH$_K$ (that uses hard constraints) and DPLM (that uses soft regularizers). For SCDH$_K$, the hyperparameter settings have been explained in Sections VI-C and VI-F. For DPLM, the hyperparameters have been tuned to get the best possible performance. As shown in Table V, SCDH$_K$ has not only higher MAP scores but also lower time costs than DPLM, on both Cifar10 and NUS-WIDE. It turns out that we do not really have to sacrifice effectiveness for efficiency (by converting those two strong constraints into regularizers), thanks to our Algorithm 1 based on closed-form solutions.

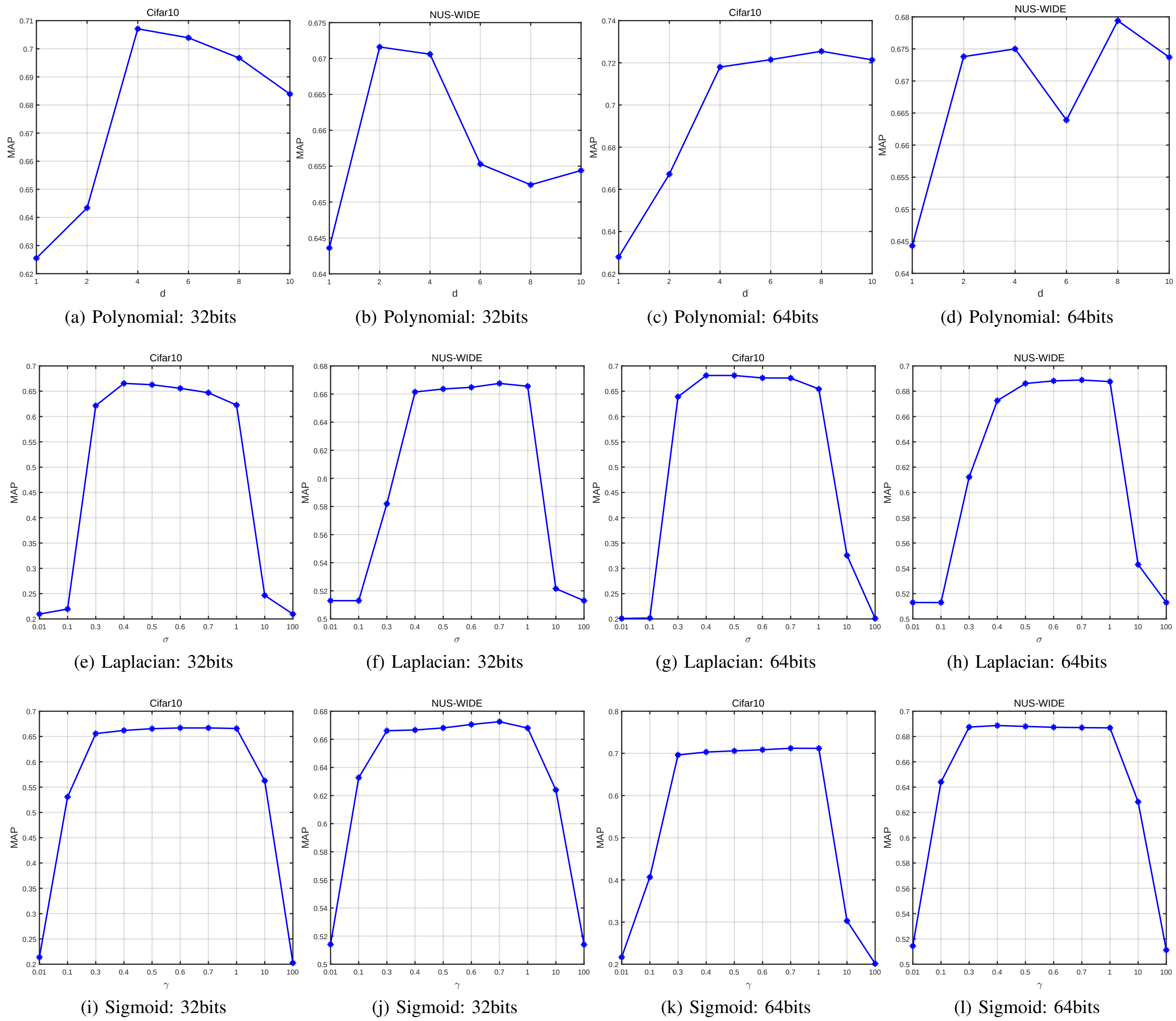
I. Shallow vs. Deep

This paper mainly focuses on exploiting the "balance" and "decorrelation" constraints in the hashing methods that are not based on deep learning (which typically require enormous computing power like GPU clusters). Nevertheless, we are curious about how our proposed SCDH/SCDH$_K$ would compete against the so-called deep hashing methods that have emerged in the last few years, such as DeepBit [55], [56], SADH [57], and DPSH [21]. Table VI shows the comparison between such deep hashing methods and the shallow hashing method SCDH/SCDH$_K$ on Cifar10 and NUS-WIDE. It is

Table VI: Retrieval performance: SCDH($_K$) vs. Deep Hashing.

