Label-Affinity Self-Adaptive Central Similarity Hashing for Image Retrieval

Yanzhao Xie, Rukai Wei, Jingkuan Song, Yu Liu, Yangtao Wang, and Ke Zhou, *Member, IEEE*

***Abstract*—Due to the usage of global similarity, the hashing methods based on predefined hash centers have achieved more accurate retrieval results than the pairwise/triplet-based methods. Nevertheless, the fixed hash centers lack the perception of data distribution and are limited by the pre-determined Hadamard matrix, which consider neither the label semantic information nor the object scale size, resulting in sub-optimal retrieval performance and weak generalization ability. In this article, we (1) adopt the label semantic information to generate self-adaptive hash centers and (2) propose the label-affinity coefficient (*lac*) that considers the scale size of each label/object appearing in the given image to calculate the real hash centroid for this image. Based on this, we propose *Label-affinity Self-adaptive Central Similarity Hashing (LSCSH)* for image retrieval. LSCSH consists of a hash code generator module and a hash center adapter module. First, we obtain the label word vector (i.e., the word vector representation of each class label) via the Word2Vector technique to generate and update the hash centers that adapt to the distribution of both label word vectors and generated hash codes. Second, we learn *lac* to indicate the dominance of different labels corresponding to objects in each given image, which considers the unequal scales of each object (corresponding to a label) to calculate a more accurate hash centroid for each image. Last but not least, we design an asynchronous learning mechanism to enable each hash code and its corresponding hash centroid to adapt to each other dynamically. We conduct extensive experiments on 5 image datasets including CIFAR-10, ImageNet, VOC2012, MS-COCO and NUS-WIDE. The experimental results demonstrate that LSCSH can achieve the state-of-the-art visual retrieval performance on both single-label and multi-label image datasets.**

***Index Terms*—Image retrieval, label semantic information, hash centers, label-affinity coefficient, hash centroid.**

Manuscript received 24 November 2022; revised 1 February 2023; accepted 13 February 2023. Date of publication 23 February 2023; date of current version 15 December 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 61902135, in part by the National Natural Science Foundation of China under Grant 62232007, in part by the Research on the supporting technologies of the metaverse in cultural Media under Grant PT252022039, and in part by Guangzhou Science and Technology Planning Project under Grant 202201010529. The Associate Editor coordinating the review of this manuscript and approving it for publication was Dr Mei-Ling Shyu. *(Yanzhao Xie and Rukai Wei are co-first authors.) (Corresponding authors: Yu Liu; Yangtao Wang.)*

Yanzhao Xie, Rukai Wei, and Ke Zhou are with the Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: yzxie@hust.edu.cn; weirukai@hust.edu.cn; zhke@hust.edu.cn).

Yu Liu is with the School of Computer of Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: liu_yu@hust.edu.cn).

Jingkuan Song is with the University of Electronic Science and Technology of China, Chengdu 4067, China (e-mail: jingkuan.song@gmail.com).

Yangtao Wang is with the School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou 511442, China (e-mail: ytaowang@gzhu.edu.cn).

The code of this work is released at: https://github.com/lzHZWZ/LSCSH_sourcecode.git.

Digital Object Identifier 10.1109/TMM.2023.3248170

I. Introduction

DEEP hashing methods dramatically improve retrieval accuracy, since deep features can accurately represent the correlation between data. Meanwhile, the hash loss function plays a vital role in this learning process because it guides the network to learn the deep features preserving the correlation. Most hash loss functions measure the relative distances between data by pairwise [1], [2], [3], [4] or triplet similarity [5], [6]. However, these hashing methods suffer from 2 severe problems. (1) They bring extra computational overhead for constructing the whole pairwise/triplet samples over the dataset, resulting in the tough preparation of large-scale training data. (2) Constrained by the current mini-batch learning manner, the ratio of similar samples and dissimilar samples in each batch is usually imbalanced, resulting in that the model failed to recognize the overall data distribution based on this mini-batch limitation and achieving suboptimal performance.

To overcome these defects, Yuan et al. [7] propose a Central Similarity Quantization (CSQ) scheme for image and video retrieval, using the hash center loss function rather than the pairwise/triplet loss function. It obtains hash centers extracted from a Hadamard matrix and generates hash codes around the fixed hash centers according to the data distribution. Specifically, hash centers are mutual orthogonal row vectors randomly extracted from the Hadamard matrix manually, and thus the average Hamming distance between hash centers is at least $K/2$, where K is the length of the hash code and each center corresponds to a data class category (i.e., a label). In this way, CSQ can push similar data to be close to their own hash center from a global perspective, and the inter-class distances between dissimilar data become big enough since there are fixed distances between initialized hash centers corresponding to class labels sampled from a Hadamard matrix.Nevertheless, fixed hash centers contain no semantic information, which has nothing to do with the object semantics and data distribution. Besides, they neglect the object scale size during the hash centroid calculation. Especially For the multi-label images [4], as shown in Fig. 1, CSQ treats each label appearing in the image as equally important, and then calculates the hash centroid (i.e., the black dot) using the average of the corresponding hash centers, resulting in that the hash code of each sample will deviate from its corresponding real hash centroid (i.e., the gray dot).

To address these issues, we expect that (1) the hash centers should contain semantic relationship between objects, and (2) the hash centroid calculation should consider the scale size of different objects appearing in the image. As a result, in this article, we propose Label-affinity Self-adaptive Central Similarity Hashing (LSCSH), a central similarity based hashing method for both single-label and multi-label image retrieval. LSCSH consists of a hash code generator module and a hash center adapter module, where the former adopts Convolution Neural Network (CNN) to generate the hash code for each image and the latter constructs hash centers according to the label semantic information, i.e., every hash center corresponds to a class label. Firstly, inspired by cross-modal hashing methods [8], [9], we obtain the label word vector (i.e., the word vector representation of each class label) via the Word2Vector technique to generate and update the hash centers that adapt to the distribution of both label word vectors and generated hash codes. Secondly, as shown in Fig. 1, we propose *label-affinity coefficient (lac)* to indicate the dominance of different labels corresponding to objects in multi-label images, which considers the unequal scales of each object (corresponding to a label) to calculate a more accurate hash centroid for each image. Last but not least, we design an asynchronous learning mechanism to enable each hash code and its corresponding hash centroid to adapt to each other dynamically. As a result, we will yield high-quality semantic hash code for each sample after completing the model training. To the best of our knowledge, this is the first time that utilizes the label semantic information to generate self-adaptive hash centers. We conduct extensive experiments on 5 image datasets including CIFAR-10, ImageNet, VOC2012, MS-COCO and NUS-WIDE. The experimental results demonstrate that LSCSH can achieve the state-of-the-art image retrieval performance on both single-label and multi-label image datasets.

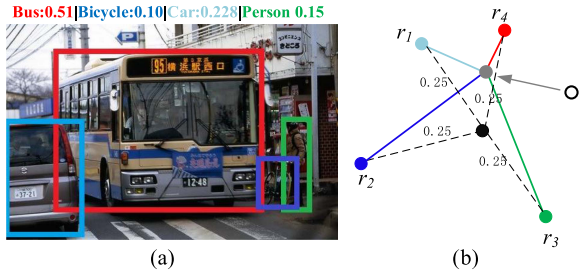


Fig. 1. An illustration of a hash centroid calculation using our LSCSH on a multi-label image. (a) The multi-label image has four labels: *bus, car, bicycle and person*. Note that each label corresponds to a hash center. The decimals above the image denote the *label-affinity coefficient (lac)* related to the scales of the corresponding objects. (b) The hollow dot denotes the continuous hash code of a sample. The color (i.e., red, light blue, blue and green) dots respectively correspond to hash centers about the aforementioned four labels. The gray dot and black dot respectively denote the real hash centroid (i.e., the weighted sum of hash centers based on our proposed *lac*) and the average hash centroid (i.e., the weighted sum of hash centers based on equal weight). Our goal is to push the hash code of a multi-label sample to be close to the real hash centroid represented by the gray dot, as shown by the solid gray arrow.

II. Related Works

Deep hashing methods can be classified into supervised [4], [10], [11] or unsupervised [12], [13], [14] methods according to whether they use image annotations or not. Additionally, our work makes full use of the label semantic [9] information to generate hash centers, so we mainly talk about supervised hashing methods. The existing supervised deep hashing methods [4], [15], [16], [17], [18] usually adopt a CNN backbone to capture the feature of images and then design a hash layer to map the features to compact binary codes, which explore the supervised information to generate discriminative and high-quality hash codes. They mainly fall into two lines and we discuss them below.

