IEEE Transactions on Cybernetics

Deep Incremental Hashing for Semantic Image Retrieval with Concept Drift

Journal:	*IEEE Transactions on Cybernetics*
Manuscript ID	CYB-E-2022-01-0038
Manuscript Type:	Regular Paper
Date Submitted by the Author:	05-Jan-2022
Complete List of Authors:	Xu, Huihui; South China University of Technology, School of Computer Science & Engineering Tian, Xing; South China University of Technology, School of Computer Science and Engineering Ng, Wing; South China University of Technology, School of Computer Science and Engineering
Keywords:	Image Retrieval, Concept Drift, Incremental Hashing, Non-Stationary Environment, Deep Hashing

Deep Incremental Hashing for Semantic Image Retrieval with Concept Drift

Hui-Hui Xu, Xing Tian*, *Member, IEEE*, Wing W. Y. Ng*, *Senior Member, IEEE*

***Abstract*—Hashing methods are widely used for content-based image retrieval due to their attractive time and space efficiencies. Several dynamic hashing methods have been proposed for image retrieval tasks in non-stationary environments. However, concept drift problems in non-stationary data scenarios are seldomly considered which lead to significant deterioration of performance. Therefore, we propose Deep Incremental Hashing (DIH). For the learning part, similarity-preserving goal codes of each newly arriving data chunk are computed using the product of its label matrix and a random Gaussian matrix generated offline. A point-wise loss function is then devised to guide the learning of a deep hash neural network. To retain the learned knowledge of former chunks, a weighting-based method is utilized to combine different hash tables trained at different time steps to form a multi-table hashing system. Experimental results on 13 simulated concept drift environments show that DIH adapts to non-stationary data environments well and yields better retrieval performance than existing dynamic hashing methods.**

***Index Terms*—Image Retrieval, Concept Drift, Incremental Hashing, Non-Stationary Environment, Deep Hashing.**

I. Introduction

LARGE amounts of multimedia data are uploaded to the Internet every day. Due to the expensive manual annotations and individual subjectivity, automatic feature extraction and content-based image retrieval (CBIR) have gained growing attention to handle massive data streams. To search nearest neighbours for query images, tree-based retrieval methods [1], [2] are proposed to index image features. However, retrieval performances of these methods deteriorate when facing high-dimensional data. Hashing-based methods [3] aim to find an embedding from the high-dimensional feature space to the Hamming space with the original semantic neighbourhood structure preserved. Each image is hashed to a short binary code. During retrieval, a bulk of hash codes can be loaded to memory occupying rather small space. Moreover, the similarity search can be done by computing the Hamming distance between query codes and codes of images in the database using very efficient XOR operations. With the original neighbourhood structure preserved in Hamming space, similar images can be returned by sorting Hamming distances between hash codes of images in the database and query images.

Existing hashing methods are generally designed under the assumption of stationary data environments. However, real-world environments is non-stationary with data chunks appearing sequentially. This dynamical condition of data environments often comes with concept drift problems [4]. Data of never-seen classes may even emerge. Moreover, new observations of already seen classes may also appear in the newest data chunk [5]. In other words, the distribution of a semantic class may change along with time. Both of these two phenomena belong to concept drifts. Without being updated, a static retrieval system trained based on the previous data cannot adapt to the new data environment, which leads to the deterioration of performance of the retrieval system. To handle the retrieval task in non-stationary environments with concept drift, a hashing system should not only adapt to new data environment but also retain former learned knowledge. An intuitive method is to retrain the current model based on all previously arriving images. However, the time and computation cost will increase with the growing number of arriving images, which is unrealistic to real applications. Several dynamic hashing methods [6]–[8] are proposed to learn hashing system in non-stationary data environments where data comes in forms of streams or sequential chunks. Though they train hash functions in dynamic setting, most of them ignore the intrinsic dynamical property of online scenarios, i.e. concept drift. Incremental hashing [6] is the first method to handle concept drift problems using multiple hash tables and a weighting scheme. However, most existing dynamic hashing methods devise a shallow classifier and take feature vectors as input. Therefore, retrieval performances of these methods rely heavily on the quality of image features and they show limited capability to distinguish targeted high-level semantic information from images. Recently, a number of deep hashing methods devise deep neural networks to extract high-level distinctive features from images and output hash codes in an end-to-end manner, which show better performance than traditional methods. Hence, we propose the Deep Incremental Hashing (DIH) in this paper, which cannot only achieve satisfactory retrieval accuracy, but also adapt to non-stationary environments where concept drift happens.

