A Scalable Optimization Mechanism for Pairwise based Discrete Hashing

Xiaoshuang Shi, Fuyong Xing, Zizhao Zhang, Manish Sapkota, Zhenhua Guo, and Lin Yang

Abstract—Maintaining the pairwise relationship among originally high-dimensional data into a low-dimensional binary space is a popular strategy to learn binary codes. One simple and intuitive method is to utilize two identical code matrices produced by hash functions to approximate a pairwise real label matrix. However, the resulting quartic problem in term of hash functions is difficult to directly solve due to the non-convex and non-smooth nature of the objective. In this paper, unlike previous optimization methods using various relaxation strategies, we aim to directly solve the original quartic problem using a novel alternative optimization mechanism to linearize the quartic problem by introducing a linear regression model. Additionally, we find that gradually learning each batch of binary codes in a sequential mode, i.e. batch by batch, is greatly beneficial to the convergence of binary code learning. Based on this significant discovery and the proposed strategy, we introduce a scalable symmetric discrete hashing algorithm that gradually and smoothly updates each batch of binary codes. To further improve the smoothness, we also propose a greedy symmetric discrete hashing algorithm to update each bit of batch binary codes. Moreover, we extend the proposed optimization mechanism to solve the non-convex optimization problems for binary code learning in many other pairwise based hashing algorithms. Extensive experiments on benchmark single-label and multi-label databases demonstrate the superior performance of the proposed mechanism over recent state-of-the-art methods on two kinds of retrieval tasks: similarity and ranking order. *All source codes will be available online.*

1 Introduction

Hashing has become a popular tool to tackle large-scale tasks in information retrieval, computer vision and machine leaning communities, since it aims to encode originally high-dimensional data into a variety of compact binary codes with maintaining the similarity between neighbors, leading to significant gains in both computation and storage [1] [2] [3].

Early endeavors in hashing focus on data-independent algorithms, like locality sensitive hashing (LSH) [4] and min-wise hashing (MinHash) [5] [6]. They construct hash functions by using random projections or permutations. However, due to randomized hashing, in practice they usually require long bits to achieve high precision per hash table and multiple tables to boost the recall. To learn compact binary codes, data-dependent algorithms using available training data to learn hash functions have attracted increasing attention. Based on whether utilizing semantic label information, data-dependent algorithms can be categorized into two main groups: unsupervised and supervised. Unlike unsupervised hashing [7] [8] [9] that explores data intrinsic structures to preserve similarity relations between neighbors without any supervision, supervised hashing [10] [11] [12] employs semantic information to learn hash functions, and thus it usually achieves better retrieval accuracy than unsupervised hashing on semantic similarity measures.

Among supervised hashing algorithms, pairwise based hashing, maintaining the relationship of similar or dissimilar pairs in a Hamming space, is one popular manner to exploit label information. Numerous pairwise based algorithms have been proposed in the past decade, including spectral hashing (SH) [7], minimal loss hashing (MLH) [10], binary reconstruction embedding (BRE) [8] and kernel-based supervised hashing (KSH) [13], etc. Although these algorithms have been demonstrated effective in many large-scale tasks, their employed optimization strategies are usually insufficient to explore the similarity information defined in the non-convex and non-differential objective functions. In order to handle these non-smooth and non-convex problems, four main strategies have been proposed: symmetric/asymmetric relaxation, and asymmetric/symmetric discrete. Symmetric relaxation [7] [11] [13] [14] is to relax discrete binary vectors in a continuous feasible region followed by thresholding to obtain binary codes. Although symmetric relaxation can simplify the original optimization problem, it often generates large accumulated errors between hash and linear functions. To reduce the accumulated error, asymmetric relaxation [15] utilizes the element-wise product of discrete and its relaxed continuous matrices to approximate a pairwise label matrix. Asymmetric discrete hashing [16] [17] [18] usually utilizes the product of two distinct discrete matrices to preserve pair relations into a binary space. Symmetric discrete hashing [19] [20] firstly learns binary codes with preserving symmetric discrete constraints and then trains classifiers based on the learned discrete codes. Although most of hashing algorithms with these four strategies have achieved promising performance, they have at least one of the following four major disadvantages: (i)

X. Shi is with the J. Crayton Pruitt Family Department of Biomedical Engineering, University of Florida, Gainesville, FL, USA, e-mail: xsshi2015@ufl.edu.
F. Xing and M. Sapkota is with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA, e-mail: (f.xing@ufl.edu and manish.sapkota@gmail.com).
Z. Zhang is with the Department of Computer Science and Engineering, University of Florida, Gainesville, FL, USA, e-mail: mr.zizhaozhang@gmail.com.
Z. Guo is with Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, Guangdong, China, e-mail: zhenhua.guo@sz.tsinghua.edu.cn.
L. Yang is with the College of Engineering, Westlake University, Hangzhou, Zhejiang, China, e-mail: yanglin@westlake.edu.cn.

Learning binary codes employs relaxation and thresholding strategies, thereby producing large accumulated errors; (ii) Learning binary codes requires high storage and computation costs, i.e. $\mathcal{O}(n^2)$, where n is the number of training samples, thereby limiting their applications to large-scale tasks; (iii) The pairwise label matrix usually emphasizes the difference of dissimilar samples but neglects the relevance among similar samples. Specifically, for multi-label data, they usually consider that any two samples are similar if they share at least one common label, while they do not take into account the number of shared labels, which determines the ranking order among similarity samples. Hence, existing optimization methods might fail to preserve the relevance information among similar data, potentially leading to inferior performance on ranking order; (iv) The employed optimization methods focus on one type of optimization problems and it is difficult to directly apply them to other problems.

Motivated by aforementioned observations, in this paper, we propose a novel, simple, general and scalable optimization method that can solve various pairwise based hashing models for directly learning binary codes. The main contributions are summarized as follows:

- We propose a novel alternative optimization mechanism to reformulate one typical quartic problem, in term of hash functions in the original objective of KSH [13], into a linear problem by introducing a linear regression model.
- We present and analyze a significant discovery that gradually updating each batch of binary codes in a sequential mode, i.e. batch by batch, is greatly beneficial to the convergence of binary code learning.
- We propose a scalable symmetric discrete hashing algorithm with gradually updating each batch of one discrete matrix. To make the update step more smooth, we further present a greedy symmetric discrete hashing algorithm to greedily update each bit of batch discrete matrices. Then we demonstrate that the proposed greedy hashing algorithm can be used to solve other optimization problems in pairwise based hashing.
- Extensive experiments on three benchmark databases: CIFAR-10 [21], NUS-WIDE [22] and COCO [23], demonstrate the superior performance of the proposed method over recent state-of-the-art algorithms, with low time costs.

2 Related Work

Based on the manner of using the label information, supervised hashing can be classified into three major categories: point-wise, multi-wise and pairwise.

Point-wise based hashing formulates the searching into one classification problem based on the rule that the classification accuracy with learned binary codes should be maximized. Supervised discrete hashing (SDH) [24], Fast supervised discrete hashing (FSDH) [25], Supervised quantization hashing (SQH) [26] are non-deep hashing algorithms and they leverage the linear regression model to generate optimal binary codes. Deep learning of binary hash codes (DLBHC) [27] and deep supervised convolutional hashing (DSCH) [28] employ convolutional neural networks to simultaneously learn image representations and hash codes in a point-wised manner. Point-wise based hashing is scalable and its optimization problem is relatively easier than multi-wise and pairwise based hashing; however, its rule is inferior compared to the other two types of supervised hashing.

Multi-wise based hashing is also named as ranking based hashing that learns hash functions to maximize the agreement of ranking orders over three items between original and Hamming distances. Triplet ranking hashing (TRH) [29], column generation hashing (CGH) [30], Ranking based supervised hashing (RSH) [31], Ranking preserving hashing (RPH) [32] and Top rank supervised binary coding (Top-RSBC) [33] learn hash functions to preserve the rank order among samples. Discrete semantic ranking hashing (DSeRH) [34] learns hash functions to maintain ranking orders with preserving symmetric discrete constraints. Deep network in network hashing (DNNH) [35], deep semantic ranking hashing (DSRH) [36] and triplet-based deep binary embedding (TDBE) [37] utilize convolutional neural network to learn image representations and hash codes based on the triplet ranking loss over three items. Most of these multi-wise based hashing algorithms relax the ranking order or discrete binary codes in a continuously feasible region to solve their original non-convex and non-smooth problems.

Pairwise based hashing maintains relationship among originally high-dimensional data into a Hamming space by calculating and preserving the relationship of each pair. SH [7] constructs one graph to maintain the similarity among neighbors and then utilizes it to map the high-dimensional data into a low-dimensional Hamming space. Although the original version of SH is unsupervised hashing, it is easily converted into a supervised algorithm. Inspired by SH, many variants including anchor graph hashing [38], elastic embedding [39], discrete graph hashing (DGH) [2], and asymmetric discrete graph hashing (ADGH) [18] have been proposed. SSH [11] introduces a pairwise matrix and KSH [13] leverages the Hamming distance between pairs to approximate the pairwise matrix. This objective function is intuitive and simple, but the optimization problem is highly non-differential and difficult to directly solve. KSH utilizes a "symmetric relaxation + greedy" strategy to solve the problem. Two-step hashing (TSH) [14] and FastHash [40] relax the discrete constraints into a continuous region $[-1, 1]$. Kernel based supervised discrete hashing (KSDH) [15] adopts asymmetric relaxation to simultaneously learn the discrete matrix and a low-dimensional projection matrix for hash functions. Lin: Lin and Lin: V [16], asymmetric inner-product binary coding (AIBC) [17] and asymmetric discrete graph hashing (ADGH) [18] employ the asymmetric discrete mechanism to learn low-dimensional matrices. Column sampling based discrete supervised hashing (COSDISH) [20] adopts the column sampling strategy same as latent factor hashing (LFH) [41] but directly learns binary codes by reformulating the binary quadratic programming (BQP) problems into equivalent clustering problems. Convolutional neural network hashing (CNNH) [19] divide the optimization problem into two sub-problems [42]: (i) learning binary codes by a coordinate descent algorithm using Newton directions; (ii) training a convolutional neural

network using the learned binary codes as labels. After that, deep hashing network (DHN) [43] and deep pairwise-supervised hashing (DPSH) [44] simultaneously learn image representations and binary codes using pairwise labels. HashNet [45] learns binary codes from imbalanced similarity data. Deep supervised discrete hashing (DSDH) [46] utilizes both pairwise labels and classification information to learn binary codes. Deep cauchy hashing (DCH) [47] utilizes pairwise labels to generate compact and concentrated binary codes for efficient and effective Hamming space retrieval. Deep asymmetric pairwise hashing (DAPH) [48] and asymmetric deep supervised hashing (ADSH) [49] adopt the asymmetric discrete mechanism to learn hash functions through CNNs. Semi-supervised deep pairwise hashing (SSDPH) [50] utilizes anchor-based self-ensembling to explore the similarity relationship hidden in unlabeled data. Most of these pairwise based algorithms only consider the similarity relationship among samples but fail to take into account their relevance, thereby potentially leading to bad performance on ranking order. Unlike previous work, in this paper we present a general optimization method for binary code learning to preserve the similarity and relevance among samples.

3 Symmetric Discrete Hashing via A Pairwise Matrix

In this paper, matrices and vectors are represented by boldface uppercase and lowercase letters, respectively. For a matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, its i-th row and j-th column vectors are denoted as $\mathbf{x}_i$ and $\mathbf{x}^j$, respectively, and x_{ij} is one entry at the i-th row and j-th column. In this section, we first introduce the formulation we aim to solve. Then we provide the theorems for solving the formulation. Afterwards, we present two symmetric discrete hashing algorithms by updating batch binary codes. Finally, we analyze their time complexity.

