Pattern Recognition 139 (2023) 109483

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Deep debiased contrastive hashing

Rukai Wei [a], Yu Liu [a,*], Jingkuan Song [b], Yanzhao Xie [a], Ke Zhou [a]

[a] Huazhong University of Science and Technology, China
[b] University of Electronic Science and Technology of China, China

ARTICLE INFO

Article history:
Received 2 November 2022
Revised 17 February 2023
Accepted 26 February 2023
Available online 28 February 2023

Keywords:
Learning to hash
Contrastive learning
Instance discrimination
EM Algorithm
Neighborhood discovery

ABSTRACT

Hashing has achieved great success in multimedia retrieval due to its high computing efficiency and low storage cost. Recently, contrastive-learning-based hashing methods have achieved decent retrieval performance in label-free scenarios by learning distortion-invariant representations with Siamese networks. Their learning principle, *i.e.*, *instance discrimination*, maximizes the correlation between self-augmented views and treats all others as negative samples. However, it may learn with false negative samples that are naturally similar, resulting in biased hash learning. To bridge this flaw, we reveal the between-instance similarity of naturally similar samples by exploring the latent structure of the training data. As a result, we propose the **D**eep **D**ebiased **C**ontrastive **H**ashing (DDCH) algorithm, using the neighborhood discovery module to explore the intrinsic similarity relationship that can help contrastive hashing reduce false negatives for superior discriminatory ability. Furthermore, we elucidate the rationale for incorporating the module into the contrastive hashing framework and explain our hashing process from an Expectation-Maximization (EM) perspective. Extensive experimental results on three benchmark image datasets demonstrate that DDCH significantly outperforms the state-of-the-art unsupervised hashing methods for image retrieval.

1. Introduction

The similarity hashing methods [1–5] convert high-dimensional data to compact binary codes for fast retrieval, where the similarity in the codes should be consistent with that in the data. The labels are the supervision tokens for preserving similarity during the conversion process. Nevertheless, hand-crafted labels are expensive for large-scale datasets. As a result, unsupervised hashing methods [6–8] have attracted increasing attention.

Existing unsupervised hashing methods mainly fall into two categories: reconstruction-based and contrastive-learning-based methods. The former [9–11] generates hash codes containing the background noise that is used to reconstruct the original information, resulting in sub-optimal performance. The latter [7,12] can alleviate the above problem via aligning different views of the same sample. As the learning principle of the latter methods, *instance discrimination* maximizes the correlation between self-augmented views of the same sample while treating all others as negative samples. However, some negative samples may share the same semantics as the anchor sample. They are referred to as "false negative samples". We show a toy example in Fig. 1. Even if two images share the same semantic, *e.g.*, "cat", existing contrastive hashing methods [7,12] still falsely treat all samples as negatives. Furthermore, as shown in Fig. 2, we illustrate the histograms of the cosine similarity between the hash codes of a sample and the other 1023 negatives in a batch. It is clear that the negatives captured by the CIBHash [12] model contain a large number of similar samples, *i.e.*, they have a high probability of sharing the same semantic with the anchor sample. This indicates that the random negative sampling will introduce massive false negatives improperly. Nevertheless, existing contrastive hashing methods overlook the latent semantic structural information of data [13,14] and employ a random negative sampling strategy that introduces false negatives, causing biased hash learning and sub-optimal retrieval performance.

To address these issues, we develop an unsupervised hashing method using a debiased contrastive learning pattern. Specifically, we incorporate the neighborhood discovery module into the contrastive learning framework to construct the latent structure of the training data where we collect between-instance similarity. The neighborhood discovery module can reduce the number of false negative samples. In addition, it assists in providing cross-sample similarity information about naturally similar similar samples for hash learning. As a result, our method can reduce the mean intra-class Hamming distance while increasing the mean inter-class Hamming distance. Note that neighborhood discovery

* Corresponding author at: School of Computer of Science and Technology, HUST, China.
E-mail address: liu_yu@hust.edu.cn (Y. Liu).

https://doi.org/10.1016/j.patcog.2023.109483
0031-3203/

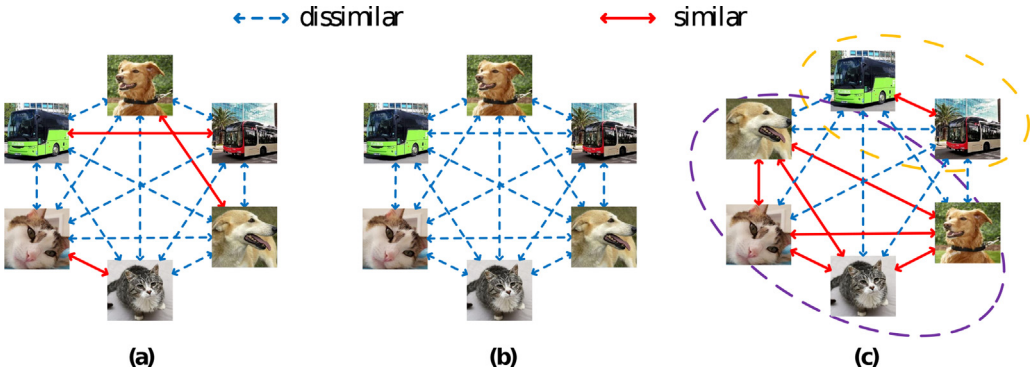

Fig. 1. The similarity and dissimilarity relationships between instances used for hash learning. The blue dashed lines imply that image pairs are dissimilar while the red lines denote that they are similar. The sub-figure (a) illustrates the relationships built by the ground-truth. The sub-figure (b) shows the similarity in perspective of contrastive hashing methods using instance discrimination, *e.g.*, DATE [7] and CIBHash [12]. Although some samples contain the same semantics, they are considered negative to each other and are referred to as *false negative samples*. The sub-figure (c) shows the similarity built by a clustering method. All images are grouped into two clusters denoted by dashed line boxes, where the similarity of naturally similar images is remarkable. The existing contrastive hashing methods neglect the latent structure of the datasets, resulting in an inability to exploit the similarity of data. The clustering method can establish between-instance similarity of naturally similar data and supplement cross-sample similarity information about naturally similar data. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

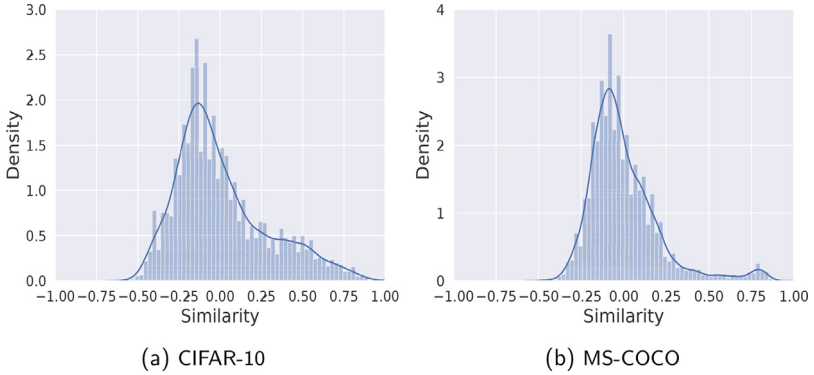

Fig. 2. The histograms of cosine similarity between the continuous hash codes of a random sample and the remaining 1023 in-batch negative samples on CIFAR-10 and MS-COCO, respectively. The 64-bit hash codes are generated by a well-trained CIBHash [12] model. When we treat the similarity of 0.5 to 1 as similar, there is a large number of negatives that are similar to the anchor sample.

is implementation-agnostic and open-ended. In practice, we offer three concrete strategies for neighborhood discovery, including online clustering, K-NN search, and threshold sifting (See Section 3.2).

Based on these motivations, we propose deep debiased contrastive hashing (DDCH). DDCH transforms images into different views and then applies the same encoder network (*e.g.*, VGG16 [15]) to extract feature vectors like conventional contrastive learning frameworks [16–18]. The features of different views are fed into the Siamese hash layers and mapped to K bits continuous hash codes, where the hash layer consists of fully-connected layers (*fc*), *batch normalization, ReLU*, and *Tanh*. The architecture is shown in Fig. 3. For pairwise hash codes presented in the blue framework, we adopt Cauchy distribution [19] based loss functions to keep their similarity in Hamming space. The supervised token, *i.e.*, pairwise similarity, is derived from the orange framework, where we perform neighborhood discovery with continuous hash codes within a mini-batch. The pairwise hash codes will be close if their self-augmented pairs are neighbors; otherwise, they will be far apart. In addition, this architecture is a symmetric framework for bidirectional cross-view supervision [20], albeit we only show an asymmetric structure in Fig. 3. We define a symmetric loss function that ensures superior alignment between different views to reduce the view gap, alleviating the over-confident problem produced by supervision under a single augmentation.

In theory, DDCH can be explained by the EM algorithm. In E-step, we estimate the posterior distribution of the latent variables (*i.e.*, pairwise similarity) via neighborhood discovery. In M-step, we evaluate the posterior distribution of hash codes by Cauchy distribution. Meanwhile, we maximize the posterior distribution to determine the revised parameter of the network (See Section 3.3). As a result, although the hash codes are generated by samples under different augmentations, they are expected to share the same distribution by the contrastive learning framework.