Taking advantage of the good representation learning ability of deep neural networks, DIH performs feature extraction and hash codes learning simultaneously. The goal code of each data is computed using its label vector and a random Gaussian matrix, which preserves the pair-wise semantic similarity information of data. With these computed goal codes, a deep

This work was supported in part by the National Natural Science Foundation of China under Grant 61876066, in part by the 67th China Postdoctoral Science Foundation under Grant 2020M672631, and in part by Guangdong Province Science and Technology Plan Project (Collaborative Innovation and Platform Environment Construction) 2019A050510006. (Corresponding Authors: Xing Tian, and Wing W. Y. Ng)

Hui-Hui Xu, Xing Tian, and Wing W. Y. Ng are with the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China and also with Brain and Affective Cognitive Research Center, Pazhou Lab, Guangzhou, 510335, China (e-mail: xuhui481@gmail.com, xingtian@scut.edu.cn, wingng@ieee.org).

neural network is trained to learn to map images to hash codes. This network can be efficiently trained by a point-wise loss function, which reduces computation cost compared with the pair-wise manner or triplet-wise manner. In non-stationary environments, the newest neural network is attained based solely on the current data chunk and the most recently updated network. Besides, a multi-table hashing system is used to retain former knowledge and new data distribution information simultaneously. A weighting scheme is devised to assign different weights to hash tables in the framework, which enhances retrieval performance efficiently.

Main contributions of this paper are as follows:

- DIH is the first end-to-end supervised incremental hashing method to deal with concept drift problems in non-stationary data environments for image retrieval. In contrast to existing dynamic hashing methods, DIH incorporates feature extraction and hash code learning under non-stationary settings, which captures high-level descriptive features from images and generates discriminative hash codes.
- The learning of DIH is guided by similarity-preserving goal codes with a computation efficient point-wise loss function. Moreover, multiple hash tables trained at different time steps are employed in the proposed hashing system. An adaptive weighting scheme is also utilized to compute weighted Hamming distance to further improve retrieval performance.
- Experimental results on 13 simulated concept drift scenarios show that DIH yields much better retrieval accuracy than other dynamic hashing methods.

The rest of this paper is organized as follows. In Section II, current hashing methods are reviewed. In Section III, the proposed DIH is described in detail. Experimental results are presented and analyzed in Section IV. In Section V, we conclude this work.

II. Related Works

Existing hashing methods can be categorized into stationary and non-stationary hashing methods, which are briefly described in Section II-A and Section II-B, respectively.

A. Stationary Hashing Methods

In stationary data environments, data distributions of different classes are static. All training data is available at the beginning. The hashing model is expected to generalize well on out-of-sample data of the to-be-learned data distribution.