3.1 Formulation

KSH [13] is one popular pairwise based hashing algorithm, which can preserve pairs' relationship with using two identical binary matrices to approximate one pairwise real matrix. Additionally, it is a quartic optimization problem in term of hash functions, and thus more typical and difficult to solve than that only containing a quadratic term with respect to hash functions. Therefore, we first propose a novel optimization mechanism to solve the original problem in KSH, and then extend the proposed method to solve other pairwise based hashing models.

Given n data points $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \in \mathbb{R}^{n\times d}$, suppose one pair $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$ when they are neighbors in a metric space or share at least one common label, and $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$ when they are non-neighbors in a metric space or have different class labels. For the single-label multi-class problem, the pairwise matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$ is defined as [11]:

$$s_{ij} = \begin{cases} 1 & (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}, \\ -1 & (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}, \\ 0 & otherwise. \end{cases} \tag{1}$$

For the multi-label multi-class problem, similar to [51], $\mathbf{S}$ can be defined as:

$$s_{ij} = \begin{cases} r_{ij} & (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}, \\ \alpha & (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}, \\ 0 & otherwise, \end{cases} \tag{2}$$

where $r_{ij} > 0$ is the relevance between $\mathbf{x}_i$ and $\mathbf{x}_j$, which is defined as the number of common labels shared by $\mathbf{x}_i$ and $\mathbf{x}_j$. $\alpha < 0$ is the weight to describe the difference between $\mathbf{x}_i$ and $\mathbf{x}_j$. In this paper, to preserve the difference between non-neighbor pairs, we empirically set $\alpha = -\frac{r_{max}}{2}$, where r_{max} is the maximum relevance among all neighbor pairs.

To encode one data point $\mathbf{x} \in \mathbb{R}^d$ into m-bit hash codes, its k-th hash function can be defined as:

$$h_k(\mathbf{x}) = sgn(\mathbf{x}\mathbf{a}_k^T + b_k), \tag{3}$$

where $\mathbf{a}_k \in \mathbb{R}^{1\times d}$ is a projection vector, and $sgn(\mathbf{x}_i\mathbf{a}_k^T + b_k) = 1$ if $\mathbf{x}_i\mathbf{a}_k^T + b_k \geq 0$, otherwise $sgn(\mathbf{x}_i\mathbf{a}_k^T + b_k) = -1$. Note that because $\mathbf{x}\mathbf{a}_k^T + b_k$ can be written as the form $\mathbf{x}\mathbf{a}_k^T$ with $\mathbf{x}$ adding one dimension and $\mathbf{a}_k$ absorbing b_k, and for simplicity we utilize $h_k(\mathbf{x}) = sgn(\mathbf{x}\mathbf{a}_k^T)$ in this paper. Let $code_m(\mathbf{x}) = \{h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_m(\mathbf{x})\}$ be hash codes of $\mathbf{x}$, and then for any pair $(\mathbf{x}_i, \mathbf{x}_j)$, we have $m \geq code_m(\mathbf{x}_i) \circ code_m(\mathbf{x}_j) \geq -m$. To approximate the pairwise matrix $\mathbf{S}$, same as [13], a least-squares style objective function is defined as:

$$\min_{\mathbf{A}} \left\| \mathbf{H}\mathbf{H}^T - \lambda\mathbf{S} \right\|_F^2, \ s.t.\ \mathbf{H} = sgn(\mathbf{X}\mathbf{A}^T), \tag{4}$$

where $\lambda = \frac{m}{r_{max}}$ and $\mathbf{A} \in \mathbb{R}^{m\times d}$ is a low-dimensional projection matrix. Eq. (4) is a quartic problem in term of hash functions, and this can be demonstrated by expanding its objective function.

3.2 Symmetric Discrete Hashing

3.2.1 Formulation transformation

In this subsection, we show the procedure to transform Eq. (4) into a linear problem. Since the objective function in Eq. (4) is a highly non-differential quartic problem in term of hash functions $sgn(\mathbf{X}\mathbf{A}^T)$, it is difficult to directly solve this problem. Here, we solve the problem in Eq. (4) via a novel alternative optimization mechanism: reformulating the quartic problem in term of hash functions into a quadratic one and then linearizing the quadratic problem. We present the detailed procedure in the following. The proofs of Proposition 1 and Theorems 1-4 are shown in the supplemental material.

Firstly, we introduce a Lemma to show one of our main motivations to transform the quartic problem into a linear problem.

Lemma 1. *When the matrix $\mathbf{A} \in \mathbb{R}^{m\times d}$ satisfies the condition: $\mathbf{X}\mathbf{A}^T = \mathbf{Y}$, it is a global solution of the following problem:*

$$\max_{\mathbf{A}} Tr\left\{\mathbf{H}^T\mathbf{Y}\right\}, \ s.t.\ \mathbf{H} = sgn(\mathbf{X}\mathbf{A}^T). \tag{5}$$

Lemma 1 is easy to solve, because when $\mathbf{X}\mathbf{A}^T = \mathbf{Y}$, $\mathbf{H} = sgn(\mathbf{X}\mathbf{A}^T) = sgn(\mathbf{Y})$ makes the objective in Eq. (5) attain the maximum. *Since $\mathbf{A}$ satisfying $\mathbf{X}\mathbf{A}^T = \mathbf{Y}$ is a global solution of the problem in Eq. (5), it suggests that the problem in term of hash functions can be transformed into a linear problem in term of $\mathbf{A}$.* Inspired by this observation, we can solve the

quartic problem in term of hash functions. For brevity, in the following we first ignore the constraint $\mathbf{H} = sgn(\mathbf{X}\mathbf{A}^T)$ in Eq. (4) and aim to transform the quartic problem in term of $\mathbf{H}$ into the linear form as the objective in Eq. (5), and then obtain the low-dimensional projection matrix $\mathbf{A}$.

To reformulate the quartic problem in term of $\mathbf{H}$ into a quadratic one, in the l-th iteration, we set one discrete matrix to be $\mathbf{H}_{l-1}$ and aim to solve the following quadratic problem in term of $\mathbf{H}$:

$$\min_{\mathbf{H}} \left\|\mathbf{H}_{l-1}\mathbf{H}^T - \lambda\mathbf{S}\right\|_F^2, \ s.t.\ \mathbf{H} \in \{-1,1\}^{n\times m}. \quad (6)$$

Note that the problem in Eq. (6) is not strictly equal to the problem in Eq. (4) w.r.t $\mathbf{H}$. However, when $\mathbf{H}_l = \mathbf{H}_{l-1}$, it is the optimal solution of both Eq. (6) and Eq. (4) w.r.t $\mathbf{H}$. The details are shown in Proposition 1.

Proposition 1. *When $\mathbf{H}_l = \mathbf{H}_{l-1}$, the optimal solution of Eq. (6) is also the optimal solution of Eq. (4) w.r.t $\mathbf{H}$.*

Inspired by Proposition 1, we aim to seek $\mathbf{H}_l = \mathbf{H}_{l-1}$ through solving the problem in Eq. (6). Because λ is known and $Tr\left\{\mathbf{S}^T\mathbf{S}\right\} = constant$, the optimization problem in Eq. (6) equals:

$$\begin{aligned}\min_{\mathbf{H}}\ & Tr\left\{\mathbf{H}\mathbf{H}_{l-1}^T\mathbf{H}_{l-1}\mathbf{H}^T\right\} - 2\lambda Tr\left\{\mathbf{H}\mathbf{H}_{l-1}^T\mathbf{S}\right\}, \\ & s.t.\ \mathbf{H} \in \{-1,1\}^{n\times m}.\end{aligned} \quad (7)$$

Since $Tr\left\{\mathbf{H}\mathbf{H}_{l-1}^T\mathbf{S}\right\}$ is a linear problem in term of $\mathbf{H}$, the main difficulty to solve Eq. (7) is caused by the non-convex quadratic term $Tr\left\{\mathbf{H}\mathbf{H}_{l-1}^T\mathbf{H}_{l-1}\mathbf{H}^T\right\}$. Thus we aim to linearize this quadratic term in term of $\mathbf{H}$ by introducing a linear regression model as follows:

Theorem 1. *Given a discrete matrix $\mathbf{H} \in \{-1,1\}^{n\times m}$ and a real non-zero matrix $\mathbf{Z} \in \mathbb{R}^{m\times m}$, $\min_{\mathbf{P}} \|\mathbf{H} - \mathbf{P}\mathbf{Z}\|_F^2 + \left\|\mathbf{P}\mathbf{\Gamma}^{\frac{1}{2}}\right\|_F^2 = Tr\left\{\mathbf{H}(\mathbf{I}_m - \mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T + \mathbf{\Gamma})^{-1}\mathbf{Z})\mathbf{H}^T\right\}$, where $\mathbf{\Gamma} \in \mathbb{R}^{m\times m}$ is a positive-definite diagonal matrix and $\mathbf{I}_m \in \mathbb{R}^{m\times m}$ is an identity matrix.*

Theorem 1 suggests that when $\mathbf{H}_{l-1}^T\mathbf{H}_{l-1} = \gamma(\mathbf{I}_m - \mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T + \mathbf{\Gamma})^{-1}\mathbf{Z})$, the quadratic problem in Eq. (7) can be linearized as a regression type. We show the details in Theorem 2.

Theorem 2. *When $\mathbf{H}_{l-1}^T\mathbf{H}_{l-1} = \gamma(\mathbf{I}_m - \mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T + \mathbf{\Gamma})^{-1}\mathbf{Z})$, where γ is a constant, the problem in Eq. (7) can be reformulated as:*

$$\begin{aligned}\min_{\mathbf{H},\mathbf{P}}\ & \gamma(\|\mathbf{H} - \mathbf{P}\mathbf{Z}\|_F^2 + \left\|\mathbf{P}\mathbf{\Gamma}^{\frac{1}{2}}\right\|_F^2) \\ & -2\lambda Tr\left\{\mathbf{H}\mathbf{H}_{l-1}^T\mathbf{S}^T\right\}, s.t.\ \mathbf{H} \in \{-1,1\}^{n\times m}.\end{aligned} \quad (8)$$

Because of $Tr\left\{\mathbf{H}^T\mathbf{H}\right\} = mn$, the problem in Eq. (8) equals:

$$\begin{aligned}\max_{\mathbf{H},\mathbf{P}}\ & Tr\left\{\mathbf{H}(\tfrac{\lambda}{\gamma}\mathbf{H}_{l-1}^T\mathbf{S} + \mathbf{Z}^T\mathbf{P}^T)\right\} \\ & -\tfrac{1}{2}(\left\|\mathbf{P}\mathbf{\Gamma}^{\frac{1}{2}}\right\|_F^2 + Tr\left\{\mathbf{P}\mathbf{Z}\mathbf{Z}^T\mathbf{P}^T\right\}), \\ & s.t.\ \mathbf{H} \in \{-1,1\}^{n\times m},\end{aligned} \quad (9)$$

which is a linear problem in term of $\mathbf{H}$.

Because Theorems 1-2 are on the basis of γ and $\mathbf{Z}$, next, we demonstrate that there exists γ and $\mathbf{Z}$ such that $\mathbf{H}_{l-1}^T\mathbf{H}_{l-1} = \gamma(\mathbf{I}_m - \mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T + \mathbf{\Gamma})^{-1}\mathbf{Z})$. The details are shown in Theorem 3.

