



# A hash centroid construction method with Swin transformer for multi-label image retrieval

Yanzhao Xie<sup>1</sup> · Yangtao Wang<sup>2</sup> · Rukai Wei<sup>1</sup> · Yu Liu<sup>3</sup> · Ke Zhou<sup>1</sup> · Lisheng Fan<sup>2</sup>

Received: 9 June 2022 / Accepted: 6 January 2023 / Published online: 27 January 2023

© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2023

## Abstract

Quantization-based hashing methods have become increasingly popular to adjust the global data distribution and accurately capture the data similarity compared with pairwise/triplet similarity-based methods. However, the existing image quantization hashing approaches adopt fixed hash centers, which consider neither the semantic information of each hash center nor the scale size of each object appearing in a multi-label image, resulting in that each hash code will deviate from its corresponding hash centroid. To address this issue, we propose HCCST, a hash centroid construction method with Swin transformer for multi-label image retrieval. HCCST consists of a hash code generation module, a hash centroid construction module and an interaction module between each hash code and its corresponding hash centroid. In the hash code generation module, we first adopt Swin transformer to extract the feature vector for each input multi-label image and then generate the initialized hash code of this image. In the hash centroid construction module, we first utilize the object semantic information to construct semantic hash centers and then consider the object scale size by learning the object weight coefficient to compute the hash centroid for each sample. After obtaining both the hash code and hash centroid of each sample, in the last interaction module, we constantly limit the distance between each hash code and its hash centroid to preserve the similarity between samples. Our model will be trained in an end-to-end manner to alternately update the net parameters of hash code generation module, hash centroid construction module and the object weight coefficient. We conduct extensive experiments on 3 multi-label image datasets including VOC2012, MS-COCO and NUS-WIDE. The experimental results demonstrate that HCCST can achieve better retrieval performance compared with the state-of-the-art image hashing methods. The open-source code of this project is released at: <https://github.com/lzHZWZ/HCCST.git>.

**Keywords** Multi-label image retrieval · Swin transformer · Hash center · Hash centroid

## 1 Introduction

Hashing [32] has been widely applied to large-scale image retrieval in the past few years owing to its compact binary codes and efficient XOR comparisons. With the rapid

development of deep learning [18], the existing deep hashing methods mainly belong to data-dependent approaches, which can capture the semantic similarity between samples from the original high-dimensional input space to low-dimensional Hamming space. By this means,

✉ Yangtao Wang  
ytaowang@gzhu.edu.cn

Yanzhao Xie  
yzzie@hust.edu.cn

Rukai Wei  
weirukai@hust.edu.cn

Yu Liu  
liu\_yu@hust.edu.cn

Ke Zhou  
zhke@hust.edu.cn

Lisheng Fan  
lsfan@gzhu.edu.cn

<sup>1</sup> Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China

<sup>2</sup> School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou, China

<sup>3</sup> School of Computer of Science and Technology, Huazhong University of Science and Technology, Wuhan, China

similar images will be mapped into hash codes with short Hamming distance while those dissimilar ones will own long Hamming distance in their output space. To keep the above principle, most solutions first generate the feature of each image and then design a hash function to transform image features into hash codes to preserve the semantic similarity between samples.

On the one hand, accurate image features will contribute to generating high-quality hash codes. During the past decades, convolution neural network (CNN) [12, 16] has made great success in computer vision tasks using its local conventional kernels to operate on the image pixels, and this way has been dominant to extract image features. However, the recent vision transformer (ViT) [10] reveals that CNN can only generate the local features because of the limitation of kernel size and pooling operation. Instead, ViT tries to capture the global representation of an input image with the well-designed self-attention mechanism and multi-layer perceptron structure in a natural language processing (NLP)-based way. Despite its good performance, ViT brings high computational complexity to compare the similarity between each two image patches. To address this issue, Swin transformer [26] designs an effective shifted window mechanism to complete the similarity computing and reduces the computational complexity to linear level. Nevertheless, Swin transformer has neither explored its effectiveness on extracting multi-label image features nor its efficiency on constructing hash centers. Based on this, we expect to improve Swin transformer to adapt to our multi-label image hashing task with more accurate image features.

On the other hand, after obtaining the image features, the hash function takes a great effect on measuring the similarity between different samples. Generally, the similarity measurement means can be divided into two categories: pairwise/triplet [2, 17]-based approaches and quantization [3, 37]-based approaches. The pairwise/triplet-based approaches first split a dataset into multiple batches and then compute the similarity between every pairwise/triplet pairs within each batch. However, this way will bring too high computational cost when constructing and comparing all pairwise/triplet pairs similarity over the dataset, which becomes not realistic for large-scale scenario. To reduce the computational complexity, the quantization-based approaches first construct multiple fixed hash centers according to the object classes appearing in a dataset, then compute a hash centroid for each given sample, and finally limit the distance between each hash code and its corresponding hash centroid. However, these hash centers will be randomly and uniformly distributed in Hamming space, which contain no semantic information at all. Besides, these methods, which neglect the scale size of each object, regard each object appearing in the image as

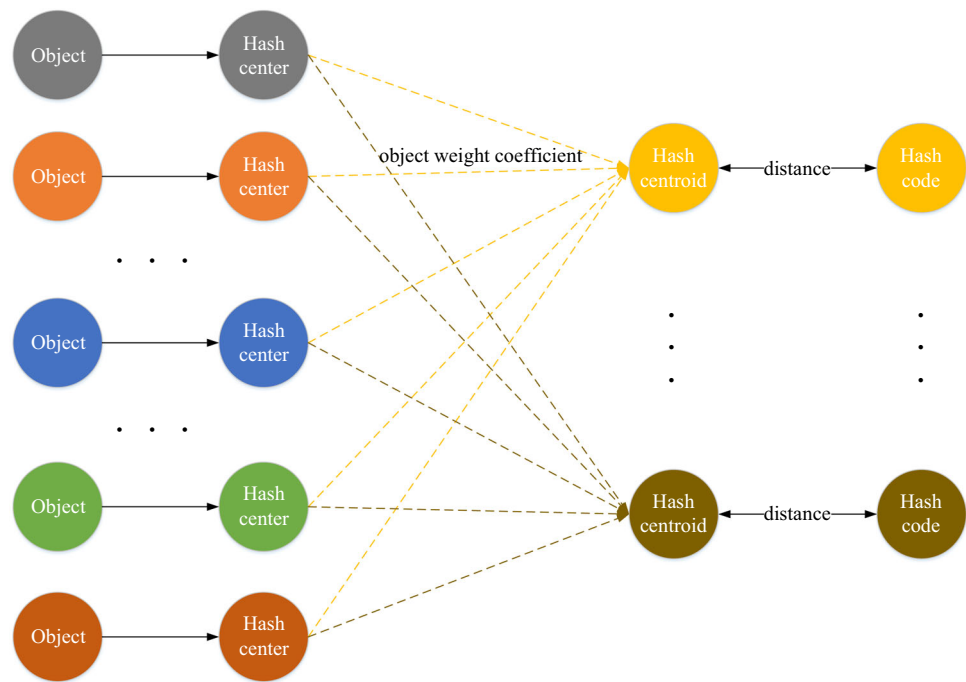
equally important to compute the hash centroid, severely leading to the inaccuracy of each hash centroid. In other words, the hash code of each sample will deviate from its real hash centroid.

To overcome the shortcomings of fixed hash centers, as shown in Fig. 1, we make full use of the object semantic information to generate semantic hash centers and compute a hash centroid for each image according to the object scale size of each object appearing in this image. Specifically, we first adopt the word2vector [8] technique to obtain the label embedding of each label (*i.e.*, object word description) and then generate semantic hash centers based on these label embeddings. To take the object scale size into consideration, we refer to European projection [33] to learn the object weight coefficient (see Sect. 3.4) of each object appearing in a given input image. Finally, we can compute a hash centroid based on the above hash centers and the corresponding object weight coefficient of this image. By this means, the hash code of each image and its corresponding hash centroid will approach each other, thus yielding high-quality and distinguishable semantic hash codes for higher image retrieval performance.

In this paper, we propose HCCST, a hash centroid construction method with Swin transformer for multi-label image retrieval. HCCST consists of a hash code generation module, a hash centroid construction module and an interaction module between each hash code and its corresponding hash centroid. In the hash code generation module, we first adopt Swin transformer to extract the feature vector for each input multi-label image and then generate the initialized hash code of this image. In the hash centroid construction module, as mentioned above, we utilize the object semantic information to construct semantic hash centers and learn the object weight coefficient to compute the hash centroid for each sample. After obtaining both the hash code and hash centroid of each sample, in the last interaction module, we constantly limit the distance between each hash code and its hash centroid to preserve the similarity between samples. Our model will be trained in an end-to-end manner to alternately update the net parameters of hash code generation module, hash centroid construction module and the object weight coefficient. We conduct extensive experiments on 3 multi-label image datasets including VOC2012 [11], MS-COCO [23] and NUS-WIDE [7]. The experimental results demonstrate that HCCST can achieve better retrieval performance compared with the state-of-the-art image hashing methods.

The remainder of this paper is organized as follows. Sect. 2 talks about recent representative multi-label image hashing methods, followed by the detailed description of our proposed HCCST in Sect. 3 and the experiments results in Sect. 4. At last, we conclude this paper in Sect. 5.

**Fig. 1** An illustration of hash center generation and hash centroid calculation. We first generate hash centers based on the label semantic information, then calculate the hash centroid for each sample according to both the above hash centers and the object weight coefficient (*i.e.*, learned parameters), and finally update the hash code and its corresponding hash centroid by constantly limiting their distance



## 2 Related works

The existing multi-label image hashing methods mainly utilize the label information to measure the similarity between different images. According to whether constructing similar sample pairs, they can be classified into two categories including pairwise/triplet-based hashing approaches and quantization-based hashing approaches.