Last but not least, we conduct extensive experiments on three public image datasets, including MS-COCO [21], NUS-WIDE [22], and CIFAR-10 [23]. The experimental results demonstrate the superiority of DDCH compared with state-of-the-art unsupervised hashing methods.

The main contributions are outlined as follows:

- We propose a debiased contrastive hashing method using the neighborhood discovery module to explore the between-instance similarity. This helps the contrastive hashing model reduce the number of false negatives for superior discriminatory ability.
- We reveal that neighborhood discovery is a unified module for open-ended implementation and is compatible with most methods for constructing similarity relationships. Additionally, we implement it with online clustering, K-NN search, and threshold sifting to achieve multi-granularity pairwise relation construction.

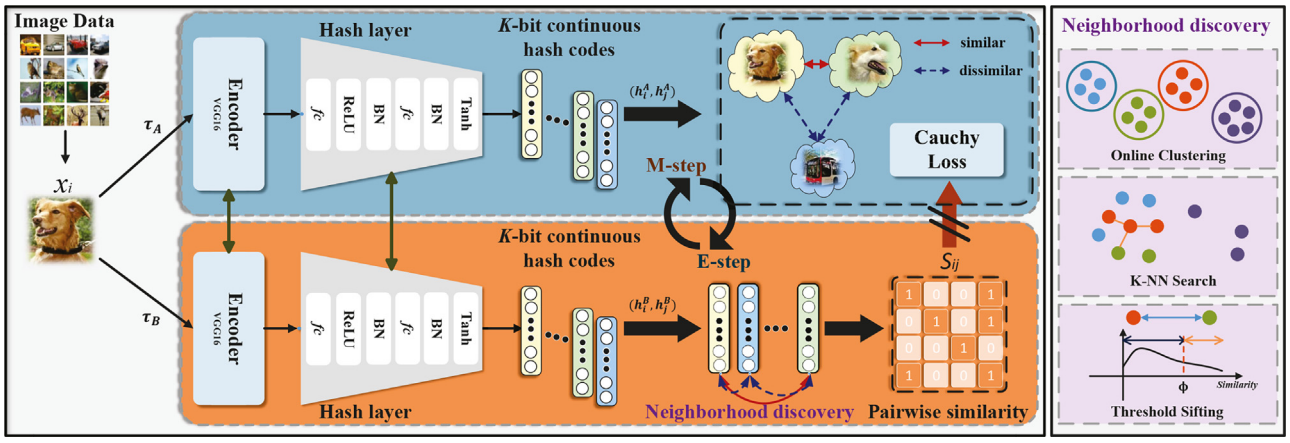

Fig. 3. DDCH: deep debiased contrastive hashing. τ_A and τ_B are two augmentation strategies. h_i^A and h_i^B are continuous hash codes corresponding to image x_i via τ_A and τ_B, respectively. DDCH can be thought of as an Expectation-Maximization (EM) algorithm implementation. The orange and blue frameworks represent the E-step branch and the M-step branch, respectively, where the encoders and hash layers in the two branches share the same parameters. (1) In the E-step branch, we perform neighborhood discovery for the hash codes in a batch, *i.e.*, $H_B = \{h_i^B\}_{i=1}^N$, and obtain pairwise similarity, *i.e.*, $s_{ij} \in \{0, 1\}$. If hash codes, *e.g.*, h_i^B and h_j^B, are neighbors, $s_{ij} = 1$; otherwise, $s_{ij} = 0$. (2) In the M-step branch, we adopt the Cauchy distribution based loss functions to learn compact hash codes under the supervision of s_{ij}. The pairwise hash codes h_i^A and h_j^A will be compelled to be close in the Hamming space when $s_{ij} = 1$; otherwise, they will be far apart. In practice, we select online clustering, K-NN search, and threshold sifting to achieve neighborhood discovery.

- We bridge the contrastive hashing and the EM algorithm. DDCH is the first work to implement and elucidate contrastive hashing from an EM perspective, aiming to provide inspiration for the hashing community.
- DDCH outperforms state-of-the-art unsupervised hashing methods in image retrieval on public benchmark datasets.

2. Related work

Since DDCH highlights on improving hashing ability in contrastive learning, we briefly review two closely related research areas, including unsupervised hashing and contrastive learning, in the following paragraphs.

2.1. Unsupervised hashing

Existing state-of-the-art unsupervised hashing methods can be broadly classified into two categories. The first category is reconstruction-based hashing methods [9,10,14,24], which employ encoder-decoder architectures to reconstruct original images from hash codes or to reconstruct pairwise similarity derived from original features in Hamming space, or implicitly reconstruct likelihood with the discriminator in Generative Adversarial Network (GAN). For example, Dai *et al.* [9] utilize the binary latent variables as the encoder-decoder bottleneck and reconstruct original images from hash codes. Yang *et al.* [14] propose to construct pairwise similarities using deep features and reconstruct the similarities in Hamming space. Shen *et al.* [10] use a deep variational Bayesian model to minimize the gap between the constructed and reconstructed latent variables from data inputs and binary outputs, respectively. The second category is contrastive-learning based hashing methods, which learns hash codes under contrastive learning frameworks [18,25]. Specifically, Luo *et al.* [7] propose a general distribution-based metric to depict the pairwise distance between images, exploring both semantic-preserving learning and contrastive learning to obtain high-quality hash codes. CIB-Hash [12] learns hash codes under the SimCLR [25] framework and compresses the model by Information Bottleneck [26]. Different from existing reconstruction-based jobs, DDCH also adopts the contrastive-learning paradigm since the reconstruction-based methods introduce background noise into hash codes, resulting in sub-optimal retrieval performance. Besides, compared to existing contrastive-learning based methods that suffer from false-negative problem, DDCH introduces the neighborhood discovery module to alleviate it. Additionally, we demonstrate its effectiveness from both experimental and theoretical aspects.

2.2. Contrastive learning

Contrastive learning is a special self-supervised learning (SSL) method. The principle of contrastive learning maximizes the identical representation between views augmented from the same image. It learns view-invariant representations by attracting the positive samples and repelling the negative samples, which is usually achieved by the InfoNCE loss function [25]. Based on this principle, Wu *et al.* [27] first propose to pre-train the representation learning model with a pre-text task of instance discrimination. Lin *et al.* [18] present a new negative sampling strategy based on the queue to provide adequate negative samples. Furthermore, Chen *et al.* [25] propose a simple contrastive framework, *i.e.*, SimCLR, for visual representations with a large batch size and an ancillary projector. Then, BYOL [17] is proposed to learn without negative pairs by training online network, predicting the target's representations, where the target network is the Exponential Moving Average (EMA) of the parameters of the online network. Chen *et al.* [16] abandon the momentum encoder and negative pairs by a well-designed asymmetric architecture and the stop-gradients. Contrastive learning assists unsupervised hashing methods in the generation of disturb-invariant hash codes. However, existing contrastive hashing methods succeed the principle of instance discrimination, *i.e.*, overlooking the natural between-instance similarity and using false negatives, resulting in biased hash learning. Due to the desire to use contrastive learning for accurate features, we propose to introduce neighborhood discovery for unsupervised hashing without bias learning.

3. Proposed method

DDCH learns a nonlinear hash function with a training set of N data points $\mathcal{X} = \{x_i \mid i = 1, 2, \ldots, N\}$, where each $x_i \in \mathbb{R}^D$ is a data point to be hashed. Let $f : x \to h \in \{0, 1\}^K$ denote the nonlinear hash function mapping data from the input space $\mathbb{R}^D$ to K-bit Hamming space $\{0, 1\}^K$. For data pairs $r_{ij} = (x_i, x_j), i = 1, \ldots, N; j = 1, \ldots, N$, DDCH encourages their hash codes to be close if they

are similar, otherwise far apart. Let $s_{ij} \in \{0, 1\}$ denote the pairwise similarity that is uncertain before training. If x_i and x_j are similar, we set $s_{ij} = 1$, otherwise $s_{ij} = 0$. DDCH uses s_{ij} learned by the neighborhood discovery module as the supervision token to learn representations and yield high-quality hash codes.

3.1. Architecture overview

The architecture of DDCH is illustrated in Fig. 3. The backbone is a Siamese network involving two branches sharing parameters. Each branch contains an encoder network (*e.g.* VGG16 [15]) and a hash layer. The hash layer contains two fully-connected (fc) layers with 1024 hidden units for each layer. In addition, the *batch normalization* [28] and *ReLU* functions, inspired by classic self-supervised representation learning frameworks [16,17], are deployed in the hash layer. The *Tanh* function is at the end of the layer.

DDCH processes a batch of data/images $\{x_i\}_{i=1}^M$ at a time, where M is the batch size. The image x_i is transformed into different views by augmentation strategies, *i.e.*, $\tau_A(\cdot)$ and $\tau_B(\cdot)$. Then, $\tau_A(x_i)$ and $\tau_B(x_i)$ are fed into the corresponding encoders and hash layers to generate continuous hash codes $h_i^A \in (-1, 1)^K$ and $h_i^B \in (-1, 1)^K$, respectively. For $\{x_i\}_{i=1}^M$, DDCH collects all h_i^A and h_i^B into $H_A = \left\{h_i^A\right\}_{i=1}^M$ and $H_B = \left\{h_i^B\right\}_{i=1}^M$, respectively. Next, DDCH performs neighborhood discovery for H_B and generates the pairwise similarity, *i.e.*, s_{ij}, depending on whether h_i^B and h_j^B are discovered to be neighbors with each other. Specifically, if h_i^B and h_j^B are neighbors, $s_{ij} = 1$; otherwise, $s_{ij} = 0$. The neighborhood discovery strategy generates similar and dissimilar relationships for the samples in H_B, being conducive to hash learning for the corresponding ones in H_A. The pairwise similarity s_{ij} can become the supervision token for pairwise hash codes (h_i^A, h_j^A) since contrastive learning frameworks expect H_B and H_A to share the same distribution. Besides, we establish a symmetric learning framework with a symmetric loss function to enable bidirectional cross-view supervision, eliminating the view gap of contrastive hashing. Furthermore, to enhance the quality of hash codes, we introduce Cauchy distribution based pairwise loss functions in Eq. (13) that is calculated with H_A and $S = \left\{s_{ij} | i = 1, \ldots, M; j = 1, \ldots, M\right\}$. We show the pseudocode describing the complete symmetric learning process in Algorithm 1.