Theorem 3. *Suppose that a full rank matrix $\mathbf{H}_{l-1}^T\mathbf{H}_{l-1} = \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^T$, where $\mathbf{\Lambda} \in \mathbb{R}^{m\times m}$ is a positive diagonal matrix and $\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{I}_m$. If $\gamma \geq \Lambda_{ii}^2$ and $\Gamma_{ii} > 0$, $(1 \leq i \leq m)$ and a real non-zero matrix $\mathbf{Z} = \mathbf{V}\mathbf{\Delta}\mathbf{U}^T$ satisfies the conditions: $\mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{V}^T = \mathbf{I}_m$, and $\mathbf{\Delta} \in \mathbb{R}^{m\times m}$ is a non-negative real diagonal matrix with the i-th diagonal element being $\Delta_{ii} = \sqrt{\frac{\gamma\Gamma_{ii}}{\Lambda_{ii}^2} - \Gamma_{ii}}$.*

$\mathbf{H}_{l-1}^T\mathbf{H}_{l-1}$ is usually a positive-definite matrix thanks to $m << n$, leading to $\Lambda_{ii} > 0$. Because of $\gamma \geq \Lambda_{ii}^2$, in practice we set $\gamma = \max_i \Lambda_{ii}^2 + \beta$ for simplicity, where $1 \leq i \leq m$ and $\beta \geq 0$ is a constant. Based on Theorem 3, it is easy to construct a real non-zero matrix $\mathbf{Z}$. There are many choices for $\mathbf{Z}$, we can simply set $\mathbf{V} = \mathbf{U}$ because of $\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{I}_m$, and hence $\mathbf{Z} = \mathbf{U}\mathbf{\Delta}\mathbf{U}^T$, where $\mathbf{U}$ can be obtained from the singular value decomposition (SVD) of $\mathbf{H}_{l-1}^T\mathbf{H}_{l-1} = \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^T$, $\mathbf{\Delta}$ is from $\Delta_{ii} = \sqrt{\frac{\gamma\Gamma_{ii}}{\Lambda_{ii}^2} - \Gamma_{ii}}$, and Γ_{ii} can be any positive constant.

With γ and $\mathbf{Z}$ obtained, we can solve Eq. (9) by alternatively updating $\mathbf{H}$ and $\mathbf{P}$ in order to attain $\mathbf{H}_l$. After obtaining $\mathbf{H}_l$, we can utilize it to replace $\mathbf{H}_{l-1}$ for next iteration until convergence. For clarity, we define that alternatively updating $\mathbf{H}$ and $\mathbf{P}$ to attain $\mathbf{H}_l$ occurs in the inner loop, and updating $\mathbf{H}_{l-1}$ with $\mathbf{H}_l$ is in the outer loop, e.g. the l-th iteration. Although we can alternatively update $\mathbf{H}$ and $\mathbf{P}$ in Eq. (9) to attain $\mathbf{H}_l$, it is computation expensive to calculate $\mathbf{P}$ and attain $\mathbf{Z}$. Actually, we can obtain $\mathbf{H}$ by using an efficient algorithm in Theorem 4, which demonstrates that there is no need to compute the matrices $\mathbf{Z}$ and $\mathbf{P}$.

Theorem 4. *For the inner t-th iteration embedded in the outer l-th iteration, the problem in Eq. (9) can be reformulated as the following problem:*

$$\begin{aligned}\max_{\mathbf{H}_l}\ & Tr\left\{\mathbf{H}_l((\gamma\mathbf{I}_m - \mathbf{H}_{l-1}^T\mathbf{H}_{l-1})\mathbf{H}_{l_{t-1}}^T\right. \\ & \left.+\lambda\mathbf{H}_{l-1}^T\mathbf{S})\right\}, s.t.\ \mathbf{H}_l \in \{-1,1\}^{n\times m},\end{aligned} \quad (10)$$

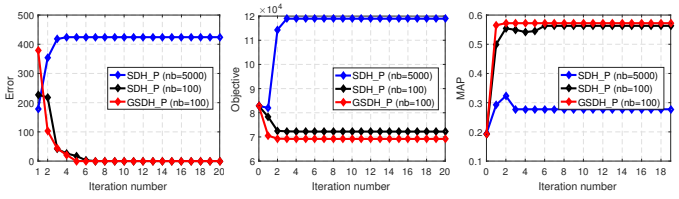
where $\mathbf{H}_l \in \mathbb{R}^{n\times m}$ denotes binary codes $\mathbf{H}$ in the outer l-th iteration, and $\mathbf{H}_{l_{t-1}}$ represents the obtained binary codes $\mathbf{H}$ at the inner t-1-th iteration embedded in the outer l-th iteration.

For the inner loop embedded in the outer l-th iteration, there are many choices for the initialization value $\mathbf{H}_{l_0}$. Here, we set $\mathbf{H}_{l_0} = \mathbf{H}_{l-1}$. At the t-th iteration, the global solution of Eq. (10) is $\mathbf{H}_{l_t} = sgn(\lambda\mathbf{S}^T\mathbf{H}_{l-1} + \mathbf{H}_{l_{t-1}}(\gamma\mathbf{I}_m - \mathbf{H}_{l-1}^T\mathbf{H}_{l-1}))$. It suggests that in the inner loop, the objective of Eq. (10) will be non-decreasing and converge to at least a local optima. Therefore, we have the following theorem.

Theorem 5. *For the inner loop embedded in the outer l-th iteration, the objective of Eq. (10) is monotonically non-decreasing in each iteration and will converge to at least a local optima.*

Although Theorem 5 suggests that the objective of Eq. (10) will converge, its convergence is largely affected by the parameter γ, which is used to balance the convergence and semantic information in $\mathbf{S}$. Usually, the larger γ, the faster convergence but the more loss of semantic information. Therefore, we empirically set a small non-negative constant for β, i.e. $0 \leq \beta \leq 100$, where $\gamma = \max_i \Lambda_{ii}^2 + \beta$.

Based on Eq. (10), the optimal solution $\mathbf{H}_l^*$ can be obtained. Then we utilize $\mathbf{H}_l^*$ to replace $\mathbf{H}_{l-1}$ in Eq. (6)



(a) $\|\mathbf{H}_l - \mathbf{H}_{l-1}\|_F$ (b) $\left\|\mathbf{H}\mathbf{H}^T - \lambda\mathbf{S}\right\|_F$ (c) Top 500

Fig. 1. The error, objective and mean average precision (MAP) of SDH_P and GSDH_P with different numbers of iterations. (In total we select 5K training and 1K query images from the CIFAR10 database, and employ all training samples as anchors. In (b) and (c), when the number of iteration is 0, the results are achieved by using the projection matrix calculated in the initialization step.)

for next iteration in order to obtain the optimal solution $\mathbf{H}^*$. Since $\mathbf{H}^* = sgn(\lambda\mathbf{S}^T\mathbf{H}^* + \mathbf{H}^*(\gamma\mathbf{I}_m - \mathbf{H}^{*T}\mathbf{H}^*))$ and $\mathbf{H} = sgn(\mathbf{X}\mathbf{A}^T)$, based on Lemma 1, $\mathbf{A}$ should satisfy $\mathbf{X}\mathbf{A}^T = \lambda\mathbf{S}^T\mathbf{H}^* + \mathbf{H}^*(\gamma\mathbf{I}_m - \mathbf{H}^{*T}\mathbf{H}^*)$. However, it is an overdetermined linear system due to $n >> d$. For simplicity, we utilize a least-squares model to obtain $\mathbf{A}$, which is $\mathbf{A} = (\lambda\mathbf{H}^{*T}\mathbf{S} + (\gamma\mathbf{I}_m - \mathbf{H}^{*T}\mathbf{H}^*)\mathbf{H}^{*T})\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}$.

3.3 Scalable Symmetric Discrete Hashing with Updating Batch Binary Codes

Based on Theorems 1-4, we present two scalable symmetric discrete algorithms by updating batch binary codes. Section 3.3.1 presents an algorithm, symmetric discrete hashing via a pairwise matrix (SDH_P) that updates all bits as a whole, and Section 3.3.2 shows a greedy algorithm, greedy symmetric discrete hashing via a pairwise matrix (GSDH_P) that greedily updates each bit for further smoothing the update step.

3.3.1 Batch-based symmetric discrete hashing

***Remark**: The optimal solution of Eq. (6) is at least the local optimal solution of Eq. (4) only when $\|\mathbf{H}_l - \mathbf{H}_{l-1}\|_F = 0$. Given an initialization $\mathbf{H}^0$, $\mathbf{H}$ can be alternatively updated by solving Eq. (10). However, with updating all binary codes at once on the non-convex feasible region, $\mathbf{H}$ might change on two different discrete matrices, which would lead to the error $\|\mathbf{H}_l - \mathbf{H}_{l-1}\|_F \neq 0$ (please see SDH_P(n_b=5000) in Fig. 1a) and the objective of Eq. (4) becomes worse (please see SDH_P(n_b=5000) in Fig. 1b), where n_b is the number of samples in each batch. Thus, we divide $\mathbf{H}$ into a variety of batches and gradually update each of them in a sequential mode, i.e. batch by batch.*

To update one batch of $\mathbf{H}$, i.e. $\mathbf{H}_b = \mathbf{H}(idx, :)$, where $idx \in \mathbb{R}^{n_b}$ is one column vector denoting the index of selected binary codes in $\mathbf{H}$, the optimization problem derived from Eq. (6) is:

$$\min_{\mathbf{H}_b} \left\|\mathbf{H}_{l-1}\mathbf{H}_b^T - \lambda\mathbf{S}_b\right\|_F^2, s.t.\ \mathbf{H}_b \in \{-1, 1\}^{n_b \times m}, \quad (11)$$

where $\mathbf{H}_b \subset \mathbf{H}$ and $\mathbf{S}_b = \mathbf{S}(:, idx) \in \mathbb{R}^{n \times n_b}$.

Because of the transduction of anchors, i.e. the prediction of each sample can be calculated as the weighed average of the prediction on anchor samples [52], it is possible to maintain the similarity among all training samples by preserving the similarity relationship between training samples and multiple anchors. Similar to previous algorithms [38] [18], we can uniformly select p ($p << n$) samples from n training samples as anchors and then construct an anchor-based pairwise matrix $\mathbf{S}_A \in \mathbb{R}^{p \times n}$. Let $\mathbf{H}_A \in \{-1, 1\}^{p \times m}$ denote binary codes of anchors, and then utilize $\mathbf{S}_A$ to replace $\mathbf{S}$ for updating $\mathbf{H}_b$, Eq. (11) becomes:

$$\min_{\mathbf{H}_b} \left\|\mathbf{H}_{Al-1}\mathbf{H}_b^T - \lambda\mathbf{S}_{Ab}\right\|_F^2, s.t.\ \mathbf{H}_b \in \{-1, 1\}^{n_b \times m}, \quad (12)$$

where $\mathbf{H}_A \subset \mathbf{H}, \mathbf{H}_b \subset \mathbf{H}$, $\mathbf{H}_{Al-1}$ denotes $\mathbf{H}_A$ obtained at the l-1-th iteration in the outer loop, and $\mathbf{S}_{Ab} = \mathbf{S}_A(:, idx) \in \mathbb{R}^{p \times n_b}$.

Because the optimization type of Eq. (12) is the same as that of Eq. (6), the problem in Eq. (12) can be firstly transformed into a quadratic problem, and then can be reformulated as a similar form to Eq. (10) based on Theorems 1-4. e.g.

$$\max_{\mathbf{H}_{bl}} Tr\left\{\mathbf{H}_{bl}((\gamma\mathbf{I}_m - \mathbf{H}_{Al-1}^T\mathbf{H}_{Al-1})\mathbf{H}_{bl_{t-1}}^T + \lambda\mathbf{H}_{Al-1}^T\mathbf{S}_{Ab})\right\}, s.t.\ \mathbf{H}_{bl} \in \{-1, 1\}^{n_b \times m}. \quad (13)$$

where $\mathbf{H}_{bl}$ denotes the batch binary codes at the l-th iteration in the outer loop.