A. Pairwise/Triplet Approaches

These approaches use pairwise/triplet similarity loss functions. By minimizing (maximizing) the Hamming distance between similar (dissimilar) data pairs, Supervised Discrete Hashing (SDH) [19] generates the discrete binary hash codes, which elaborates a new hashing framework for linear classification and proposes a novel solution for the NP-hard binary optimization. The Supervised Hashing with Kernels (KSH) [20] generates the nonlinear and discriminative hash codes. To address the issues of large computational overhead and the cumbersome model training process, KSH proposes a kernel-based hashing model using the similar/dissimilar information of samples to construct a low training cost model to help acquire high-quality hash codes. For multi-label images scenario, Deep Cauchy Hashing (DCH) [21] adopts the pairwise cross-entropy loss based on Cauchy distribution, which significantly forces the hash codes of similar data pairs to fall into shorter hamming distance scope. Label-attended Hashing (LAH) [15] combines CNN and Graph Convolution Network (GCN) [22], which first respectively extracts the features from multi-label images and aggregates the label co-occurrence information into label word vectors, and then establishes pairwise similarity preserving loss function for hashing learning. Both DCH and LAH essentially exploit the hash function based on pairwise similarity loss. Deep Co-image-label Hashing (DCILH) [23] extracts a shared Hamming space between images and corresponding labels and exploits the dependency relations among multiple labels. Convolution Neural Network Hashing (CNNH) [18] first learns approximate hash codes from pairwise similarity matrix decomposition and then simultaneously learns image features and hash functions with the raw image pixels. Triplet based hashing methods such as Deep Neural Network Hashing (DNNH) [24] use Triplet Ranking Loss to learn the similarity. Instead of using hand-engineering visual features, DNNH leverages deep neural network to extract the feature of each image and proposes a triplet ranking loss which can characterize one image is more similar to the second one than to the third one. Instance-aware Hashing (IAH) [25] utilizes the region proposal method to obtain specific objects from a multi-label image and constructs the triplet similarity based loss function and the *Softmax* loss function to optimize the whole deep hashing network as well as generates compact hash codes. Improved Deep Hashing Network (IDHN) [4] propose a soft definition for a pairwise similarity with normalized semantic labels and combines the cross-entropy loss as well as

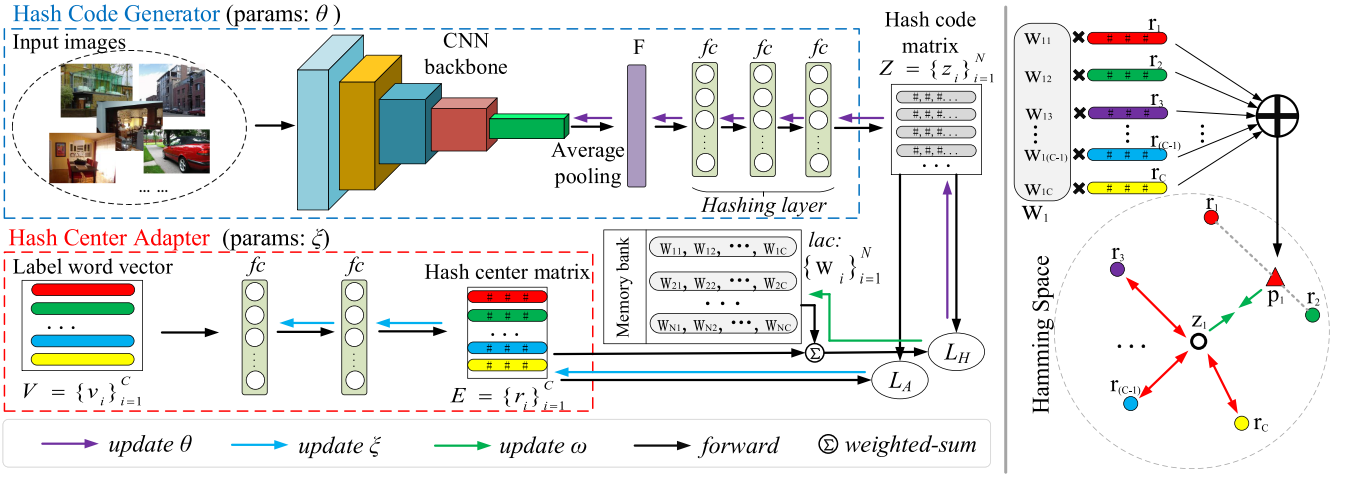


Fig. 2. The left part presents the overall framework of LSCSH that consists of a hash code generator module and a hash center adapter module. The black arrow denotes the forward propagation. L_H and L_A denote the corresponding loss function of the above two modules respectively, which update the network parameters in an asynchronous learning mechanism, denoted by the purple and blue arrow respectively. The right part displays the details of hash centroid calculation. LSCSH aims to push a continuous code (e.g., z_1) to be close to its corresponding hash centroid (e.g., p_1) and keep as large distance between this hash code and all other irrelevant hash centers (i.e., $r_3, \ldots, r_{C-1}, r_C$) as possible.

Mean Squared Error (MSE) loss to generate robust hash codes. Partial-softmax Loss based Deep Hashing (PSLDH) [26] trains a category hashing network to generate category codes, which helps yield similarity hash codes that preserve semantic information by a novel Partial-SoftMax loss. Note that Vision Transformer Hashing (VTS) [27] reveals that most image hashing methods can achieve higher retrieval performance when adopting a vision transformer backbone to extract image features. Instead, these pairwise/triplet similarity based methods inevitably suffer from huge computational overhead caused by the enumeration of all the sample pairs/triplets. In the large-scale visual scenario, it is impracticable to establish such a huge amount of sample pairs/triplets.

B. Quantization Approaches

The another approach is based on quantization loss function [16], [28], [29], [30], whose target is to alleviate the huge computational overhead caused by exhaustive enumeration of all pairwise/triplet pairs in a large-scale image dataset. This issue has been effectively explored in many representative researches such as Iterative Quantization (ITQ) [31], Deep Hashing (DH) [32] and Supervised Hashing with Binary Deep Neural Network (SH-BDHH) [29], *etc*. ITQ [31] partitions the feature space into disjoint subspaces that are quantized independently. DH [32] combines deep learning methods and designs its own quantization loss to generate binary codes. SH-BDNN [29] exploits the deep neural network and adopts Hamming distance similarity as metric to construct quantization loss based objective function. Recently, a central similarity based method called CSQ [7] yields a new research direction and attracts much more attention. It first utilizes the Hadamard matrix to obtain hash centers, and then pushes data samples to be close around the orthogonal hash centers.

As mentioned above, it is impracticable to exhaustively enumerate all the data pairs/triplets owing to the high computational complexity, especially for large-scale datasets. Although CSQ gets rid of the dependence on data pairs/triplets, it utilizes the stationary hash centers initialized with Hadamard matrix in a manual manner, which failed to consider the semantic information of each hash center. Different from the existing works, our proposed LSCSH fuses the label semantic information into its hash centers and adopts a new asynchronous learning manner to optimize the parameters of hash code generator and hash center adapter, which greatly helps generate high-quality hash codes and improve the visual retrieval performance on both single-label and multi-label image datasets.

III. Proposed Method

In this section, we first present the overall framework of LSCSH, then introduce the work-flow of these modules and the learning strategy of LSCSH in detail.

A. Overall Framework

As shown in Fig. 2, the overall framework of our LSCSH consists of a hash code generator module and a hash center adapter module. The hash code generator module uses a CNN backbone to extract the visual feature of the input image and generate its corresponding hash code. The hash center adapter module aims to adaptively construct the semantic hash centers based on the label semantic information extracted from the Word2Vector technique. After obtaining the initialized hash code and hash centers, LSCSH treat the scale size of each label/object appearing in the input image as learnable parameters to calculate the hash centroid (i.e., the weighted sum of hash centers) for this image. We design an asynchronous learning mechanism to efficiently update the network parameters to enable each hash code and its corresponding hash centroid to adapt to each other dynamically. At last, the model will output the final hash code of each sample after completing the training process. We explain the details of each module below.

B. Hash Code Generator

The hash code generator module first extracts the visual image feature and then generates the hash code for each sample. Formally, we assume that there are N samples $\{x_i\}_{i=1}^{N}$ and C categories in the training set. Let $L(x_i) = [y_{i1}, \ldots, y_{iC}]$ denote the corresponding ground truth labels of the sample x_i, where $y_{ij} \in \{0, 1\}$ is a discrete binary indicator for $j = 1, \ldots, C$, i.e., $y_{ij} = 1$ if the sample x_i has the j-th category label and otherwise $y_{ij} = 0$. This module will map each sample from the input space to Hamming space: $x_i \mapsto h_i \in \{-1, 1\}^K$. Any CNN based models can complete the feature extraction of an image in our architecture. For fair comparisons with the existing work [7], we use ResNet-50 [33] and AlexNet [34] to extract the image-level feature in our experiments. Based on this, for an image x_i with the resolution 224×224, we can respectively obtain a $7 \times 7 \times 2048$ and 4096 dimensional feature vector. For ResNet-50, We use the global average pooling to generate the image-level feature. After completing the feature extraction, a hashing layer consisting of three fully connected (fc) layers and non-linear activation functions will map these extracted image-level feature to a real-valued vector:

$$z_i = \tanh(H(\mathcal{F}_{CNN}(x_i; \theta))), \tag{1}$$

where θ denotes the parameters of hash code generator module, $\mathcal{F}_{CNN}$ denotes the feature extraction network, H denotes the hashing layer, $tanh$ denotes the activation function, and z_i denotes the K-dimensional continuous code which is on the interval $(-1, 1)$, i.e., $z_i \in (-1, 1)^K$. Note that during the training phase, we will use z_i combined with the hash center adapter module to calculate the loss. While in the inference phase, we will remove the hash center adapter module, and only preserve the hash code generator module followed by the binarization operation with the sign function to generate the hash code $h_i = sign(z_i)$ for each sample x_i. In the next part, we will introduce how to adaptively construct semantic hash centers to match these hash codes.