Methods/Length		32 bits MAP score	32 bits training time	64 bits MAP score	64 bits training time
Cifar10	DeepBit	0.1875	7731.449	0.1969	9671.132
	SADH	0.3147	9150.755	0.3308	10981.705
	DPSH	**0.7037**	10795.150	**0.7261**	12946.728
	SCDH	0.6116	9.901	0.6376	5.888
	SCDH$_K$	0.7023	16.692	0.7114	12.407
NUS-WIDE	DeepBit	0.4092	12211.901	0.4203	14903.540
	SADH	0.4564	15462.948	0.4732	19432.642
	DPSH	**0.7275**	16397.246	**0.7383**	19390.623
	SCDH	0.6553	38.028	0.6561	96.658
	SCDH$_K$	0.6737	89.394	0.6792	155.057

obvious that all the deep hashing methods are several orders of magnitude slower than SCDH/SCDH$_K$. Moreover, the two deep hashing methods DeepBit and SADH actually get lower MAP scores than SCDH/SCDH$_K$, which is probably because they are unsupervised while SCDH/SCDH$_K$ is supervised. The deep hashing method DPSH which is supervised does outperform SCDH/SCDH$_K$ in terms of MAP scores, though it requires significantly more training time than SCDH/SCDH$_K$. This demonstrates the superior ability of deep neural networks in fitting complex data. It may be possible to utilize a deep neural network instead of the kernel trick to enable SCDH for nonlinear hashing, which is a research problem to be investigated in the future.



(a) Polynomial: 32bits (b) Polynomial: 32bits (c) Polynomial: 64bits (d) Polynomial: 64bits

(e) Laplacian: 32bits (f) Laplacian: 32bits (g) Laplacian: 64bits (h) Laplacian: 64bits

(i) Sigmoid: 32bits (j) Sigmoid: 32bits (k) Sigmoid: 64bits (l) Sigmoid: 64bits

Figure 4: The MAP scores of SCDH$_K$ w.r.t. different kernels (except the Gaussian kernel that has been shown in Fig. 3).

Table VII: Retrieval performance: “Binary” vs. “Real-valued”.

Methods/Length		32 bits		64 bits	
		P@1000	retr. time	P@1000	retr. time
Cifar10	SCDH$_K$	0.7162	**0.012**	**0.7275**	**0.022**
	Real+BF	**0.7214**	28.37	0.7269	29.102
	Real+HI	0.6291	0.074	0.6376	0.129
	Real+GH	0.6332	0.037	0.6357	0.044
	Real+PQ	0.7106	4.614	0.7133	5.521
NUS-WIDE	SCDH$_K$	0.6982	**0.005**	0.7053	**0.006**
	Real+BF	**0.7034**	28.425	**0.7068**	29.380
	Real+HI	0.6342	0.147	0.6445	0.177
	Real+GH	0.6277	0.117	0.6361	0.157
	Real+PQ	0.6847	4.876	0.6863	5.869

J. Binary vs. Real-valued

Can we just use real-valued vectors rather than binary codes for our image retrieval application? In what follows, we construct a “real-valued” version of SCDH$_K$ and compare it with the original “binary” SCDH$_K$. Specifically, the real-valued model is made by removing the binary constraint from SCDH$_K$ and represent each data sample with not a binary code but a real-valued vector.

As discussed in [49], there exist several nearest-neighbor search strategies including brute-force (BF), hash index (HI), grouped hamming ranking (GH), and product quantization (PQ) [58]). The brute-force search strategy should be the most accurate but also the slowest, while the other three search strategies are approximate ones that accelerate the retrieval process in different ways. The standard binary SCDH$_K$ simply uses the hash index search approach (based on hamming ranking) as in most learning to hash papers. For the real-valued model, we combine it with each of the above mentioned four

Table VIII: The MAP scores and training time costs (in seconds) of different hashing methods, for unseen classes.