### 2.1 Pairwise/triplet-based similarity methods

One of the most common means to measure the similarity between samples is to construct pairwise or triplet pairs that determine two images are similar if they have one or multiple common labels. Wang et al. refer to the idea of spectral hashing [34] and propose DSH [24] that utilizes a pairwise loss function and pre-defines a threshold to limit the similarity between samples. Based on DSH, Lu et al. propose PCDH [5] that extends the layers of DSH by adding a discrete hashing layer and classification layer to generate more accurate hash codes. On the one hand, deeper layers can overcome the over-fitting problem caused by DSH. On the other hand, the supervised training in the classification layer can boost the model performance. Different from the above methods that only consider the pairwise similarity, Do et al. propose SH-BDNN [9] that integrates the independence of each hash bit into its loss function, which means each bit of hash code will be mapped into 1 or -1 with 50% possibility. Although SH-BDNN takes the advantage of kernel function [25] that can

reduce the network training cost, it neither utilizes the main-stream CNN nor adopts an end-to-end model, resulting in poor performance on complex datasets. Furthermore, Li et al. add a quantization loss to the pairwise loss function and propose DPSH [19], which increases the training flexibility using a sign function to approach the binary process of hash code. Similar to DPSH, HashNet [4] adopts the Tanh activation function to complete the above binarization operation, which greatly promotes the model performance. It is worth mentioning that DCH [2] proposes to replace the exponential distribution with Cauchy distribution to effectively improve the retrieval accuracy within Hamming distance 2. Based on this Cauchy distribution, Xie et al. propose LAH [36] that refers to the multi-label image recognition approach ML-GCN [6] and integrates the label dependencies into its hash learning to generate distinguishable hash codes.

Different from the above pairwise-based methods, DNNH [17] constructs a triplet loss function by comparing the similarity between three samples in a triplet. Huang et al. consider more refined degree of sample similarity and propose HMLD [31] that declares two images are more similar if they have more common labels. More recently, Lai et al. propose DLTH [20] that involves multiple triplet pairs in each batch training to effectively complete the similarity computation. Some other works, such as MIHash [1] and HALR [13], have not directly utilized the pairwise/triplet loss function, but the key idea is to construct a group to calculate the similarity between samples within this group, which can also be categorized into pairwise/triplet-

based approaches. On the one hand, they have to calculate the similarity between every two or three samples within a mini-batch, which will bring great computational cost. On the other hand, the ratio of positive and negative samples tends to be severely unbalanced, leading to the model instability during the training process.

## 2.2 Quantization-based similarity methods

The other is the quantization-based similarity hashing methods that refer to the idea of product quantization [14] to divide a high-dimensional space into the Cartesian product of multiple low-dimensional sub-spaces. This way can effectively alleviate the high computational complexity brought by pairwise/triplet comparisons as well as avoid too large quantization loss during the binarization process. Long *et al.* propose DQN [3] which first divides the original vector space into  $M$  sub-vectors and then combines all sub-vectors to obtain the hash code for each query. The key issue of this way is to find and measure several center points that make each sample approach one or more center points. Inspired by this means, researchers begin to explore how to construct hash centers to generate hash codes.

In 2018, Ji *et al.* propose HMOH [21, 22] that indicates the Hadamard matrix is an orthogonal binary matrix, so they directly extract each column vector as a hash center corresponding to a class category. Based on HMOH, Feng *et al.* propose CSQ [37] for multi-label image and video retrieval. Specifically, CSQ first chooses multiple hash centers from the Hadamard matrix or Bernoulli distribution, then matches one or more hash centers corresponding to each image. However, CSQ neglects the object scale size within each image, which utilizes the same object weight coefficient for each object to calculate the hash centroid. In other words, CSQ treat each object within a multi-label image as equally important and allocate an equal weight to each object to construct the hash centroid. However, this way will result in that the hash code of each sample deviate from its corresponding real hash centroid owing to that each object occupies a different scale size from others within the same image. Based on CSQ, PSLDH [30] extracts the hash vector representations from one-hot vectors, which will be used as the supervised information to guide the hash network training. Similar to CSQ, PSLDH also fixes its hash centers and neither captures the label semantic information nor considers the object scale size, resulting in very limited performance improvement. Among these supervised hashing methods, Sablayrolles *et al.* [29] propose two new evaluation metrics to test the retrieval performance on unseen classes by utilizing the classification probabilities to obtain binary hash codes.

To overcome these shortcomings, our proposed HCCST integrates the label semantic information to generate hash

centers and learn the object weight coefficient to calculate the hash centroid for each sample, which helps obtain high-quality semantic hash codes by constantly limiting the distance between each hash code and its corresponding real hash centroid.

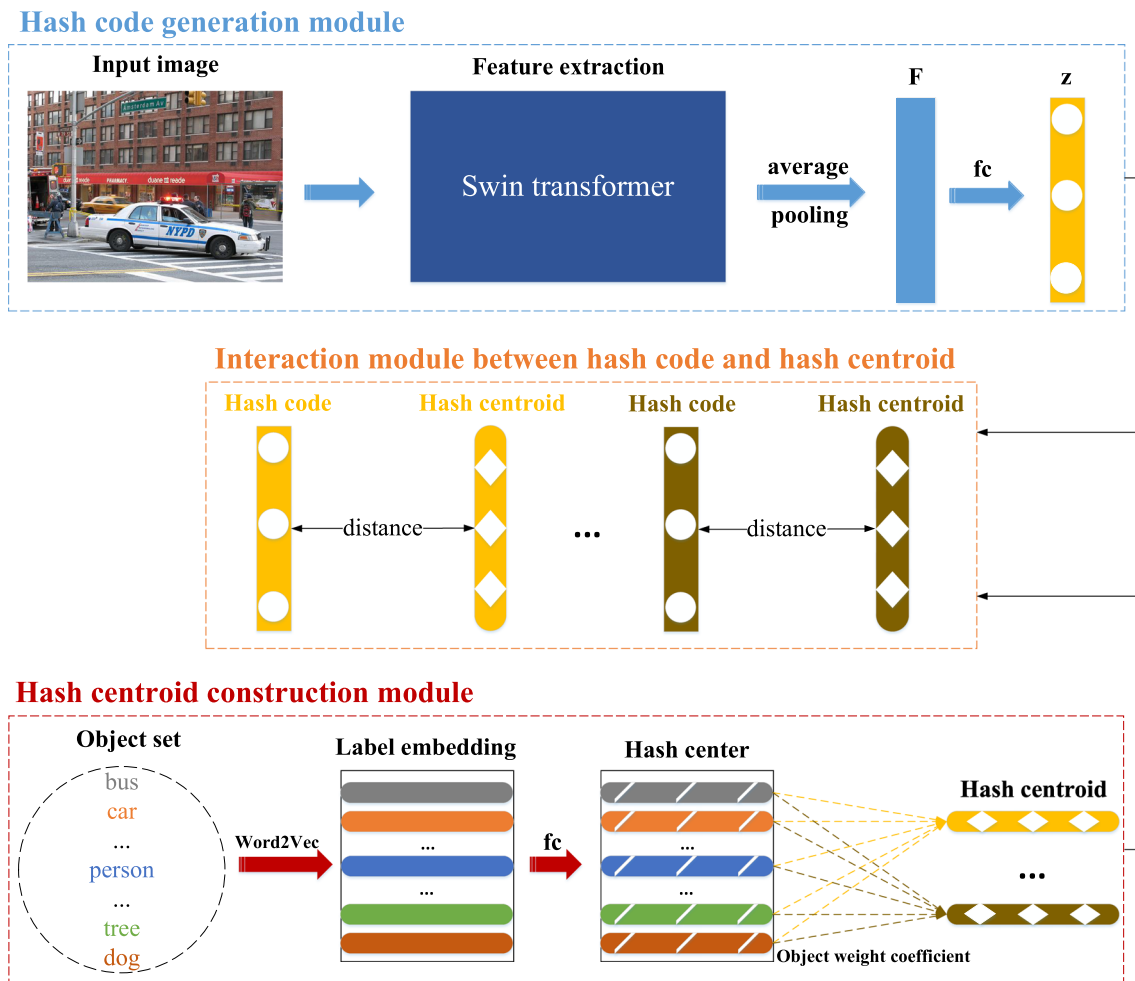
## 3 Proposed methodology

In this section, we elaborate the structure design of our proposed HCCST that consists of a hash code generation module, a hash centroid construction module and an interaction module between each hash code and its corresponding hash centroid. We first introduce the overall framework of HCCST in Fig. 2 and then describe the work-flow of each module in detail.

### 3.1 Overall framework

As shown in Fig. 2, in the hash code generation module, we adopt the Swin transformer [26] structure to extract the feature vector of each multi-label image, which will be mapped into an initialized real-valued hash code via 3 fully connected ( $fc$ ) layers. At the same time, in the hash centroid construction module, we first obtain the label embedding of each object via the Word2Vec [8] technique, then generate the hash centers via 2  $fc$  layers, and finally calculate the hash centroid for each sample according to both the above hash centers and the object weight coefficient (*i.e.*, learned parameters). Based on this, in the last interaction module between hash code and hash centroid, we begin to train the network in an end-to-end manner by constantly limiting the distance between each hash code and its corresponding hash centroid.