For clarity, we present the detailed optimization procedure to attain $\mathbf{H}$ by updating each batch $\mathbf{H}_b$ and calculate the projection matrix $\mathbf{A}$ in Algorithm 1 (SDH_P). Its outer loop for updating $\mathbf{H}_l$ contains steps 2-13 and the inner loop for updating $\mathbf{H}_{bl_t}$ has steps 8-10. We present its convergence analysis in the supplemental material. For Algorithm 1, with gradually updating each batch of $\mathbf{H}$, the error $\|\mathbf{H}_l - \mathbf{H}_{l-1}\|_F$ usually converges to zero (please see SDH_P(n_b=100) in Fig. 1a) and the objective of Eq. (4) also converges to a better local optima (please see SDH_P(n_b=100 in Fig. 1b). Besides, we also display the similarity retrieval performance in term of mean average precision (MAP) with a small batch size and different iterations in Fig. 1c. Note that for SDH_P ($n_b = 5000$), although it achieves a smaller objective value with the iteration number being 0 than that with a larger iteration number in Fig. 1b, it still obtains worse MAP shown in Fig. 1c. This might be because when

Algorithm 1: SDH_P

Input: Data $\mathbf{X} \in \mathbb{R}^{n \times d}$, pairwise matrix $\mathbf{S}_A \in \mathbb{R}^{p \times n}$, bit number m, parameters λ, $\beta > 0$, batch size n_b, anchor index $a\_idx \in \mathbb{R}^p$, outer and inner loop maximum iteration number L_1, L_2.
Output: $\mathbf{A} \in \mathbb{R}^{m \times d}$ and $\mathbf{H} \in \{-1, 1\}^{n \times m}$.

1: **Initialize:** Let $\mathbf{X}_A = \mathbf{X}(a\_idx, :)$, set $\mathbf{A}$ to be the left-eigenvectors of $\mathbf{X}^T\mathbf{S}_A^T\mathbf{X}_A$ corresponding to its largest m eigenvalues, calculate $\mathbf{H} = sgn(\mathbf{X}\mathbf{A}^T)$ and $\mathbf{H}_A = \mathbf{H}(a\_idx, :)$.
2: **while not converge or reach maximum iterations**
3. $index \leftarrow randperm(n)$;
4. **for** $i = 1$ to $\frac{n}{n_b}$ **do**
5. $idx \leftarrow index((i-1)n_b + 1 : in_b)$;
6: Do the SVD of $\mathbf{H}_A^T\mathbf{H}_A = \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^T$;
7: $\gamma \leftarrow max(diag(\mathbf{\Lambda}^2)) + \beta$;
8: **repeat**
9: $\mathbf{H}(idx, :) \leftarrow sgn(\lambda\mathbf{S}_A(:, idx)^T\mathbf{H}_A + \mathbf{H}(idx, :)(\gamma\mathbf{I}_m - \mathbf{H}_A^T\mathbf{H}_A))$;
10: **until convergence**
11: $\mathbf{H}_A = \mathbf{H}(a\_idx, :)$;
12: **end for**
13: **end while**
14: Do the SVD of $\mathbf{H}_A^T\mathbf{H}_A = \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^T$;
15: $\gamma \leftarrow max(diag(\mathbf{\Lambda})) + \beta$;
16: $\mathbf{A} = (\lambda\mathbf{H}_A^T\mathbf{S}_A + (\gamma\mathbf{I}_m - \mathbf{H}_A^T\mathbf{H}_A)\mathbf{H}^T)\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}$.

Algorithm 2: GSDH_P

Input: Data $\mathbf{X} \in \mathbb{R}^{n\times d}$, pairwise matrix $\mathbf{S}_A \in \mathbb{R}^{p\times n}$, bit number m, parameters λ, $\beta > 0$, batch size n_b, anchor index $a\_idx \in \mathbb{R}^p$, outer/inner maximum iteration number L_1, L_2.

Output: $\mathbf{A} \in \mathbb{R}^{m\times d}$ and $\mathbf{H} \in \{-1,1\}^{n\times m}$.

Initialize: Let $\mathbf{X}_A = \mathbf{X}(a\_idx,:)$, set $\mathbf{A}$ to be the left-eigenvectors of $\mathbf{X}^T\mathbf{S}_A^T\mathbf{X}_A$ corresponding to its largest m eigenvalues, calculate $\mathbf{H} = sgn(\mathbf{X}\mathbf{A}^T)$, and $\mathbf{H}_A = \mathbf{H}(a\_idx,:)$.
while not converge or reach maximum iterations
  $index \leftarrow randperm(n)$;
  for $i = 1$ to $\frac{n}{n_b}$ **do**
    $idx \leftarrow index((i-1)n_b + 1 : in_b)$;
    for $j = 1$ to m **do**
      Calculating $\widetilde{\mathbf{S}}_A(:, idx)$ with fixing $\mathbf{h}^k$, $k = 1, 2, \cdots, m$, $k \neq j$;
      repeat
        $\mathbf{H}(idx, j) \leftarrow sgn(\lambda\widetilde{\mathbf{S}}_A(:, idx)^T\mathbf{H}_A(:, j) + \beta\mathbf{H}(idx, j))$;
      until convergence
      $\mathbf{H}_A(:, j) = \mathbf{H}(a\_idx, j)$;
    end for
  end for
end while
Do the SVD of $\mathbf{H}_A^T\mathbf{H}_A = \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^T$;
$\gamma \leftarrow max(diag(\mathbf{\Lambda})) + \beta$;
$\mathbf{A} = (\lambda\mathbf{H}_A^T\mathbf{S}_A + (\gamma\mathbf{I}_m - \mathbf{H}_A^T\mathbf{H}_A)\mathbf{H}^T)\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}$.

the iteration number is 0, there exists a large accumulated quantization error between binary codes $\mathbf{H} = sgn(\mathbf{X}\mathbf{A}^T)$ and its relaxed matrix $\mathbf{X}\mathbf{A}^T$. However, when the iteration number is greater than 0, $\mathbf{H}$ is directly learned as the binary codes of training data. Additionally, for the iteration number 0, the projection matrix $\mathbf{A}$ is only determined by the pairwise label matrix $\mathbf{S}$, while for the iteration number larger than 0, $\mathbf{A}$ is learned based on both $\mathbf{H}$ and $\mathbf{S}$. This might enhance the similarity among binary codes of both training and query data.

3.3.2 Batch-based greedy symmetric discrete hashing

To make the update step more smooth, we greedily update each bit of the batch matrix $\mathbf{H}_b$. Suppose that $\mathbf{h}_b^j = \mathbf{H}(idx, j)$ is the j-th bit of $\mathbf{H}_b$, it can be updated by solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{h}_b^j} \left\| \mathbf{h}_A^j\mathbf{h}_b^{jT} - \lambda(\mathbf{S}_{Ab} - \textstyle\sum_{k\neq j}^m \mathbf{h}_A^k\mathbf{h}_b^{kT}) \right\|_F^2, \\ s.t.\ \mathbf{h}_b^j \in \{-1,1\}^{n_b}, \end{aligned} \tag{14}$$

where $\mathbf{h}_A^j \subset \mathbf{h}^j$, $\mathbf{h}_b^j \subset \mathbf{h}^j$, $\mathbf{h}_A^j$ and $\mathbf{h}_A^k$ represent the j-th and k-th bits of $\mathbf{H}_A$, respectively, and $\mathbf{h}_b^k$ is the k-th bit of $\mathbf{H}_b$.

The problem in Eq. (14) can also be firstly transformed into a quadratic problem and then solved using Theorems 1-4. Similar to Eq. (10), the problem in Eq. (14) can be transformed to:

$$\begin{aligned} \max_{\mathbf{h}_{bl}^j} Tr\left\{\mathbf{h}_{bl}^j(\beta\mathbf{h}_{bl_{t-1}}^{jT} + \lambda\mathbf{h}_{Al-1}^{jT}\widetilde{\mathbf{S}}_{Ab})\right\}, \\ s.t.\ \mathbf{h}_{bl}^j \in \{-1,1\}^{n_b}, \end{aligned} \tag{15}$$

where $\mathbf{h}_{bl}^j$ is the $\mathbf{h}_b^j$ obtained at the l-th iteration in the outer loop, $\mathbf{h}_{bl_{t-1}}^j$ is the $\mathbf{h}_b^j$ obtained at the t-1-th iteration in inner loop embedded in the l-th outer loop and $\widetilde{\mathbf{S}}_{Ab} = \mathbf{S}_{Ab} - \sum_{k\neq j}^m \mathbf{h}_A^k\mathbf{h}_b^{kT}$.

In summary, we show the detailed optimization procedure in Algorithm 2 (GSDH_P). Its outer loop for updating $\mathbf{H}_l$ contains steps 2-14 and the inner loop for updating $\mathbf{h}_{bl_t}^j$ has steps 8-10. The error $\|\mathbf{H}_l - \mathbf{H}_{l-1}\|_F$ and the objective of Eq. (4) in Algorithm 2 and its retrieval performance in term of MAP with different numbers of iterations are shown in Fig. 1a, b and c, respectively.

Out-of-sample: In the query stage, $\mathbf{H}$ is employed as the binary codes of training data. We can adopt two strategies to encode the query data point $\mathbf{q} \in \mathbb{R}^d$: (i) encoding it using $h(\mathbf{q}) = sgn(\mathbf{q}\mathbf{A}^T)$; (ii) similar to previous algorithms [53] [19], employing $\mathbf{H}$ as labels to learn a classification model, like least-squares, decision trees (DT) or convolutional neural networks (CNNs), to classify $\mathbf{q}$.

3.4 Time Complexity Analysis

In Algorithm 1, $n >> d >> m$ and $n >> p >> m$. Step 1 calculating matrices $\mathbf{A}$ and $\mathbf{H}$ requires $\mathcal{O}(pnd)$ and $\mathcal{O}(ndm)$ operations, respectively. For the outer loop stage, the time complexity of steps 6, 7, 9 and 11 is $\mathcal{O}(pm^2)$, $\mathcal{O}(m)$, $\mathcal{O}(pmn_b)$ and $\mathcal{O}(pm)$, respectively. Hence, the outer loop stage spends $\mathcal{O}(L_1L_2npm)$ operations. Steps 14-16 to calculate the projection matrix $\mathbf{A}$ spend $\mathcal{O}(pm^2)$, $\mathcal{O}(m)$ and $max(\mathcal{O}(pnm), \mathcal{O}(nd^2))$, respectively. Therefore, the total complexity of Algorithm 1 is $max(\mathcal{O}(npd), \mathcal{O}(nd^2), \mathcal{O}(L_1L_2npm))$. Empirically, $L_1 \leq 20$ and $L_1 \leq 3$.

For Algorithm 2, step 1 calculating $\mathbf{A}$, $\mathbf{H}$ and $\mathbf{M}$ spends at most $max(\mathcal{O}(pnd), \mathcal{O}(nd^2))$. In the loop stage, the major steps both 7 and 9 require $\mathcal{O}(pn_b)$ operations. Hence, the total time complexity of the loop stage is $\mathcal{O}(L_1L_2npm)$. Additionally, calculating the final $\mathbf{A}$ costs the same time to the steps 14-16 in Algorithm 1. Therefore, the time complexity of Algorithm 2 is $max(\mathcal{O}(npd), \mathcal{O}(nd^2), \mathcal{O}(L_1L_2npm))$.

4 Extension to Other Hashing Algorithms

In this subsection, we illustrate that the proposed algorithm GSDH_P is suitable for solving many other pairwise based hashing models.

Two-step hashing algorithms [14] [40] iteratively update each bit of the different loss functions defined on the Hamming distance of data pairs so that the loss functions of many hashing algorithms such as BRE [8], MLH [10] and EE [39] are incorporated into a general framework, which can be written as:

$$\min_{\mathbf{h}^j} \mathbf{h}^j\mathbf{L}\mathbf{h}^{jT},\ s.t.\ \mathbf{h}^j \in \{-1,1\}^n \tag{16}$$

where $\mathbf{h}^j$ represents the j-th bit of binary codes $\mathbf{H} \in \{-1,1\}^{n\times m}$ and $\mathbf{L} \in \mathbb{R}^{n\times n}$ is obtained based on different loss functions with fixing all bits of binary codes except $\mathbf{h}^j$.