(a) Caltech256 {Train:Test = 70% classes : 30% classes}

Methods /Length	8 bits		16 bits		32 bits		64 bits	
	MAP score	training time	MAP score	training time	MAP score	training time	MAP score	training time
LSH	0.0154	0.002	0.0332	0.003	0.0830	0.004	0.1431	0.005
PCAH	0.0548	0.493	0.1009	0.721	0.1586	1.108	0.2088	1.520
ITQ	0.0750	1.094	0.1327	2.174	0.2232	4.100	0.2981	6.184
DGH	0.0388	36.002	0.0834	36.649	0.1131	51.957	0.1914	80.088
SGH	0.0555	3.255	0.1153	3.721	0.1974	3.957	0.2608	4.573
CCA-ITQ	0.0831	2.819	0.1916	5.515	0.2657	7.784	0.3564	14.344
SDH	0.1235	11.341	0.2556	20.128	0.3184	31.998	0.3591	56.078
FSDH	0.1287	4.159	0.2289	6.335	0.3109	10.027	0.4066	18.889
FastH	0.1484	251.928	0.3233	292.161	0.4442	449.008	0.565	790.680
COSDISH	0.0999	4.822	0.1919	7.720	0.3440	17.490	0.4927	56.555
FSSH	0.1163	17.085	<u>0.3791</u>	17.722	<u>0.5197</u>	18.181	<u>0.5783</u>	20.421
SCDH	<u>0.1584</u>	3.525	0.2978	5.245	0.4123	6.191	0.5424	6.986
SCDH$_K$	**0.276**	9.291	**0.4623**	14.033	**0.5992**	15.576	**0.6303**	17.102

(b) Cifar10 {Train:Test = 70% classes : 30% classes}

Methods /Length	8 bits		16 bits		32 bits		64 bits	
	MAP score	training time	MAP score	training time	MAP score	training time	MAP score	training time
LSH	0.0871	0.001	0.0971	0.001	0.1160	0.001	0.1222	0.002
PCAH	0.0958	0.258	0.0926	0.261	0.1034	0.313	0.1066	0.482
ITQ	0.1100	0.853	0.1291	1.346	0.1400	3.574	0.1445	6.277
DGH	0.0916	71.532	0.0893	78.963	0.0945	88.605	0.1016	114.303
SGH	0.1099	1.652	0.1059	2.036	0.1142	3.023	0.1156	6.542
CCA-ITQ	0.1607	1.718	0.1653	3.489	0.2188	5.215	0.2197	8.989
SDH	0.2002	6.833	0.2045	15.385	0.2608	19.804	0.2849	33.855
FSDH	0.1781	3.156	0.2292	4.995	0.2776	6.180	0.2778	7.443
HC-SDH	n/a	n/a	0.4205	3.229	0.4270	3.765	0.4295	3.803
FastH	0.3237	378.261	0.4324	541.176	0.5067	812.296	0.5149	1662.214
COSDISH	0.2288	5.119	0.2756	7.867	0.3607	22.784	0.3846	99.903
FSSH	<u>0.4331</u>	63.881	<u>0.4861</u>	67.397	<u>0.5308</u>	70.957	<u>0.5597</u>	74.212
SCDH	0.3730	3.091	0.3992	4.138	0.4984	6.187	0.5108	7.020
SCDH$_K$	**0.4450**	7.704	**0.5126**	8.270	**0.5609**	10.314	**0.5807**	11.950

(c) NUS-WIDE {Train:Test = 70% concepts : 30% concepts}

Methods /Length	8 bits		16 bits		32 bits		64 bits	
	MAP score	training time	MAP score	training time	MAP score	training time	MAP score	training time
LSH	0.2039	0.156	0.2081	0.559	0.2176	0.753	0.2176	0.771
PCAH	0.2412	0.890	0.2572	1.703	0.2588	2.049	0.2613	2.558
ITQ	0.2576	3.057	0.2580	7.021	0.2638	12.098	0.2647	17.291
DGH	0.2068	398.513	0.2148	505.936	0.2155	799.486	0.2187	916.381
SGH	0.2147	8.891	0.2223	12.411	0.2274	39.229	0.2344	143.068
CCA-ITQ	0.3048	6.464	0.3380	8.001	0.3448	17.176	0.3691	27.981
SDH	0.2859	25.959	0.3204	35.711	0.3302	58.444	0.3399	138.303
FSDH	0.3144	7.464	0.3257	11.685	0.3352	27.895	0.3605	44.902
FastH	—	—	—	—	—	—	—	—
COSDISH	0.3662	11.422	0.3777	25.042	0.3829	90.281	0.3914	282.240
FSSH	0.4100	675.187	<u>0.4268</u>	685.416	0.4302	696.424	0.4381	718.315
SCDH	<u>0.4136</u>	6.219	0.4247	11.408	<u>0.4354</u>	26.034	<u>0.4397</u>	59.489
SCDH$_K$	**0.4326**	37.024	**0.4380**	45.143	**0.4515**	58.865	**0.4587**	105.689

search strategies, i.e., "+BF", "+HI", "+GH" and "+PQ"[16]. The results on Cifar10 and NUS-WIDE are reported in Table VII, where the effectiveness is measured by the precision of the top-1000 search results (P@1000) on the test set, and the efficiency is measured by the retrieval time (in seconds). It can be seen that the standard binary SCDH$_K$ reaches similarly high P@1000 scores as the thorough brute-force search strategy, but only with a fraction of the retrieval time. Moreover, by enforcing the binary constraint and directly optimizing the binary codes to represent the data, SCDH$_K$ outperforms the real-valued model using any of the other three search strategies.