C. Hash Center Adapter

As we know, the word description naturally contains the semantic relationships between objects. Intuitively, it is common sense that the label 'dog' will be more similar to label 'cat' than 'bus'. Thus, the hash center of label 'cat' should be closer to that of label 'dog' than 'bus' in the Hamming space. Our adaptive hash centers can greatly incorporate label semantic information via keeping the relative distance between different labels. In addition, our scheme is able to better adapt to the distribution of a given dataset, thus promoting the generalization ability of the model. Based on these considerations, different from the previous works that construct fixed hash centers without considering semantic correlations, in this part, we adaptively generate and adjust the hash centers according to the label word vector. Specially, we first utilize the BERT [35] model to convert each object (i.e., the word description of each label) into a word vector to obtain a label word vector matrix $V = \{v_i\}_{i=1}^{C} \in \mathbb{R}^{C \times D}$, where C is the number of class categories mentioned in Section III-B and $D = 768$ is a default dimension of each word vector in BERT. Note that we have also tried other Word2Vector methods (i.e., GoogleNews [36], GloVe [37] and FastText [38]) and find BERT can bring better results. Next, we design 2 fc layers to generate and adjust the hash centers. By this means, we can embed the semantic relationship between different objects into hash centers. The mapping function $g : v_i \mapsto r_i$ denotes the hash center adapter function from label word vectors to continuous hash centers and this process can be formulated as:

$$r_i = g(v_i; \xi), \tag{2}$$

where ξ denotes the parameters of g, r_i denotes the i-th hash centers. In this way, we will obtain C continuous hash centers $E = \{r_j\}_{j=1}^{C} \in \mathbb{R}^{C \times K}$ based on label word vectors with randomly initialized parameters ξ of this module. After obtaining both hash codes and hash centers, our goal is to enable each hash center and each hash code to dynamically adapt to each other to generate high-quality hash codes. As we see in Fig. 2, most of the hash centers are irrelevant to a given sample if the corresponding labels/objects have not appeared in this sample. We aim to ensure that each hash code of a sample will be close to the relevant hash centers and far away from the irrelevant hash centers. To achieve this goal, we propose to calculate the hash centroid (i.e., the weighted sum of the relevant hash centers) for each image, which considers the scale size of each label/object appearing in this image. We describe the calculation process in Section III-D.

D. Hash Centroid Calculation

Previous works adopt the fixed hash centers and neglect the object size of different objects in the hash centroid calculation process. In this part, based on the constructed hash centers, we propose label-affinity coefficient (*lac*) to calculate the real hash centroid for each image. Inspired by LTHNet [39], as shown in Fig. 2, *lac* is maintained as a table in memory dubbed as *Memory bank* that records all $\{w_i\}_{i=1}^{N}$ for all samples one by one during the model training, where $w_i = \{w_{ij}\}_{j=1}^{C}$ and w_{ij} denotes the label weight of the j-th label/object appearing in sample x_i that satisfies $\sum_{j=1}^{C} w_{ij} = 1$. As a result, the hash centroid p_i of sample x_i can be calculated as:

$$p_i = \sum_{j=1}^{C} w_{ij} r_j, \tag{3}$$

where r_j denotes the j-th hash centers. We further explain how to obtain w_{ij} below.

As mentioned in Section III-B, $L(x_i) = [y_{i1}, \ldots, y_{iC}]$ denotes the corresponding ground truth labels of the sample x_i, where $y_{ij} \in \{0, 1\}$ is a discrete binary indicator for $j = 1, \ldots, C$, i.e., $y_{ij} = 1$ if the sample x_i has the j-th category label and otherwise $y_{ij} = 0$. CSQ [7] neglects the object scale size and regards each object in an image as equally important. Take the right part of Fig. 2 for example, we assume an image x_1 includes two labels that correspond to the index 0,1 of its label vector (i.e., $\underbrace{[1, 1, 0, 0, \ldots]}_{C}$). That is to say, $y_{11} = y_{12} = 1$. CSQ will directly set and fix $w_{11} = w_{12} = 1/2$ and the value of all

other remaining elements in w_1 will be set as 0. Thus, the centroid p_1 is only influenced by hash centers r_1 and r_2. However, different objects in the same image will occupy different scales, which means different hash centers own different weights on the hash centroid. To obtain the accurate hash centroid of sample x_i, we treat w_i as learned parameters and refer to European projection [40] to update w_i during the model iteration.

Specifically, for each sample x_i, we denote $|L(x_i)|$ as the number of non-zero elements in $L(x_i)$. Based on this, the corresponding position of each non-zero element in w_i will be initialized as $1/|L(x_i)|$. Since the weighted sum in w_i equals 1 (i.e., $\sum_{j=1}^{C} w_{ij} = 1$), we combine European projection [40] and gradient descent to effectively learn and update w_i, which can reflect the scale size of objects in x_i and bring the accurate hash centroid p_i to help yield high-quality hash codes. Next, we will introduce how to train our model by constantly limiting the distance between each hash code z_i and its corresponding hash centroid p_i in Section III-E and Section III-F.

E. Loss Functions of LSCSH

After obtaining both the initialized hash code z_i and the corresponding hash centroid p_i of each sample x_i, we begin to train our LSCSH by constantly limiting the distance between z_i and p_i. To achieve this goal, we design a *Weighted-Softmax Loss* to achieve the homogeneous polymerization and heterogeneous dispersion goal as follows:

$$
\begin{aligned}
L_{ws} &= -\frac{1}{N}\sum_{i=1}^{N}\log\frac{SIM\_POS(i)}{SIM\_POS(i)+SIM\_NEG(i)},\\
SIM\_POS(i) &= \exp\left(\frac{sim(z_i,p_i)}{\tau}\right),\\
SIM\_NEG(i) &= \sum_{r_j^{(n)}\in\mathcal{S}_{neg}}\exp\left(\frac{sim(z_i,r_j^{(n)})}{\tau}\right),
\end{aligned}
\tag{4}
$$

where $j \in \{1, 2, \ldots, C\}$, $sim(\cdot,\cdot)$ denotes the similarity measurement (e.g., cosine distance or Euclidean distance, and here we adopt the cosine distance), τ is the scale parameter for adapting the scale of hypersphere space to fit the requirements of feature representation. $\mathcal{S}_{neg}$ represents a subset of hash centers, each of which element corresponds to the 0 element in the i-th label vector, i.e., these hash centers have no substantial effect on the hash centroid calculation. $r_{(\cdot)}^{(n)}$ with superscript (n) denotes the hash center comes from $\mathcal{S}_{neg}$. In this way, this loss not only inherits the advantages of conventional hash loss that can push each hash code to be close to its corresponding hash centroid to keep the intra-class hash codes to be high polymerization, but also considers the inter-class discriminative features to bring a bigger margin between the cluster of hash codes from different classes. Note that this *Weighted-Softmax Loss* will take participate in the whole learning process including hash code optimization, hash center optimization and hash centroid optimization.

1) Hash Code Optimization: The hash code generator module aims to generate and update each hash code according to its corresponding current hash centroid. The *Weighted-Softmax Loss* (see (4)) here can be interpreted as intra-class and inter-class loss for a hash code given its hash centroid, which aims to ensure that every two hash codes become similar (dissimilar) to each other if their samples belong to the same (different) class(es). In addition, owing to that the output from the hashing layer in Fig. 2 is real-valued vectors, we have to utilize a quantization loss to approximate these floating-point numbers to 1 or -1 as follows:

$$
L_q = \frac{1}{N}\sum_{i=1}^{N}||h_i - z_i||_2^2, \tag{5}
$$

where h_i and z_i respectively denote the binary and real-valued hash codes corresponding to the sample x_i. As we see, the hash centroid calculation is determined by all labels/objects appearing in an image. To prevent a certain label weight from dominating the loss function during the updating process as well as increase the robustness of our model, we utilize the maximum entropy regularization to optimize these label weights as:

$$
R(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} w_{ij}\log(w_{ij}). \tag{6}
$$

As a result, the whole *Hash Loss* in the hash code generator module consists of the above three loss functions, and we formulate this *Hash Loss* as:

$$
L_H = L_{ws} + \mu L_q + \lambda R, \tag{7}
$$

where μ and λ are two weighted factors. In the experiments, we set $\mu = 0.05$ and $\lambda = 0.001$ by grid search. We directly use L_H to update the parameter θ (i.e., hash code optimization) via gradient descent in Section III-F.

2) Hash Center Optimization: The hash center adapter module aims to generate semantic hash centers based on the label word vectors extracted from the BERT [35] model. As we see, this Word2Vector method not only maps the label words into vectors, but also maintains the semantic similarity between different label word vectors, which effectively integrates the semantic relationship into hash centers. This way can greatly keep the relative distances between hash centers according to the corresponding label semantic correlation (e.g., the distance between label *Bus* and *Car* is less than that between label *Person* and *Bus*). In order to make the distribution of hash centers t_{ij} close to the distribution of label word vectors s_{ij}, we refer to the commonly-used Kullback-Leibler (KL) divergence that has been proved valid in representation learning [41] and design a *Label-center Loss* to achieve this approximation:

$$
\begin{aligned}
L_{label-center} &= \sum_{i=1}^{C}\sum_{j=1}^{C} s_{ij}\log\frac{s_{ij}}{t_{ij}},\\
s_{ij} &= \frac{1}{2}(\cos(v_i, v_j)+1),\\
t_{ij} &= \frac{1}{2}(\cos(r_i, r_j)+1),
\end{aligned}
\tag{8}
$$

where v_i and r_i respectively denote the i-th label word vector and hash center. In addition, to indicate the distance between

different hash centers for greatly distinguishing the generated hash codes, we formulate the *Inter-class Loss* as:

$$L_{inter} = -\sum_{i=1}^{C}\sum_{j=1,j\neq i}^{C} ||r_i - r_j||_2^2, \tag{9}$$

where *Inter-class Loss* aims to increase the inter-class separability between r_i and r_j, guiding the hash center adapter network to put as large distance as possible between any two hash centers.

Based on the above design, given a current hash code set, we will optimize the hash center adapter module towards the following three aspects: (i) L_{ws} aims to decrease the distance between each hash code and its own corresponding hash centroid, (ii) $L_{label-center}$ aims to maintain the similarity between the distribution of hash centers and label word vectors, and (iii) L_{inter} aims to make the distances between different hash centers as large as possible. As a result, we formulate the *Adapter Loss* as:

$$L_A = L_{ws} + \alpha_1 L_{label-center} + \alpha_2 L_{inter}, \tag{10}$$

where α_1 and α_2 are two weighted factors to trade-off the loss items. In our experiments, we set $\alpha_1 = 0.5$ and $\alpha_2 = 10.0$ by grid search. We directly use L_A to update the parameter ξ (i.e., hash center optimization) via gradient descent in Section III-F.