The algorithms [14] [40] firstly relax $\mathbf{h}^j \in \{-1,1\}^n$ into $\mathbf{h}^j \in [-1,1]^n$ and then employ L-BFGS-B [54] to solve the relaxed optimization problem, followed by thresholding to attain the binary vector $\mathbf{h}^j$. However, our optimization mechanism can directly solve Eq. (16) without relaxing $\mathbf{h}^j$.

Since $Tr(\mathbf{h}^j\mathbf{h}^{jT}\mathbf{h}^j\mathbf{h}^{jT}) = const$ and $Tr(\mathbf{L}\mathbf{L}^T) = const$, the problem in Eq. (16) can be equivalently reformulated as:

$$\min_{\mathbf{h}^j} \left\| \mathbf{h}^j\mathbf{h}^{jT} - (-\mathbf{L}) \right\|_F^2\ s.t.\ \mathbf{h}^j \in \{-1,1\}^n, \tag{17}$$

whose optimization type is same as the objective of Eq. (4) w.r.t $\mathbf{H}$. Replacing the constraint $\mathbf{h}^j \in \{-1,1\}^n$ with $\mathbf{h}^j = sgn(\mathbf{X}\mathbf{a}_j^T)$, where $\mathbf{a}_j \in \mathbb{R}^d$ is the j-th row vector of $\mathbf{A}$, Eq. (17) becomes:

$$\min_{\mathbf{a}_j} \left\|\mathbf{h}^j\mathbf{h}^{jT} - (-\mathbf{L})\right\|_F^2 \ s.t.\ \mathbf{h}^j = sgn(\mathbf{X}\mathbf{a}_j^T), \tag{18}$$

Since $\mathbf{L} \in \mathbb{R}^{n\times n}$ will consume large computation and storage costs for large n, we select p training anchors to construct $\mathbf{L}_A \in \mathbb{R}^{p\times n}$ based on different loss functions. Replacing $\mathbf{L}$ in Eq. (18) with $\mathbf{L}_A$, it becomes:

$$\min_{\mathbf{a}_j} \left\|\mathbf{h}_A^j\mathbf{h}^{jT} - (-\mathbf{L}_A)\right\|_F^2, \ s.t.\ \mathbf{h}^j = sgn(\mathbf{X}\mathbf{a}_j^T), \tag{19}$$

Similar to solving Eq. (4), we can firstly obtain $\mathbf{h}^j$ and then calculate $\mathbf{a}_j$. To attain $\mathbf{h}^j$, we still gradually update each batch $\mathbf{h}_b^j$ by solving the following problem:

$$\min_{\mathbf{h}_b^j} \left\|\mathbf{h}_A^j\mathbf{h}_b^{jT} - (-\mathbf{L}_{Ab})\right\|_F^2, \tag{20}$$

where $\mathbf{L}_{Ab} = \mathbf{L}_A(:, idx) \in \mathbb{R}^{p\times n_b}$.

The optimization type of Eq. (20) is the same as that of Eq. (14). $\mathbf{h}^j$ can be obtained by gradually updating $\mathbf{h}_b^j$ as shown in GSDH_P. After obtaining $\mathbf{h}^j$, $\mathbf{a}_j$ can be attained by using $\mathbf{a}_j = (\beta\mathbf{h}^{jT} - \mathbf{h}_A^{jT}\mathbf{L}_A)\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}$.

Based on Eqs. (16)-(20), many pairwise based hashing models can learn binary codes by using GSDH_P. For instances, we show the performance on solving the optimization model in BRE [8] [40]:

$$\mathcal{L}(\mathbf{h}_i, \mathbf{h}_j) = [m\delta(s_{ij} < 0) - d_H(\mathbf{h}_i, \mathbf{h}_j)]^2 \tag{21}$$

where $d_H(\mathbf{h}_i, \mathbf{h}_j)$ is the Hamming distance between $\mathbf{h}_i$ and $\mathbf{h}_j$, and $\delta(\cdot) \in \{0, 1\}$ is an indicator function. Here, $s_{ij} < 0$ denotes $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$.

One typical model with a hinge loss function [10] [40]:

$$\mathcal{L}(\mathbf{h}_i, \mathbf{h}_j) = \begin{cases} [0 - d_H(\mathbf{h}_i, \mathbf{h}_j)]^2 & (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M} \\ [max(0.5m - d_H(\mathbf{h}_i, \mathbf{h}_j), 0)]^2 & otherwise. \end{cases} \tag{22}$$

In this paper, the optimization model in BRE solved by GSDH_P is named as GSDH_P$_{BRE}$. Similarly, the hinge loss function Eq. (22) solved by GSDH_P is named as GSDH_P$_{Hinge}$.

5 Experimental Results and Analysis

We evaluate the proposed algorithms SDH_P and GSDH_P by using one benchmark single-label database: CIFAR-10 [21] and two popular multi-label databases: NUS-WIDE [22] and COCO [23]. The CIFAR-10 database contains 60K color images of ten object categories, with each category consisting of 6K images. The NUS-WIDE database contains around 270K images collected from Flickr, with each image consisting of multiple semantic labels. Totally, this database has 81 ground truth concept labels. Here, similar to [24], we choose the images associated with the 21 most frequent labels. In total, there are around 195K images. The COCO database consists of about 328K images belonging to 91 objects types. We adopt the 2014 training and validation datasets. They have around 83K training and 41K validation images belonging to 80 object categories.

5.1 Comparison with Non-deep Hashing Algorithms

5.1.1 Experimental Setting

We evaluate SDH_P and GSDH_P on two kinds of image retrieval tasks: similarity and ranking order, where the similarity retrieval task regards the pair to be similar if they share at least one common labels, and the ranking order retrieval task considers the number of shared labels, which determines the order of returned samples. We adopt the single-label database CIFAR-10 for similarity retrieval and utilize two multi-label databases NUS-WIDE and COCO for ranking order retrieval. We compare them against ten state-of-the-art non-deep hashing algorithms including six point-wise and pairwise based algorithms: KSH [13], CCA-ITQ [55], SDH [24], COSDISH [20], KSDH [15], ADGH [18], four ranking algorithms: RSH [31], CGH [30], Top-RSBC [33] and DSeRH [34]. Note that ADGH cannot be directly applied to multi-label databases, so we only utilize it for similarity retrieval. Additionally, we compare BRE [8] and MLH [10] and FastH [40] with GSDH_P$_{BRE}$ and GSDH_P$_{Hinge}$ to illustrate the generalization of GSDH_P on similarity retrieval. For KSH and KSDH, we employ the same pairwise matrix as SDH_P and GSDH_P for tackling similarity and raking order retrieval tasks. For SDH_P and GSDH_P, we empirically set $p = 1000$, $n_b = 100$, $\beta = 10$, $L_1 = 20$ and $L_2 = 3$ in our experiments.

To evaluate the non-deep hashing methods, we utilize MAP, Precision and Recall to evaluate their performance on the similarity retrieval task, and employ two main criteria: Normalized Discounted Cumulative Gain (NDCG) [56] and average cumulative gain (ACG) [31], to assess their performance on the ranking order task.

Because all non-deep hashing methods have similar test time, we only show their training time for better comparison. We utilize MATLAB and conduct all experiments on a 3.50GHz Intel Xeon E5-1650 CPU with 128GB memory.

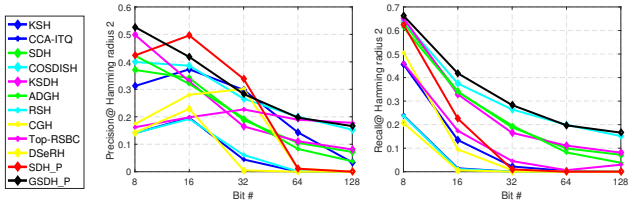
5.1.2 Experiments for Similarity Retrieval

We partition the CIFAR-10 database into training and query sets, which consist of 50K and 10K images, respectively. Each image is aligned and cropped to 32×32 pixels and then represented by a 512-dimensional GIST feature vector [57]. In our experiments, we kernelize GIST feature vectors by using the same kernel type in KSH [13] and uniformly selecting 1K samples from the training set as anchors. Since some comparative multi-wise based algorithms are extremely slow when using a large number of training data, we utilize only a subset of data to train models for the non-deep hashing algorithms: KSH, CCA-ITQ, SDH, COSDISH, KSDH, ADGH, RSH, CGH, Top-RSBC and DSeRH. Similar to KSH, we uniformly pick up 1K and 100 images from each category for training and testing, respectively. Then, we evaluate the proposed algorithms and the six non-ranking algorithms (KSH, CCA-ITQ, SDH, COSDISH, KSDH and ADGH) using all training and query images. Their retrieval performance in term of MAP with 500 retrieved samples is shown in Table 1. As we can see, when using 10K training and 1K query images, both SDH_P and GSDH_P achieve better MAPs than the other algorithms at 8-, 16-, 32- and 64-bit. The gain of GSDH_P ranges from 1.35% to 4.88% over the best competitor except SDH_P. Additionally, GSDH_P

TABLE 1
Similarity retrieval performance (MAP) on top 500 retrieved samples and training time (seconds) of different hashing methods on the single-label database CIFAR-10. We bold the best accuracy and underline the second best one at each setting.

Method	GIST ($n = 10000$)				
	MAP (Top 500)				Time
	8-bit	16-bit	32-bit	64-bit	64-bit
KSH	0.4281	0.4706	0.5098	0.5270	4.6×10^3
CCA-ITQ	0.2014	0.1984	0.2147	0.2316	0.9
SDH	0.4970	0.5370	0.5670	0.5781	7.3
COSDISH	0.5116	0.5912	0.5980	0.6162	1.5×10^1
KSDH	0.5370	0.5740	0.5900	0.6000	3.5
ADGH	0.5360	0.5700	0.5980	0.6020	1.4
RSH	0.2121	0.1889	0.1812	0.1810	7.5×10^4
CGH	0.3013	0.3040	0.3255	0.3305	9.2×10^2
Top-RSBC	0.2568	0.2404	0.2400	0.2544	9.3×10^4
DSeRH	0.2020	0.2070	0.2087	0.2116	2.7×10^3
SDH_P	**0.5755**	0.5981	0.6102	0.6222	1.1×10^1
GSDH_P	0.5600	**0.5992**	**0.6272**	**0.6300**	2.4×10^1

Method	GIST (Full)				
	MAP (Top 500)				Time
	8-bit	16-bit	32-bit	64-bit	64-bit
KSH	0.4057	0.4725	0.5126	0.5317	3.5×10^4
CCA-ITQ	0.2338	0.2224	0.2473	0.2745	4.7
SDH	0.4723	0.5700	0.5920	0.6038	5.2×10^1
COSDISH	0.5593	0.6065	0.6125	0.6255	9.2×10^1
KSDH	0.5687	0.5955	0.6015	0.6091	7.6×10^1
ADGH	0.5731	0.6097	0.6113	0.6119	4.4
SDH_P	**0.5735**	**0.6172**	0.6222	0.6279	3.3×10^1
GSDH_P	0.5680	0.6142	**0.6239**	**0.6333**	6.9×10^1



(a) Precision (b) Recall

Fig. 2. Precision and Recall vs. Bits of various algorithms on the CIFAR-10 database with Hamming radius being 2.

obtains higher MAPs than SDH_P at 16-, 32- and 64-bit. When using 50K training and 10K query images, GSDH_P outperforms the other comparative non-deep hashing algorithms except ADCG at 8-bit. Fig. 2 presents the precision and recall of various hashing algorithms at 8-, 16-, 32-, 64- and 128-bits on the CIFAR-10 database with Hamming radius being 2. It further demonstrates the superior performance of GSDH_P over the other hashing algorithms. Although SDH_P achieves best precision at 16- and 32-bit, its recall is very low compared to other algorithms.