Existing stationary hashing methods can be categorized into data-independent and data-dependent hashing methods. Locality-sensitive hashing (LSH) [3] and its variants [9], [10] are classical data-independent hashing methods, which directly generate random hash functions from a Gaussian distribution. However, they need longer codes to yield satisfactory performance which require higher costs in both computation and storage. Several data-dependent hashing methods are proposed to solve this problem. Iterative Quantization (ITQ) [11] is one of representative data dependent hashing methods, which aims to decrease the quantization error by iteratively finding an optimal rotation matrix. Spectral Hashing [12] formulates the hashing code learning to a graph partitioning problem and uses a spectral method to relax the discrete optimization. Anchor Graph Hashing [13] uses anchors to automatically estimate the similarity structure among data. Instead of using hyper-planes to partition the input space, Spherical Hashing [14] devises hyper-spheres to generate efficient partitions. Aforemential methods aim at approximating the similarity distance in original data space. Ordinal Constraint Hashing (OCH) [15], Ordinal Embedding Hashing [16] and Probability Ordinal-preserving Semantic Hashing (POSH) [17] are proposed to approximate ordinal relations in the original data space to guide the learning process. EC-IO [18] is proposed to simultaneously encode class-level and instance-level information based on distribution preservation and error correction. Aforementioned methods are based on supervised setting, Semi-supervised hashing (SSH) [19] uses labeled data to fit similarity structure and uses unlabelled data to mitigate overfitting.

Recently, deep learning is introduced to hashing and shows better results than above traditional hashing methods. Using an end-end manner, deep neural networks (DNNs) extract more descriptive high-level semantic features from images. Lots of stationary hashing methods based on DNNs are proposed. Asymmetric Deep Supervised Hashing (ADSH) [20] uses two asymmetric hash functions for query images and database images. Hash codes of database images and the network parameters are efficiently learned using an iterative alternating algorithm. An adversarial strategy for semi-supervised hashing [21] is proposed to fully utilize the unlabelled data. HashNet [22], Deep pairwise-supervised hashing (DPSH) [23], Deep Supervised Hashing (DSH) [24], and Deep Hashing Network (DHN) [25] use pair-wise loss functions to train DNNs. In contrast, Deep Supervised Hashing using Triplet Labels (DTSH) [26], and Network in Network Hashing (NINH) [27] use triplet-wise loss functions. Graph Convolutional Network Hashing (GCNH) [28] devises Graph Convolution Networks (GCNs) to generate effective hash codes from affinity graph. Hashing with Mutual Information (MIHash) [29] proposes a mutual information metric for hashing. Using a query-database symmetry, MIHash aims to maximize the distance between the distribution of similar points and the distribution of dissimilar points for query data. Deep Multi-View Enhancement Hashing (D-MVE-Hash) [30] is proposed to exploit the multi-view information to generate more effective hash codes . Deep Reinforcement Learning for Image Hashing (DRLIH) [31] introduces reinforcement learning into hashing and uses recurrent neural networks (RNNs) to model the correlation between hashing functions. Deep Self-Taught Hashing (DSTH) [32] devises deep features from pretrained networks to generate pseudo labels for unsupervised learning of hash codes. Most of these methods are proposed for stationary data environments. Deep Incremental Hashing Network (DIHN) [33] incrementally learns hash codes when one data chunk with new classes comes and does not update hash codes of database images. However, DIHN is not aiming at handling data streams and requires learned binary codes before incremental learning, which does not accord with assumptions of this work.