3) Hash Centroid Optimization: As mentioned before, the hash centroid of each image is calculated based on both hash centers and *lac*, where the former is generated by hash center adapter and the latter will be obtained by *Euclidean Projection* [40] and gradient descent technique. Given hash codes and hash centers, *lac* will be updated by the gradient vector of the objective function L_H (see (7)) with respect to w_i, which can be formulated as:

$$\nabla w_{ij} = \frac{\partial L_H}{\partial w_{ij}^{t-1}}, \quad \hat{w}_{ij}^t = w_{ij}^{t-1} - \eta \nabla w_{ij}, \tag{11}$$

where w_{ij}^t denotes the value of w_{ij} at t-th iteration. As we know, the sum of all elements in w_i equals 1 and we need to preserve this principle after each iteration. Since *Euclidean Projection* [40] is suitable for this optimization, we refer to it to project $\hat{w}_i^t = \{\hat{w}_{ij}^t\}_{j=1}^C$ onto $w_i^t = \{w_{ij}^t\}_{j=1}^C$ and output the optimal value of w_{ij} as follows:

$$w_i^t = \min_{w_i^t} \frac{1}{2} \| w_i^t - \hat{w}_i^t \|^2, \quad \sum_{j=1}^{C} w_{ij} = 1, w_{ij} \geq 0. \tag{12}$$

In this way, w_i^t reflects the label weights of x_i in the t-th iteration and we combine w_i^t and hash centers to calculate the hash centroid p_i in this iteration. Next, we will introduce the training work-flow of LSCSH via our synchronous learning mechanism in Section III-F.

F. Asynchronous Learning Mechanism

After constructing the above loss functions, we begin to train our network and update the network parameters. We design an asynchronous learning mechanism to optimize our LSCSH. There are mainly three learning parameters, i.e., θ, ξ and *lac*, where θ and ξ can be optimized by back propagation shown in the purple and blue arrows in Fig. 2, while *lac* is maintained in a memory bank [42]. We update the hash code generator module and hash center adapter module in an asynchronous learning manner. In brief, we first utilize the current parameter ξ of the hash center adapter, parameter θ of the hash code generator and *lac* to generate the current hash codes and calculate hash loss L_H according to a batch of input data. Afterwards, *lac*, L_H and θ will be updated in sequence. For every batch of data in each iteration, the above procedure will be carried out until all data has been processed. Next, based on the obtained θ and *lac* after each iteration, we begin to calculate adapter loss L_A and update all hash codes as well as parameter ξ. In this way, the loss functions (i.e., L_H and L_A), hash codes and hash centers will be calculated or updated by forward propagation, while at the same time, the parameters (i.e., θ, ξ and *lac*) will be updated by back propagation and gradient descent. The above two branches will continue to alternate until the network converges or reaches the predefined max epochs. More specifically, we summarize the pseudocode of the network training process in Algorithm 1 and you may check our open-sourced code for more details.

Algorithm 1: Optimization for LSCSH

Input:
N: number of samples;
$\mathcal{B}$: batch size;
θ and ξ: parameters of two network branches;
w: label-affinity coefficient (*lac*);
η_*: learning rate;
$optimizer_*$: optimizer.
Initialization:
randomly initialize θ and ξ;
initialize each nonzero element of w_i as $\frac{1}{|L(x_i)|}$.

repeat
  $r = g(v;\xi)$;
  for *j=1 to* $\frac{N}{\mathcal{B}}$ **do**
    $z_j \leftarrow f(x_j;\theta)$;
    Compute L_H using (w, r, z_j);
    $\widehat{w} \leftarrow w - \eta_1 \frac{\partial L_H}{\partial w}$;
    $w \leftarrow Euclidean\_Projection(\widehat{w})$;
    Compute L_H using (w, r, z_j);
    $\delta\theta \leftarrow \partial_\theta L_H$;
    $\theta \leftarrow optimizer_\theta(\theta, \delta\theta, \eta_2)$;
  $z \leftarrow f(x;\theta)$;
  Compute L_A using (w, r, z);
  $\delta\xi \leftarrow \partial_\xi L_A$;
  $\xi \leftarrow optimizer_\xi(\xi, \delta\xi, \eta_3)$;
until *network converges or reaches the max epochs*;
Output: f_θ, g_ξ, w.

IV. Experiments

In this section, we first introduce the experimental settings, then present the experimental results, next perform ablation studies, and finally illustrate the visualization results.

TABLE I
MAP COMPARISONS ON MULTI-LABEL IMAGES WITH RESNET-50

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
ITQ-CCA* [16]	0.505	0.511	0.513	0.514	0.566	0.562	0.530	0.502	0.435	0.435	0.435	0.435
BRE* [1]	0.494	0.503	0.514	0.525	0.592	0.622	0.630	0.634	0.485	0.525	0.544	0.544
KSH* [20]	0.607	0.619	0.621	0.626	0.521	0.534	0.534	0.536	0.394	0.407	0.404	0.399
SDH* [19]	0.562	0.566	0.612	0.627	0.554	0.564	0.572	0.580	0.575	0.590	0.603	0.613
CNNH* [18]	0.618	0.629	0.631	0.642	0.599	0.617	0.624	0.620	0.655	0.659	0.688	0.647
DNNH* [24]	0.621	0.637	0.645	0.653	0.644	0.651	0.647	0.647	0.703	0.738	0.715	0.754
DHN* [2]	0.668	0.683	0.693	0.707	0.719	0.731	0.740	0.745	0.712	0.759	0.758	0.771
HashNet* [17]	0.732	0.739	0.741	0.744	0.745	0.773	0.785	0.788	0.757	0.775	0.781	0.790
IDHN* [4]	0.678	0.699	0.703	0.725	0.713	0.686	0.676	0.586	0.757	0.764	0.769	0.772
DCH* [21]	0.739	0.740	0.748	0.749	0.759	0.801	0.811	0.825	0.773	0.795	0.811	0.818
CSQ [7]	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 [26]	0.751	0.761	0.769	0.777	0.782	0.835	0.850	0.854	0.809	0.823	0.839	0.847
LSCSH	**0.766**	**0.778**	**0.788**	**0.796**	**0.821**	**0.855**	**0.878**	**0.882**	**0.823**	**0.839**	**0.855**	**0.869**

The values of * are from the corresponding references. The other values are re-produced by us under the same settings.

A. Experimental Settings

Datasets: We use 3 multi-label image datasets (i.e., VOC2012 [43], NUS-WIDE [44] and MS-COCO [45]) and 2 single-label image datasets (i.e., ImageNet [46] and CIFAR-10 [47]) to test the performance of LSCSH. Specifically, we utilize all the 20 and 80 categories of VOC2012 and MS-COCO, and choose images from 21 most frequent categories of NUS-WIDE for evaluation. Similar to CSQ [7], we use the 100 most frequent categories from ImageNet and all the CIFAR-10 categories respectively. More details about the dataset splitting are listed in Table II.

Baselines: Following the mainstream methods [7], we compare the performance of LSCSH with 11 state-of-the-art hashing approaches including 4 shallow ones (i.e., SDH [19], ITQ-CCA [16], KSH [20] and BRE [1]) and 8 deep ones (i.e., CNNH [18], DNNH [24], DHN [2], HashNet [17], IDHN [4], DCH [21], CSQ [7] and PSLDH [26]). Our LSCSH adopt the commonly-used ResNet-50 [33] pre-trained on ImageNet to extract the features of the fully connected layer. For comparison fairness, we also re-produce the state-of-the-art methods with the same backbone (i.e., ResNet-50) to calculate their mAP values. The parameters of all comparison methods are selected according to their default settings.

Evaluation metrics: We utilize 3 standard evaluation metrics for image retrieval: Mean Average Precision (mAP), Precision-Recall curves (PR) and Precision curves within Hamming distance 2 (P@H$\leq$2). Note that for fair comparisons, we respectively adopt mAP@1000 for ImageNet, mAP@5000 for MS-COCO and NUS-WIDE, mAP@ALL for CIFAR-10 [47] and VOC2012 [43].

Implementation details: We conduct all the experiments using PyTorch on a linux server with 64-core CPUs, 4 NVIDIA Tesla A100 GPUs, 256 GB memory size and 1,024 TB disk capacity. LSCSH consists of two main branches: *hash code generator* and *hash center adapter*. The former is composed of a ResNet-50/AlexNet backbone pre-trained on ImageNet and a randomly initialized hash layer, where we set the learning rate as 2e-6 to fine-tune the backbone and the learning rate as 5e-4 to learn the hash layer from the scratch. The latter is a fully connected neural network and we set the learning rate as 5e-4 to train this branch. We use Adam optimizer with a batch size of 64 during the training phase. More implementation details are recorded in our open-sourced project: https://github.com/lzHZWZ/LSCSH_sourcecode.git.