To illustrate the generation of the proposed algorithm GSDH_P, we uniformly pick up 1K and 100 images from each category for training and testing, respectively. We repeat this process 10 times and report the average MAP of KSH, BRE, MLH, FastH and GSDH_P with top 500 samples returned in Table 2. Note that GSDH_P$_{KSH}$+DT denotes learning classification models by using decision trees as classifiers and the learned binary codes of GSDH_P$_{KSH}$ as labels. Similar definitions for GSDH_P$_{BRE}$+DT and GSDH_P$_{Hinge}$+DT. Here, we utilize GSDH_P$_{KSH}$ to represent GSDH_P for a clear comparison. Table 2 illustrates that GSDH_P can achieve significantly better performance than KSH, BRE and MLH. Meanwhile, it also outperforms FastH with lower training costs.

TABLE 2
Ranking performance (MAP) on top 500 retrieved samples and training time (seconds) of KSH, BRE, MLH and their variants on the CIFAR-10 database. We bold the best accuracy and underline the second best one at each setting.

Method	GIST ($n = 10000$)				
	MAP (Top 500)				Time
	8-bit	16-bit	32-bit	64-bit	64-bit
KSH	0.4270	0.4658	0.5126	0.5340	4.6×10^3
BRE	0.2201	0.2181	0.2355	0.2533	1.3×10^5
MLH	0.1141	0.2210	0.2673	0.2676	2.5×10^2
GSDH_P$_{KSH}$	0.5528	**0.5996**	0.6126	0.6162	2.4×10^1
GSDH_P$_{BRE}$	0.5478	0.5960	**0.6158**	**0.6251**	5.8×10^1
GSDH_P$_{Hinge}$	**0.5582**	0.5961	0.6136	0.6240	1.2×10^2
FastH$_{KSH}$	0.5434	0.5831	0.6111	0.6234	4.1×10^2
FastH$_{BRE}$	0.5236	0.5819	0.6104	0.6132	4.1×10^2
FastH$_{Hinge}$	0.5266	0.5775	0.6200	0.6290	3.7×10^2
GSDH_P$_{KSH}$+DT	**0.5482**	0.5934	0.6230	0.6301	1.7×10^2
GSDH_P$_{BRE}$+DT	0.5225	0.5863	**0.6248**	0.6322	1.8×10^2
GSDH_P$_{Hinge}$+DT	0.5376	**0.5990**	0.6215	**0.6350**	2.1×10^2

5.1.3 *Experiments for Ranking Order*

For the NUS-WIDE database, we partition all images into training and test sets, including around 185K training and 10K query images, respectively. Each image is represented by the provided 500 Bag-of-Words (BoW) features. For the COCO database, we adopt 83K training images and select 10K validation images to construct training and query sets, respectively. Then we resize each image to 32×32 pixels and represent each one by using a 2048-dimensional CNN feature vector, which is extracted by a popular and powerful neural network: ResNet50 [58]. In our experiments, similar to KSH [13], we kernelize feature vectors by uniformly selecting 1K samples from the training set as anchors. Firstly, we uniformly select 10K training and 1K query images to evaluate all ranking and non-ranking algorithms except ADGH. Then, we utilize all training and query images to evaluate the proposed algorithms and three scalable algorithms: CCA-ITQ, SDH and COSDISH.

Table 3 presents their ranking performance in term of NDCG with 50 samples retrieved and training time of various algorithms on NUS-WIDE and COCO. It shows that GSDH_P has superior ranking performance to the other nine non-deep hashing algorithms when 10K training images are used. On NUS-WIDE, its gain in term of NDCG is from 2.35% to 13.42% over the best competitors except SDH_P. When all training images are used, GSDH_P significantly outperforms CCA-ITQ, SDH and COSDISH, its gain ranges from 8.58 % to 15.13% over the best competitors at all four bits. On COCO, when using 10K training images, the gain of GSDH_P in term of NDCG is 7.40%, 5.44%, 4.79% and 0.92% compared to the best competitor except SDH_P at 8-, 16-, 32- and 64-bit, respectively; when using all training images, the NDCG of GSDH_P is 4.09%, 7.30%, 2.23% and 1.74% higher than the best competitor except SDH_P at the four bits, respectively.

Fig. 3 presents the ACG of various algorithms on NUS-WIDE and COCO at 8-, 16-, 32-, 64- and 128-bit with Hamming radius being 2, and their NDCGs with 5, 10, 20, 50, 100 and 200 samples retrieved at 64-bit. It further illustrates that GSDH_P outperforms the other hashing algorithms.

5.2 Comparison with Deep Hashing Algorithms

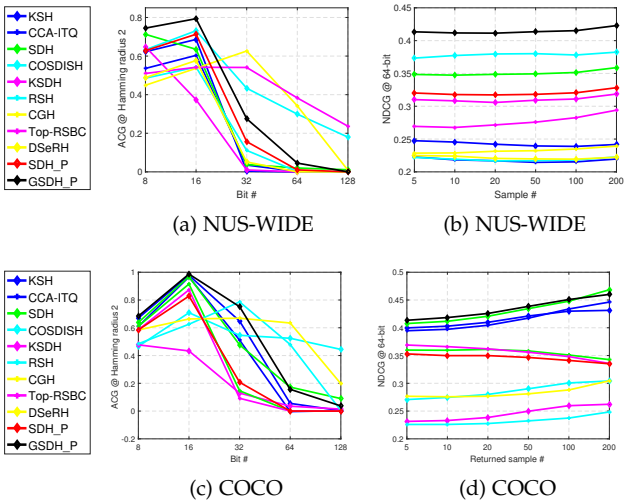
5.2.1 *Experimental Setting*

To better illustrate the strength of GSDH_P, we utilize the binary codes learned by GSDH_P as labels to train clas-

TABLE 3
Ranking order performance (NDCG) on top 50 retrieved samples and training time (seconds) of different hashing algorithms on NUS-WIDE and COCO. We bold the best accuracy and underline the second best one at each setting.

NUS-WIDE					
Method	BoW ($n = 10000$)				
	NDCG (Top 50)				Time
	8-bit	16-bit	32-bit	64-bit	64-bit
KSH	0.2755	0.2730	0.2554	0.2395	2.1×10^3
CCA-ITQ	0.2522	0.2337	0.2248	0.2145	0.4
SDH	0.3108	0.3674	0.3674	0.3494	2.5×10^1
COSDISH	0.3169	0.3606	0.3989	0.3801	3.5×10^1
KSDH	0.3227	0.3125	0.3091	0.3092	5.3
RSH	0.2040	0.2061	0.2088	0.2193	9.2×10^4
CGH	0.2163	0.2228	0.2475	0.2323	1.5×10^3
Top-RSBC	0.1962	0.2238	0.2564	0.2758	8.9×10^4
DSeRH	0.2158	0.2264	0.2271	0.2210	8.2×10^3
SDH_P	0.2916	0.3014	0.3038	0.3174	4.3
GSDH_P	**0.3588**	**0.3889**	**0.4083**	**0.4311**	5.5×10^1
Method	BoW (Full)				
	NDCG (Top 50)				Time
	8-bit	16-bit	32-bit	64-bit	64-bit
CCA-ITQ	0.2562	0.2551	0.2528	0.2475	8.8
SDH	0.3381	0.3927	0.3838	0.3827	3.8×10^2
COSDISH	0.3649	0.3539	0.4279	0.4323	3.5×10^2
SDH_P	0.3010	0.3094	0.3431	0.3313	7.7×10^1
GSDH_P	**0.4152**	**0.4521**	**0.4646**	**0.4875**	5.0×10^2
COCO					
	NDCG (Top 50)				Time
	8-bit	16-bit	32-bit	64-bit	64-bit
KSH	0.2352	0.3226	0.3810	0.4168	2.7×10^3
CCA-ITQ	0.2293	0.3367	0.3947	0.4103	0.3
SDH	0.1911	0.3218	0.3911	0.4345	2.1×10^1
COSDISH	0.1623	0.2382	0.2438	0.2890	2.2×10^1
KSDH	0.1623	0.2000	0.2292	0.2597	2.7
RSH	0.2156	0.2751	0.3299	0.3583	6.1×10^4
CGH	0.1566	0.1928	0.2176	0.2324	1.9×10^3
Top-RSBC	0.1547	0.1843	0.2045	0.2845	8.6×10^4
DSeRH	0.2144	0.2556	0.3210	0.3560	1.7×10^3
SDH_P	0.1932	0.2476	0.3165	0.3465	2.7
GSDH_P	**0.2526**	**0.3550**	**0.4136**	**0.4385**	5.5×10^1
Method	Full				
	NDCG (Top 50)				Time
	8-bit	16-bit	32-bit	64-bit	64-bit
CCA-ITQ	0.1831	0.2426	0.2902	0.3032	1.0×10^1
SDH	0.1684	0.2430	0.3537	0.3846	3.2×10^2
COSDISH	0.1557	0.1741	0.2013	0.2146	1.9×10^2
SDH_P	0.1756	0.2427	0.3252	0.3471	2.3×10^1
GSDH_P	**0.2240**	**0.3160**	**0.3760**	**0.4020**	6.1×10^2

sification models. Following many previous deep hashing algorithms [43] [45], we adopt the AlexNet architecture [59] with pre-trained on the ImageNet database [60], and name this method as GSDH_P*. Additionally, we compare GSDH_P* against eight popular deep hashing algorithms: CNNH [19], DNNH [35], DHN [43], DPSH [44], Hashnet [45], DSDH [46], ADSH [49] and DCH [47]. For fairness, we re-implement all the eight deep hashing algorithms using the pre-trained AlexNet architecture via the Pytorch framework based on the provided public codes. For GSDH_P*, we also empirically set $p = 1000$, $n_b = 100$, $\beta = 10$, $L_1 = 20$ and $L_2 = 3$ in our experiments. Additionally, we employ the Adam optimizer [61] to update the network parameters, with the initialization momentum parameters $\beta_1 = 0.99$ and $\beta_2 = 0.999$. We set the maximum learning rate to be 0.00005, with the learning rate decreasing by 50% every 10 epochs. We totally run 100 epochs for training with the mini-batch size being 100, and run each deep hashing algorithm on a machine by using one Nvidia GeForce GTX 1080 Ti.



(a) NUS-WIDE (b) NUS-WIDE

(c) COCO (d) COCO

Fig. 3. ACG vs. Bits and NDCG vs. retrieved samples of various algorithms on NUS-WIDE and COCO.