Formally, given a multi-label image dataset containing  $N$  samples  $\{x_1, x_2, \dots, x_N\}$ , there exist  $C$  class categories in the label set. For each  $x_i \in \{x_1, x_2, \dots, x_N\}$ , its ground truth labels  $y_i = [y_{i1}, y_{i2}, \dots, y_{iC}]$ , where  $y_{ij} \in \{0, 1\}$  for  $\forall j \in \{1, 2, \dots, C\}$ . In the hash code generation module, we input  $x_i$  to the Swin transformer structure and adopt average pooling to obtain the feature vector  $F_i$ , followed by 3  $fc$  layers to generate its initialized  $K$ -dimensional real-valued hash code  $z_i$ . In the hash centroid construction module, each label/object (word description) will be mapped into a  $D$ -dimensional label embedding via the Word2Vec [8] technique. Based on this, we can generate  $C$  hash centers that contain the label semantic information by transforming the  $D \times C$ -dimensional label embedding matrix into a  $K \times C$ -dimensional hash center matrix  $E = \{e_1, e_2, \dots, e_C\}$ . Next, we aim to calculate the hash centroid  $r_i$  for each sample  $x_i$ . Note that given  $x_i$ , we initialize and record the object weight coefficient  $w_i =$



**Fig. 2** The overall framework of HCCST consists of a hash code generation module, a hash centroid construction module and an interaction module between each hash code and its corresponding hash centroid

$[w_{i1}, w_{i2}, \dots, w_{iC}]$  according to those objects appearing in  $x_i$ , and  $w_i$  will be automatically updated during the network back propagation to indicate the scale size of each object. As a result, each hash centroid  $r_i$  corresponds to the weighted sum of all hash centers and the object weight coefficient  $w_i$ :

$$r_i = \sum_{j=1}^C w_{ij} e_j. \quad (1)$$

At last, in the interaction module between hash code and hash centroid, for each sample  $x_i$ , we will constantly limit the distance between its hash code  $z_i$  and hash centroid  $r_i$ . In this way, our model will be trained in an end-to-end manner after limited iterations, and the final binary hash code  $h_i$  of  $x_i$  will be obtained by conducting the binarization operation on  $z_i$ . The detailed work-flow of the above modules will be described as follows.

### 3.2 Hash code generation module

As shown in the blue frame of Fig. 2, we adopt the Swin transformer [26] structure to extract the image feature and generate the initialized hash code for sample  $x_i$ . Theoretically, other backbones such as ResNet [12] can also complete this process. The reason why we choose Swin transformer is that this structure can effectively extract accurate image features in an NLP-based way, which will greatly boost the model performance combined with hash centers generated from label embeddings. Tables 2 and 3 also verify the effectiveness of this backbone.

Specifically, for each sample  $x_i$  with resolution  $224 \times 224$ , we conduct the average pooling operation on the last block of Swin-T [26] to generate the feature vector  $F_i$ , followed by a hash layer consisting of 3  $fc$  layers and a  $Tanh$  activation function to generate the  $K$ -dimensional initialized hash code  $z_i \in (-1, 1)^K$ :



$$F_i = \text{Avg}_{\text{pooling}}(\mathcal{F}_{\text{Swin}}(x_i); \theta_1), z_i = \text{Tanh}(fc(F_i); \theta_2), \quad (2)$$

where  $\text{Avg}_{\text{pooling}}$  denotes the average pooling operation,  $\mathcal{F}_{\text{Swin}}$  denotes the Swin transformer structure,  $\theta_1$  denotes the network parameters of this structure,  $\theta_2$  denotes the parameters of  $fc$  layers. Note that we will update the model parameters by limiting the distance between  $z_i$  and its corresponding hash centroid  $r_i$  during the training phase. After our model converges, we will remove other modules and only preserve the hash code generation module to obtain the final binary hash code  $h_i \in \{-1, 1\}^K$  for each sample  $x_i$  during the inference phase via performing the  $\text{sign}$  function on  $z_i$ :

$$h_i = \text{sign}(z_i). \quad (3)$$

### 3.3 Hash centroid construction module

Previous works [37] adopt the fixed hash centers that neither contain any semantic information nor consider the object scale size of each sample, resulting in the inaccuracy during the hash centroid calculation. To overcome these shortcomings, we utilize the Word2Vec technique and consider the object weight coefficient to calculate the hash centroid for each sample.

As shown in the red frame of Fig. 2, there exist  $C$  objects (word description such as “bus”, “car” and *etc.*) in the label set. We adopt the BERT [8] model to map each object into a  $D$ -dimensional label embedding  $d \in \{d_1, d_2, \dots, d_C\}$ , where  $D = 768$  denotes the default output dimension of a word in BERT. In this way, each label embedding contains the semantic information of one object, which will help generate semantic hash centers. To keep the same dimension between each hash center and hash code, we further use 2  $fc$  layers and a  $\text{Tanh}$  activation function to transform each label embedding  $d_j$  into a  $K$ -dimensional hash center  $e_j$  as follows:

$$e_j = \text{Tanh}(fc(d_j); \theta_3), \quad (4)$$

where  $j \in \{1, 2, \dots, C\}$  and  $\theta_3$  denotes the network parameters of the above  $fc$  layers. By this means, we can first construct  $C$  hash centers and then train the network by limiting the distance between each hash code and these hash centers instead of comparing the pairwise/triplet similarity between samples. However, on the one hand, each multi-label image  $x_i$  only contains one or multiple objects, which means not all the  $C$  objects appear in  $x_i$ . As a result, each sample only corresponds to several related hash centers.

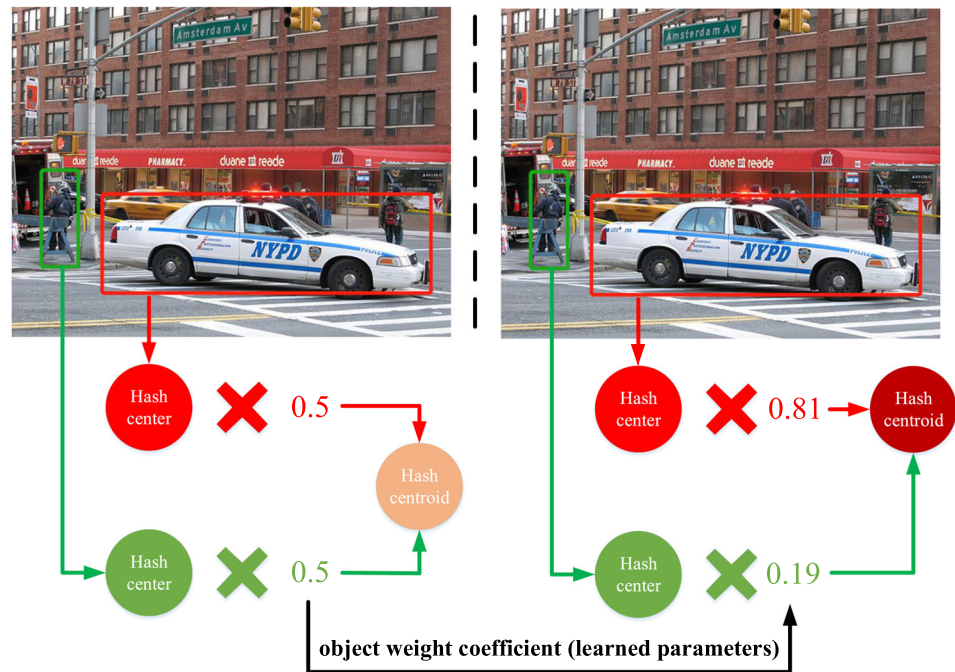
Given the ground truth labels  $y_i = [y_{i1}, y_{i2}, \dots, y_{iC}]$  of sample  $x_i$ , if the  $j$ -th object appears in  $x_i$ ,  $y_{ij} = 1$ , and otherwise  $y_{ij} = 0$ . In the hash centroid construction,

referring to CSQ [37], we record the object weight coefficient  $w_i = [w_{i1}, w_{i2}, \dots, w_{iC}]$  that satisfies  $\sum_{j=1}^C w_{ij} = 1$  for  $x_i$ , but CSQ neglects the object scale size and regards each object appearing in  $x_i$  as equally important and allocate the same weight to these objects. Take Fig. 3 for example,  $x_i$  only contains the 2-th and 5-th objects (*i.e.*, “car” and “person”), which means only  $y_{i2} = y_{i5} = 1$ . Based on the generated hash centers, CSQ will directly set  $w_{i2} = w_{i5} = 0.5$  to calculate the weighted sum as the hash centroid of  $x_i$ . However, as we see, the scale size of each object is different from others in the same image. “Car” occupies larger scale than “person”, so the hash centroid of this image will be closer to the hash center of “car” than “person”, resulting in that each hash code will deviate from its corresponding real hash centroid. To address this issue, we treat the object weight coefficient  $w_i$  as learned parameters and refer to European projection [33] to optimize these parameters, making that the final object weight coefficient indicate the real scale size of each object shown in Fig. 3. As we see, the learned object weight coefficient forces each hash centroid to approach more important hash center(s) of each sample, which will contribute to high-quality hash codes in the following interaction module.

Specifically, for each sample  $x_i$ , we denote  $|y_i|$  as the number of nonzero elements in  $y_i$ . Correspondingly, each nonzero elements in  $w_i$  will be initialized as  $\frac{1}{|y_i|}$ . For example, only  $w_{i2}$  and  $w_{i5}$  in Fig. 3 will be initialized as  $\frac{1}{2}$ , other elements in  $w_i$  will be set as 0. Since the weighted sum in  $w_i$  equals 1, *i.e.*,  $\sum_{j=1}^C w_{ij} = 1$ , European projection can effectively optimize this vector whose sum of elements equals 1. The detailed update process of  $w_i$  will be introduced in Algorithm 1 of Sect. 3.4. Overall, on the one hand, this module generates  $C$  semantic hash centers based on the object semantic information. On the other hand, each hash centroid  $r_i$  corresponding to sample  $x_i$  will be calculated according to the weighted sum of both its object weight coefficient  $w_i$  and hash centers. The next module presents the interaction process between each hash code and its hash centroid to update the network parameters including  $\theta_1$ ,  $\theta_2$ ,  $\theta_3$ , and all  $w_i$  in an end-to-end manner.

### 3.4 Interaction module between hash code and hash centroid

After generating the initialized hash code  $z_i$  and hash centroid  $r_i$  of each sample  $x_i$ , in this section, we begin to train our HCCST by constantly limiting the distance between  $z_i$  and  $r_i$  shown in the orange frame of Fig. 2. The optimization process can be divided into the following 3 parts: hash code optimization (*i.e.*,  $\theta_1$  and  $\theta_2$ ), hash center optimization (*i.e.*,  $\theta_3$ ) and hash centroid optimization (*i.e.*, object weight coefficient  $w_i$ ). We first introduce the

**Fig. 3** Object weight coefficient

optimization criteria and then talk about the training workflow of HCCST. These training processes will continue to alternate until the network converges.