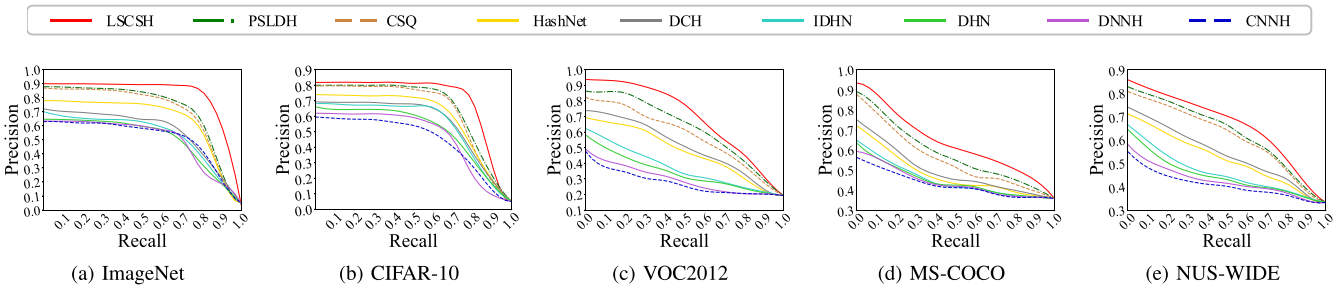
TABLE II
EXPERIMENTAL SETTINGS FOR ALL DATASETS

Dataset	Label Type	#Train	#Test	#Retrieval
VOC2012	multi	4,000	1,000	6,540
MS-COCO	multi	10,000	5,000	112,217
NUS-WIDE	multi	10,000	2,100	149,685
CIFAR-10	single	5,000	1,000	54,000
ImageNet	single	10,000	5,000	128,495

B. Experimental Results

In this section, we first evaluate the performance of LSCSH including mAP, precision and recall compared with the state-of-the-art candidates. In addition, for better revealing the dynamic change of generated hash centers during the model training phase, we also record the mean Hamming distances between hash centers. Finally, we test the model performance under a cross-dataset scenario to verify the good generalization ability of LSCSH. All the evaluation results are described as follows.

mAP comparisons: Tables I and III respectively list the mAP results on multi-label images and single-label images of LSCSH compared with other baselines when using ResNet-50 as the backbone. As shown in Table I, LSCSH achieves the best performance on all cases. Especially, compared with PSLDH, LSCSH promotes the mAP result by at least 1.9%, 2.3% and 1.8% at all hash code length on these 3 multi-label datasets. Furthermore, we analyze the experimental results on single-label images. As shown in Table III, the performance of LSCSH presents a same trend as that in Table I. LSCSH again achieves the optimal performance on ImageNet and CIFAR-10, which averagely



(a) ImageNet (b) CIFAR-10 (c) VOC2012 (d) MS-COCO (e) NUS-WIDE

Fig. 3. The PR on ImageNet, CIFAR-10, VOC2012, MS-COCO and NUS-WIDE at 64 bits.

TABLE III
MAP COMPARISONS ON SINGLE-LABEL IMAGE DATASETS WITH RESNET-50

Method	ImageNet				CIFAR-10			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
ITQ-CCA*	0.226	0.436	0.548	0.576	0.281	0.288	0.295	0.266
BRE*	0.063	0.253	0.330	0.358	0.193	0.193	0.208	0.257
KSH*	0.160	0.298	0.342	0.394	0.298	0.346	0.356	0.359
SDH*	0.299	0.455	0.555	0.585	0.297	0.447	0.559	0.563
CNNH*	0.315	0.473	0.554	0.596	0.499	0.521	0.532	0.545
DNNH*	0.353	0.522	0.530	0.610	0.556	0.558	0.581	0.605
DHN*	0.367	0.523	0.542	0.627	0.590	0.603	0.621	0.674
HashNet*	0.622	0.701	0.725	0.739	0.511	0.628	0.663	0.683
IDHN*	0.645	0.714	0.726	0.741	0.622	0.701	0.711	0.715
DCH*	0.652	0.737	0.745	0.758	0.668	0.694	0.681	0.678
CSQ	0.851	0.861	0.869	0.873	0.691	0.732	0.742	0.748
PSLDH	0.854	0.873	0.881	0.886	0.728	0.741	0.750	0.757
LSCSH	**0.861**	**0.882**	**0.899**	**0.903**	**0.740**	**0.759**	**0.781**	**0.783**

The values of * are from the corresponding references. The other values are re-produced by us under the same settings.

TABLE IV
MAP COMPARISONS ON IMAGENET WITH DIFFERENT BACKBONES

Method	ResNet-50				AlexNet			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
CNNH*	0.315	0.473	0.554	0.596	0.282	0.453	0.465	0.548
DNNH*	0.353	0.522	0.530	0.610	0.303	0.457	0.531	0.572
DHN*	0.367	0.523	0.542	0.627	0.318	0.473	0.542	0.569
HashNet*	0.622	0.701	0.725	0.739	0.506	0.631	0.648	0.684
DCH*	0.652	0.737	0.745	0.758	0.529	0.637	0.642	0.664
CSQ	0.851	0.861	0.869	0.873	0.612	0.643	0.658	0.689
PSLDH	0.854	0.873	0.881	0.886	0.621	0.650	0.671	0.702
LSCSH	**0.861**	**0.882**	**0.899**	**0.903**	**0.625**	**0.663**	**0.682**	**0.723**

The values of * are from the corresponding references. The other values are re-produced by us under the same settings.

boosts the mAP value by 1.5% and 2.8% at all hash code length on single-label images. It's worth mentioning that we add another backbone (i.e., AlexNet) to exclude the dependence of our approach on a certain specific backbone. As shown in Table IV that lists the evaluation results on ImageNet, LSCSH also produces higher mAP values than other candidates even if using a different backbone. This phenomenon reveals that our model can produce better results using different backbones in all cases by constructing semantic hash centers in an asynchronous learning mechanism. All the results in Tables I, III and IV verify the superiority of LSCSH.

Precision and recall comparisons: Fig. 3 shows the PR curves on ImageNet, CIFAR-10, VOC2012, MS-COCO and NUS-WIDE at 64 bits. As we see, the precision of our LSCSH outperforms other candidates under a same recall rate on all datasets. Especially on single-label images, LSCSH achieves accurate and stable precision at a lower recall rate. Although the precision on multi-label images decreases more rapidly as the recall rate increases, LSCSH still obtain a better performance than other methods. This is beacuse our LSCSH can not only push those similar images to approach the same hash center(s), but also greatly separate different hash centers based on the label semantic information, thus yielding distinguishable semantic hash codes.

Precision within Hamming distance 2: We evaluate the precision curves within Hamming distance 2 (P@H$\leq$ 2) at 16 bits, 32 bits, 48 bits and 64 bits on ImageNet, CIFAR-10, VOC2012, MS-COCO and NUS-WIDE in Fig. 4. Note that P@H$\leq$ 2 curves can well reflect the polymerization degree of hash codes in a Hamming ball with radius 2. The results indicate our proposed LSCSH outperforms all other state-of-the-art methods at all code lengths on all datasets. Particularly, owing to that our *Weighted-Softmax Loss* (see (4)) is designed with the intra-calss cohesion as well as inter-class dispersion property, the precision curves on different datasets of LSCSH are relatively smooth than baselines, which also demonstrates owns the ability of producing high-quality and distinguishable hash codes.

Hamming distance between hash centers: Referring to DCH [21], we calculate the mean Hamming distance between continuous hash centers on a single-label dataset (i.e., ImageNet) and a multi-label dataset (i.e., MS-COCO) with the increase of epochs. As shown in Fig. 5, no matter what code length we use, the hash centers will be greatly separated from each other and the mean Hamming distance will constantly approach half of the length within 30 epochs. Compared with CSQ that manually picks hash centers, on the one hand, our self-adaptive scheme can effectively bring high-quality hash centers based on the initial label semantic information at all code lengths on both single-label and multi-label image datasets. On the other hand, each hash code will constantly get close to its corresponding one or more hash centers via the asynchronous learning mechanism. As a result, our model will produce distinguishable semantic hash codes.

Generalization ability: To verify the generalization ability of LSCSH, we test the model performance under a cross-dataset scenario. Specifically, we first train the model on MS-COCO, and then test the retrieval performance on VOC2012. Afterwards, we exchange the roles of MS-COCO and VOC2012 to conduct the same experiment as well. Note that VOC2012 is a subset of MS-COCO in terms of label classes. As shown in

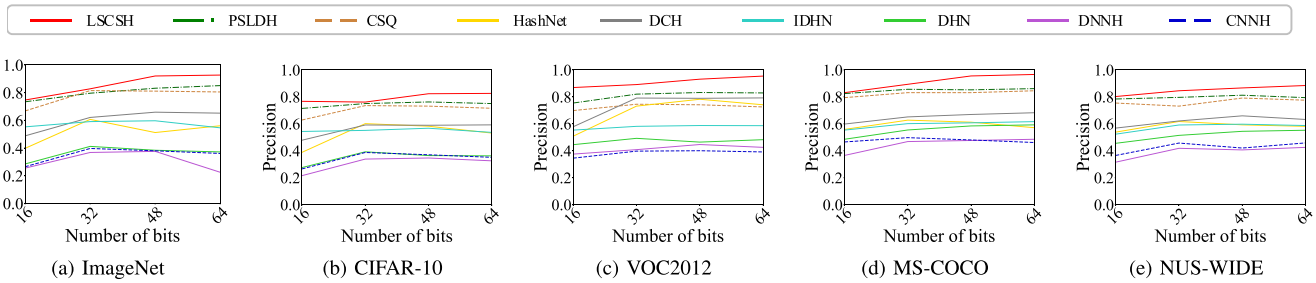


Fig. 4. The P@H≤2 curves at 16 bits, 32 bits, 48 bits and 64 bits on ImageNet, CIFAR-10, VOC2012, MS-COCO and NUS-WIDE.

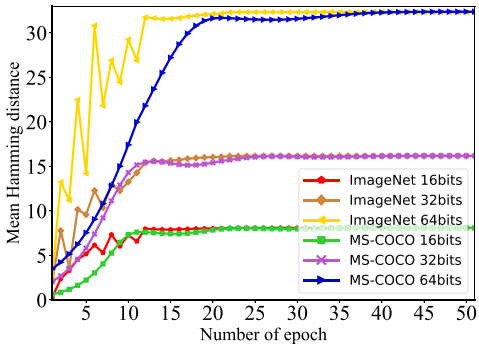


Fig. 5. Mean Hamming distance between hash centers.