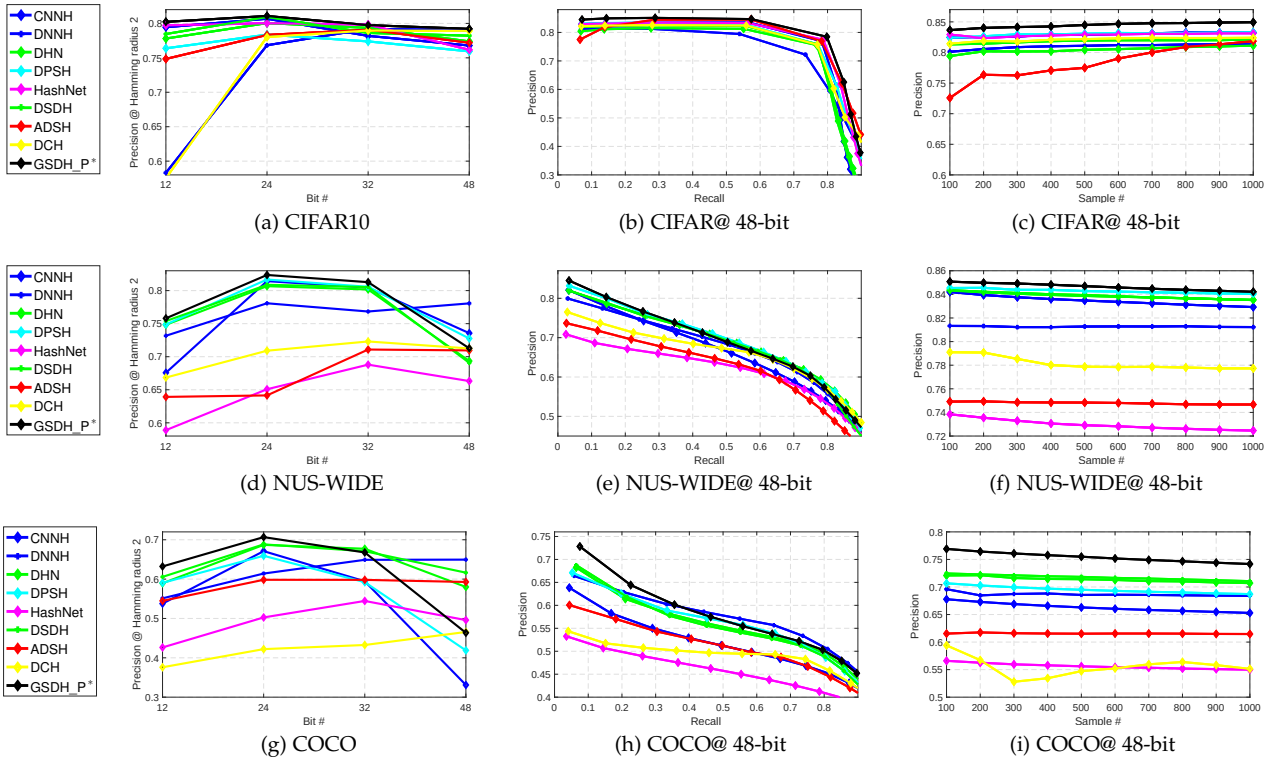
We evaluate GSDH_P* and the eight deep hashing algorithms on similarity and ranking order retrieval tasks. For similarity retrieval, we conduct experiments on CIFAR-10, NUS-WIDE and COCO databases. Following [19] [35], on CIFAR-10 and NUS-WIDE, we randomly select 100 images per class to construct a query set, adopt the remaining images as a base set for retrieval, and then uniformly select 500 images of each class from the base set to construct a training set; on COCO, we randomly select at least 20 images per concept for querying, utilize the rest of images as a base set for retrieval, and then uniformly choose 100 images of each concept from the base set for training. Totally there are 1600 query and 8000 training images. Additionally, we utilize the MAP over all images in the base set to evaluate the hashing algorithms on CIFAR-10, and employ the MAP over the top 5K retrieved samples from the base set to assess them on NUS-WIDE and COCO. Moreover, we adopt the metrics, Precision within Hamming radius being 2, precision-curves and precision with different numbers of returned samples, to assess the mentioned hashing algorithms on the three databases. Note that for similarity retrieval, any two samples are defined as neighbors if they share at least one common label. For ranking order retrieval, we conduct experiments on two multi-label databases NUS-WIDE and COCO. We evaluate all algorithms by using the NDCG over top 50 returned samples. Also, we show their performance on the NDCG with different numbers of returned samples and the ACG within Hamming radius being 2. Note that GSDH_P* utilize the pairwise matrix in Eq. (2) for multi-label databases on both similarity and ranking order retrieval tasks.

5.2.2 *Experimental Results for Similarity Retrieval*

Table 4 presents the MAP of various deep hashing algorithms on CIFAR-10 and their MAP@5000 on NUS-WIDE and COCO. It suggests that on CIFAR-10 and NUS-WIDE, GSDH_P* achieves superior performance over the other deep hashing algorithms at 24-, 32- and 48-bit, and obtains the competitive performance to that of the best competitors at 12-bit; on COCO, GSDH_P* attains the best accuracy at the four bit settings. Although GSDH_P* only attains slightly better MAP than the best competitor HashNet on

TABLE 4
Similarity retrieval performance (MAP) of GSDH_P* and eight popular deep hashing algorithms on CIFAR-10, NUS-WIDE and COCO. We bold the best accuracy and underline the second best one at each setting.

Method	CIFAR-10 MAP				NUS-WIDE MAP @5000				COCO MAP @5000			
	12-bit	24-bit	32-bit	48-bit	12-bit	24-bit	32-bit	48-bit	12-bit	24-bit	32-bit	48-bit
CNNH	0.7752	0.7854	0.7908	0.7967	0.7597	0.8009	0.8020	0.8198	0.6138	0.6605	0.6694	0.6671
DNNH	0.7232	0.7604	0.7710	0.7671	0.7841	0.8094	0.7992	0.8100	0.6438	0.6601	0.6837	0.6803
DHN	0.7676	0.7818	0.7814	0.7772	0.7892	0.8058	0.8152	0.8213	0.6291	0.6810	0.6905	0.6974
DPSH	0.7590	0.7753	0.7791	0.7978	**0.7939**	0.8140	0.8202	0.8321	0.6322	0.6648	0.6672	0.6848
HashNet	**0.7998**	0.8037	0.8079	0.8097	0.6924	0.7057	0.7253	0.7548	0.5457	0.6170	0.6285	0.6449
DSDH	0.7792	0.7913	0.7961	0.8020	0.7862	0.8080	0.8172	0.8304	0.6446	0.6776	0.6834	0.7007
ADSH	0.7626	0.7862	0.7987	0.8061	0.7128	0.7031	0.7723	0.7677	0.5953	0.6162	0.6104	0.6134
DCH	0.7625	0.7912	0.7928	0.7965	0.7423	0.7574	0.7681	0.7867	0.5452	0.6072	0.6295	0.6452
GSDH_P*	0.7947	**0.8091**	**0.8138**	**0.8206**	0.7902	**0.8220**	**0.8241**	**0.8352**	**0.6555**	**0.6881**	**0.6942**	**0.7275**



(a) CIFAR10 (b) CIFAR@ 48-bit (c) CIFAR@ 48-bit

(d) NUS-WIDE (e) NUS-WIDE@ 48-bit (f) NUS-WIDE@ 48-bit

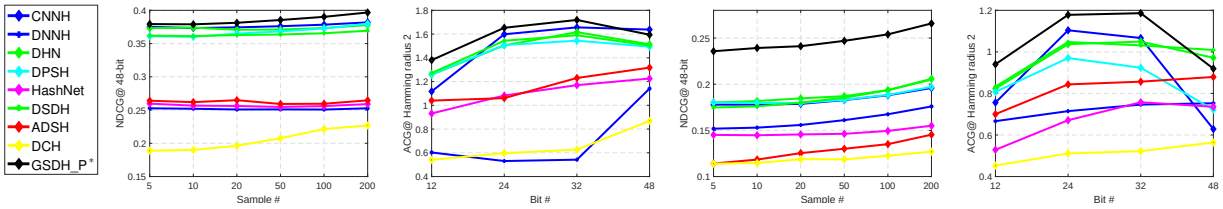
(g) COCO (h) COCO@ 48-bit (i) COCO@ 48-bit

Fig. 4. Similarity retrieval performance of different deep hashing algorithms and GSDH_P*: (a)-(c) Precision@ Hamming radius being 2, Precision-recall, and Precision vs. samples on CIFAR-10, (d)-(f) Precision@ Hamming radius being 2, Precision-recall, and Precision vs. samples on COCO, (g)-(i) Precision@ Hamming radius being 2, Precision-recall, and Precision vs. samples on COCO.

CIFAR-10, it can attain much better performance than HashNet on NUS-WIDE and COCO. Similar observations can be found on DPSH (the best competitor on NUS-WIDE). Note that several deep hashing algorithms, like CNNH and DHN, obtain better results than their originally reported accuracy, probably because we adopt a different framework and learning schedule. Fig. 4 illustrates that GSDH_P* attains better performance than the other deep hashing algorithms on the three databases, except the precision-recall curves at 48-bit on NUS-WIDE and COCO, where GSDH_P* achieves a higher precision with a relatively small recall, e.g. 0.3. It infers that GSDH_P* can attain better performance on retrieving a small number of top samples on NUS-WIDE and COCO, probably because we preserve the similarity among training samples via Eq. (2), which assigns larger weights to more relevant samples among similar data. Both Table 4 and Fig. 4 demonstrate the effectiveness and strength of the proposed method. Similar findings can be observed at other bits.

5.2.3 *Experimental Results for Ranking Order*

Table 5 presents the NDCG of GSDH_P* and eight deep hashing algorithms on top 50 returned samples from the base set of NUS-WIDE and COCO. It suggests that GSDH_P* obtains the best performance on the two databases, especifically on COCO, where the gain of GSDH_P* is 10.40%, 17.57%, 26.05% and 32.03% over the best competitors at the four bit settings. The pairwise based deep hashing algorithms, CNNH, DHN, DPSH, HashNet, DSDH, ADSH and DCH obtain inferior ranking performance, probably because their pairwise label matrices fail to consider the relevance among similarity samples. Note that we have tried to utilize Eq. (2) to replace their pairwise matrices, but they attain worse performance or even do not work at all. Fig. 5 presents the NDCG and ACG of the deep hashing algorithms on NUS-WIDE and COCO, and it further demonstrates the superior performance of GSDH_P*.



(a) NUS-WIDE (b) NUS-WIDE (c) COCO (c) COCO

Fig. 5. Ranking performance of different deep hashing algorithms and GSDH_P*.

TABLE 5
Ranking order performance (NDCG) on top 50 retrieved samples of different deep hashing algorithms on NUS-WIDE and COCO. We bold the best accuracy and underline the second best one at each setting.

NUS-WIDE				
Method	NDCG (Top 50)			
	12-bit	24-bit	32-bit	48-bit
CNNH	0.2946	0.3476	0.3627	0.3701
DNNH	0.2572	0.2694	0.2697	0.2870
DHN	0.2946	0.3211	0.3558	0.3711
DPSH	0.3027	0.3303	0.3392	0.3684
HashNet	0.1931	0.2074	0.2250	0.2547
DSDH	0.2895	0.3185	0.3519	0.3636
ADSH	0.2348	0.2374	0.2684	0.2758
DCH	0.1684	0.1921	0.2089	0.2176
GSDH_P*	**0.3147**	**0.3665**	**0.3779**	**0.3855**
COCO				
Method	NDCG (Top 50)			
	12-bit	24-bit	32-bit	48-bit
CNNH	0.1440	0.1727	0.1761	0.1828
DNNH	0.1299	0.1582	0.1639	0.1611
DHN	0.1452	0.1736	0.1789	0.1870
DPSH	0.1299	0.1497	0.1624	0.1831
HashNet	0.1084	0.1208	0.1355	0.1423
DSDH	0.1450	0.1643	0.1788	0.1854
ADSH	0.1160	0.1231	0.1301	0.1301
DCH	0.0938	0.1084	0.1157	0.1186
GSDH_P*	**0.1603**	**0.2041**	**0.2255**	**0.2469**

6 Conclusion and Future Work

In this paper, we propose a novel, scalable and general optimization method to directly solve the non-convex and non-smooth problems in term of hash functions. We firstly solve a quartic problem in a least-squares model that utilizes two identical code matrices produced by hash functions to approximate a pairwise label matrix, by reformulating the quartic problem in term of hash functions into a quadratic problem, and then linearize it by introducing a linear regression model. Additionally, we find that gradually learning each batch of binary codes is beneficial to the convergence of learning process. Based on this finding, we propose a symmetric discrete hashing algorithm to gradually update each batch of the discrete matrix, and a greedy symmetric discrete hashing algorithm to greedily update each bit of batch discrete matrices. Finally, we extend the proposed greedy symmetric discrete hashing algorithm to handle other optimization problems. The comparison with non-deep hashing algorithms on similarity and ranking order retrieval tasks demonstrates that the proposed method can effectively solve multiple optimization problems and preserve the relevance information in the pairwise label matrix, with acceptably low time costs. Moreover, the comparison with deep hashing algorithms further illustrates the effectiveness and strength of binary codes learned by the proposed optimization mechanism.