B. Non-stationary Hashing Methods

For real-world applications, the data generally come in a streaming manner and are unavailable in advance. Several dynamic hashing methods are proposed. Incremental hashing (ICH) [6] trains a hash table in a semi-supervised manner and combines different hash tables trained at different time steps to boost the performance. Online Sketch Hashing (SketchHash) [7] uses data sketching to reduce the training overhead. Hadamard Codebook guided Online Hashing (HMOH) [8] uses Hadamard matrix as codebook and updates model parameters using kernelized perceptron algorithm. Adaptive Hashing (AdaptHash) [34] updates model parameters by stochastic gradient decent in a boosting manner when new data chunks come. Using the pair-wise similarity information, online hashing (OKH) [35] designs a loss function with the idea of PA strategy. Online Supervised Hashing (OSH) [36] uses a randomly generated codebook and assigns each column as the target hash code of one appearing class during learning. Mutual Information Hashing (MIHash) [37] introduces a new metric inspired by mutual information. The model parameters are updated to maximize the distance between two approximated distributions in reservoir pool with a sampling method. Incremental Hash-bit Learning (IBL) [38] is a semi-supervised online hashing method and generates a hash table by assessing all ever-trained hash functions in the hash function pool according to three metrics. Online Selection Hashing (OSelH) [39] computes goal codes in a bit-wise manner to better adapt to characteristics of data and designs an evaluating method to choose better hash functions. In addition to using similarity information, Similarity-Preserving Linkage Hashing (SPLH) [40] investigates the inter-class relationship to learn efficient codes. The method proposed in [41] fixes the hash functions and efficiently updates proposed projection functions, which are used to transform the mapped binary codes in the same Hamming space. Performances of existing dynamic hashing methods depend heavily on the quality of inputted features. Moreover, most of them do not consider concept drift problems. When new classes emerge or the data distribution changes over time, the retrieval performance of these methods will deteriorate.

III. Deep Incremental Hashing

To handle aforementioned issues, we propose Deep Incremental Hashing (DIH) which utilizes similarity-preserving goal codes to guide the learning of the network. Essentially, the training of the newest network is based only on the current data chunk and the parameters of the most recently updated network. To better retain former knowledge and adapt to the current data environment, DIH devises a multi-table hashing system with an adaptive weighting scheme to combine multiple hash tables trained at different time steps.

In non-stationary environments, not all data is available in advance and concept drift problems may happen. Without loss of generality, we assume that a new data chunk with n images arrives at each time step. Let $D^t = (X^t, Y^t)$ denote the new data chunk arriving at the time step t ($t = 1, 2, 3, ...$), in which $X^t = \{x_i^t\}_{i=1}^n$ and $Y^t = \{y_i^t\}_{i=1}^n \in \{0,1\}^{c^t \times n}$ denote the set of images in D^t and the label matrix of D^t, respectively. y_i^t denotes the one-hot label vector of image x_i^t with length c^t. The set of all existing classes until the time step t is denoted as $L^t = \{l_j\}_{j=1}^{c^t}$. The j^{th} element in y_i^t equals to 1 if x_i^t belongs to the class l_j, 0 otherwise. Considering that images of new classes may emerge in non-stationary data environments, the value of c^t, i.e. the cardinality of L^t, may change over time. The database for retrieval at the time step t consists of all existing data chunks, i.e. $\{X^i\}_{i=1}^t$.

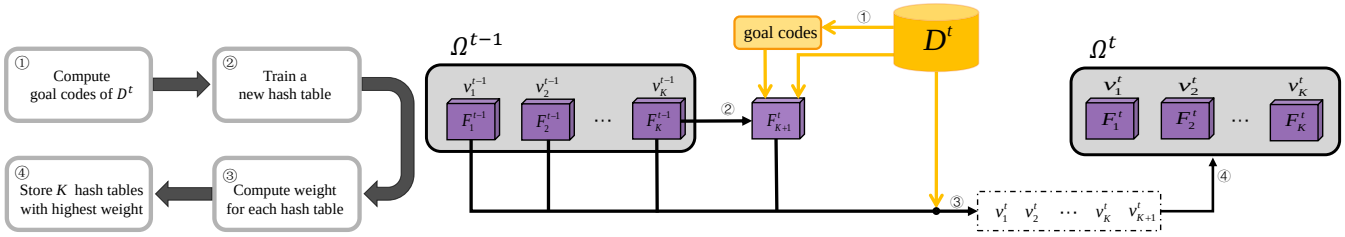
The retrieval system of DIH at the time step t is a hash table pool $\Omega^t = \{(F_i^t, v_i^t)\}_{i=1}^K$, in which F_i^t, v_i^t and K denote the i^{th} hash table in Ω^t, the weight of the i^{th} hash table, and the number of hash tables employed in Ω^t, respectively. The weight value for each hash table falls in the range $[0, 1]$, i.e. $v_i^t \in [0, 1]$. Each hash table corresponds to a deep hash neural network trained at different time step. At the time step $t = 1$, goal codes of images in D^t are firstly computed. Then, the hash table F_1^1 is trained based on the data chunk D^1. Considering that only one data chunk exists and the amount of data maybe not enough for training, the network of F_1^1 is initialized with parameters of a pretrained CNN network [5], [42]. Guided by goal codes of the newly arriving data chunk, the hash table F_1^1 is trained using a point-wise loss function. There is only one hash table existing in the hash table pool, so $v_1^1 = 1$. Based on output logits of the corresponding network of F_1^1, hash codes of both database images and query images are computed. Then, Hamming distances between hash codes of the query image and images stored in the database are computed and ranked. Images with the smallest Hamming distance to the query image are returned. For any two hash codes $b_1, b_2 \in \{-1, 1\}^r$, where r is the hash code length, their Hamming distance $\mathcal{H}(b_1, b_2)$ is computed as follows:

$$\mathcal{H}(b_1, b_2) = \frac{1}{2}\|b_1 - b_2\|_1 = \frac{1}{2}(r - b_1 \cdot b_2). \tag{1}$$

At the time step $t>1$, goal codes of the newly arriving data chunk D^t are firstly computed. A new deep hash network is trained based solely on D^t, with parameters initialized as the most recently trained network. After training, this newly trained network is added into the hash table pool of DIH. Weights of all hash tables are computed to evaluate their adaptability to the current data environment. For efficiency, only K hash tables with highest weights are stored over time. The weighted Hamming distances between query and images in the database are computed to get final retrieval results, which will be given in Section III-C. The overview and framework of the updating procedure in DIH at the time step t are shown in Fig. 1

A. Goal Code Computing

Hashing using a codebook has been recently introduced in [8], [36]. In [36], each column of the ECOCs is treated as the goal code of each class. In [8], each column of the Hadamard matrix is treated as the goal code of each class. A random matrix generated from the Gaussian distribution is further utilized to make the method applicable to the case where the number of classes is not equal to the size of the Hadamard matrix. With goal codes well preserving the original



(a) Overview (b) Framework

Fig. 1: Overview and Framework of the updating procedure in DIH at the time step t

neighbourhood structure, the classifier is trained efficiently using kernelized perceptron algorithm.

In DIH, goal codes of the newly arriving data chunk are computed firstly at each time step using a random Gaussian matrix and the label matrix. To attain the random Gaussian matrix, a vector $\gamma_i \in \mathcal{R}^r$ is generated for each newly appearing class l_i in L^t at first. Each element of the vector is sampled from the standard Gaussian distribution, i.e. $N(0,1)$. This vector is bound with the newly appearing class and does not change over time. In others words, the corresponding random Gaussian vector for each seen class is generated at the first time step they appear and do not change after. Let Γ^t denote the set of vectors generated for all existing classes, i.e. $\Gamma^t = \{\gamma_i\}_{i=1}^{c^t}$. For an image x^t and its label vector $y^t \in \{0,1\}^{c^t}$, its goal code $g_{x^t} \in \{-1,1\}^r$ for training the deep hash network is computed as follows:

$$g_{x^t} = \mathcal{F}^t(y^t) = sgn(W^t y^t), \tag{2}$$

where $\mathcal{F}^t$ and $W^t \in R^{r \times c^t}$ denote the goal code mapping function and the random Gaussian matrix utilized at the time step t, respectively. The random Gaussian matrix W^t is built by concatenating all vectors in Γ^t, i.e.