[16]For "+HI" and "+GH", the number of clusters and the number of candidates are set to 10 and 2000 respectively. For "+PQ", the CPU version of the faiss [58] implementation is employed, for a fair comparison.

K. Unseen Classes

The experimental results in Table II are obtained under the traditional configuration where each class has some examples for training and some examples for testing, as in most learning to hash papers [14], [18], [17], [52], [26], [24], [45], [44]. However, a new evaluation protocol *"retrieval of unseen classes"* [53] has recently been proposed to measure the

(a) “Killer Whale” (b) “Bowling Ball” (c) “Homer Simpson”

Figure 5: The three randomly selected queries for our image retrieval case study. Their top-20 search results on Caltech256 are shown in Figs. 6 , 7 and 8 respectively.

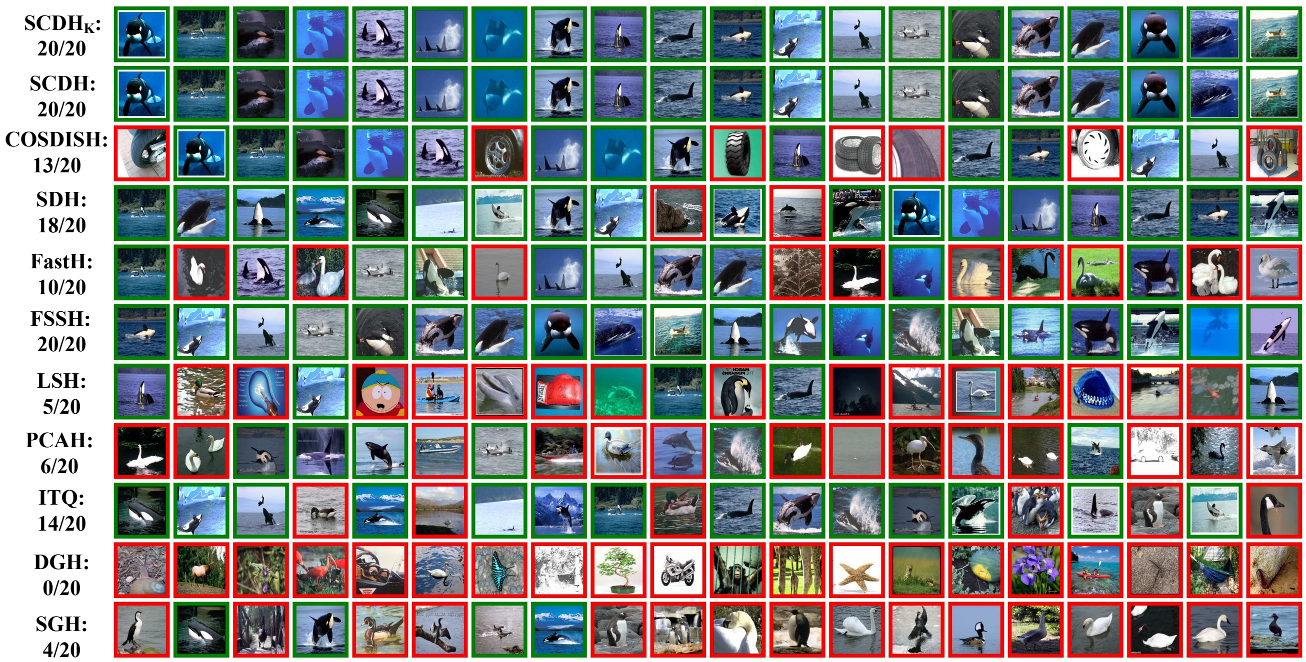


Figure 6: Retrieval results: “Killer Whale” (the bounding boxes are green for correct results and red for wrong ones).

generalization ability of the learned hash functions on unseen classes (i.e., the classes not appeared in the training stage at all).