TABLE V
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	0.569	0.647	0.683	0.716	0.671	0.685	0.691	0.697
PSLDH	0.599	0.665	0.694	0.721	0.679	0.698	0.703	0.717
LSCSH	**0.635**	**0.679**	**0.715**	**0.742**	**0.689**	**0.723**	**0.725**	**0.735**

All the results of CSQ and PSDH are based on our experiments.

Table V, compared with the lastest state-of-the-art hashing methods PSLDH and CSQ, our LSCSH averagely improve the mAP results by 5.4%, 4.0%, 3.9% and 3.6% at 16 bits, 32 bits, 48 bits and 64 bits respectively. These results indicate that although LSCSH learns the hash centers specific to the class of the training dataset, the over-fit phenomenon will hardly occur on this dataset, which implies our method brings better generalization ability than CSQ and PSLDH.

C. Ablation Studies

In this section, we perform ablation studies to explore the influence of key components and parameters including (i) vision transformer based backbone, (ii) *lac*, (iii) different combinations of loss items, (iv) different word embedding techniques, (v) the scale parameter τ in (4), the weighted factor μ, λ in (7) and the weighted factor α_1, α_2 in (10), and (vi) adaptive hash centers.

Vision transformer based backbone: Inspired by the recent good work VTS [27], we also change the original CNN backbone into a vision transformer (i.e., VTS16) backbone to yield semantic hash codes. As shown in Table VI, compared with the strategy that adopts fixed hash centers (i.e., CSQ+VTS16), we evaluate the mAP results on MS-COCO at 16 bits, 32 bits and 64 bits with our method using dynamic semantic hash centers (i.e., LSCSH+VTS16). On the one hand, CSQ+VST16 obtains limited performance improvement at 16 bits, owing to that in the case of short code lengths with extremely limited representation ability, those hashing methods that consider neither semantic information nor object size cannot generate hash codes with a sufficient degree of differentiation. With the increase of code length, the performance of CSQ+VTS16 also gradually increases, which indicates the stronger feature extraction ability of VTS16 will in some degree make up for the defeat of CSQ that failed to adopt semantic hash centers. On the other hand, even though using the same powerful feature extraction backbone, our LSCSH obtains higher retrieval performance and outperforms the candidate with a significant margin by 9.5%, 7.7% and 3.1% at 16 bits, 32 bits and 64 bits. This phenomenon verifies that our proposed semantic hash centers can further optimize the model no matter what backbone we adopt.

TABLE VI
PERFORMANCE COMPARISONS WITH VTS16 BACKBONE

Method	MS-COCO		
	16 bits	32 bits	64 bits
CSQ+VTS16	0.779	0.874	0.911
LSCSH+VTS16	**0.853**	**0.941**	**0.939**

The values of CSQ+VTS16 are from [27].

Effect of lac: Our proposed *lac* can indicate the scale size of each object, which will greatly affect the hash centroid calculation in the training phase. We investigate the effect of priori *lac* and our learned *lac* on multi-label images, where priori *lac* means its each nonzero element equals a predefined value $\frac{1}{|L(x_i)|}$ if there are $|L(x_i)|$ valid labels in the image x_i (see Section III-B) and learned *lac* is obtained by our model in an asynchronous learning mechanism. As listed in Table VII, our learned *lac* yields better performance than priori *lac* in all cases, because different scales of objects in a multi-label image really affect the feature of each sample. Our scheme consider this issue to calculate a more accurate hash centroid for each sample, thus the learned *lac* can push the hash code of this sample to be closer to its real hash centroid to generate more high-quality hash codes.

Different combinations of loss items: In order to verify the effectiveness of our designed *Weighted-Softmax Loss* in (4) termed as WS), we compare it with *Partial-Softmax* loss (termed as PS) and cross entropy (termed as CE) based loss on ImageNet and MS-COCO during the hash center adapter stage, where PS

TABLE VII
MAP RESULTS WITH PRIORI *LAC* AND LEARNED *LAC*

lac	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
priori *lac*	0.755	0.767	0.774	0.783	0.796	0.841	0.857	0.860	0.813	0.828	0.845	0.856
learned *lac*	**0.766**	**0.778**	**0.788**	**0.796**	**0.821**	**0.855**	**0.878**	**0.882**	**0.823**	**0.839**	**0.855**	**0.869**

TABLE VIII
MAP RESULTS WITH DIFFERENT COMBINATIONS OF LOSS ITEMS

Loss					Dataset							
L			$L_{label-center}$	L_{inter}	ImageNet				MS-COCO			
WS	PS	CE			16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
✓	×	×	✓	×	0.103	0.168	0.194	0.387	0.387	0.516	0.542	0.569
✓	×	×	×	✓	0.841	0.872	0.874	0.880	0.795	0.826	0.847	0.868
×	×	×	✓	✓	0.811	0.861	0.872	0.886	0.788	0.824	0.829	0.840
×	×	✓	✓	✓	0.853	0.871	0.881	0.888	0.792	0.829	0.837	0.851
×	✓	×	✓	✓	0.857	0.879	0.887	0.894	0.802	0.837	0.851	0.863
✓	×	×	✓	✓	**0.861**	**0.882**	**0.899**	**0.903**	**0.821**	**0.855**	**0.878**	**0.882**

loss and CE loss are respectively adopted in PSLDH [26] and CSQ [7]. Note that the hash codes and hash centers are binary values in CSQ, so it can directly use the binary cross entropy loss to measure the distance between each hash code and its corresponding hash center(s). However, considering that the continuous values of hash codes and hash centers in our LSCSH, we first use *tanh* to restrict each element of hash code h_i and its corresponding p_i (i.e., $\sum_{j=1}^{C} w_{ij} r_j$) into interval $(-1, 1)$. Next, we transform $\sum_{j=1}^{C} w_{ij} r_j$ and h_i into probability values (i.e., p_i and q_i) belonging to interval (0,1). Finally, we design the Cross entropy based loss $L_{CE} = -\frac{1}{N}\sum_{i=1}^{N}(p_i \log(q_i) + (1 - p_i)\log(1 - q_i))$ to limit the distance between p_i and q_i, where $p_i = \frac{1}{2}(1 + \sum_{j=1}^{C} w_{ij} r_j)$ and $q_i = \frac{1}{2}(1 + h_i)$.

In Table VIII, We record the model performance of using WS, PS or CE loss combined with our $L_{label-center}$ and L_{inter}. As we see, our model will achieve the highest mAP if using the combination of our proposed WS loss, Label-center loss and Inter-class loss. The performance will decline if we remove anyone from them. The main reason is that they work together to enable separated and distinguishable hash centers that absorb label semantic information. In addition, compared with CE loss adopted by CSQ, our designed WS loss can effectively limit the distance between each hash code and its corresponding hash centroid, so the performance will drop if we replace WS loss with CE loss. Besides, PS loss failed to play a good impact in our LSCSH owing to that it disabled the impact of *lac*. These results show the effectiveness of our design principle on the combinations of three loss items (i.e., the last row of Table VIII). The main reason is that our designed WS loss not only penalizes significantly on the sample whose hash code owns long distance from its target hash centroid but also makes full use of w_{ij} to complete the hash centroid calculation. As a result, our loss items greatly contribute to high-quality hash codes.

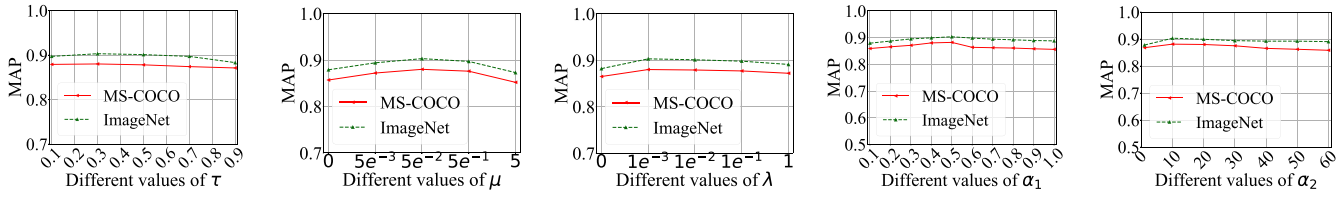
Different word embedding techniques: As mentioned in Section III-C, we evaluate the performance of LSCSH using commonly-used word embedding techniques (including CategoryVector, FastText [38], GloVe [37], GoogleNews [36] and BERT [35]) to extract the word embedding of each label. Note that CategoryVector denotes the ground-truth labels vector of each single-label or multi-label image. We list the mAP results with different word embedding techniques on ImageNet and MS-COCO in Table IX. As we see, CategoryVector brings the lowest mAP than other candidates, owing to that it can hardly retain the semantic correlation between objects/classes. Although other 4 NLP based embedding techniques can achieve better performance, BERT yields highest mAP at all code lengths on these two datasets. The main reason lies in that BERT adopts the latest transformer structure that can better maintain the semantic topology relationships between different label embeddings to obtain separated hash centers.

Scale parameter τ and weighted factor μ, λ, α_1, α_2: As mentioned in (4), τ is the scale parameter of our designed *Weighted-Softmax Loss* to control the size of hypersphere space related to feature representation. As for μ, λ and α_1, α_2, they are weighted factors to respectively adjust the proportion of loss items in (7) and (10). Fig. 6 presents the mAP change with different parameters settings at 64 bits code length on ImageNet and MS-COCO.