$$W^t = [\gamma_1, \gamma_2, ..., \gamma_{c^t}]. \tag{3}$$

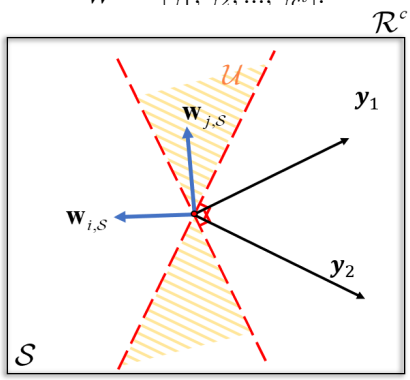


Fig. 2: Illustration of the proof for Lemma 1

1) Effectiveness proof of goal codes to preserve the intra-chunk similarity relationship: A probabilistic proof of the property of goal codes to preserve the semantic similarity relationship inside each newly arriving data chunk is given. For simplicity, the time symbol t associated with all notations is ignored to make the proof clear.

Lemma 1 For any two images x_1, x_2 and their label vectors $y_1, y_2 \in \{0,1\}^c$ in the newly arriving data chunk,

$$E[\mathcal{H}(g_{x_1}, g_{x_2})] = \frac{r}{\pi} arccos(\frac{y_1 \cdot y_2}{\|y_1\|_2 \times \|y_2\|_2}). \tag{4}$$

Without loss of generality, we suppose that y_1 and y_2 are not collinear. In other words, x_1 and x_2 belong to different set of semantic classes. According to Eq.1, the Hamming distance between their goal codes can be demonstrated as:

$$\begin{aligned} \mathcal{H}(g_{x_1}, g_{x_2}) &= \frac{1}{2}\|\mathcal{F}(y_1) - \mathcal{F}(y_2)\|_1 \\ &= \frac{1}{2}\|sgn(Wy_1) - sgn(Wy_2)\|_1. \end{aligned} \tag{5}$$

Let $w_i = [W_{i1}, W_{i2}, ...W_{ic}]^T$ denote the i^{th} row vector in W. As shown in Fig. 2, $\mathcal{S}$ denotes the 2D plane in $\mathcal{R}^c$ spanned by y_1 and y_2. $\mathcal{U}$ denotes the area demarcated by each orthogonal line of y_1 and y_2 on $\mathcal{S}$. $w_{i,\mathcal{S}}$ denotes the projection vector of w_i on $\mathcal{S}$. It should be noted that the value of the i^{th} element in $\mathcal{F}(y_1)$ or $\mathcal{F}(y_2)$ only depends on the sign of the dot product of $w_i \cdot y_1$ or $w_i \cdot y_2$. Then, if $w_i \cdot y_1$ and $w_i \cdot y_2$ are both positive or negative, the absolute value of the i^{th} element in $\mathcal{F}(y_1) - \mathcal{F}(y_2)$ will becomes 0, 2 otherwise. Therefore, the contribution to $\|\mathcal{F}(y_1) - \mathcal{F}(y_2)\|_1$ only comes from those w_i, where the two dot products $w_i \cdot y_1$, $w_i \cdot y_2$ have different signs. This is equivalent to the case where $w_{i,\mathcal{S}}$ is inside $\mathcal{U}$. Reversely, for $w_{j,\mathcal{S}}$ shown in Fig. 2 which is outside $\mathcal{U}$, the contribution to $\|\mathcal{F}(y_1) - \mathcal{F}(y_2)\|_1$ is zero due to $sgn(w_j \cdot y_1) - sgn(w_j \cdot y_2) = 0$. Therefore, the Hamming distance between these two goal codes can be further demonstrated as follows:

$$\begin{aligned} \mathcal{H}(g_{x_1}, g_{x_2}) &= \frac{1}{2}(\mathcal{V}_1 + \mathcal{V}_2 + ... + \mathcal{V}_r), \text{where} \\ \mathcal{V}_i &= \begin{cases} 0 & \text{if } w_{i,\mathcal{S}} \text{ is outside } \mathcal{U} \\ 2 & \text{if } w_{i,\mathcal{S}} \text{ is inside } \mathcal{U}. \end{cases} \end{aligned} \tag{6}$$