Following the configuration in [53], [54], we randomly select about 70% of the classes and use their examples to learn the hash functions, while the examples in the rest 30% classes are reserved for the purpose of evaluation only. Specifically, on Caltech256, we have 180 classes for training and the other 76 classes for testing; on Cifar10, we have 7 classes for training and the other 3 classes for testing; on the multi-labeled dataset NUS-WIDE, the examples labeled by at least one of the selected 7 classes are used for training and the remaining examples are used for testing. Under the same settings as described before in Section VI-C, we conduct experiments on the retrieval of unseen classes and report the results in Table VIII. Similar to the previous experimental results, SCDH/SCDH$_K$ demonstrates not only higher effectiveness (in terms of MAP scores) than all the other hashing methods in comparison but also higher efficiency (in terms of training speeds) than the supervised ones among them.

L. Case Study

Here we examine the top-20 image retrieval results for three randomly selected queries — “Killer Whale” (Fig. 5a), “Bowling Ball” (Fig. 5b) and “Homer Simpson” (Fig. 5c) —- on Caltech256 (as described in Section VI-A), using different hashing methods under our investigation, with the code length set to 32 bits.

In Figs. 6 , 7 and 8, the first six rows correspond to six supervised methods while the remaining five rows correspond to five unsupervised methods. Overall, the supervised methods perform far better than the unsupervised methods. In particular, our proposed method SCDH$_K$ could achieve 100% accuracy (20/20) for every given query, significantly outperforming other methods such as COSDISH, SDH and FastH. Furthermore, the performances of SCDH/SCDH$_K$ are more stable than those of the other methods across different queries.

Specifically, in Fig. 6, both SCDH and SCDH$_K$ could recognize “Killer Whale” perfectly under different color backgrounds while the other methods would make some mistakes. For example, the competitive method COSDISH

Figure 7: Retrieval results: “Bowling Ball” (the bounding boxes are green for correct results and red for wrong ones).

Figure 8: Retrieval results: “Homer Simpson” (the bounding boxes are green for correct results and red for wrong ones).

often confuses tires with killer whales; FastH and SGH often incorrectly returns swans that are similar to killer whales from the appearance. In Fig. 7, SCDH$_K$ successfully tells the difference between “Bowling Ball” and other ball-like objects, but the other methods including COSDISH, ITQ, and SGH often fail to distinguish them and thus perform badly. In Fig. 8, SCDH/SCDH$_K$ again could reach 100% accuracy, but COSDISH would collapse: it could not find any image of “Homer Simpson” at all.

It is worth mentioning that FSSH, probably the strongest baseline method, also performs well for the three given queries. Nevertheless, FSSH is still slightly inferior to SCDH$_K$ in the case of “Bowling Ball” (Fig. 7), which reflects the outstanding ability of our proposed methods.

In summary, these three concrete queries have intuitively illustrated the substantial performance improvements that SCDH/SCDH$_K$ could make on existing methods for large-scale image retrieval.

VII. Conclusion

In this paper, we improve supervised discrete hashing by maintaining two strong constraints (balance and decorrelation of hash bits) and propose a fast optimization algorithm for it. Although such constraints are known to be beneficial for hashing in previous studies, to our knowledge this is the first time that the hard discrete optimization problem with all those constraints is shown to have efficient solutions. The developed algorithm SCDH, and its kernelized variant SCDH_K, can learn the binary codes and the hash function from labelled data simultaneously. They have been demonstrated to outperform state-of-the-art supervised learning to hash methods for large-scale image retrieval in terms of both MAP scores and training speeds.

Acknowledgment

Thanks to the support of China Scholarship Council (CSC), Yong has been funded as a visiting PhD student at Birkbeck and UCL from 01/2018 to 01/2019. This work is also supported in part by China Postdoctoral Science Foundation.

References

[1] Ruslan Salakhutdinov and Geoffrey E. Hinton, *Semantic Hashing*. In Int. J. Approx. Reasoning, 969–978, 2009. 1

[2] Aristides Gionis, Piotr Indyk and Rajeev Motwani, *Similarity Search in High Dimensions via Hashing*. In VLDB, 518–529, 1999. 1, 6

[3] Yunchao Gong, Svetlana Lazebnik, Albert Gordo and Florent Perronnin, *Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval*. In IEEE Trans. on Pattern Anal. Mach. Intell., 2916–2929, 2013. 1, 6

[4] Xi Li, Guosheng Lin, Chunhua Shen, Anton van den Hengel and Anthony R. Dick, *Learning Hash Functions Using Column Generation*. In ICML, 142–150, 2013.