First, we record the mAP results by varying τ from 0.1 to 0.9 in Fig. 6(a). Intuitively, τ will influence the distribution density of feature representation located in a hypersphere space. According to our observation, the results indicate that a smaller τ will contribute to better hash codes, but a too small τ may lead to an unconsolidated feature distribution, thus the model cannot effectively capture the information of sample clusters. In our experiment, $\tau = 0.3$ makes a great impact on our designed *Weighted-Softmax Loss* to bring the highest mAP. Second, in order to investigate the influence of μ and λ in (7), we respectively record the performance change with different μ and λ in Fig. 6(b) and (c). According to our setting, μ and λ respectively adjust the scale of *quantization loss* and *Regularization* item, and we find these two hyperparameters with diffent values will

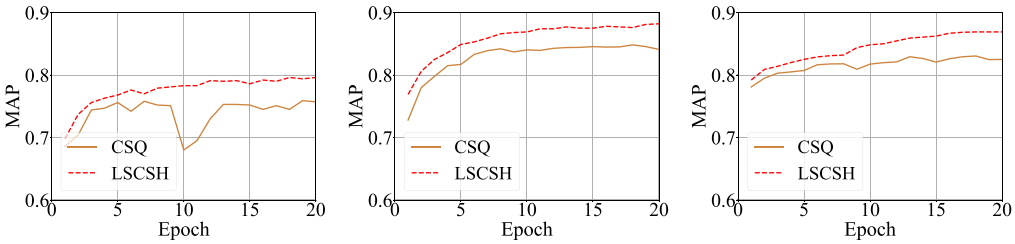
TABLE IX
MAP RESULTS WITH DIFFERENT WORD EMBEDDING TECHNIQUES

Word2Vec	ImageNet				MS-COCO			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
CategoryVector	0.840	0.871	0.875	0.883	0.791	0.823	0.847	0.859
GoogleNews	0.850	0.873	0.884	0.893	0.795	0.835	0.850	0.861
GloVe	0.853	0.879	0.888	0.889	0.804	0.838	0.859	0.868
FastText	0.858	0.880	0.892	0.896	0.811	0.842	0.866	0.872
BERT	**0.861**	**0.882**	**0.899**	**0.903**	**0.821**	**0.855**	**0.878**	**0.882**



(a) mAP with different τ (b) mAP with different μ (c) mAP with different λ (d) mAP with different α_1 (e) mAP with different α_2

Fig. 6. The mAP change with different $\tau, \mu, \lambda, \alpha_1, \alpha_2$ at 64 bits code length on ImageNet and MS-COCO.



(a) VOC2012 (b) MS-COCO (c) NUS-WIDE

Fig. 7. The mAP change with different epochs at 64 bits code length on VOC2012, MS-COCO and NUS-WIDE.

not affect the model performance too much. That is to say, in (7), it is the *Weighted-Softmax Loss* that contributes more to high-quality hash codes. Therefore, we choose $\mu = 0.05$ and $\lambda = 0.001$ when our model achieves the highest mAP. At last, we analyze the effect of α_1 and α_2 in (10). As shown in Fig. 6(d), LSCSH obtains a better mAP when $\alpha_1 = 0.5$ according to our experimental results. Since α_2 controls the scale of *Inter-class* loss that will influence the separation degree of hash centers corresponding to different classes, we alter α_2 in a big range from 1 to 60 for evaluation. From Fig. 6(e), $\alpha_2 = 10$ can lead to the highest mAP. Besides, the model performance will only slightly fluctuate with different settings of α_1 and α_2, which reflects that our method is not sensitive to different values of them. Note that we adopt the control variable method to search the optimal hyperparameter settings, which means we only vary one hyperparameter (i.e., τ, μ, λ, α_1, α_2) and fix others as the values corresponding to best mAP.

Effect of adaptive hash centers: To further illustrate the superiority of adaptive hash centers adopted by LSCSH compared with fixed hash centers in CSQ, we plot the mAP curve with different epochs at 64 bits code length on VOC2012, MS-COCO and NUS-WIDE in Fig. 7. Experimental results show our LSCSH can achieve better mAP than CSQ at each epoch. Specifically, Fig. 7(a) reveals that CSQ has not reached its best mAP (see Table I) when LSCSH has converged within 19 epochs with higher mAP. Additionally, the mAP curve of CSQ is severely fluctuating during the first 20 epochs, while our LSCSH presents a more smoothing and stable performance. From Fig. 7(b) and (c), we can discover the mAP gap of these two methods is small at the beginning of training, but this gap becomes bigger as the epoch increases. Fig. 7(c) also verifies the more stable and higher performance of LSCSH compared with CSQ. The main reason is that the way of constructing adaptive hash centers can better adapt to different data distribution, which greatly alleviates the instability during our model training. Moreover, the asynchronous leaning mechanism can guide each hash codes and hash centroid to adapt to each other in time under a batch-level view. As a result, our LSCSH yields higher performance during its smooth and stable training.

D. Visualization Results

In this section, we evaluate the effect of LSCSH by illustrating the visualization results of CAMs and t-SNE distribution on randomly selected images. Besides, we also present the visualization result of *lac* corresponding to each object scale.

CAMs and t-SNE distribution Visualization: We visualize the CAMs [48] of 3 randomly selected images and t-SNE distribution of hash codes on 10 randomly selected classes (each class

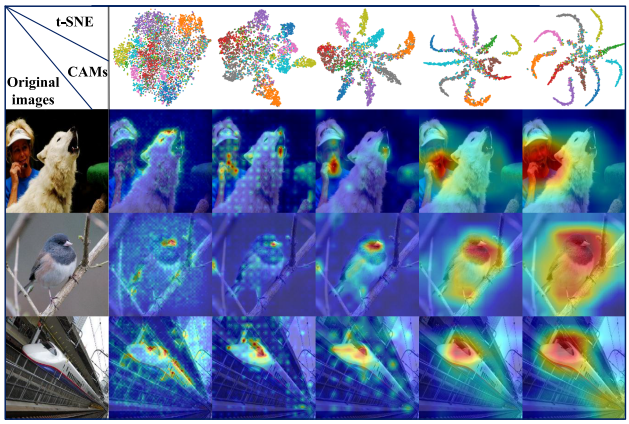


Fig. 8. Visualization of CAMs and t-SNE distribution.

Fig. 9. Visualization of *lac*.

contains 500 samples) from ImageNet. These images are sampled from the train set because our proposed *lac* only exists in the training phase and *lac* will be removed on the inference phase (refer to Section III-B). As shown in Fig. 8, we choose 5 stages from the beginning to the end during our model training, and find LSCSH can gradually correctly highlight the valid regions for input images with model iterations. To our satisfaction, the hash codes become clear and distinguishable to gather in different centers after limited iterations from the initial mess. These visualization results reflect LSCSH can extract accurate image features and generate high-quality hash codes.

Visualization of lac: As mentioned in Section III-B, *lac* only participates in the training phase to indicate each object scale size. We randomly select 3 multi-label images from the training set and list their *lac* visualization results in Fig. 9. The decimals below each image denote the relative scale of an object in this image. For example, the scale ratio between *Dog* and *Chair* is approximately $0.28 : 0.72$ in the leftmost image. In our scheme, we can obtain the final *lac* matrix at the end of model training, of which each row vector corresponds to the *lac* of a sample mentioned in Section III-F. Obviously, the weights in *lac* can accurately reflect the scales of objects (i.e., the weights approximate to the bounding box in the object detection task) appearing in each multi-label image. This effectiveness of *lac* greatly helps calculate the hash centroid (see Section III-D) based on the generated hash centers for each sample.

Retrieval results: In this experiment, we randomly select two query images from single-label ImageNet and another two query images from multi-label MS-COCO to conduct the similarity retrieval. Fig. 10 lists the top-10 returned retrieval results. As we see, on ImageNet, for an image containing "bulbul," the all returned results also contain this object. Besides, the "blood-hound" appears in each returned image for its given query object. As for the multi-label images on MS-COCO, LSCSH further achieves the satisfied retrieval performance, i.e., given those objects in a query sample, at least one object appears in the returned images in our experiment. On the whole, our scheme is able to return the similar images for each given query on both single-multi and multi-label images, which illustrates LSCSH can generate high-quality semantic hash codes.

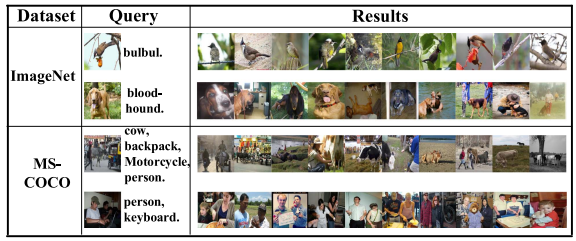


Fig. 10. Top-10 returned retrieval results on ImageNet and MS-COCO.

V. Conclusion

In this article, we propose LSCSH that learns the hash centers based on the label semantic correlation and utilizes *lac* to indicate the scales of different objects in each image. First, we use the commonly-used backbone to extract the features of data samples and design another multi-layer perception as a hash center generator. Second, the hash center generator considers the relationships of label embeddings, which keeps not only the distance of hash centers big enough, but relative distance of them complying with the semantic information of label word. Third, our designed *Weighted-Softmax Loss* can push each hash code to be close to its corresponding hash centroid as well as stay away from the improper hash centers. Last but not least, the asynchronous learning mechanism effectively optimizes our model. The results of extensive experiments on 5 image datasets demonstrate that LSCSH can achieve better image retrieval performance than the state-of-the-art image hashing methods. In addition, we deeply analysis the performance of our proposed LSCSH under different module combination or weighted factors.

In the future, we intend to improve our scheme from the following four aspects. First, we attempt to use other techniques (e.g., graph neural network) to better capture the relationship of data samples or annotations. Second, the *lac* is actually a spare table in memory, so we would like to compress this table to save the memory space for large-scale image datasets. Third, we will further explore to reduce the gap between image features and label embeddings by adopting multi-modal backbones like CLIP to complete both the feature extraction and hash centers generation, so as to yield high-quality hash codes. Four, we may adjust the structure of LSCSH to apply it to more complex tasks like scene image retrieval.

Acknowledgment

This work was also achieved in Key Laboratory of Information Storage System and Ministry of Education of China. We

miss Prof. Fuhao Zou deeply and greatly appreciate his guidance and assistance in this research field.