The projected vector is isotropic on $\mathcal{S}$ because the projection of a Gaussian vector remains Gaussian. Therefore, we have:

$$\begin{aligned} p(w_{i,\mathcal{S}} \text{ is inside } \mathcal{U}) &= \frac{\theta_{y_1 y_2}}{\pi}, \\ E(\mathcal{V}_i) &= 2\frac{\theta_{y_1 y_2}}{\pi}. \end{aligned} \tag{7}$$

Finally, we get the proof and the expectation value of the Hamming distance of two goal codes is:

$$\begin{aligned} E[\mathcal{H}(g_{x_1}, g_{x_2})] &= \frac{1}{2}\sum_{i=1}^{r} E(\mathcal{V}_i) \\ &= \frac{r}{\pi} arccos(\frac{y_1 \cdot y_2}{\|y_1\|_2 \times \|y_2\|_2}). \end{aligned} \tag{8}$$

For the case where samples in the data chunk have only one label, the angular distance between label vectors of two data points is either 0 or $\frac{\pi}{2}$. Thus, we have:

$$E[\mathcal{H}(g_{x_1}, g_{x_2})] = \begin{cases} 0 & \text{if } y_1 \cdot y_2 = 1 \\ r/2 & \text{if } y_1 \cdot y_2 = 0. \end{cases} \tag{9}$$

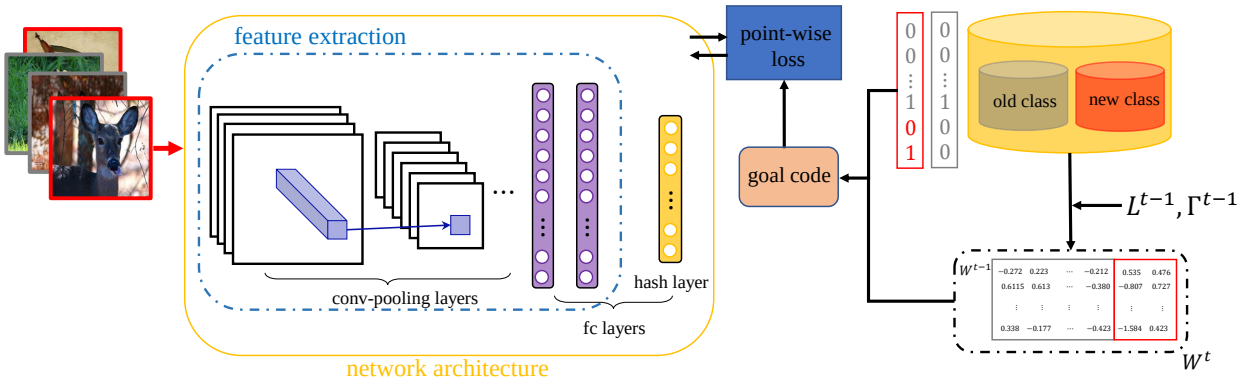


Fig. 3: The training procedure of one hash table at the time step $t > 1$ in DIH

Therefore, goal codes of two images computed by Eq.2 have large Hamming distance if these two images belong to different classes. Their Hamming distance is 0 if these two images belong to the same class. For the case where samples in the data chunk have multiple labels, the angular distance between label vectors of two data points falls in the range $[0, \frac{\pi}{2}]$. Two goal codes have large Hamming distance if these two images have no common annotations ($y_1 \cdot y_2 = 0$). At the meantime, their Hamming distance is small if their label vectors have large intersection.

According to Lemma 1, these goal codes can effectively preserve the intra-chunk pair-wise semantic similarity relationship in the original label space for both single-label and multi-label data chunks. Goal codes of the newly arriving data chunk can be treated as the expected outputs of the deep hash neural network and guide the network to capture the semantic similarity relationship inside the newly arriving data chunk.