[5] Jingkuan Song, Yang Yang, Yi Yang, Zi Huang and Heng Tao Shen, *Inter-media Hashing for Large-Scale Retrieval from Heterogeneous Data Sources*. In SIGMOD, 785–796, 2013. 1

[6] Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola and Josh Attenberg, *Feature Hashing for Large Scale Multitask Learning*. In ICML, 1113–1120, 2009. 1

[7] Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P. Drake, Jane M. Landolin and Adam M. Phillippy, *Assembling Large Genomes with Single-Molecule Sequencing and Locality-Sensitive Hashing*. In Nature Biotechnology, 623–630, 2015. 1

[8] Baris Coskun, Bulent Sankur and Nasir D. Memon, *Spatio-Temporal Transform Based Video Hashing*. In IEEE Trans. on Multimedia, 1190–1208, 2006. 1

[9] Yair Weiss, Antonio Torralba and Robert Fergus, *Spectral Hashing*. In NIPS, 1753–1760, 2008. 1, 2, 3, 6

[10] Wei Liu, Jun Wang, Sanjiv Kumar and Shih-Fu Chang, *Hashing with Graphs*. In ICML, 1–8, 2011. 1, 2

[11] Wei Liu, Cun Mu, Sanjiv Kumar and Shih-Fu Chang, *Discrete Graph Hashing*. In NIPS, 3419–3427, 2014. 1, 3, 5, 6

[12] Qing-Yuan Jiang and Wu-Jun Li, *Scalable Graph Hashing with Feature Transformation*. In IJCAI, 2248–2254, 2015. 1, 6, 9

[13] Dell Zhang, Jun Wang, Deng Cai and Jinsong Lu, *Self-Taught Hashing for Fast Similarity Search*. In SIGIR, 18–25, 2010. 1, 2, 3

[14] Jun Wang, Ondrej Kumar and Shih-Fu Chang, *Semi-Supervised Hashing for Scalable Image Retrieval*. In CVPR, 3424–3431, 2010. 1, 2, 12

[15] Mohammad Norouzi and David J. Fleet, *Minimal Loss Hashing for Compact Binary Codes*. In ICML, 353–360, 2011. 1

[16] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang and Shih-Fu Chang, *Supervised Hashing with Kernels*. In CVPR, 2074–2081, 2012. 1, 2, 4, 6, 9

[17] Fumin Shen, Chunhua Shen, Wei Liu and Heng Tao Shen, *Supervised Discrete Hashing*. In CVPR, 37–45, 2015. 2, 6, 12

[18] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel and David Suter, *Fast Supervised Hashing with Decision Trees for High-Dimensional Data*. In CVPR, 1971–1978, 2014. 1, 4, 6, 12

[19] Tiezheng Ge, Kaiming He and Jian Sun, *Graph Cuts for Supervised Binary Coding*. In ECCV, 250–264, 2014. 1

[20] Guosheng Lin, Chunhua Shen, David Suter and Anton van den Hengel, *A General Two-Step Approach to Learning-Based Hashing*. In ICCV, 2552–2559, 2013. 1

[21] Wu-Jun Li, Sheng Wang and Wang-Cheng Kang, *Feature Learning Based Deep Supervised Hashing with Pairwise Labels*. In IJCAI, 1711–1717, 2016. 1, 2, 10

[22] Guiguang Ding, Yuchen Guo and Jile Zhou, *Collective Matrix Factorization Hashing for Multimodal Data*. In CVPR, 2083–2090, 2014. 1

[23] Gou Koutaki, Keiichiro Shirai and Mitsuru Ambai, *Hadamard Coding for Supervised Discrete Hashing*. In IEEE Trans. Image Processing, 5378–5392, 2018. 6, 7

[24] Sen Su, Gang Chen, Xiang Cheng and Rong Bi, *Deep Supervised Hashing with Nonlinear Projections*. In IJCAI, 2786–2792, 2017. 1, 6, 12

[25] Qing-Yuan Jiang and Wu-Jun Li, *Asymmetric Deep Supervised Hashing*. In AAAI, 3342–3349, 2018. 1

[26] Wang-Cheng Kang, Wu-Jun Li and Zhi-Hua Zhou, *Column Sampling Based Discrete Supervised Hashing*. In AAAI, 1230–1236, 2016. 2, 3, 6, 12

[27] Griffin Gregory, Holub Alex and Perona Pietro, *Caltech-256 Object Category Dataset*. In California Institute of Technology, 2007. 6

[28] Alex Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*. In Master's thesis, Department of Computer Science, University of Toronto, 2009. 6

[29] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*. In NIPS, 1106–1114, 2012.

[30] Peichao Zhang, Wei Zhang, Wu-Jun Li and Minyi Guo, *Supervised Hashing with Latent Factor Models*. In SIGIR, 173–182, 2014. 6

[31] Yunchao Gong and Svetlana Lazebnik, *Iterative Quantization: A Procrustean Approach to Learning Binary Codes*. In CVPR, 817–824, 2011. 6

[32] Zhongming Jin, Yao Hu, Yue Lin, Debing Zhang, Shiding Lin, Deng Cai and Xuelong Li, *Complementary Projection Hashing*. In ICCV, 257–264, 2013. 6

[33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li and Fei-Fei Li, *ImageNet: A Large-Scale Hierarchical Image Database*. In CVPR, 248–255, 2009.