### 3.4.1 Hash code optimization

In this part, we optimize the hash code  $z_i$  based on its current hash centroid  $r_i$ . Referring to Cauchy distribution (see reference [2] for details), we design a *center loss* to limit the distance between  $z_i$  and  $r_i$ :

$$d_i = \frac{K}{2} (1 - \cos(z_i, r_i)),$$

$$L_{\text{center}} = -\frac{1}{N} \sum_{i=1}^N \log\left(\frac{\gamma}{\gamma + d_i}\right), \quad (5)$$

where  $K$  denotes the length of each hash code,  $d_i$  denotes the distance between  $z_i$  and  $r_i$ ,  $\gamma = 0.15$  denotes the scale parameter of Cauchy distribution. In this way,  $L_{\text{center}}$  will penalize significantly on the sample whose hash code owns long distance from its target hash centroid. Besides, owing to that  $z_i \in (-1, 1)^K$  is a real-valued vector (see Equation (2)), to speed up the network training, we adopt a *quantization loss* to make each floating point number in  $z_i$  to get close to 1 or -1:

$$L_q = \frac{1}{N} \sum_{i=1}^N \|h_i - z_i\|_2^2, \quad (6)$$

where  $h_i$  denotes the binary hash code of  $z_i$ . As a result, the whole loss in the hash code generation module can be formulated as  $L_1$ :

$$L_1 = L_{\text{center}} + L_q, \quad (7)$$

where  $\theta_1$  and  $\theta_2$  will be updated via gradient descent  $\frac{\partial L_1}{\partial \theta_1}$  and  $\frac{\partial L_1}{\partial \theta_2}$ , respectively.

### 3.4.2 Hash center optimization

In this part, we will update the  $C$  hash centers  $\{e_1, e_2, \dots, e_C\}$  based on the given hash code  $z_i$  and object weight coefficient  $w_i$ . As shown in Fig. 2, we obtain each label embedding via the Word2Vec technique, which contains the semantic information of each object. In order to integrate the above semantic information into hash centers, we design a *KL-divergence loss* to make the distribution of hash centers (i.e.,  $q_{ij}$ ) and label embeddings (i.e.,  $p_{ij}$ ) get close to each other:

$$p_{ij} = \frac{1}{2} (\cos(d_i, d_j) + 1),$$

$$q_{ij} = \frac{1}{2} (\cos(e_i, e_j) + 1), \quad (8)$$

$$L_{kl} = \sum_{i=1}^C \sum_{j=1}^C p_{ij} \log \frac{p_{ij}}{q_{ij}},$$

where  $d_i$  and  $e_i$ , respectively, denote the  $i$ -th label embedding and  $i$ -th hash center. In addition, on the one hand, to ensure the great separability between different hash centers that correspond to different class categories, we add a *class loss* to maximize the distance between these hash centers:

$$L_{\text{class}} = - \sum_{i=1}^C \sum_{j=1}^C \|e_i - e_j\|_2^2. \quad (9)$$

On the other hand, each hash code should get close to its corresponding hash centroid, and this goal will be completed by  $L_{\text{center}}$  (see Equation (5)). As a result, the whole loss in the hash centroid construction module can be formulated as  $L_2$ :

$$L_2 = L_{\text{center}} + L_{kl} + L_{\text{class}}, \quad (10)$$

where  $\theta_3$  will be updated via gradient descent  $\frac{\partial L_2}{\partial \theta_3}$ .

Next, we refer to Euclidean projection [33] to project  $\hat{w}_i^t$  onto a probability simplex to output the final value of  $w_i^t$  as follows:

$$w_i^t = \min_{w_i^t} \frac{1}{2} \|w_i^t - \hat{w}_i^t\|^2, \quad (12)$$

$$s.t., \sum_{j=1}^C w_{ij}^t = 1, w_{ij}^t \geq 0,$$

where  $w_{ij}^t$  denotes the  $j$ -th element in  $w_i^t$  for  $\forall j \in \{1, 2, \dots, C\}$ . We present the computation process of

---

**Algorithm 1:** Euclidean Projection of  $\hat{w}_i^t$  onto  $w_i^t$ .

---

**Input:**  $\hat{w}_i^t \in R^C$ .

**Computation:**

1. Sort  $\hat{w}_i^t = [\hat{w}_{i1}^t, \hat{w}_{i2}^t, \dots, \hat{w}_{iC}^t]$  into  $u = [u_1, u_2, \dots, u_C]: u_1 \geq u_2 \geq \dots \geq u_C$ ;
2. Find  $p = \max\{j \in \{1, 2, \dots, C\}: u_j + \frac{1}{j}(1 - \sum_{i=1}^j u_i) > 0\}$ ;
3. Define  $q = \frac{1}{p}(1 - \sum_{i=1}^p u_i)$ .

**Output:**  $w_i^t \in R^C$ ,  $s.t.$ ,  $w_{ij}^t = \max\{\hat{w}_{ij}^t + q, 0\}, j \in \{1, 2, \dots, C\}$ .

---

### 3.4.3 Hash centroid optimization

In this part, we will update the hash centroid (*i.e.*, the object weight coefficient  $w_i$ ) corresponding to each sample  $x_i$  based on the given hash code  $z_i$  and hash centers

$w_i^t$  in Algorithm 1 and this optimization has been proved in reference [33]. You may check reference [33] for more details if interested.

---

**Algorithm 2:** Training work-flow of HCCST.

---

**Input:**  $\{x_1, x_2, \dots, x_N\}$ .

**Computation:**

1. Initialize the parameters of  $\theta_1, \theta_2, \theta_3$  and  $w_i$ ;
2. Update  $\theta_1, \theta_2$  and  $w_i$  with fixed  $\theta_3$ ;
3. Update  $\theta_3$  with fixed  $\theta_1, \theta_2$  and  $w_i$ ;
4. Alternately conduct step 2 and step 3 until the model converges.

**Output:**  $\theta_1, \theta_2, \theta_3$  and  $w_i$ .

---

$\{e_1, e_2, \dots, e_C\}$ . As mentioned in Sect. 3.3,  $w_i$  records the scale size of each object appearing in  $x_i$  and the weighted sum in  $w_i$  equals 1, *i.e.*,  $\sum_{j=1}^C w_{ij} = 1$ . We have to ensure this principle in each iteration. The parameter optimization process of  $w_i$  is described as follows.

First,  $w_i$  will be updated by the gradient vector of its loss function  $L_1$  (see Equation (7)) with respect to  $w_i$ :

$$\nabla w_i^t = \frac{\partial L_1}{\partial w_i^{t-1}}, \quad (11)$$

$$\hat{w}_i^t = w_i^{t-1} - \nabla w_i^t,$$

where  $t$  denotes the  $t$ -th iteration during the model training.

### 3.4.4 Training work-flow

In this part, we design an alternating learning strategy to train our HCCST and update the network parameters (*i.e.*,  $\theta_1, \theta_2, \theta_3$  and  $w_i$ ) based on the above optimization criteria. Specifically, the training work-flow can be divided into the following 4 steps. (i) We randomly initialize the parameters of  $\theta_1, \theta_2$  and  $\theta_3$  and set each nonzero element of  $w_i$  as  $\frac{1}{|y_i|}$ . (ii) Given  $\theta_1, \theta_2, \theta_3$  and  $w_i$ , we first fix  $\theta_3$ , and then calculate  $L_1$  (see Equation (7)) to update  $\theta_1, \theta_2$ , and  $w_i$  via gradient descent or Euclidean projection for each batch of data. This process will be repeatedly conducted with multiple batches until all the  $N$  samples have been



processed to update our model. (iii) We first fix the above  $\theta_1$ ,  $\theta_2$ , and  $w_i$ , and then calculate  $L_2$  (see Equation (10)) to update  $\theta_3$  via gradient descent. Similarly, this process will also be repeatedly conducted to cover all the  $N$  samples. (iv) Step (ii) and step (iii) will continue to alternate until our model converges or reaches the pre-defined max epochs. We also summarize the training work-flow in Algorithm 2. Note that after completing this training process, we first directly remove both the hash centroid construction module and the interaction module and then only keep the hash code generation module to obtain the binary hash code of each sample (see Equation (4)) in the inference phase.

## 4 Experiments

In this section, we first introduce the experimental settings including datasets, implementation details and evaluation metrics, then present the experimental results compared with the state-of-the-art image hashing methods, and finally conduct ablation studies to analyze the effect of key components.

### 4.1 Experimental settings

#### 4.1.1 Datasets

We use 3 multi-label image datasets including VOC2012 [11], MS-COCO [23] and NUS-WIDE [7] to test the performance of HCCST. For fair comparisons, we adopt the same dataset partition as the mainstream methods like CSQ. Specifically, VOC2012 is randomly divided into training set, test set and retrieval set, which, respectively, contain 4000, 1000, and 6540 images. Similarly, we, respectively, randomly allocate 10,000, 5000, and 114,217 images to the training set, test set and retrieval set of MS-COCO. Note that we utilize all the 20 and 80 categories of VOC2012 and MS-COCO. As for NUS-WIDE, we only use its 21 most frequently categories and randomly allocate 10,500, 2100, and 149,685 images to its training set, test set and retrieval set. For each dataset, we first utilize its training set to train our model and then evaluate the performance on its test set and retrieval set.

#### 4.1.2 Implementation details

All the experiments are conducted with PyTorch on 4 NVIDIA Tesla A100 GPUs. Our model is easy to deploy, which consists of the hash code generation module and the hash centroid construction module, where the former contains the Swin-T [26] backbone followed by 3  $fc$  hash layers to generate the initialized hash code and the latter

contains 2  $fc$  layers to construct the initialized hash centers and hash centroid. We adopt the Adam optimizer with a batchsize of 64 and, respectively, set a learning rate of  $2e-6$  to fine-tune the backbone,  $5e-4$  to learn the hash layers, and  $5e-4$  to construct the hash centers. We choose  $K \in \{16, 32, 48, 64\}$  (*i.e.*, hash code length) to list the experimental results. More details about our open-source project are released at: <https://github.com/lzHZWZ/HCCST.git>.