REFERENCES

[1] B. Kulis and T. Darrell, "Learning to hash with binary reconstructive embeddings," in *Proc. Adv. Neural Inf. Process. Syst. 22: 23rd Annu. Conf. Neural Inf. Process. Syst.*, 2009, pp. 1042–1050.

[2] H. Zhu, M. Long, J. Wang, and Y. Cao, "Deep hashing network for efficient similarity retrieval," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2415–2421.

[3] W. Li, S. Wang, and W. Kang, "Feature learning based deep supervised hashing with pairwise labels," in *Proc. 25th Int. Joint Conf. Artif. Intell.*, 2016, pp. 1711–1717.

[4] Z. Zhang, Q. Zou, Y. Lin, L. Chen, and S. Wang, "Improved deep hashing with soft pairwise similarity for multi-label image retrieval," *IEEE Trans. Multimedia*, vol. 22, no. 2, pp. 540–553, Feb. 2020.

[5] D. Zhai et al., "Parametric local multiview hamming distance metric learning," *Pattern Recognit.*, vol. 75, pp. 250–262, 2018.

[6] M. Norouzi, D. J. Fleet, and R. Salakhutdinov, "Hamming distance metric learning," in *Proc. Adv. Neural Inf. Process. Syst. 25: 26th Annu. Conf. Neural Inf. Process. Syst.*2012, pp. 1070–1078.

[7] L. Yuan et al., "Central similarity quantization for efficient image and video retrieval," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 3080–3089.

[8] Q. Lin, W. Cao, Z. He, and Z. He, "Mask cross-modal hashing networks," *IEEE Trans. Multimedia*, vol. 23, pp. 550–558, 2021.

[9] D. Mandal, P. Rao, and S. Biswas, "Semi-supervised cross-modal retrieval with label prediction," *IEEE Trans. Multimedia*, vol. 22, no. 9, pp. 2345–2353, Sep. 2020.

[10] C. Ma et al., "Toward making unsupervised graph hashing discriminative," *IEEE Trans. Multimedia*, vol. 22, no. 3, pp. 760–774, Mar. 2020.

[11] Y. Wu, S. Wang, and Q. Huang, "Online fast adaptive low-rank similarity learning for cross-modal retrieval," *IEEE Trans. Multimedia*, vol. 22, no. 5, pp. 1310–1322, May 2020.

[12] S. Li, Z. Chen, X. Li, J. Lu, and J. Zhou, "Unsupervised variational video hashing with 1D-CNN-LSTM networks," *IEEE Trans. Multimedia*, vol. 22, no. 6, pp. 1542–1554, Jun. 2020.

[13] P. F. Zhang, Y. Li, Z. Huang, and X. S. Xu, "Aggregation-based graph convolutional hashing for unsupervised cross-modal retrieval," *IEEE Trans. Multimedia*, vol. 24, pp. 466–479, 2022.

[14] H. Zhang et al., "Deep unsupervised self-evolutionary hashing for image retrieval," *IEEE Trans. Multimedia*, vol. 23, pp. 3400–3413, 2021.

[15] Y. Xie et al., "Label-attended hashing for multi-label image retrieval," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, 2020, pp. 955–962.

[16] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. 24th IEEE Conf. Comput. Vis. Pattern Recognit.*, 2011, pp. 817–824.

[17] Z. Cao, M. Long, J. Wang, and P. S. Yu, "HashNet: Deep learning to hash by continuation," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 5609–5618.

[18] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan, "Supervised hashing for image retrieval via image representation learning," in *Proc. 28th AAAI Conf. Artif. Intell.*, 2014, pp. 2156–2162.

[19] F. Shen, C. Shen, W. Liu, and H. T. Shen, "Supervised discrete hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 37–45.

[20] W. Liu, J. Wang, R. Ji, Y. G. Jiang, and S. F. Chang, "Supervised hashing with kernels," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 2074–2081.

[21] Y. Cao, M. Long, B. Liu, and J. Wang, "Deep cauchy hashing for hamming space retrieval," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1229–1237.

[22] X. Zhou, et al., "Graph convolutional network hashing," *IEEE Trans. Cybern.*, vol. 50, no. 4, pp. 1460–1472, Apr. 2020.

[23] X. Shen et al., "Deep co-image-label hashing for multi-label image retrieval," *IEEE Trans. Multimedia*, vol. 24, pp. 1116–1126, 2022.

[24] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3270–3278.

[25] H. Lai, P. Yan, X. Shu, Y. Wei, and S. Yan, "Instance-aware hashing for multi-label image retrieval," *IEEE Trans. Image Process.*, vol. 25, no. 6, pp. 2469–2479, Jun. 2016.

[26] R. Tu, X. Mao, J. Guo, W. Wei, and H. Huang, "Partial-softmax loss based deep hashing," in *Proc. WWW '21: Web Conf. ACM / IW3C2*, 2021, pp. 2869–2878.

[27] S. R. Dubey, S. K. Singh, and W. T. Chu, "Vision transformer hashing for image retrieval," in *Proc. IEEE Int. Conf. Multimedia Expo*., 2022, pp. 1–6.

[28] Y. Cao et al., "Learning to hash with dimension analysis based quantizer for image retrieval," *IEEE Trans. Multimedia*, vol. 23, pp. 3907–3918, 2021.

[29] T. Do, A. Doan, and N. Cheung, "Learning to hash with binary deep neural network," in *Proc. Comput. Vis. - ECCV*, 2016, vol. 9909, pp. 219–234.

[30] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 4, pp. 744–755, Apr. 2014.

[31] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2916–2929, Dec. 2013.

[32] V. E. Liong, J. Lu, G. Wang, P. Moulin, and J. Zhou, "Deep hashing for compact binary codes learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 2475–2483.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. 25: 26th Annu. Conf. Neural Inf. Process. Syst.*2012, pp. 1106–1114.

[35] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol.*, 2019, pp. 4171–4186.

[36] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. 1st Int. Conf. Learn. Representations*, 2013, *arXiv:1301.3781*.

[37] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2014, pp. 1532–1543.

[38] A. Joulin et al., "Fasttextzip: Compressing text classification models," 2016, *arXiv:1612.03651*.

[39] Y. Chen et al., "Long-tail hashing," in *Proc. 44th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2021, pp. 1328–1338.

[40] W. Wang and M. Á. Carreira-Perpiñán, "Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application," 2013, *arXiv:1309.1541*.

[41] Z. Wang, Y. Wang, Z. Wu, J. Lu, and J. Zhou, "Instance similarity learning for unsupervised feature representation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 10336–10345.

[42] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 3733–3742.

[43] M. Everingham et al., "The pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, 2015.

[44] T. Chua et al., "NUS-WIDE: A real-world web image database from national university of Singapore," in *Proc. 8th ACM Int. Conf. Image Video Retrieval*, 2009, Art. no. 48.

[45] T. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Comput. Vis. - ECCV*, 2014, pp. 740–755.

[46] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[47] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep. TR-2009, 2009.

[48] R. R. Selvaraju, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," *Int. J. Comput. Vis.*, vol. 128, no. 2, pp. 336–359, 2020.

Yanzhao Xie received the M.A.Sc. degree in 2019 from the School of Software Engineering, Huazhong University of Science and Technology, Wuhan, China, where he is currently working toward the Ph.D. degree with the Wuhan National Laboratory for Optoelectronics. He has authored or coauthored papers in international conferences and journals including IJCAI, CIKM, ICMR, APWeb-WAIM, and Neural Computing and Applications, Multimedia Tools and Applications. His main research interests include deep learning, image recognition, graph neural network, multimedia content analysis, and reinforcement learning.

Rukai Wei received the B.S. degree in 2022 from the School of software engineering, Huazhong university of science and technology, Wuhan, China, where he is currently working toward the Ph.D. degree with the Wuhan National Laboratory for Optoelectronics. His research interests include information retrieval, deep learning, and machine learning.

Jingkuan Song is currently a Full Professor with the University of Electronic Science and Technology of China. He has authored or coauthored more than 200 peer-reviewed papers, including more than 40 IEEE/ACM Transactions such as TPAMI, IJCV, TIP, and more than 100 Chinese Computing Federation A ranked papers such as ACM MM, NeurIPS, CVPR, and ICLR. His research interests mainly include multimedia analysis and computer vision. He was the recipient of the ACM SIGMM Rising Star Award 2021, Best Paper Award in ICPR in 2016, Mexico, Best Student Paper Award in Australian Database Conference in 2017, Australia, and Best Paper Honorable Mention Award in 2017, Japan. He also an Associate Editor for IEEE TMM, ACM TOMM, and the Guest Editor of TMM, WWWJ, and PR.

Yu Liu received the Ph.D. degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2017. From 2018 to 2021, he was a Postdoctoral Researcher with HUST. He is currently an Associate Researcher with the School of Computer Science and Technology, HUST. He has authored or coauthored some papers in international journals and conferences, including ACM MM, IJCAI, SIGMOD, DAC, IEEE TRANSACTIONS ON CYBERNETICS, CIKM, ICME, ICCD, and FGCS. His research interests include building the cognitive storage systems with machine learning technologies and large-scale multimedia search technologies.

Yangtao Wang received the Ph.D. degree in 2021 from the Huazhong University of Science and Technology, Wuhan, China, where he is currently a Full-Time Lecturer with the School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou, China. He has published papers in international conferences and journals including CIKM, ICMR, IJCAI, Pattern Recognition, and WWWJ. His main research interests include deep learning and image recognition.

Ke Zhou (Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer science and technology in 1996, 1999, and 2003, respectively, from the Huazhong University of Science and Technology, Wuhan, China, where he is currently a Professor. He has more than 100 publications in journals and international conferences, including SIGMOD, FAST, ATC, DAC, ACM MM, TOS, and TPDS. His main research interests include computer architecture, mass data storage, and management.