[34] Ruimao Zhang, Liang Lin, Rui Zhang, Wangmeng Zuo and Lei Zhang, *Bit-Scalable Deep Hashing With Regularized Similarity Learning for Image Retrieval and Person Re-Identification*. In IEEE Trans. on Image Processing, 4766–4779, 2015. 1

[35] Brian Kulis and Kristen Grauman, *Kernelized Locality-Sensitive Hashing for Scalable Image Search*. In ICCV, 2130–2137, 2009. 4

[36] Ke Jiang, Qichao Que and Brian Kulis, *Revisiting Kernelized Locality-Sensitive Hashing for Improved Large-Scale Image Retrieval*. In CVPR, 4933–4941, 2015. 4

[37] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo and Yantao Zheng, *NUS-WIDE: A Real-World Web Image Database from National University of Singapore*. In CIVR, 2009. 6

[38] Zijia Lin, Guiguang Ding, Mingqing Hu and Jianmin Wang, *Semantics-Preserving Hashing for Cross-View Retrieval*. In CVPR, 3864–3872, 2015. 6

[39] Dongqing Zhang and Wu-Jun Li, *Large-Scale Supervised Multimodal Hashing with Semantic Correlation Maximization*. In AAAI, 2177–2183, 2014. 3

[40] Ning Li, Chao Li, Cheng Deng, Xianglong Liu and Xinbo Gao, *Deep Joint Semantic-Embedding Hashing*. In IJCAI, 2397–2403, 2018. 2

[41] Weiming Hu, Yabo Fan, Junliang Xing, Liang Sun, Zhaoquan Cai and Stephen J. Maybank, *Deep Constrained Siamese Hash Coding Network and Load-Balanced Locality-Sensitive Hashing for Near Duplicate Image Detection*. In IEEE Trans. on Image Processing, 4452–4464, 2018. 2

[42] Fumin Shen, Xin Gao, Li Liu, Yang Yang and Heng Tao Shen, *Deep Asymmetric Pairwise Hashing*. In ACM Multimedia, 1522–1530, 2017. 2

[43] Haomiao Liu, Ruiping Wang, Shiguang Shan and Xilin Chen, *Deep Supervised Hashing for Fast Image Retrieval*. In CVPR, 2064–2072, 2016. 2

[44] Xin Luo, Liqiang Nie, Xiangnan He, Ye Wu, Zhen-Duo Chen and Xin-Shun Xu, *Fast Scalable Supervised Hashing*. In SIGIR, 735–744, 2018. 2, 6, 9, 12

[45] Jie Gui, Tongliang Liu, Zhenan Sun, Dacheng Tao and Tieniu Tan, *Fast Supervised Discrete Hashing*. In IEEE Trans. on Pattern Anal. Mach. Intell., 490–496, 2018. 2, 6, 9, 12

[46] Fumin Shen, Xiang Zhou, Yang Yang, Jingkuan Song, Heng Tao Shen and Dacheng Tao, *A Fast Optimization Method for General Binary Code Learning*. In IEEE Trans. on Image Processing, 5610–5621, 2016. 2, 3, 10

[47] Shaobo Lin and Jinshan Zeng, *Fast Learning With Polynomial Kernels*. In IEEE Trans. Cybernetics, 3780–3792, 2019. 8

[48] Mika Sayama and Taro Tezuka, *Fisher Laplacian Kernel for Image Analysis*. In IDT/IIMSS/STET, 377–384, 2014. 8

[49] Deng Cai, *A Revisit of Hashing Algorithms for Approximate Nearest Neighbor Search*. In CoRR abs/1612.07545, 1–14, 2016. 11

[50] Thomas Ricatte, Gemma C. Garriga, Rémi Gilleron and Marc Tommasi, *Learning from Multiple Graphs Using a Sigmoid Kernel*. In ICMLA, 140–145, 2013. 8

[51] Frances Y. Kuo, Ian H. Sloan and Henryk Wozniakowski, *Multivariate integration for analytic functions with Gaussian kernels*. In Math. Comput., 829–853, 2017. 8

[52] Thanh-Toan Do, Anh-Dzung Doan and Ngai-Man Cheung, *Learning to Hash with Binary Deep Neural Network*. In ECCV, 219–234, 2016. 2, 3, 12

[53] Alexandre Sablayrolles, Matthijs Douze, Nicolas Usunier and Herve Jegou, *How should we evaluate supervised hashing?*. In ICASSP, 1732–1736, 2017. 12, 13