#### 4.1.3 Evaluation metrics

We compare the performance of HCCSH with the state-of-the-art image hashing methods including CNNH [35], DNNH [17], DHN [39], HashNet [4], IDHN [38], DCH [2], CSQ [37] and PSLDH [30]. We adopt the commonly used image hashing evaluation metrics including mean average precision (mAP), precision–recall curves (PR), precision curves within top- $N$  returned images ( $P@N$ ) and precision curves within Hamming distance 2 ( $P@H \leq 2$ ). For fair comparisons, we, respectively, adopt mAP@all for VOC2012, mAP@5000 for MS-COCO and NUS-WIDE.

### 4.2 Experimental results

In this section, we evaluate the performance of HCCST from the following 6 aspects: we (i) compare the mAP results of HCCST with the state-of-the-art image hashing methods on VOC2012, MS-COCO and NUS-WIDE at all hash code lengths (*i.e.*, 16 bits, 32 bits, 48 bits and 64 bits), (ii) compare the mAP results with different backbones on MS-COCO and NUS-WIDE at all hash code lengths, (iii) compare the PR,  $P@N$  and  $P@H$  results on VOC2012, MS-COCO and NUS-WIDE, (iv) calculate the mean Hamming distance between hash centers on VOC2012, MS-COCO and NUS-WIDE at 64 bits, (v) evaluate the generalization ability of HCCST on VOC2012 and MS-COCO at all hash code lengths, and (vi) draw the loss curves to reveal the optimization process on VOC2012, MS-COCO and NUS-WIDE at 64 bits.

#### 4.2.1 mAP comparisons with the state-of-the-art methods

Table 1 lists the mAP results of HCCST compared with other baselines. As we see, HCCST yields the highest performance on all cases. Especially, HCCST averagely achieves higher mAP than others by at least 13.3%, 7.7%, 1.9% on VOC2012, MS-COCO and NUS-WIDE, respectively. Besides, with the hash code length increasing, the performance of HCCST also gradually increases to reach its peak values at 64 bits. This reflects that HCCST owns a good semantic representation ability to measure the similarity between samples at both short and long hash code lengths. Note that compared with those candidates like

CSQ and PSLDH that adopt fixed hash centers without semantic information, as expected, our HCCST can construct semantic hash centers to automatically make each hash code adapt to its corresponding hash centroid, leading to higher mAP values on all these 3 datasets. As a whole, the superiority of HCCST mainly results from 2 aspects. (i) We adopt Swin-T to extract more accurate image features to help generate high-quality semantic hash codes; (ii) based on the semantic hash centers, the object weight coefficient brings a real hash centroid for each sample, which make each hash code approach the most related one or multiple hash centers, thus yielding distinguishable hash codes.

#### 4.2.2 mAP comparisons with different backbones

In this part, we adopt different backbones to compare the performance of HCCST with others to exclude the dependence of our approach on a certain feature extraction network. As shown in Tables 2 and 3, we adopt both Swin-T and the commonly used ResNet-101 [12] as backbones to evaluate the mAP results on MS-COCO and NUS-WIDE. As we see, HCCST again achieves the highest mAP at all hash code lengths when we change the backbone. To be sure, a more powerful feature extraction network tends to bring higher performance like the better results with Swin-T than ResNet-101. However, even if we utilize ResNet-101 to train the model, compared with other baselines with the same backbone, HCCST averagely obtains higher mAP by at least 2.9% and 1.1% on MS-COCO and NUS-WIDE, respectively. These results indicate our proposed HCCST benefits more from the semantic hash centers and hash centroid construction scheme. In other words, it is the combination of hash centers and object weight coefficient that greatly promotes the model performance. We also

further verify the effectiveness of object weight coefficient in Sect. 4.3.

#### 4.2.3 PR, P@N and P@H comparisons with the state-of-the-art methods

Figures 4 and 5, respectively, present the PR and P@N curves on VOC2012, MS-COCO and NUS-WIDE at 64 bits. As shown in Fig. 4, at a same recall rate, the precision of our HCCST outperforms others on all these 3 datasets. Besides, we calculate the precision within top-1000 to top-6000 on VOC2012, top-1000 to top-5000 on both MS-COCO and NUS-WIDE. From Fig. 5, HCCST not only achieves the highest precision but also presents more stable and smooth curve than other candidates. As for the P@H curves within Hamming distance 2, the results in Fig. 6 well reflect the polymerization degree of hash codes in a Hamming ball with radius 2. It can be found that HCCST obtains the highest precision on all these 3 datasets at all hash code lengths, which implies that our designed *center loss* (see Eq. 5)) can not only penalize significantly on the sample whose hash code owns long distance from its target hash centroid, but also push those similar samples to be close to the same hash center(s). On the whole, all the experimental results in Figs. 4, 5 and Fig. 6 verify the superiority of our scheme that is able to adaptively adjust the distance between each hash code and its corresponding hash centroid to generate high-quality semantic hash codes.

#### 4.2.4 Hamming distance between hash centers

High-quality hash codes rely on distinguishable semantic hash centers. As shown in Fig. 7, we refer to DCH [2] to calculate the mean Hamming distance between hash centers on VOC2012, MS-COCO and NUS-WIDE at 64 bits during the training phase. With the increase of epoch, the

**Table 1** mAP comparisons on VOC2012, MS-COCO and NUS-WIDE

Method	VOC2012				MS-COCO				NUS-WIDE			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
CNNH	0.618	0.629	0.631	0.642	0.599	0.617	0.620	0.636	0.635	0.642	0.659	0.647
DNNH	0.621	0.637	0.645	0.653	0.644	0.651	0.655	0.657	0.703	0.738	0.742	0.754
DHN	0.698	0.713	0.715	0.721	0.719	0.731	0.740	0.745	0.712	0.759	0.764	0.771
HashNet	0.732	0.739	0.741	0.744	0.745	0.773	0.788	0.791	0.757	0.775	0.781	0.790
IDHN	0.702	0.739	0.743	0.745	0.769	0.795	0.796	0.801	0.765	0.784	0.795	0.802
DCH	0.739	0.740	0.748	0.749	0.755	0.801	0.811	0.825	0.762	0.795	0.799	0.808
CSQ	0.746	0.755	0.762	0.767	0.778	0.828	0.836	0.841	0.801	0.818	0.831	0.835
PSLDH	0.751	0.761	0.769	0.777	0.782	0.835	0.850	0.854	0.809	0.823	0.839	0.847
<b>HCCST</b>	<b>0.874</b>	<b>0.898</b>	<b>0.905</b>	<b>0.913</b>	<b>0.860</b>	<b>0.899</b>	<b>0.935</b>	<b>0.933</b>	<b>0.815</b>	<b>0.850</b>	<b>0.860</b>	<b>0.866</b>

Bold represents the optimal result

**Table 2** mAP comparisons on MS-COCO with different backbones

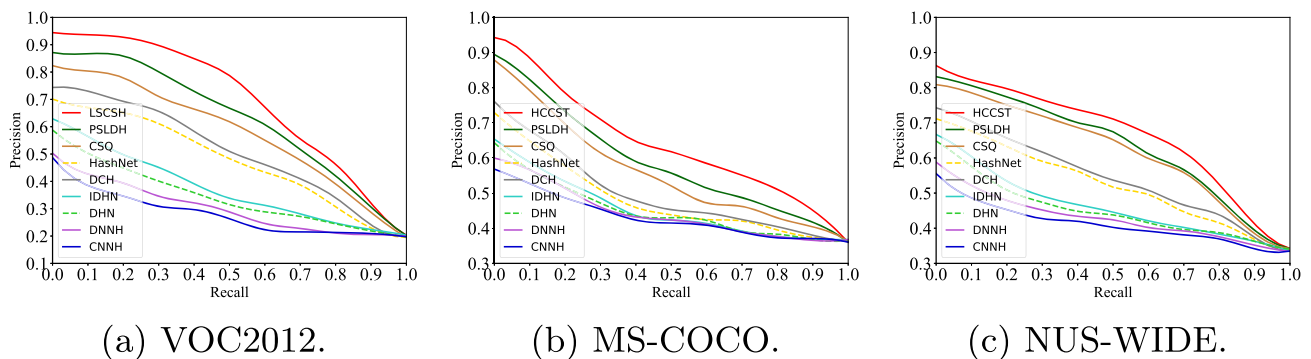
Method	Swin-T				ResNet-101			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
CNNH	0.625	0.637	0.648	0.665	0.599	0.617	0.620	0.636
DNNH	0.692	0.672	0.689	0.698	0.644	0.651	0.655	0.657
DHN	0.747	0.773	0.784	0.797	0.719	0.731	0.740	0.745
HashNet	0.783	0.791	0.805	0.819	0.745	0.773	0.788	0.791
IDHN	0.742	0.787	0.805	0.828	0.769	0.795	0.796	0.801
DCH	0.798	0.817	0.835	0.848	0.755	0.801	0.811	0.825
CSQ	0.831	0.851	0.859	0.879	0.778	0.828	0.836	0.841
PSLDH	0.854	0.873	0.881	0.896	0.782	0.835	0.850	0.854
<b>HCCST</b>	<b>0.860</b>	<b>0.899</b>	<b>0.935</b>	<b>0.933</b>	<b>0.821</b>	<b>0.855</b>	<b>0.878</b>	<b>0.882</b>