For the test, we disable the neighborhood discovery module and generate continuous hash codes using the well-trained encoder and hash layer. Note that we directly share weights between the two branches of the Siamese framework [16,25], creating symmetrical network structures. Afterward, these continuous hash codes will be constrained to the discrete binary hash codes in $\{-1, 1\}^K$ by $sgn(\cdot)$, used to evaluate the retrieval performance.

```
# f: hash function
# ND: neighborhood discovery
# hyper-params: hyper-params of the neighborhood discovery
    strategies, i.e., number of clusters, number of nearest
     neighbors, and sifting threshold.
# CL: Cauchy distribution based loss functions in Equation
    {(14)}.
# load a mini-batch X with M samples
for x in loader:
# random augmentation
X_A, X_B = aug_A(X), aug_B(X)
# generate continuous hash codes
H_A, H_B = f(X_A), f(X_B)
# neighborhood discovery
S_A=ND(H_A.detach(), hyper-params)
S_B=ND(H_B.detach(), hyper-params)
# symmetric loss in Equation {(16)}
L = CL(H_A, S_B)/2 + CL(H_B, S_A)/2
# optimization step
L.backward() # back-propagate
update(f) # Adam update
```

Algorithm 1. DDCH Pseudocode, PyTorch-like.

3.2. Neighborhood discovery

Since neighborhood discovery can mine the latent structure of the datasets to reduce the number of false negatives and narrow the Hamming distance of similar samples, we introduce the algorithm to achieve debiased contrastive hashing. We provide three strategies for effective neighborhood discovery as follows.

Online Clustering (DDCH -C). Clustering methods [29,30] are an intuitive pattern for revealing between-instance similarity and are widely used in contrastive learning [31,32]. In practice, we incorporate K-Means into the contrastive learning framework via an online pattern. Hash codes within a mini-batch will be clustered into k classes. Specifically, if h_i^B and h_j^B are in the same cluster, $s_{ij} = 1$; otherwise, $s_{ij} = 0$. Formally, we define pairwise similarity, *i.e.*, s_{ij}, as follows:

$$s_{ij} = \begin{cases} 1 & \text{if } h_i^B \simeq h_j^B, \\ 0 & otherwise, \end{cases} \tag{1}$$

where $h_i^B \simeq h_j^B$ means that h_i^B and h_j^B are in the same cluster.

K-NN Search (DDCH -K). In the K-nearest neighbors search (K-NN) algorithm, s_{ij} can be formulated as

$$s_{ij} = \begin{cases} 1, & \text{if } h_i^B \in \mathcal{N}(h_j^B, \mathcal{K}) \text{ or } h_j^B \in \mathcal{N}(h_i^B, \mathcal{K}), \\ 0, & otherwise, \end{cases} \tag{2}$$

where $\mathcal{N}(h_i, \mathcal{K})$ represents the set of $\mathcal{K}$ nearest neighbors of h_i. The K-NN algorithm can provide a local perspective for similarity construction, which is different from the clustering method that concentrates on the global data distribution.

Threshold Sifting (DDCH -T). In addition, we can use the threshold sifting method to directly divide in-batch samples by the predefined threshold. We define pairwise similarity as

$$s_{ij} = \begin{cases} 1, & \text{if } sim(h_i^B, h_j^B) \geq \phi, \\ 0, & \text{if } sim(h_i^B, h_j^B) < \phi, \end{cases} \tag{3}$$

where $sim(h_i, h_j)$ is the cosine distance between h_i^B and h_j^B, ϕ is the threshold to sift positive samples.

3.3. DDCH Under EM perspective

In theory, we believe that the learning process of DDCH is the same as that of the EM algorithm. Let θ denote the parameters of hash function $f(\cdot)$. $H = f_\theta(\tau(\mathcal{X}))$ represents the continuous hash codes, where $\mathcal{X} = \{x_i\}_{i=1}^N$ is the training data and τ is the augmentation. Let $r_{ij} \in \left\{(x_i, x_j)\right\}_{i=1,j=1}^N$ and $s_{ij} \in \{0, 1\}$ denote the data pair (x_i, x_j) and the corresponding pairwise similarity, respectively. We estimate θ by logarithm Maximum A Posteriori Estimation (MAP),

$$\begin{aligned} \theta_{MAP} &= arg\max_\theta \left[\log L(\theta) + \log P(\theta)\right] \\ &= arg\max_\theta \left[\sum_{i=1,j=1}^{N} \int_\tau p(\tau) \log p(\tau(r_{ij}) \mid \theta) \mathrm{d}\tau + \log P(\theta)\right], \end{aligned} \tag{4}$$

where $L(\theta)$ is the likelihood function, $P(\theta)$ is the priori distribution of model parameters, $p(\tau)$ is the distribution of augmentations. We first discuss the solution for $L(\theta)$. We introduce the latent variable s and the corresponding distribution $q(s)$, satisfying

$\int_s q(s)\mathrm{d}s = 1$. As a result, we acquire a new formula,

$$\log L(\theta) = \underbrace{\sum_{i,j}^{N} \int_{\tau} p(\tau) \int_{s} q(s_{ij}) log \frac{p(\tau(r_{ij}), s_{ij} \mid \theta)}{q(s_{ij})} \mathrm{d}s\mathrm{d}\tau}_{\text{ELBO}} - \underbrace{\sum_{i,j}^{N} \int_{\tau} p(\tau) \int_{s} q(s_{ij}) log \frac{p(s_{ij} \mid \tau(r_{ij}), \theta)}{q(s_{ij})} \mathrm{d}s\mathrm{d}\tau}_{\text{KL-divergence}}, \quad (5)$$

where the latter item is the KL-divergence between $q(s_{ij})$ and $p(s_{ij} \mid \tau(r_{ij}), \theta)$. Since the KL-divergence item is non-negative, the former item becomes the evidence lower bound (ELBO). Obviously, the closer $q(s_{ij})$ and $p(s_{ij} \mid \tau(r_{ij}), \theta)$ are, the easier the optimization of ELBO and $L(\theta)$. It means the closer the distribution between H_A and H_B, the easier the solution of the EM algorithm. We believe it is the key to integrating the EM algorithm and contrastive learning. As a result, we assume $q(s_{ij}) = p(s_{ij} \mid \tau(r_{ij}), \theta)$, resulting in the KL-divergence item equaling 0. The following optimization of ELBO is equivalent to maximizing $L(\theta)$.

E-step. In this step, we aim to estimate the distribution $q(s_{ij})$. For the KL-divergence item in Eq. (5), we approximate it by sampling the augmentation from $p(\tau)$ only once [16], denoted as $\tau_B(\cdot)$. Since the KL-divergence item is set to 0, we can estimate $q(s_{ij})$ by the observed posterior distribution $p(s_{ij} \mid \tau_B(r_{ij}))$. Then, we use continuous hash codes generated in the orange branch of Fig. 3 to perform neighborhood discovery. The distribution $q^t(s_{ij})$ represents relationships between observed hash codes at step t in the orange branch. We can formulate the distribution under DDCH-C as

$$q^t(s_{ij}) = p\left(s_{ij} \mid \tau_B(r_{ij}), \theta^{t-1}\right) = \begin{cases} p(s_{ij}=1 \mid \tau_B(r_{ij}), \theta^{t-1}) = \begin{cases} 1 & \text{if } h_i^B \simeq h_j^B, \\ 0 & otherwise, \end{cases} \\ p(s_{ij}=0 \mid \tau_B(r_{ij}), \theta^{t-1}) = \begin{cases} 0 & \text{if } h_i^B \simeq h_j^B, \\ 1 & otherwise. \end{cases} \end{cases} \quad (6)$$

Note that θ is fixed in this step, as it can reveal the rationality to *stop gradients* in the orange branch [16].