[54] Thanh-Toan Do, Khoa Le, Tuan Hoang, Huu Le, Tam V. Nguyen and Ngai-Man Cheung, *Simultaneous Feature Aggregating and Hashing for Compact Binary Code Learning*. In IEEE Trans. Image Processing, 4954–4969, 2019. 13

[55] Kevin Lin, Jiwen Lu, Chu-Song Chen and Jie Zhou, *Learning Compact Binary Descriptors with Unsupervised Deep Neural Networks*. In CVPR, 1183–1192, 2016. 10

[56] Kevin Lin, Jiwen Lu, Chu-Song Chen, Jie Zhou and Ming-Ting Sun, *Unsupervised Deep Learning of Compact Binary Descriptors*. In IEEE Trans. Pattern Anal. Mach. Intell., 1501–1514, 2019. 10

[57] Fumin Shen, Yan Xu, Li Liu, Yang Yang, Zi Huang and Heng Tao Shen, *Unsupervised Deep Hashing with Similarity-Adaptive and Discrete Optimization*. In IEEE Trans. Pattern Anal. Mach. Intell., 3034–3044, 2018. 10

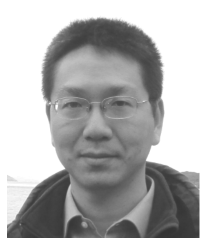
[58] Jeff Johnson, Matthijs Douze and Hervé Jégou, *Billion-scale similarity search with GPUs*. In CoRR abs/1702.08734, 1–12, 2017. 11, 12

Yong Chen received his Ph.D. in Computer Science and Engineering from Beihang University (BUAA), Beijing, China, in 2019. He is currently working as a "Boya" Postdoc in the Key Lab of Machine Perception, School of Electronics Engineering and Computer Science, Peking University, Beijing, China. He has been funded as a visiting Ph.D. student at Birkbeck and UCL from January 2018 to January 2019. His research interests include machine learning, data mining, big data and numerical optimization. For more information, please refer to https://scholar.google.com/citations?user=bakW4s4AAAAJ&hl=zh-CN.

Zhibao Tian received B.Sc. degree from computer science and technology, China University of Geosciences, Beijing, PR China, in 2017. Now he is a master student in the State Key Lab of Software Development Environment, School of Computer Science and Engineering, Beihang University, Beijing, China. He received gold award at the 40th ACM/ICPC International College Student Programming Contest, Asia Changchun Station during his undergraduate period. His main research interests include machine learning, data mining, and big data, especially information retrieval and recommender system for large-scale S&T resources.

Hui Zhang received the M.S. and Ph.D. degrees in computer science from Beihang University, Beijing, China, in 1994 and 2009, respectively. He is a Professor and also the Deputy Director at State Key Laboratory of Software Development Environment (SKLSDE), School of Computer Science and Engineering, Beihang University. He had been working in the University of Chicago and Argonne National Laboratory, Chicago, IL, USA, from 2007 to 2008 as a Guest Researcher. His main research interests include e-science archives management, data mining, and information retrieval.

Jun Wang is a Professor in Computer Science from the University College London and the Founding Director of the M.Sc. Web Science and Big Data Analytics. Prof. Jun Wang's main research interests are in the areas of AI and intelligent systems, including (multiagent) reinforcement learning, deep generative models, and their diverse applications on information retrieval, recommender systems and personalization, data mining, smart cities, bot planning, computational advertising etc. His team won the first global real-time bidding algorithm contest with 80+ participants worldwide. Jun has published over 100 research papers and is a winner of multiple "Best Paper" awards. He was a recipient of the Beyond Search – Semantic Computing and Internet Economics award by Microsoft Research and also received Yahoo! FREP Faculty award. He has served as an Area Chair in ACM CIKM and ACM SIGIR. His recent service includes co-chair of Artificial Intelligence, Semantics, and Dialog in ACM SIGIR 2018. For more information, please refer to http://www0.cs.ucl.ac.uk/staff/Jun.Wang/bio.html.



Dell Zhang is a Reader in Computer Science at Birkbeck, University of London (UoL), a Senior Member of ACM, a Senior Member of IEEE, and a Fellow of RSS. He is currently on leave from Birkbeck and working for Blue Prism AI Labs. He got his PhD from Southeast University, Nanjing, China, and then worked as a Research Fellow at the Singapore-MIT Alliance (SMA) until he moved to the UK in 2005. His research interests include natural language processing, information retrieval, and machine learning. He has published more than 100 research articles, received multiple best paper awards, and won prizes from several data science competitions. For more information, please refer to http://www.dcs.bbk.ac.uk/~dell/.