Bold represents the optimal result

**Table 3** mAP comparisons on NUS-WIDE with different backbones

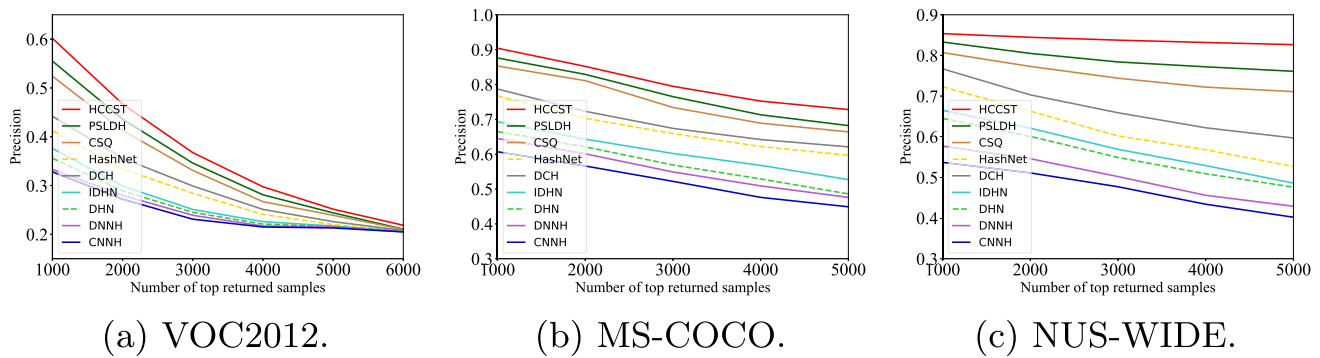
Method	Swin-T				ResNet-101			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
CNNH	0.655	0.667	0.668	0.659	0.635	0.642	0.659	0.647
DNNH	0.712	0.742	0.759	0.768	0.703	0.738	0.742	0.754
DHN	0.737	0.763	0.778	0.788	0.712	0.759	0.764	0.770
HashNet	0.773	0.781	0.799	0.809	0.757	0.773	0.780	0.790
IDHN	0.772	0.792	0.815	0.818	0.765	0.784	0.795	0.802
DCH	0.788	0.817	0.835	0.848	0.762	0.795	0.799	0.808
CSQ	0.801	0.831	0.839	0.845	0.801	0.818	0.831	0.835
PSLDH	0.811	0.843	0.850	0.857	0.809	0.823	0.839	0.847
<b>HCCST</b>	<b>0.815</b>	<b>0.850</b>	<b>0.860</b>	<b>0.866</b>	<b>0.813</b>	<b>0.839</b>	<b>0.850</b>	<b>0.859</b>

Bold represents the optimal result

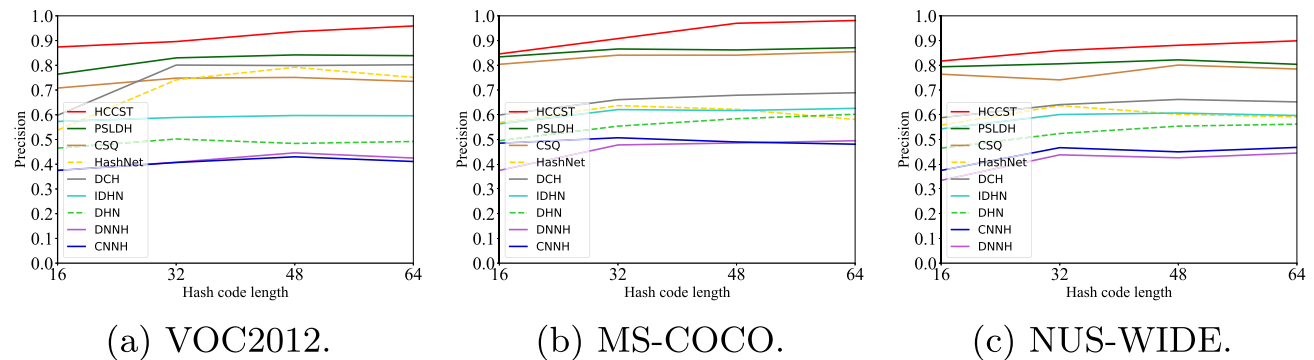
**Fig. 4** The PR curves on VOC2012, MS-COCO and NUS-WIDE at 64 bits

semantic hash centers will be gradually separated from others and the mean Hamming distance will get close to half of the current hash code length. For example, when we test HCCST on VOC2012 at 64 bits, we find the mean Hamming distance between the 20 hash centers of this dataset will reach around 32 within 30 epochs. In addition,

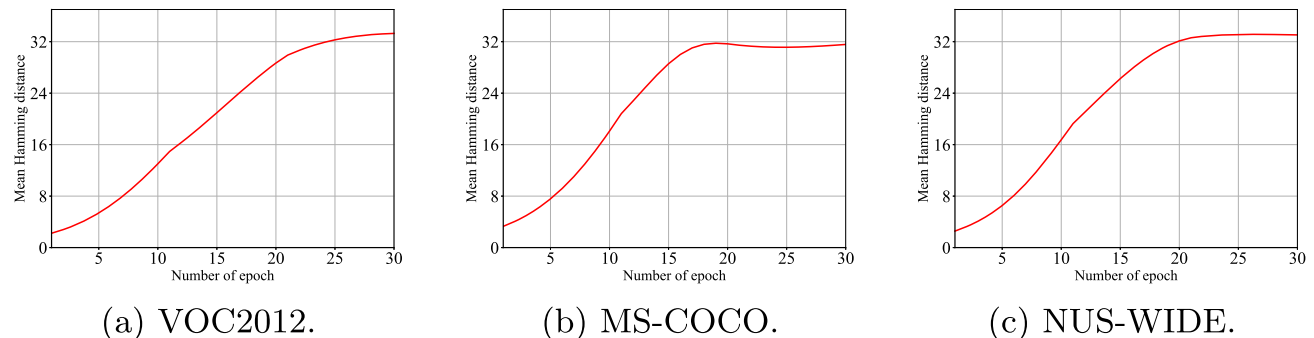
the results on MS-COCO and NUS-WIDE also present the same trend. This phenomenon reflects our approach can automatically construct distinguishable semantic hash centers that help to obtain the real hash centroid to bring high-quality hash codes.



**Fig. 5** The P@N curves on VOC2012, MS-COCO and NUS-WIDE at 64 bits



**Fig. 6** The  $P@H \leq 2$  curves on VOC2012, MS-COCO and NUS-WIDE



**Fig. 7** Hamming distance between hash centers on VOC2012, MS-COCO and NUS-WIDE at 64 bits

#### 4.2.5 Generalization ability

Most supervised hashing methods usually ignore testing the retrieval performance on unseen classes [29], resulting in a lack of evaluation about the model generalization ability. Hence, we train our model on one dataset and test its retrieval performance on another different dataset including multiple classes that have never appeared during the model training process. As mentioned before, VOC2012 and MS-COCO, respectively, own 20 and 80 classes, where the label set of MS-COCO is a superset of the label set of VOC2012. Based on this, on the one hand, we first utilize the train set of MS-COCO to train our model and then test

the retrieval performance of this model on the retrieval set as well as test set of VOC2012. In this way, we can explore the generalization ability of HCCST under different data distributions (*i.e.*, different datasets). On the other hand, we exchange the roles of VOC2012 and MS-COCO and carry out the same experiment, to verify the out-of-domain retrieval performance (*i.e.*, the retrieval performance of a method on classes that have never appeared during the training process) of our HCCST. The experimental results listed in Table 4 record the mAP values of our proposed HCCST compared with the recent state-of-the-art methods including CSQ and PSLDH. It can be seen that HCCST achieves the best performance under the above cross-

**Table 4** Generalization ability comparisons

Method	MS-COCO → VOC2012				VOC2012 → MS-COCO			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
CSQ	0.579	0.649	0.691	0.719	0.674	0.686	0.701	0.705
PSLDH	0.603	0.668	0.704	0.721	0.682	0.698	0.713	0.722
<b>HCCST</b>	<b>0.645</b>	<b>0.689</b>	<b>0.725</b>	<b>0.752</b>	<b>0.689</b>	<b>0.733</b>	<b>0.735</b>	<b>0.739</b>

Bold represents the optimal result

validation settings and averagely improves the mAP results by at least 1.0%, 3.1%, 2.9% and 2.3% at 16 bits, 32 bits, 48 bits and 64 bits. These results manifest our HCCST with semantic hash centers can recognize those unseen samples and own good generalization ability compared with other state-of-the-art methods.

Furthermore, we follow the new evaluation protocol of supervised hashing task and compare the retrieval performance of our HCCST and **Classifier+LSH (CL-LSH)** [29]. Specifically, we randomly select 20 classes from MS-COCO as the query/retrieval set and remove those samples whose ground-truth labels include any one of these selected labels. Then, the remaining samples are used as the training set. Obviously, these two sets have no common labels. For fair comparison, the LSH index is constructed based on the output of the last activation layer in Swin transformer (*i.e.*, the backbone of our feature extraction network) under its classification task. Then, we refer to the source code of CL-LSH [29] and construct a LSH-based index for these classification outputs. We evaluate the mAP@5000 results of our HCCST with CL-LSH at 16 bits, 32 bits, 48 bits and 64 bits in Table 5. The results show that HCCST achieves a better performance than CL-LSH and averagely improves the mAP@5000 results by 11.9%, 11.8%, 12.7% and 12.1% at different lengths. The main reason is that multi-label images contain more complex semantic information than single-label images, but CL-LSH nearly completely relies on the classification ability of its adopted classification model, which may neglect the label dependencies between different objects. Our HCCST focuses on multi-label images by generating each semantic hash center based on the word embedding of each object. In addition, it considers the object size scale to yield a more accurate hash centroid for each sample based on the above hash centers. As a result, HCCST can greatly distinguish similar

or dissimilar samples in the Hamming space to bring higher retrieval performance.

#### 4.2.6 Loss curves

As mentioned in Algorithm 2, we design an alternating learning strategy to train our model, where  $L_1$  (see Equation (7)) and  $L_2$  (see Equation (10)), respectively, alternately update their own network branch when fixing another branch. In this part, to reveal the optimization process, we record the change of loss curves of  $L_1$  and  $L_2$  on VOC2012, MS-COCO and NUS-WIDE at 64 bits with the training epoch increasing. As shown in Fig. 8,  $L_1$  will gradually decrease and converge within around 55 epochs on all these 3 datasets, which reflects that our model is efficiently optimizing the hash code and object weight coefficient of each sample during this period. At the same time, we also find that  $L_2$  will reach a certain stable value on these 3 datasets as the gradient decreases in Fig. 9. Note that  $L_2$  will converge earlier within 40 epochs than  $L_1$ . This phenomenon reveals during the training process, our model goes through 2 stages. At the first stage (*i.e.*, 0–40 epochs), hash codes, hash centers and object weight coefficient will be alternately updated, and hash centers will tend to be stable after completing this stage. At the second stage (*i.e.*, 41–55 epochs), our model will continue to optimize the hash code and object weight coefficient with the nearly fixed hash centers. At last, our model will converge to output the final semantic hash code for each given sample. On the whole, this optimization process seems interesting and reasonable, which records and reflects how this alternating learning strategy updates our model.