M-step. In this step, we aim to optimize $L(\theta)$ based on the fixed $q^t(s_{ij})$ from Eq. (6). Specifically, we sample $\tau_A(\cdot)$ and ignore the constant items. As a result, we transform the optimization of $L(\theta)$ to the maximization of the item

$$\tilde{L}(\theta) = \sum_{i,j}^{N} \int_{s} q^t(s_{ij}) \log p\left(\tau_A(r_{ij}) \mid s_{ij}, \theta\right) p\left(s_{ij} \mid \theta\right) \mathrm{d}s, \quad (7)$$

where $q^t(s_{ij})$ is a constant obtained in E-step. Next, we estimate the priori distribution $p\left(s_{ij} \mid \theta\right)$ of DDCH-C as follows:

$$p(s_{ij} \mid \theta) = \left(\frac{n-1}{kn-1}\right)^{s_{ij}} \cdot \left(\frac{kn-n}{kn-1}\right)^{1-s_{ij}}, \quad (8)$$

where k denotes the cluster number, n denotes the number of samples in one cluster under the assumption of sample equilibrium. Note that the sample equilibrium is unnecessary for DDCH because the experimental results demonstrate that DDCH is insensitive to the initial distribution of samples (See Section 4.5). Since Hamming distances satisfying the Cauchy distribution can provide high retrieval precision for hash codes [19], we adopt the Cauchy distribution to define the posterior distribution and use the pairwise similarity calculated by Hamming distances. As a result, we acquire posterior distribution

$$p\left(\tau_A(r_{ij}) \mid s_{ij}, \theta\right) = \sigma\left(d\left(h_i^A, h_j^A\right)\right)^{s_{ij}} \left(1 - \sigma\left(d\left(h_i^A, h_j^A\right)\right)\right)^{1-s_{ij}}, \quad (9)$$

where

$$\sigma\left(d\left(h_i, h_j\right)\right) = \frac{\gamma}{\gamma + d\left(h_i, h_j\right)}, \quad (10)$$

γ is the scale parameter of the Cauchy distribution, the Hamming distance between continuous hash codes is defined below.

$$d\left(h_i, h_j\right) = \frac{K}{2}\left(1 - \cos\left(h_i, h_j\right)\right). \quad (11)$$

For the prior distribution $P(\theta)$, we transform it into a regularization of the continuous hash codes generated with θ. The Cauchy distribution are introduced again to control the quantization error caused by continuous relaxation. $P(\theta)$ is defined below.

$$P(\theta) = \sum_{i=1}^{N} \log \frac{\gamma}{\gamma + d\left(|h_i^A|, \mathbf{1}\right)}. \quad (12)$$

Incorporating Eqs. (7) and (12) into the MAP estimation framework, we obtain the loss function

$$L(\tau_A(\mathcal{X}), \tau_B(\mathcal{X})) = L_E + \lambda L_Q, \quad (13)$$

where λ is a hyper-parameter to acquire the preferred trade-off between the Cauchy cross- entropy loss L_E and the Cauchy quantization loss L_Q [19]. Note that switching the position of $\tau_A(\mathcal{X})$ and $\tau_B(\mathcal{X})$ leads to different results, in which case the pairwise similarity s_{ij} will be derived from the results of H_A. The Cauchy cross-entropy loss L_E is derived as

$$L_E = -\sum_{i,j}^{N} \left(s_{ij} \log\left(\frac{n-1}{kn-1}\sigma\left(d\left(h_i^A, h_j^A\right)\right)\right)\right) - \sum_{i,j}^{N} \left((1-s_{ij}) \log\left(\frac{kn-n}{kn-1}\left(1 - \sigma\left(d\left(h_i^A, h_j^A\right)\right)\right)\right)\right). \quad (14)$$

Note that since the assumption of sample equilibrium, *i.e.*, the coefficient consisting of k and n, is a constant, and the gradient of the coefficient equals 0 in the training process, the assumption will not be decisive for hash learning. Besides, the Cauchy quantization loss L_Q is derived as

$$L_Q = \sum_{i=1}^{N} \log\left(1 + \frac{d(|h_i|, \mathbf{1})}{\gamma}\right). \quad (15)$$

3.4. Overall learning objective

Leveraging the outcomes of the neighborhood discovery module under one augmentation to supervise the hash learning in the branch under the other augmentation would result in an overconfident problem due to the view gaps between the various augmentations. Moreover, it will cause limited alignment across different views, resulting in sub-optimal representations [33]. As a result, we provide a symmetric learning pattern with a symmetric loss function to enable bidirectional cross-view supervision that it helps exploit the similarities and differences between views for superior retrieval performance. Following [16,17], the loss function is defined below.

$$L = \frac{1}{2}L(\tau_A(\mathcal{X}), \tau_B(\mathcal{X})) + \frac{1}{2}L(\tau_B(\mathcal{X}), \tau_A(\mathcal{X})). \quad (16)$$

4. Experiments

In this section, we conduct extensive experiments compared with state-of-the-art unsupervised and self-supervised hashing methods to verify the superiority of DDCH. Note that our results are generated by DDCH-C if without a special statement.

Table 1
The comparisons of DDCH and state-of-the-art unsupervised/self-supervised hashing methods in terms of mAP.

Method	Reference	CIFAR-10			MS-COCO			NUS-WIDE		
		16bits	32bits	64bits	16bits	32bits	64bits	16bits	32bits	64bits
DeepBit [6]	CVPR16	0.194	0.249	0.277	0.407	0.419	0.430	0.392	0.403	0.429
SGH [9]	ICML17	0.435	0.437	0.433	0.594	0.610	0.618	0.593	0.590	0.607
BGAN [34]	AAAI18	0.525	0.531	0.562	0.645	0.682	0.707	0.684	0.714	0.730
BinGAN [38]	NIPS18	0.476	0.512	0.520	0.651	0.673	0.696	0.654	0.709	0.713
GreedyHash [36]	NIPS18	0.448	0.473	0.501	0.582	0.688	0.710	0.633	0.691	0.731
DVB [10]	IJCV19	0.403	0.422	0.446	0.570	0.629	0.623	0.604	0.632	0.665
DistillHash [39]	CVPR19	0.284	0.285	0.288	-	-	-	0.667	0.675	0.677
TBH [37]	CVPR20	0.532	0.573	0.578	0.706	0.735	0.722	0.717	0.725	0.735
DATE [7]	ACM MM21	0.567	0.621	0.636	-	-	-	0.765	0.784	0.790
CIBHash [12]	IJCAI21	0.590	0.622	0.641	0.713	0.742	0.757	0.770	0.785	0.792
DDCH	**Ours**	**0.611**	**0.648**	**0.658**	**0.721**	**0.759**	**0.779**	**0.781**	**0.798**	**0.808**

4.1. Datasets

We use public datasets commonly used in hashing tasks [10,12,34] to evaluate the performance.

CIFAR-10 [35] consists of 60,000 images containing 10 classes. We follow the common setting [10,12] and select 5000 images (500 per class) as the query set and 5000 images (500 per class) as the training set. The remaining images are regarded as the database.

NUS-WIDE [22] is a multi-label dataset containing 269,648 images from 81 categories. We select the 21 most frequently used for our task [1,12]. We select 100 images per class as the query set and use the remaining images as the database. Furthermore, we select 500 images per class from the database for training.

MS-COCO [21] is a commonly used multi-label dataset for detection, segmentation and captioning. We randomly select 5000 images from the subset as the query set. The remaining images constitute the database, where 10,000 images are used for training.

4.2. Evaluation metrics and baselines

We evaluate all methods by the standard metrics [36,37] consisting of Mean Average Precision (mAP), Precision-Recall (P-R) curves, and Precision curves within Hamming radius 2 (P@H$\leq$2). According to [12,37], we adopt mAP@1000 for CIFAR-10 as well as mAP@5000 for MS-COCO and NUS-WIDE. We compare DDCH with several state-of-the-art unsupervised/self-supervised hashing methods, including DeepBit [6], SGH [9], BGAN [34], BinGAN [38], GreedyHash [36], DVB [10], DistillHash [39], TBH [37], DATE [7], and CIBHash [12].

4.3. Implementation details

We exploit the same augmentation strategies as CIBHash [12], including cropping, color distortions, Gaussian blur, *etc*., to transform the resized images into different views. For fairness, we deploy a pre-trained VGG16 network [15] with fixed parameters as the encoder network [10,12,36,37]. The hash layer contains two fully-connected layers, where the 4096-dimensional feature vector from the encoder is first converted to 1024 dimensions, then to k dimensions. Inspired by the classic self-supervised learning frameworks [16,17], the hash layer deploys *batch normalization* and *ReLU* after each fully-connected layer. The *Tanh* function is deployed at the end of the hash layer to constrain the hash code to be within $(-1, 1)$. We prefabricate all feature vectors via the pre-trained encoder, achieving fast training with a large batch size. We set the default batch size $M = 1024$, hyper-parameters $\gamma = 2.0$ (Eq. (10)), and $\lambda = 0.05$ (Eq. (13)). We run the model on *PyTorch* and utilize *Adam* as the optimizer, using a learning rate of 0.0005. All results given by default are obtained at the optimal k, $\mathcal{K}$, and ϕ.

4.4. Result analysis

mAP comparisons. We list the mAP results in Table 1. Obviously, DDCH achieves the best performance on all three datasets. Compared with the state-of-the-art contrastive hashing method, *i.e*., CIBHash, DDCH promotes mAP at least 1.1%, 1.4%, and 2.6% at different code lengths on MS-COCO, NUS-WIDE, and CIFAR-10, respectively. In particular, the increases at 64 bits are the largest (*i.e*., 2.91%, 2.02%, and 2.65%, respectively). Additionally, the significance tests indicates that DDCH significantly beats CIBHash with $p < .001$ (e.g., the p-values for mAP@64-bits on CIFAR-10, MS-COCO, and NUS-WIDE are $5.53e^{-5}$, $9.59e^{-5}$, and $2.08e^{-4}$, respectively). These results demonstrate that DDCH has a greater ability to generate distinguishable hash codes. We believe ND helping to exploit the structural information of the datasets to reduce the detriment from false-negatives matters a lot.