In this paper we only focus on solving the problems in pairwise based hashing, in the future, it is worth extending the proposed mechanism to solve the problems in multi-wise based hashing, whose objective is also highly non-differential, non-convex and more difficult to directly solve. Additionally, we train a convolutional neural network to learn image representations and binary codes individually, and the model performance might be further improved by simultaneously learning image representations and binary codes. Therefore, it is promising to utilize various networks and the proposed optimization mechanism to learn image features and binary codes simultaneously, and apply them to numerous applications, like text and pathology images.

References

[1] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large image databases for recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[2] W. Liu, C. Mu, S. Kumar, and S.-F. Chang, "Discrete graph hashing," in *Advances in Neural Information Processing Systems*, 2014, pp. 3419–3427.

[3] Y. Cao, H. Qi, W. Zhou, J. Kato, K. Li, X. Liu, and J. Gui, "Binary hashing for approximate nearest neighbor search on big data: A survey," *IEEE Access*, vol. 6, pp. 2039–2054, 2018.

[4] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *the ACM symposium on Theory of computing*, 1998, pp. 604–613.

[5] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, 2000.

[6] A. Shrivastava and P. Li, "In defense of minhash over simhash," in *Artificial Intelligence and Statistics*, 2014, pp. 886–894.

[7] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems*, 2009, pp. 1753–1760.

[8] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Advances in Neural Information Processing Systems*, 2009, pp. 1042–1050.

[9] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2916–2929, 2013.

[10] M. Norouzi and D. M. Blei, "Minimal loss hashing for compact binary codes," in *International Conference on Machine Learning*, 2011, pp. 353–360.

[11] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large-scale search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 12, pp. 2393–2406, 2012.

[12] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua, "Ldahash: Improved matching with smaller descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 1, pp. 66–78, 2012.

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

[14] G. Lin, C. Shen, D. Suter, and A. van den Hengel, "A general two-step approach to learning-based hashing," in *IEEE International Conference on Computer Vision*, 2013, pp. 2552–2559.

[15] X. Shi, F. Xing, J. Cai, Z. Zhang, Y. Xie, and L. Yang, "Kernel-based supervised discrete hashing for image retrieval," in *European Conference on Computer Vision*, 2016, pp. 419–433.

[16] B. Neyshabur, N. Srebro, R. R. Salakhutdinov, Y. Makarychev, and P. Yadollahpour, "The power of asymmetry in binary hashing," in *Advances in Neural Information Processing Systems*, 2013, pp. 2823–2831.

[17] F. Shen, W. Liu, S. Zhang, Y. Yang, and H. T. Shen, "Learning binary codes for maximum inner product search," in *IEEE International Conference on Computer Vision*, 2015, pp. 4148–4156.

[18] X. Shi, F. Xing, K. Xu, M. Sapkota, and L. Yang, "Asymmetric discrete graph hashing," in *AAAI Conference on Artificial Intelligence*, 2017, pp. 2541–2547.

[19] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning." in *AAAI Conference on Artificial Intelligence*, vol. 1, 2014, pp. 2156–2162.

[20] W.-C. Kang, W.-J. Li, and Z.-H. Zhou, "Column sampling based discrete supervised hashing," in *AAAI Conference on Artificial Intelligence*, 2016.

[21] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[22] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng, "Nus-wide: a real-world web image database from national university of singapore," in *ACM International Conference on Image and Video Retrieval*, 2009, p. 48.

[23] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*. Springer, 2014, pp. 740–755.

[24] F. Shen, C. Shen, W. Liu, and H. Tao Shen, "Supervised discrete hashing," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 37–45.

[25] G. Koutaki, K. Shirai, and M. Ambai, "Fast supervised discrete hashing and its analysis," *arXiv preprint arXiv:1611.10017*, 2016.

[26] X. Wang, T. Zhang, G.-J. Qi, J. Tang, and J. Wang, "Supervised quantization for similarity search," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2018–2026.

[27] K. Lin, H.-F. Yang, J.-H. Hsiao, and C.-S. Chen, "Deep learning of binary hash codes for fast image retrieval," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015, pp. 27–35.

[28] M. Sapkota, X. Shi, F. Xing, and L. Yang, "Deep convolutional hashing for low dimensional binary embedding of histopathological images," *IEEE Journal of Biomedical and Health Informatics*, 2018.

[29] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, "Hamming distance metric learning," in *Advances in Neural Information Processing Systems*, 2012, pp. 1061–1069.

[30] X. Li, G. Lin, C. Shen, A. Hengel, and A. Dick, "Learning hash functions using column generation," in *International Conference on Machine Learning*, 2013, pp. 142–150.

[31] J. Wang, W. Liu, A. X. Sun, and Y.-G. Jiang, "Learning hash codes with listwise supervision," in *IEEE International Conference on Computer Vision*, 2013, pp. 3032–3039.

[32] Q. Wang, Z. Zhang, and L. Si, "Ranking preserving hashing for fast similarity search," in *International Joint Conference on Artificial Intelligence*, 2015, pp. 3911–3917.

[33] D. Song, W. Liu, R. Ji, D. A. Meyer, and J. R. Smith, "Top rank supervised binary coding for visual search," in *IEEE Conference on Computer Vision*, 2015, pp. 1922–1930.

[34] L. Liu, L. Shao, F. Shen, and M. Yu, "Discretely coding semantic rank orders for supervised image hashing," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1425–1434.

[35] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3270–3278.

[36] F. Zhao, Y. Huang, L. Wang, and T. Tan, "Deep semantic ranking based hashing for multi-label image retrieval," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1556–1564.

[37] B. Zhuang, G. Lin, C. Shen, and I. Reid, "Fast training of triplet-based deep binary embedding networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5955–5964.

[38] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *International Conference on Machine Learning*, 2011, pp. 1–8.

[39] M. A. Carreira-Perpinán, "The elastic embedding algorithm for dimensionality reduction," in *International Conference on Machine Learning*, vol. 10, 2010, pp. 167–174.

[40] G. Lin, C. Shen, and A. van den Hengel, "Supervised hashing using graph cuts and boosted decision trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 11, pp. 2317–2331, 2015.

[41] P. Zhang, W. Zhang, W.-J. Li, and M. Guo, "Supervised hashing with latent factor models," in *ACM SIGIR Conference on Research & Development in Information Retrieval*, 2014, pp. 173–182.

[42] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," in *ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010, pp. 18–25.

[43] H. Zhu, M. Long, J. Wang, and Y. Cao, "Deep hashing network for efficient similarity retrieval." in *AAAI Conference on Artificial Intelligence*, 2016, pp. 2415–2421.

[44] W.-J. Li, S. Wang, and W.-C. Kang, "Feature learning based deep supervised hashing with pairwise labels," *arXiv preprint arXiv:1511.03855*, 2015.

[45] Z. Cao, M. Long, J. Wang, and S. Y. Philip, "Hashnet: Deep learning to hash by continuation." in *IEEE Conference on Computer Vision*, 2017, pp. 5609–5618.

[46] Q. Li, Z. Sun, R. He, and T. Tan, "Deep supervised discrete hashing," in *Advances in neural information processing systems*, 2017, pp. 2482–2491.

[47] Y. Cao, M. Long, B. Liu, J. Wang, and M. KLiss, "Deep cauchy hashing for hamming space retrieval," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1229–1237.

[48] F. Shen, X. Gao, L. Liu, Y. Yang, and H. T. Shen, "Deep asymmetric pairwise hashing," in *ACM international conference on Multimedia*. ACM, 2017, pp. 1522–1530.

[49] Q.-Y. Jiang and W.-J. Li, "Asymmetric deep supervised hashing," in *AAAI Conference on Artificial Intelligence*, 2018.

[50] X. Shi, Z. Guo, F. Xing, Y. Liang, and L. Yang, "Anchor-based self-ensembling for semi-supervised deep pairwise hashing," *International Journal of Computer Vision*, pp. 1–18, 2020.

[51] X. Shi, M. Sapkota, F. Xing, F. Liu, L. Cui, and L. Yang, "Pairwise based deep ranking hashing for histopathology image classification and retrieval," *Pattern Recognition*, vol. 81, pp. 14–22, 2018.

[52] W. Liu, J. He, and S.-F. Chang, "Large graph construction for scalable semi-supervised learning," in *International Conference on Machine Learning*, 2010, pp. 679–686.

[53] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter, "Fast supervised hashing with decision trees for high-dimensional data," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1963–1970.

[54] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software*, vol. 23, no. 4, pp. 550–560, 1997.

[55] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2916–2929, 2013.

[56] K. Järvelin and J. Kekäläinen, "Ir evaluation methods for retrieving highly relevant documents," in *ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, 2000, pp. 41–48.

[57] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[59] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[60] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

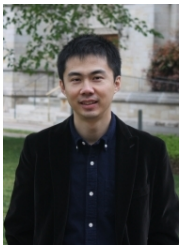
Xiaoshuang Shi received the B.S. degree in automation from Northwestern Polytechnical University, China, and M.S. degree in automation from Tsinghua University, China, in 2009 and 2013, respectively. From Sep. 2013 to Apr. 2015, he was a Research Assistant in Shenzhen Key Laboratory of Broadband Network & Multimedia, Graduate School at Shenzhen, Tsinghua University, China. Now, he is a Ph.D. candidate in the J. Crayton Pruitt Family Department of Biomedical Engineering at University of Florida, Gainesville, USA. His current research interests include large-scale image retrieval, pattern recognition and medical image analysis.

Fuyong Xing is an Assistant Professor in the Department of Biostatistics and Informatics at Colorado School of Public Health, University of Colorado Anschutz Medical Campus. He received his Ph.D. from the University of Florida, his M.S. from Rutgers, The State University of New Jersey-New Brunswick and bachelor?s degree from Xi?an Jiaotong University. His research interests focus on medical image computing, biomedical imaging informatics, machine learning and deep learning.

Zizhao Zhang received the B.S. degree in Cognitive Science from Xiamen University. He is currently a PhD student in the department of Computer and Information Science and Engineering, University of Florida. His research interests include in computer vision, deep learning, and computer-aided medical image diagnosis.

Manish Sapkota received the Computer Engineering degree from Purbanchal University - Nepal in 2007. He is currently a PhD candidate in Department of Electrical and Computer Engineering at University of Florida, USA. His research interests include computer vision, machine learning, content-based image retrieval and high performance computing.

Zhenhua Guo received the M.S. and Ph.D degree in computer science from Harbin Institute of Technology and the Hong Kong Polytechnic University in 2004 and 2010 respectively. Since April 2010, he has worked in Graduate School at Shenzhen, Tsinghua University. His research interests include pattern recognition, texture classification, biometrics, video surveillance, etc.



Lin Yang is working in the College of Engineering, Westlake University. He was an associate professor in the J. Crayton Pruitt Family Department of Biomedical Engineering, the Department of Electrical and Computer Engineering, and the Department of Computer and Information Science and Engineering at University of Florida from 2014 to 2020. He was an assistant professor in the Department of Radiology and Pathology, and graduate faculty in the Department of Biomedical Engineering at Rutgers University from 2009 to 2011. He was an assistant professor in the Department of Biomedical Informatics and the Department of Computer Science at University of Kentucky from 2011 to 2014. His major research interest is focused on biomedical image analysis, imaging informatics, computer vision, biomedical informatics, and machine learning. He is also working on high performance computing and computed aided health care and information technologies.