#### 4.3 Ablation studies

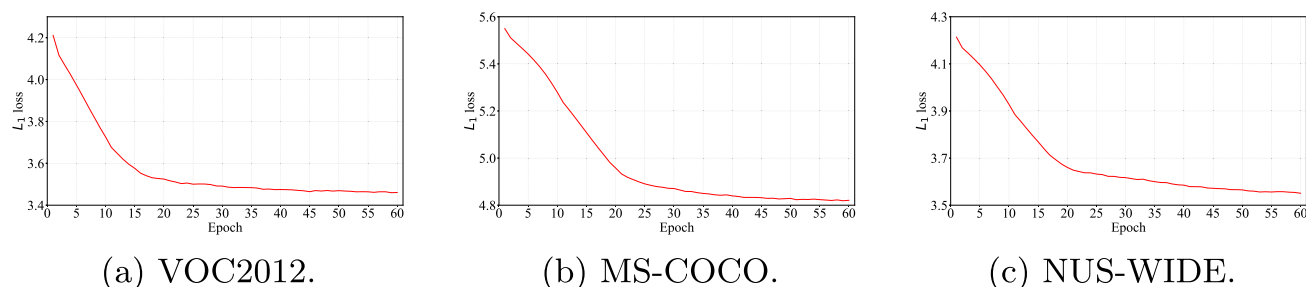
In this section, we conduct ablation studies to explore the influence of key components on our model. We mainly (i) observe the performance change with different Word2Vec techniques and (ii) record the object weight coefficient effect during the model training phase.

**Table 5** mAP@5000 comparisons under protocol [29] on MS-COCO

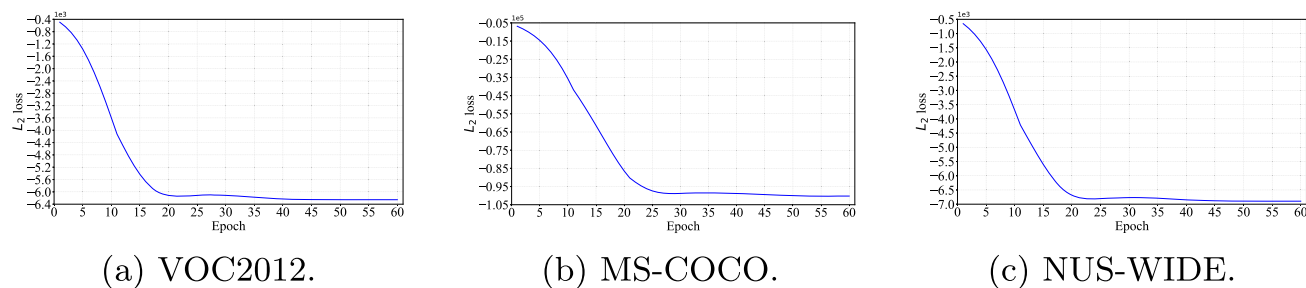
Method	16 bits	32 bits	48 bits	64 bits
CL-LSH	0.551	0.593	0.622	0.642
<b>HCCST</b>	<b>0.617</b>	<b>0.663</b>	<b>0.701</b>	<b>0.720</b>

Bold represents the optimal result





**Fig. 8** The change of  $L_1$  loss on VOC2012, MS-COCO and NUS-WIDE at 64 bits



**Fig. 9** The change of  $L_2$  loss on VOC2012, MS-COCO and NUS-WIDE at 64 bits

#### 4.3.1 Different Word2Vec techniques

Our HCCST constructs the semantic hash centers based on the label embeddings extracted from Word2Vec techniques. In this part, we record the mAP results of our model on VOC2012, MS-COCO and NUS-WIDE with different Word2Vec techniques including commonly used GoogleNews [27], GloVe [28], FastText [15] and BERT [8]. As shown in Table 6, all these 4 NLP-based techniques can bring good performance on these 3 datasets, but BERT yields the optimal mAP values at all hash code lengths. The main reason lies in two aspects. One is that BERT adopts the transformer structure that can keep the semantic relationships between different label embeddings. The other is that the hash center(s) and hash centroid generated from this transformer-based Word2Vec technique can well adapt to the hash code generated from the Swin transformer structure, thus greatly boosting the model performance

during the training phase. As a result, we adopt BERT as the default Word2Vec tool to extract the label embeddings.

#### 4.3.2 Object weight coefficient

As mentioned in Sect. 3.3, during the training phase, we first initialize the object weight coefficient of each sample and then refer to European projection to update this coefficient to construct the real hash centroid corresponding to this sample. In this part, we, respectively, record the mAP results on VOC2007, MS-COCO and NUS-WIDE using initialized fixed object weight coefficient (termed as fixed *owc*) and learned object weight coefficient (termed as learned *owc*) in Table 7. It can be seen that our learned *owc* can contribute to higher mAP values than fixed *owc* at all hash code lengths. The main reason is that this learned *owc* can help construct the real hash centroid, which makes each hash code approach its most related one or multiple

**Table 6** mAP results with different Word2Vec techniques



Word2Vec	VOC2012				MS-COCO				NUS-WIDE			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
GoogleNews [27]	0.852	0.876	0.886	0.897	0.799	0.845	0.855	0.867	0.781	0.825	0.839	0.851
GloVe [28]	0.857	0.881	0.890	0.889	0.806	0.848	0.865	0.869	0.783	0.831	0.840	0.852
FastText [15]	0.863	0.887	0.896	0.898	0.813	0.852	0.876	0.875	0.799	0.841	0.847	0.858
<b>BERT</b>	<b>0.875</b>	<b>0.898</b>	<b>0.905</b>	<b>0.913</b>	<b>0.860</b>	<b>0.899</b>	<b>0.935</b>	<b>0.933</b>	<b>0.815</b>	<b>0.850</b>	<b>0.860</b>	<b>0.866</b>

Bold represents the optimal result

**Table 7** Effect of object weight coefficient

<i>owc</i>	VOC2012				MS-COCO				NUS-WIDE			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
Fixed <i>owc</i>	0.841	0.866	0.872	0.879	0.834	0.858	0.892	0.895	0.785	0.819	0.825	0.836
<b>Learned <i>owc</i></b>	<b>0.874</b>	<b>0.898</b>	<b>0.905</b>	<b>0.913</b>	<b>0.860</b>	<b>0.899</b>	<b>0.935</b>	<b>0.933</b>	<b>0.815</b>	<b>0.850</b>	<b>0.860</b>	<b>0.866</b>

Bold represents the optimal result

image	epoch <i>owc</i>	epoch-0	epoch-5	epoch-10	epoch-15	epoch-20	epoch-25	epoch-30	epoch-35
 labels: banana, bench		0.500	0.495	0.484	0.460	0.380	0.315	0.242	0.240
		0.500	0.505	0.516	0.540	0.620	0.686	0.758	0.760
 labels: cat, handbag		0.500	0.506	0.518	0.540	0.572	0.596	0.603	0.604
		0.500	0.494	0.482	0.460	0.428	0.404	0.397	0.396

**Fig. 10** Two examples: the change of the learned object weight coefficient during the training phase

hash center(s) with an accurate coefficient, resulting in that those similar samples get close to the similar hash center(s). In addition, we also illustrate the change of the learned *owc* in Fig. 10. We randomly choose two images from MS-COCO and record the *owc* of each image during the training phase. As we see, for the first image that contains two objects (*i.e.*, banana and beach), we initialize the *owc* of each object as 0.5. With the training epoch increasing, our model will gradually converge and the *owc* of “banana” and “bench” will be stable at around “0.24” and “0.76”, respectively, within 30–35 epochs. The second image that contains “cat” and “handbag” further illustrates the learned *owc* of these two objects. These results reveal our proposed *owc* scheme is able to indicate the object scale size of each object, thus contributing to constructing a real hash centroid for each sample to promote the model performance.

## 5 Conclusion and future work

In this paper, we propose HCCST, a hash centroid construction method with Swin transformer for multi-label image retrieval. First, we adopt Swin transformer to extract

the feature vector for each input multi-label image and then generate the initialized hash code of this image. Second, we utilize the object semantic information to construct semantic hash centers and then consider the object scale size by learning the object weight coefficient (*i.e.*, *owc*) to compute the hash centroid for each sample. At last, we constantly limit the distance between each hash code and its hash centroid to preserve the similarity between samples. Our model will be trained in an end-to-end manner to alternately update the net parameters. We conduct extensive experiments on 3 multi-label image datasets including VOC2012, MS-COCO, NUS-WIDE. On the one hand, the results compared with the recent state-of-the-art hashing methods verify that HCCST can boost the image retrieval performance by constructing semantic hash centers and weighted hash centroids. On the other hand, the ablation studies reveal that the learned *owc* can effectively measure the object scale size to limit the distance between each hash code and its real hash centroid than the fixed *owc*. In the future, we aim to design graph convolution network based on the label embeddings to construct semantic hash centers in order to better capture the relationships between objects.

**Acknowledgements** Authors thank for the support of the National Natural Science Foundation of China Grant (No. 62232007), the

Research on the supporting technologies of the metaverse in cultural Media (PT252022039), Guangzhou Science and Technology Planning Project (No.202201010529), the National Natural Science Foundation of China (No.61902135), the National Natural Science Foundation of China (No.61871139) and the International Science and Technology Cooperation Projects of Guangdong Province (No.2020A0505100060).