Notably, for fair baseline comparisons, we report the mAP results in Table 1 using VGG16 as the encoder backbone [15]. We further list the mAP results of DDCH using different representation learning backbones in Table 2, including VGG16 [15], ResNet50 [40], and ViT-B/16 [41]. The results under ResNet50 are close to those of supervised methods [1,19]. Meanwhile, ViT-B/16 boosts the mAP 42.1% compared with VGG16 on CIFAR-10. Obviously, accurate features can substantially improve hash quality. In addition, we believe that co-training the backbone and the hash layer will further improve the performance.

Precision and recall comparisons. The retrieval performance in terms of P@H$\leq$2 at different code lengths, P-R curves at 64 bits, and precision curves with different number of top returned images at 64 bits are shown in Fig. 4. As shown in Fig. 4 (a) and (d), DDCH can achieve better P@H$\leq$2 results at all code lengths than all comparison methods. Especially at 64 bits, the P@H$\leq$2 results overtake 6.4% and 8.9% on MS-COCO and CIFAR-10, respectively. It validates that DDCH can concentrate hash codes of similar data points together to be within the Hamming radius 2. The superior results also are shown in Fig. 4 (b), (c), (e) and (f). DDCH outperforms the state-of-the-art methods by large margins in terms of the P-R curves and P@N on MS-COCO and CIFAR-10. **Intra-distance and inter-distance comparisons.** We report the mean intra-distance d_{intra} and the mean inter-distance d_{inter} on CIFAR-10 in Table 3.

Table 2
The mAP performance with various representation backbones.

Encoder	MS-COCO			CIFAR-10		
	16bits	32bits	64bits	16bits	32bits	64bits
VGG16	0.721	0.759	0.779	0.611	0.648	0.658
ResNet50	0.777	0.807	0.826	0.675	0.734	0.756
ViT-B/16	**0.816**	**0.866**	**0.875**	**0.882**	**0.923**	**0.935**

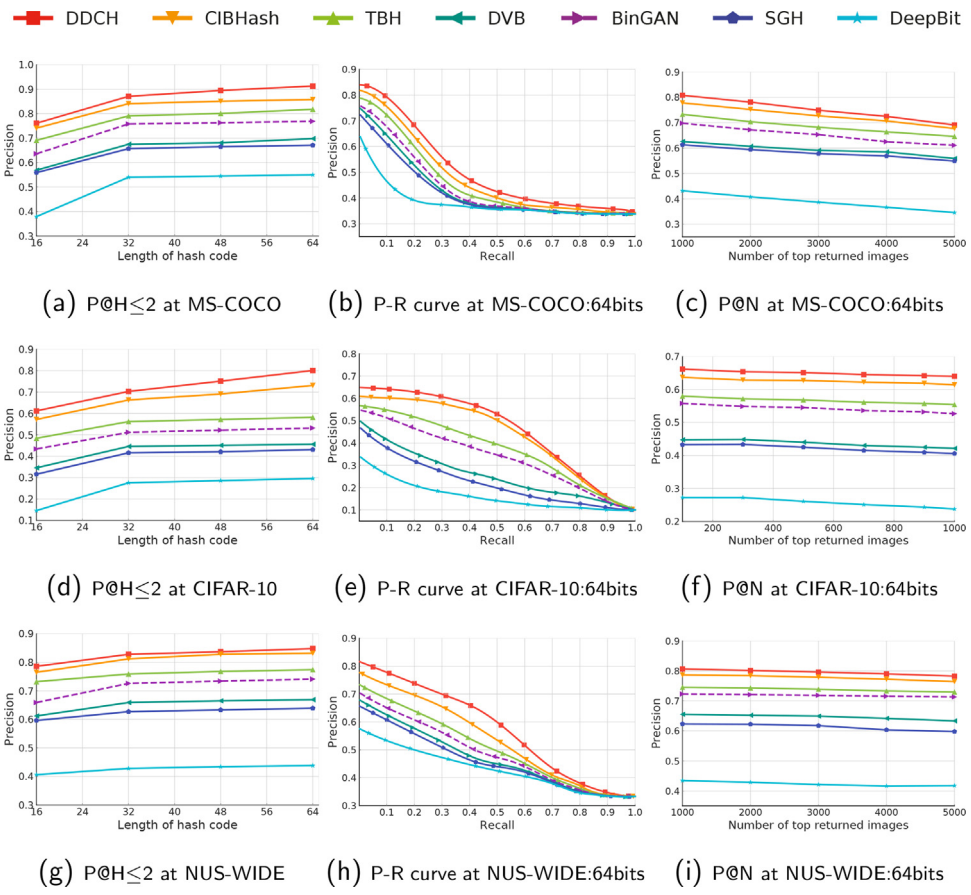

Fig. 4. P@H≤2, P-R curves, and P@N of DDCH and comparison methods on CIFAR-10 and MS-COCO.

Table 3
The d_{intra} and d_{inter} on CIFAR-10.

Method	$d_{intra}\downarrow$			$d_{inter}\uparrow$		
	16bits	32bits	64bits	16bits	32bits	64bits
DATE	2.78	5.87	11.78	8.34	17.08	33.25
CIBHash	2.73	5.68	11.09	8.48	17.21	33.72
DDCH (Ours)	**2.71**	**5.42**	**10.31**	**8.69**	**17.37**	**34.83**

It is clear that smaller d_{intra} indicates better cluster performance, while larger d_{inter} indicates greater disentangle ability on confusion samples. Compared with existing contrastive learning based methods [7,12], DDCH achieves the smallest d_{intra} and the largest d_{inter}, which implies excellent retrieval performance.

4.5. Ablation study

In this section, we investigate the mAP performance of DDCH with different components. **ND** represents neighborhood discovery, including the **offline** and **online** neighborhood discovery patterns. **SL** denotes the symmetric loss function (See Eq. (16)), and **SE** means that the input in a batch satisfies sample equilibrium by pre-clustering (See Eq. (8)). The results of all variants at different code lengths on MS-COCO and CIFAR-10 are listed in Table 4. Furthermore, we investigate the retrieval performance with different neighborhood discovery strategies in Table 5.

ND. We first explore the effect of neighborhood discovery (ND) in DDCH. As shown in the first row of Table 4, disabling the neighborhood discovery plunges all results to a bottom. Therefore, ND is crucial to exploring between-instance similarities and achieving debiased contrastive hash learning. In addition, we study the effect of offline ND, *i.e.*, we perform ND before every epoch with all training samples to generate s_{ij}. Although, in theory, this pattern should generate better results due to the global view rather than the batch view, the results in the second row of Table 4 do not show the prospective advantages compared with the results of the online pattern in the fifth row. The reason may be that unsupervised ND strategies create supervisory information with noise, and when this information is overused, it interferes with the capture of valid features by contrastive learning. We further discuss this issue in Section 5.

Table 4
The mAP under different components.

ND		SL	SE	MS-COCO			CIFAR-10		
offline	online			16bits	32bits	64bits	16bits	32bits	64bits
✗	✗	√	✗	0.699	0.726	0.745	0.563	0.608	0.623
√	✗	√	✗	0.718	0.754	**0.780**	0.581	0.634	0.649
✗	√	✗	✗	0.704	0.747	0.767	0.555	0.629	0.646
✗	√	√	√	**0.723**	**0.761**	0.778	0.607	0.646	0.654
✗	√	√	✗	0.721	0.759	0.779	**0.611**	**0.648**	**0.658**

Table 5
The mAP under different neighborhood discovery strategies.

variants	MS-COCO			NUS-WIDE			CIFAR-10		
	16bits	32bits	64bits	16bits	32bits	64bits	16bits	32bits	64bits
DDCH-C	**0.721**	**0.759**	**0.779**	0.781	0.798	0.808	0.611	**0.648**	0.658
DDCH-K	0.707	0.743	0.761	**0.789**	**0.799**	**0.819**	**0.618**	0.647	**0.661**
DDCH-T	0.570	0.643	0.666	0.775	0.791	0.802	0.478	0.538	0.573

SL. The symmetric loss function is designed for bidirectional cross-view supervision. To determine its effectiveness for hash learning, we test its influence and show the results in the second row of Table 4. Obviously, the mAP performance will drop at all code lengths when disabling **SL**, especially at 16 bits. We believe that the symmetric loss function is crucial to alleviating the over-confident problem as well as achieving superior alignment between different views.

SE. We pre-cluster feature vectors into k clusters. In the training phase, we randomly select equal amounts of data from each cluster to construct the input of a batch. As shown in the first row of Table 4, enabling SE will slightly boost the mAP performance on MS-COCO at 16 and 32 bits while slightly decreasing it on CIFAR-10. These results confirm the prophesy of Eq. (14). As a result, DDCH is robust to unbalanced training data.

ND Strategies. We investigate the mAP performance under different ND strategies and report the results in Table 5. We can draw the following two conclusions from the results: (1) Neighborhood discovery with online clustering (DDCH-C) achieves optimal performance on MS-COCO while DDCH-K excels on NUS-WIDE and CIFAR-10, which implies that K-NN search is more suitable for smaller datasets with fewer semantic categories. (2) DDCH-T achieves inferior performance on all datasets. We suspect that the cosine similarity used as the sifting metric is incapable of discriminating between true neighbors.

4.6. Sensitivity study

In this section, we study the mAP performance w.r.t different parameters' settings, including parameters in neighborhood discovery (ND), *i.e.*, number of clustering k in DDCH-C, number of nearest neighbors $\mathcal{K}$ in DDCH-K, threshold ϕ in DDCH-T, batch sizes, and hyper-parameters γ and λ.

Parameters of ND. To determine the number of clusters k, the number of nearest neighbors $\mathcal{K}$, and the sifting threshold ϕ for DDCH, we show the performance varying the three parameters on MS-COCO, CIFAR-10, and NUS-WIDE in Fig. 5(a)-(i). The following four conclusions can be drawn from the results. (1) When k

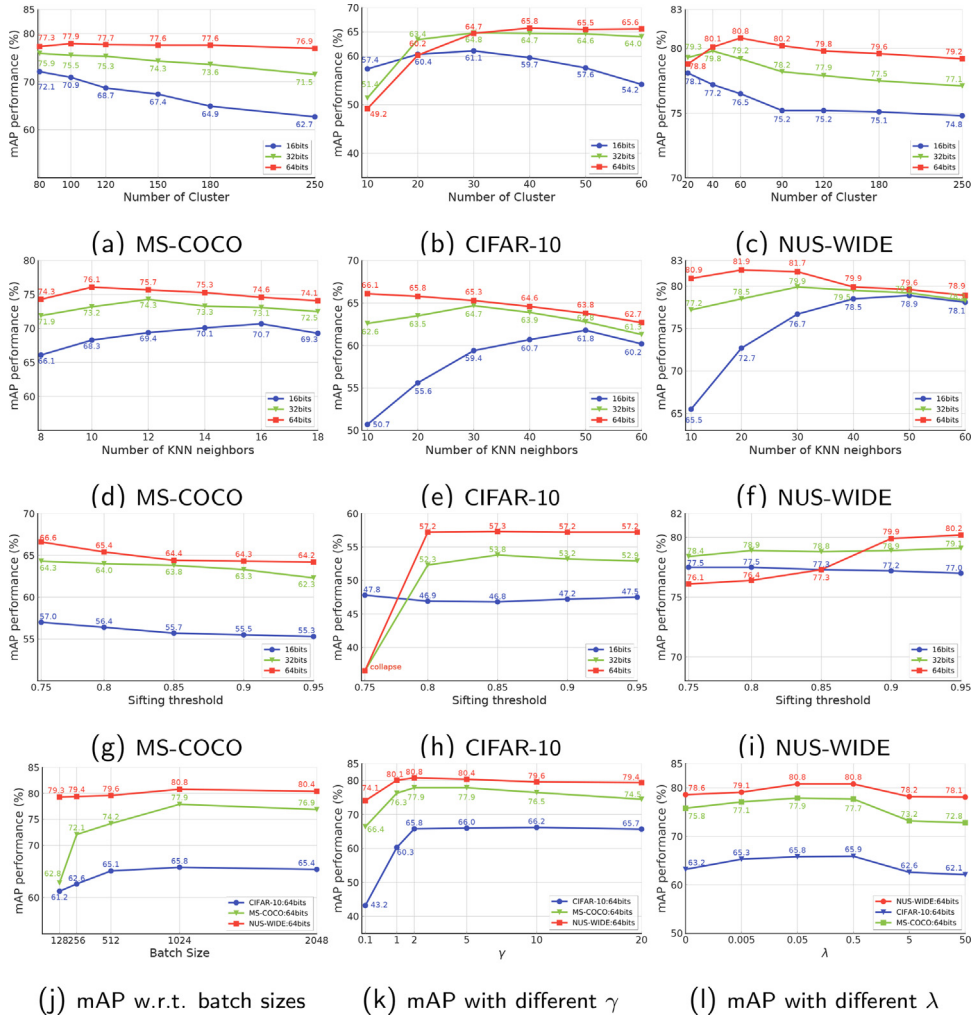

Fig. 5. Sensitivity study for DDCH w.r.t number of clusters, batch size, and hyper-parameters.

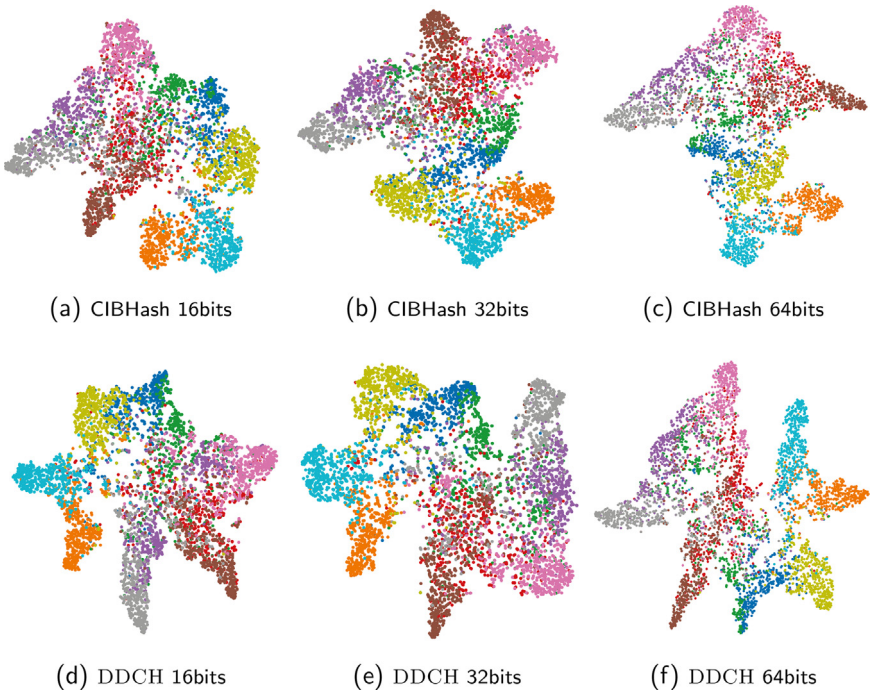

Fig. 6. The t-SNE visualization of the learned hash codes for CIFAR-10 test samples. The scatters of the same color indicate the same category.

is greater than two times the actual number of categories, DDCH achieves relatively stable performance at 32 and 64 bits on three datasets as (a), (b), and (c) show. As a result, the selection of k does not become a bottleneck limiting the practicality of DDCH when reasonably predicting the number of categories. (2) DDCH-T is insensitive to the threshold ϕ when we set ϕ from 0.8 to 0.95, but at the same time, we can not obtain satisfying results. Furthermore, we also notice that the small ϕ will lead to trivial solutions in our experiments, in which all the codes are discovered to be neighbors with each other. (3) DDCH-K is a bit more sensitive to the number of nearest neighbors, *i.e.*, $\mathcal{K}$, especially on MS-COCO, since MS-COCO has more complex class divisions. (4) The short hash codes prefer small k and large $\mathcal{K}$. Intuitively, hash codes with small lengths contain less semantic information and show fewer diversities, resulting in performance degradation with a large number of clusters or a small number of neighbors. **Batch Size.** We test the performance with different batch sizes since contrastive learning based methods rely on large batch sizes to provide sufficient negative samples. According to the best performance at 64 bits shown in Fig. 5(a), (b), and (c), we set $k = 100$, $k = 40$, and $k = 60$ for MS-COCO, CIFAR-10, and NUS-WIDE, respectively. Then, we test the mAP performance varying with the batch size at 64 bits. As shown in Fig. 5(d), the results on CIFAR-10 and NUS-WIDE are relatively stable, while the results on MS-COCO are stable after the batch size exceeds 512. Our findings show that smaller batch sizes can lead to decreased mAP performance due to two factors: 1) the relationships constructed by the ND module are less accurate with smaller batch sizes, and 2) smaller batch sizes provide fewer negative samples for our contrastive learning framework, which requires a large number of negative samples for effective learning, as seen in other methods such as SimCLR and MoCo. Obviously, a large batch size can provide superior performance. Since all feature vectors can be pre-computed, the training process barely consumes GPU memory in the big batch size setting, making DDCH competent for resource-constrained scenarios.

Hyper-parameters γ and λ**.** Fig. 5 (e) shows the mAP curves w.r.t different γ of Cauchy distribution [19]. Large γ results in loose hash codes while small one provides over tight constraint. We obtain the best mAP performance on MS-COCO when $\gamma = 2$ while the counterpart on CIFAR-10 is $\gamma = 10$. Fig. 5 (f) shows the mAP curves w.r.t different λ trading-off loss items in Eq. (13). A large λ will weaken the L_E function and EM effect, while a small λ is incapable of reducing the quantization error caused by continuous relaxation. As a result, $\lambda = 0.05$ is apt for hashing.

4.7. Visualization

Visualization of Hash Codes by t-SNE. Fig. 6 shows the t-SNE visualization [42] of the hash codes with different code lengths on CIFAR-10. The hash codes generated by DDCH show clear discriminative structures, where the most hash codes in different categories are well separated. Besides, hash codes at 64 bits have farther inter-class distances than those at 16 bits. It also explains why the retrieval results at 64 bits are better than those at 16 bits in the previous experiments. **Top-10 Retrieved Results.** We visualize the top 10 retrieved images for three query images selected from MS-COCO, NUS-WIDE, and CIFAR-10, respectively. As shown in Fig. 7, DDCH can yield more relevant and accurate retrieval results than CIBHash.