**Data availability statement** The datasets generated during and/or analyzed during the current study are available in the open-source GitHub repository: <https://github.com/lzHZWZ/HCCST.git>

## Declarations

**Conflict of interests** The authors have no competing interests to declare that are relevant to the content of this article.

## References

- Çakir F, He K, Bargal SA, Sclaroff S (2019) Hashing with mutual information. *IEEE Trans Pattern Anal Mach Intell* 41(10):2424–2437
- Cao Y, Long M, Liu B, Wang J (2018) Deep cauchy hashing for hamming space retrieval. In: 2018 IEEE conference on computer vision and pattern recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018. p 1229–1237. Computer Vision Foundation / IEEE Computer Society
- Cao Y, Long M, Wang J, Zhu H, Wen Q (2016) Deep quantization network for efficient image retrieval. In: Schuurmans, D., Wellman, M.P. (eds.) *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, February 12–17, 2016, Phoenix, Arizona, USA. p 3457–3463. AAAI Press
- Cao Z, Long M, Wang J, Yu PS (2017) Hashnet: Deep learning to hash by continuation. In: *IEEE international conference on computer vision, ICCV 2017, Venice, Italy, October 22–29, 2017*. p 5609–5618. IEEE Computer Society
- Chen Y, Lu X (2020) Deep discrete hashing with pairwise correlation learning. *Neurocomputing* 385:111–121
- Chen Z, Wei X, Wang P, Guo Y (2019) Multi-label image recognition with graph convolutional networks. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16–20, 2019*. p 5177–5186. Computer Vision Foundation / IEEE
- Chua T, Tang J, Hong R, Li H, Luo Z, Zheng Y (2009) NUS-WIDE: a real-world web image database from national university of singapore. In: Marchand-Maillet, S., Kompatsiaris, Y. (eds.) *Proceedings of the 8th ACM international conference on image and video retrieval, CIVR 2009, Santorini Island, Greece, July 8–10, 2009*. ACM
- Devlin J, Chang M, Lee K, Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: Burstein, J., Doran, C., Solorio, T. (eds.) *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2–7, 2019, Vol 1 (Long and Short Papers)*. p 4171–4186. Association for Computational Linguistics
- Do T, Doan A, Cheung N (2016) Learning to hash with binary deep neural network. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) *Computer Vision - ECCV 2016 - 14th European conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part V. lecture notes in computer science, vol 9909*, p 219–234. Springer
- Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, Dehghani M, Minderer M, Heigold G, Gelly S, Uszkoreit J, Houlsby, N (2021) An image is worth 16x16 words: Transformers for image recognition at scale. In: 9th international conference on learning representations, ICLR 2021, Virtual Event, Austria, May 3–7, 2021. OpenReview.net
- Everingham M, Eslami SMA, Gool LV, Williams CKI, Winn JM, Zisserman A (2015) The pascal visual object classes challenge: A retrospective. *Int J Comput Vis* 111(1):98–136
- He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE conference on Computer vision and pattern recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016. p 770–778. IEEE Computer Society
- He K, Çakir F, Bargal SA, Sclaroff S (2018) Hashing as tie-aware learning to rank. In: 2018 IEEE conference on computer vision and pattern recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018. p 4023–4032. Computer Vision Foundation / IEEE Computer Society
- Jégou H, Douze M, Schmid C (2011) Product quantization for nearest neighbor search. *IEEE Trans Pattern Anal Mach Intell* 33(1):117–128
- Joulin A, Grave E, Bojanowski P, Douze M, Jégou H, Mikolov T (2016) Fasttext.zip: Compressing text classification models. *CoRR arXiv: abs/1612.03651*
- Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Bartlett, P.L., Pereira, F.C.N., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) *Advances in Neural Information Processing Systems 25: 26th annual conference on neural information processing systems 2012. Proceedings of a meeting held December 3–6, 2012, Lake Tahoe, Nevada, United States*. p 1106–1114
- Lai H, Pan Y, Liu Y, Yan S (2015) Simultaneous feature learning and hash coding with deep neural networks. In: *IEEE conference on computer vision and pattern recognition, CVPR 2015, Boston, MA, USA, June 7–12, 2015*. p 3270–3278. IEEE Computer Society
- Lecun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436
- Li W, Wang S, Kang W (2016) Feature learning based deep supervised hashing with pairwise labels. In: Kambhampati, S. (ed.) *Proceedings of the twenty-fifth international joint conference on artificial intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016*. p 1711–1717. IJCAI/AAAI Press
- Liang Y, Pan Y, Lai H, Liu W, Yin J (2022) Deep listwise triplet hashing for fine-grained image retrieval. *IEEE Trans Image Process* 31:949–961
- Lin M, Ji R, Liu H, Sun X, Chen S, Tian Q (2020) Hadamard matrix guided online hashing. *Int J Comput Vis* 128(8):2279–2306
- Lin M, Ji R, Liu H, Wu Y (2018) Supervised online hashing via hadamard codebook learning. In: Boll, S., Lee, K.M., Luo, J., Zhu, W., Byun, H., Chen, C.W., Lienhart, R., Mei, T. (eds.) 2018 ACM multimedia conference on multimedia conference, MM 2018, Seoul, Republic of Korea, October 22–26, 2018. p 1635–1643. ACM
- Lin T, Maire M, Belongie SJ, Hays J, Perona P, Ramanan D, Dollár P, Zitnick CL (2014) Microsoft COCO: common objects in context. In: Fleet, D.J., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) *Computer Vision - ECCV 2014 - 13th european conference, Zurich, Switzerland, September 6–12, 2014, proceedings, Part V. Lecture Notes in Computer Science, vol 8693*, p 740–755. Springer
- Liu H, Wang R, Shan S, Chen X (2016) Deep supervised hashing for fast image retrieval. In: 2016 IEEE conference on computer vision and pattern recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016. p 2064–2072. IEEE Computer Society

25. Liu W, Wang J, Ji R, Jiang Y, Chang S (2012) Supervised hashing with kernels. In: 2012 IEEE conference on computer vision and pattern recognition, Providence, RI, USA, June 16–21, 2012. p 2074–2081. IEEE Computer Society
26. Liu Z, Lin Y, Cao Y, Hu H, Wei Y, Zhang Z, Lin S, Guo B (2021) Swin transformer: Hierarchical vision transformer using shifted windows. In: 2021 IEEE/CVF international conference on computer vision, ICCV 2021, Montreal, QC, Canada, October 10–17, 2021. p 9992–10002. IEEE
27. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. In: Bengio, Y., LeCun, Y. (eds.) 1st international conference on learning representations, ICLR 2013, Scottsdale, Arizona, USA, May 2–4, 2013, Workshop Track Proceedings
28. Pennington J, Socher R, Manning CD (2014) Glove: Global vectors for word representation. In: Moschitti, A., Pang, B., Daelemans, W. (eds.) Proceedings of the 2014 conference on empirical methods in natural language processing, EMNLP 2014, October 25–29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL. p 1532–1543. ACL
29. Sablayrolles A, Douze M, Usunier N, Jégou H (2017) How should we evaluate supervised hashing? In: 2017 IEEE international conference on acoustics, speech and signal processing, ICASSP 2017, New Orleans, LA, USA, March 5–9, 2017. pp. 1732–1736. IEEE
30. Tu R, Mao X, Guo J, Wei W, Huang H (2021) Partial-softmax loss based deep hashing. In: Leskovec, J., Grobelnik, M., Najork, M., Tang, J., Zia, L. (eds.) WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19–23, 2021. pp. 2869–2878. ACM / IW3C2
31. Wang D, Huang H, Lin H, Mao X (2017) Supervised hashing for multi-labeled data with order-preserving feature. In: Cheng, X., Ma, W., Liu, H., Shen, H., Feng, S., Xie, X. (eds.) Social media processing - 6th national conference, SMP 2017, Beijing, China, September 14–17, 2017, Proceedings. communications in computer and information Science, vol 774, p 16–28. Springer
32. Wang J, Zhang T, Song J, Sebe N, Shen HT (2018) A survey on learning to hash. IEEE Trans Pattern Anal Mach Intell 40(4):769–790
33. Wang W, Carreira-Perpiñán MÁ (2013) Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application. CoRR [arXiv: abs/1309.1541](https://arxiv.org/abs/1309.1541)
34. Weiss Y, Torralba A, Fergus R (2008) Spectral hashing. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) Advances in neural information processing Systems 21, Proceedings of the twenty-second annual conference on neural information processing systems, Vancouver, British Columbia, Canada, December 8–11, 2008. p 1753–1760. Curran Associates, Inc
35. Xia R, Pan Y, Lai H, Liu C, Yan S (2014) Supervised hashing for image retrieval via image representation learning. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the twenty-eighth AAAI conference on artificial intelligence, July 27 –31, 2014, Québec City, Québec, Canada. p 2156–2162. AAAI Press
36. Xie Y, Liu Y, Wang Y, Gao L, Wang P, Zhou K (2020) Label-attended hashing for multi-label image retrieval. In: Bessiere, C. (ed.) Proceedings of the twenty-ninth International joint conference on artificial intelligence, IJCAI 2020. p 955–962. ijcai.org
37. Yuan L, Wang T, Zhang X, Tay FEH, Jie Z, Liu W, Feng J (2020) Central similarity quantization for efficient image and video retrieval. In: 2020 IEEE/CVF conference on computer vision and pattern recognition, CVPR 2020, Seattle, WA, USA, June 13–19, 2020. p 3080–3089. Computer vision foundation / IEEE
38. Zhang Z, Zou Q, Lin Y, Chen L, Wang S (2020) Improved deep hashing with soft pairwise similarity for multi-label image retrieval. IEEE Trans Multimed 22(2):540–553
39. Zhu H, Long M, Wang J, Cao Y (2016) Deep hashing network for efficient similarity retrieval. In: Schuurmans, D., Wellman, M.P. (eds.) Proceedings of the thirtieth AAAI conference on artificial intelligence, February 12–17, 2016, Phoenix, Arizona, USA. p 2415–2421. AAAI Press

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.