5. Discussion

Learning with more views. In our implementation, the augmentation strategies are kept in line with CIBHash [12], and we learn view-invariant hash codes with only two views, *i.e.*, we sample the augmentation only once in both E-step and M-step. In practice, DDCH can achieve better retrieval performance with more views. We update DDCH to accommodate learning with multi-views, and show the new framework in Fig. 8. In E-step (orange branch), we perform neighborhood discovery for the hash codes from each view, *i.e.*, each augmentation, to obtain the pairwise similarity. Then, we can generate the joint pairwise similarity as the supervision token for hash learning of each view in the M-step (blue branch) by the voting mechanism. We represent the mAP results of the updated version at 64 bits in Table 6, where Multi-E and Multi-M denote the numbers of views in E-step and M-step,

Fig. 7. Retrieval comparisons to CIBHash [12] at 64 bits on three datasets. We report precision within the top-10 retrieved images (P@10) and the results demonstrate that our proposed DDCH outperforms CIBHash with more relevant and accurate returned images as well as fewer contradictions.

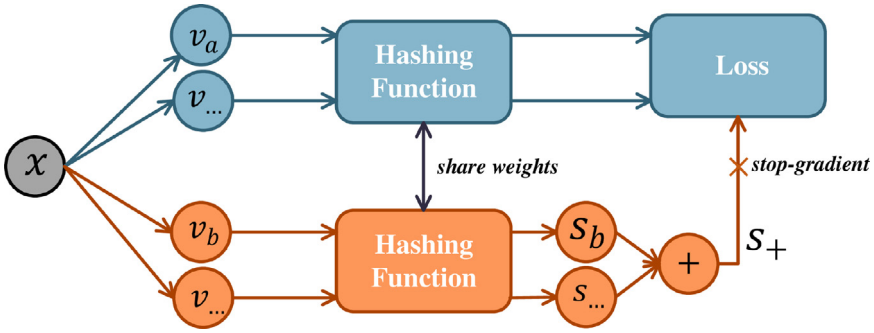

Fig. 8. The framework of updated DDCH when learning with multi-views. v is the augmented view of the image x. s is the pairwise similarity corresponding to the view. The joint pairwise similarity s_+ is the result of multiple views *voting* in E-step (orange branch), which is different from the baseline that generates pairwise similarity with only one view. Besides, the hash learning of each view in M-step (blue branch) is supervised by the joint pairwise similarity. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 6
mAP results at 64 bits when learning with multi-views.

Multi-E	Multi-M	CIFAR10-64bit	MS-COCO-64bit
1	1	0.658	0.779
1	2	0.661	0.785
2	1	0.663	0.787
2	2	**0.667**	0.790
3	3	0.664	**0.791**

respectively. Note that the baseline results of DDCH are listed in the first row of Table 6. These results demonstrate that more views are conducive to learning the more robust hash codes. Furthermore, we notice that mAP does not rise linearly with the number of views. We ascribe this phenomenon to the redundancy of views. As a result, we believe that the quality of the view also affects performance improvement.

Neighborhood Discovery Position. We are interested in what kind of information should be used to do neighborhood discovery in DDCH. Intuitively, more accurate features imply higher quality hash codes. Since the ***tanh*** function may cause information loss, we first deploy the neighborhood discovery module before the ***tanh*** function, *i.e.*, we run the E-step branch without the ***tanh*** function (wo/*tanh*). As shown in Table 7, compared with results of DDCH-C, there is no improvement in mAP. After that, we further resorted to the original features from the $fc-7$ layer of VGG. As a result, we run the E-step branch without the hash layer (wo/h), *i.e.*, clustering with high-dimensional features derived from the pre-trained backbone. However, the results are worse than the first attempt. These results show that the alignment of the data distribution is key to contrastive hashing.

Search for a near-optimal cluster number. The optimal values of the parameters vary with different datasets, where the hyper-parameters (*e.g.*, cluster number) in the neighborhood discovery module greatly affect the performance. As a result, we discuss searching for a near-optimal value, *e.g.*, a near-optimal number of clusters k, for the ND module. We focus on two solutions: 1) We can use non-parametric methods, e.g., Bayesian non-parametric (BNP) mixture models, and more specifically, the DPM models [43,44] to implement the neighborhood discovery module, since the ND module is open-ended and can be implemented by any similarity construction method. These non-parametric models do not take a hyper-parameter (e.g., k) as input and scale well to large-scale datasets. Therefore, we use the DeepDPM model [45] to implement ND, and the results are reported in Table 8. The results show that non-parametric methods can also achieve decent performance. 2) Hyper-parameter optimization methods can approximate a satisfying k. Table 8 lists the mAP performance using the Bayesian optimization method [46] to search for a optimal k. In addition, the model selection based method (*i.e.*, a hyper-parameter

Table 7
mAP under different neighborhood discovery positions.

clustering position			MS-COCO			CIFAR-10		
w/*tanh*	wo/*tanh*	wo/h	16bits	32bits	64bits	16bits	32bits	64bits
✗	√	✗	0.715	0.753	0.776	0.605	0.639	0.652
✗	✗	√	0.691	0.712	0.731	0.577	0.597	0.612
√	✗	✗	**0.721**	**0.759**	**0.779**	**0.611**	**0.648**	**0.658**

Table 8
The mAP under different search methods.

Methods	MS-COCO			CIFAR-10		
	16bits	32bits	64bits	16bits	32bits	64bits
Bayesian Optimization	**0.726**	0.756	0.778	0.610	0.648	**0.660**
DeepDPM	0.715	0.751	0.768	0.603	0.644	0.653
Baseline (Grid Search)	0.721	**0.759**	**0.779**	**0.611**	**0.648**	0.658

Table 9
The mAP under different supervision manners.

SM	MS-COCO			NUS-WIDE			CIFAR-10		
	16bits	32bits	64bits	16bits	32bits	64bits	16bits	32bits	64bits
supervised-DCH	0.667	0.682	0.689	0.677	0.690	0.699	0.734	0.748	0.759
supervised-DDCH	0.709	0.702	0.699	0.724	0.717	0.712	**0.762**	**0.768**	**0.768**
unsupervised-MoCo(H)	0.715	0.749	0.766	0.770	0.789	0.794	0.590	0.632	0.647
unsupervised-DDCH	**0.721**	**0.759**	**0.779**	**0.781**	**0.798**	**0.808**	0.611	0.648	0.658

optimization method) is capable of searching for a near-optimal configuration. However, they are mostly computationally expensive since the training process would have to be repeated numerous times.

Extension to supervised learning. In this section, we discuss two issues: 1) Does multi-view learning improve the performance of supervised hashing? 2) Does contrastive hashing have the potential to surpass traditional supervised hashing?

We first run DDCH using the similarities obtained from the hand-crafted classification labels. The mAP performance comparisons between supervised-DDCH and supervised-DCH are shown in the first two rows in Table 9. Note that supervised-DDCH is different from DCH mainly in that DDCH add an additional regularization term to maximize the consistency between different views augmented from one image. As we can see from the results, hash learning under multiple views can generate transformation-invariant hash codes, offering superior retrieval performance [47].

We also compare the performance of supervised-DDCH and several unsupervised ones, including unsupervised-DDCH and unsupervised-MoCo(H). For unsupervised-MoCo(H), we modify the original MoCo V2 framework to adapt to hash learning following [12]. We note that unsupervised methods outperform supervised ones over MS-COCO and NUS-WIDE. We have reason to believe that similarity relationships constructed from labels may be less reliable than those constructed between visual features in multi-label datasets. This is because, according to most supervised methods, two images are considered similar as long as they share at least one label, which can result in biased hash learning. On the other hand, the similarity constructed by unsupervised-DDCH can reveal more details, such as size information, which may not be fully realized by the supervised approach.

6. Limitations and future works

Although DDCH yields optimal results, there are still some issues we need to address in the future.

- Since the parameters of the model will be fixed upon being deployed, the hash function cannot adapt to changes in data distribution, which means that the retrieval performance will decline as the amount of data increases.
- The proposed method can only reduce the number of false-negative samples, not eliminate them. We would like to investigate more effective ways to further improve the performance of contrastive hashing.
- DDCH learns continuous hash codes in the training phase and employs *sgn* to meet the binary constraint in the test, causing considerable information loss. It is necessary to research the operative binary optimization algorithms to achieve superior retrieval performance with binary codes.

7. Conclusion

We present a debiased contrastive learning framework for unsupervised hashing tasks, namely DDCH, that outperforms existing unsupervised hashing methods in three benchmark image datasets. To alleviate the issues of instance discrimination, we incorporate the neighborhood discovery module to provide similar and dissimilar information for samples, using latent structural information in the datasets and reducing the detriment from false negatives, resulting in debiased contrastive hashing. The introduction of neighborhood discovery brings both a high quality of hash codes and superior image retrieval performance. In addition, we explain the rationale for introducing neighborhood discovery in contrastive learning from an Expectation-Maximization (EM) perspective. We hope our study can inspire the community to rethink the role of contrastive learning for unsupervised hashing tasks.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

This work was also achieved in Key Laboratory of Information Storage System and Ministry of Education of China. It was supported by the National Natural Science Foundation of China No. 61902135, the National Natural Science Foundation of China Grant No. 62232007, Natural Science Foundation of Hubei Province No. 2022CFB060, and Ministry of Education - China Mobile Research Funding No. MCM20180406.

References

[1] L. Yuan, T. Wang, X. Zhang, F.E.H. Tay, Z. Jie, W. Liu, J. Feng, Central similarity quantization for efficient image and video retrieval, in: CVPR, 2020, pp. 3080–3089.
[2] F. Yang, Y. Liu, X. Ding, F. Ma, J. Cao, Asymmetric cross-modal hashing with high-level semantic similarity, Pattern Recognit. 130 (2022) 108823.
[3] C. Fu, G. Wang, X. Wu, Q. Zhang, R. He, Deep momentum uncertainty hashing, Pattern Recognit. 122 (2022) 108264.
[4] Y. Duan, N. Chen, P. Zhang, N. Kumar, L. Chang, W. Wen, MS2Gah: multi-label semantic supervised graph attention hashing for robust cross-modal retrieval, Pattern Recognit. 128 (2022) 108676.
[5] C. Deng, Z. Chen, X. Liu, X. Gao, D. Tao, Triplet-based deep hashing network for cross-modal retrieval, IEEE Trans. Image Process. 27 (8) (2018) 3893–3903.
[6] K. Lin, J. Lu, C.-S. Chen, J. Zhou, Learning compact binary descriptors with unsupervised deep neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 1183–1192.
[7] X. Luo, D. Wu, Z. Ma, C. Chen, M. Deng, J. Huang, X.-S. Hua, A statistical approach to mining semantic similarity for deep unsupervised hashing, in: MM '21, 2021, pp. 4306–4314.
[8] Y. Wang, J. Song, K. Zhou, Y. Liu, Unsupervised deep hashing with node representation for image retrieval, Pattern Recognit. 112 (2021) 107785.
[9] B. Dai, R. Guo, S. Kumar, N. He, L. Song, Stochastic generative hashing, in: International Conference on Machine Learning, PMLR, 2017, pp. 913–922.
[10] Y. Shen, L. Liu, L. Shao, Unsupervised binary representation learning with deep variational networks, volume 127, 2019, pp. 1614–1628.
[11] R.-C. Tu, X. Mao, W. Wei, MLS3RDUH: deep unsupervised hashing via manifold based local semantic similarity structure reconstructing, in: C. Bessiere (Ed.), IJCAI, ijcai.org, 2020, pp. 3466–3472.
[12] Z. Qiu, Q. Su, Z. Ou, J. Yu, C. Chen, Unsupervised hashing with contrastive information bottleneck, in: Z.-H. Zhou (Ed.), IJCAI, 2021, pp. 959–965.
[13] J. Zhang, Y. Peng, M. Yuan, Unsupervised generative adversarial cross-modal hashing, AAAI, 2018.
[14] E. Yang, C. Deng, T. Liu, W. Liu, D. Tao, Semantic structure-based unsupervised deep hashing, in: IJCAI, 2018, pp. 1064–1070.
[15] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, ICLR, 2015.
[16] X. Chen, K. He, Exploring simple siamese representation learning, in: CVPR, 2021, pp. 15750–15758.
[17] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P.H. Richemond, E. Buchatskaya, C. Doersch, B.A. Pires, Z. Guo, M.G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, M. Valko, Bootstrap your own latent - A new approach to self-supervised learning, Advances in Neural Information Processing Systems, 2020.
[18] K. He, H. Fan, Y. Wu, S. Xie, R.B. Girshick, Momentum contrast for unsupervised visual representation learning, in: CVPR, 2020, pp. 9726–9735.
[19] Y. Cao, M. Long, B. Liu, J. Wang, Deep cauchy hashing for hamming space retrieval, in: CVPR, Computer Vision Foundation / IEEE Computer Society, 2018, pp. 1229–1237.
[20] H. Alwassel, D. Mahajan, B. Korbar, L. Torresani, B. Ghanem, D. Tran, Self–supervised learning by cross-modal audio-video clustering, in: H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, H.-T. Lin (Eds.), Advances in Neural Information Processing Systems, 2020.
[21] T.-Y. Lin, M. Maire, S.J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C.L. Zitnick, Microsoft COCO: common objects in context, in: ECCV, volume 8693, 2014, pp. 740–755.
[22] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, Y. Zheng, NUS-WIDE: a real-world web image database from national university of singapore, CIVR, 2009.
[23] B. Kulis, T. Darrell, Learning to hash with binary reconstructive embeddings, Adv. Neural Inf. Process. Syst. 22 (2009).
[24] C. Deng, E. Yang, T. Liu, J. Li, W. Liu, D. Tao, Unsupervised semantic-preserving adversarial hashing for image search, IEEE Trans. Image Process. 28 (8) (2019) 4032–4044.
[25] T. Chen, S. Kornblith, M. Norouzi, G.E. Hinton, A simple framework for contrastive learning of visual representations, in: ICML, volume 119, 2020, pp. 1597–1607.
[26] N. Tishby, N. Zaslavsky, Deep learning and the information bottleneck principle, in: ITW, 2015, pp. 1–5.
[27] Z. Wu, Y. Xiong, S.X. Yu, D. Lin, Unsupervised feature learning via non-parametric instance discrimination, in: CVPR, 2018, pp. 3733–3742.
[28] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: ICML, 37, 2015, pp. 448–456.
[29] X. Yang, C. Deng, F. Zheng, J. Yan, W. Liu, Deep spectral clustering using dual autoencoder network, in: CVPR, 2019, pp. 4066–4075.
[30] J. MacQueen, et al., Some methods for classification and analysis of multivariate observations, in: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, volume 1, Oakland, CA, USA, 1967, pp. 281–297.
[31] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, A. Joulin, Unsupervised learning of visual features by contrasting cluster assignments, Advances in Neural Information Processing Systems, 2020.
[32] J. Li, P. Zhou, C. Xiong, S.C.H. Hoi, Prototypical contrastive learning of unsupervised representations, ICLR, 2021.
[33] T. Wang, P. Isola, Understanding contrastive representation learning through alignment and uniformity on the hypersphere, in: ICML, volume 119, 2020, pp. 9929–9939.
[34] J. Song, T. He, L. Gao, X. Xu, A. Hanjalic, H.T. Shen, Binary generative adversarial networks for image retrieval, in: AAAI, 2018, pp. 394–401.
[35] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images, Citeseer, 2009.
[36] S. Su, C. Zhang, K. Han, Y. Tian, Greedy hash: Towards fast optimization for accurate hash coding in CNN, in: S. Bengio, H.M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, 2018, pp. 806–815.
[37] Y. Shen, J. Qin, J. Chen, M. Yu, L. Liu, F. Zhu, F. Shen, L. Shao, Auto-encoding twin-bottleneck hashing, in: CVPR, 2020, pp. 2815–2824.
[38] M. Zieba, P. Semberecki, T. El-Gaaly, T. Trzcinski, Bingan: Learning compact binary descriptors with a regularized GAN, in: Advances in Neural Information Processing Systems, 2018, pp. 3612–3622.
[39] E. Yang, T. Liu, C. Deng, W. Liu, D. Tao, Distillhash: Unsupervised deep hashing by distilling data pairs, in: CVPR, 2019, pp. 2946–2955.
[40] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: CVPR, 2016, pp. 770–778.
[41] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16x16 words: Transformers for image recognition at scale, ICLR, 2021.
[42] L. Van der Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (11) (2008).
[43] T.S. Ferguson, A bayesian analysis of some nonparametric problems, Annal. Stat. (1973) 209–230.
[44] C.E. Antoniak, Mixtures of dirichlet processes with applications to bayesian nonparametric problems, Annal. Stat. (1974) 1152–1174.
[45] M. Ronen, S.E. Finder, O. Freifeld, DeepDPM: Deep clustering with an unknown number of clusters, in: CVPR, IEEE, 2022, pp. 9851–9860.
[46] M. Emmerich, Single-and multi-objective evolutionary design optimization assisted by gaussian random field metamodels, 2005 Ph.D. thesis.
[47] Y.K. Jang, G. Gu, B. Ko, I. Kang, N.I. Cho, Deep hash distillation for image retrieval, in: S. Avidan, G.J. Brostow, M. Cissé, G.M. Farinella, T. Hassner (Eds.), ECCV, volume 13674, 2022, pp. 354–371.

Rukai Wei received a B.S. degree in the School of Software Engineering, Huazhong University of Science and Technology (HUST), in 2022. He is currently a Ph.D. candidate at the Wuhan National Laboratory for Optoelectronics (HUST). His current research interests include information retrieval, deep learning, and machine learning, etc.

Yu Liu received a Ph.D. degree in computer science from Huazhong University of Science and Technology (HUST) in 2017. Currently, he is an associate professor in the Dept. of Software Engineering, HUST, China. His current research interests include machine learning, large-scale multimedia search, big data, storage, etc.

Jingkuan Song is currently a professor at the University of Electronic Science and Technology of China. He obtained his PhD degree in information technology from The University of Queensland (UQ), Australia, in 2014. His research interests include large-scale multimedia retrieval, image/video segmentation, graph learning, etc.

Yanzhao Xie received a Master's degree from the School of Software Engineering, Huazhong University of Science and Technology (HUST). He is currently a Ph.D. candidate at the Wuhan National Laboratory for Optoelectronics (HUST). His current research interests include machine learning, graph neural networks, reinforcement learning, etc.

Ke Zhou is currently a professor at Huazhong University of Science and Technology (HUST). He received the B.S., M.S., and Ph.D. degrees from HUST in 1996, 1999, and 2003, respectively. His main research interests include network/cloud storage, data security and service, and parallel I/O.