B. Network Training for A Hash Table

As shown in Fig. 3, the network architecture of DIH is composed of feature extraction layers and hash layer. At the time step $t > K$, the network of F^t_{K+1} is initialized with parameters of F^{t-1}_K, i.e. the most recently trained network. Only images of the newly arriving data chunk are used for the updating procedure.

Owing to the similarity-preserving property of goal codes, the network is trained to directly fit these goal codes. Because the thresholding procedure of sign function makes the network optimization infeasible, continuous $sigmod$ or $tanh$ function is commonly used for relaxation. However, these activation functions would slow down the network learning and affect the convergence of the network [42]. Therefore, the mean squared error loss of the hash layer outputs and goal codes is used in DIH to make outputs of the network approach discrete bits. Let b^{t*}_i denote the hash layer output of x^t_i during training. The point-wise loss function is devised:

$$\mathcal{L}(x^t_i, y^t_i, F^t_{K+1}) = \|b^{t*}_i - g_{x^t_i}\|^2_2. \tag{10}$$

Empirically, with the same number of iterations, we found that the proposed learning scheme can help the network converge faster than the pair-wise manner [24] and the triplet-wise manner [43] . Besides, the computation cost of the proposed scheme is less than the other two. The mining of triplets or the computation of similarity graph would slow down the network training, which is important in non-stationary environments. In the proposed scheme, computation of the mean squared loss is linear to the batch size. Goal codes of the data and the random Gaussian matrix can be generated offline.

After training F^t_{K+1}, weights for all $K+1$ hash tables are computed. The hash table with the smallest weight is deleted from the hash table pool, while the newly trained hash table is added. After updating the hash table pool, hash codes of D^t are computed based on all existing K hash tables in the hash table pool. Since the database for retrieval at the time step t is composed of all existing data chunks [6], [38], [44], the j^{th} image in the newly arriving data chunk will become the $[n(t-1)+j]^{th}$ image in the database, i.e. $x_{n(t-1)+j}$. Its hash code based on the i^{th} hash table in Ω^t stored in the database is computed as follows:

$$b_{n(t-1)+j,i} = F^t_i(x_{n(t-1)+j}) = sgn(b^*_{n(t-1)+j}). \tag{11}$$

C. Multi-Table Hashing System in DIH

Weights of hash tables in the hash table pool are used for both the deletion of the out-of-date hash table and the ensemble of multiple hash tables during retrieval. After training a new hash table, this new hash table will be added to current hash table pool. Since the newly arriving data chunk represents the current data environment, all existing hash tables are evaluated based on D^t to quantify their adaptability to the current data environment. The evaluation standard is based on the common idea that hash codes of two similar data should have a small Hamming distance while hash codes of two dissimilar data should have a large Hamming distance. Since the evaluation procedure is carried out at each time step, in this section, we ignore the superscript t to make our description clear. Let x_j, x_k denote two images of the newly arriving data chunk. $\mathcal{H}_i(x_j, x_k)$ denotes the Hamming distance of their hash codes based on the i^{th} hash table in the current hash table pool, i.e.

$$\mathcal{H}_i(x_j, x_k) = \mathcal{H}(b_{j,i}, b_{k,i}) = \mathcal{H}[F_i(x_j), F_i(x_k)]. \tag{12}$$

S denotes the pair-wise similarity matrix of the newly arriving data chunk, which is computed as follows:

$$S = f(Y^T Y), \text{where } f(x) = \begin{cases} 1 & \text{if } x>0 \\ 0 & \text{if } x = 0. \end{cases} \tag{13}$$

$d^+_{j,i}$ denotes the mean Hamming distance of $b_{j,i}$ to hash codes of the data based on F_i where $S_{jk} = 1$. $d^-_{j,i}$ denotes the mean Hamming distance of $b_{j,i}$ to hash codes of the data based on F_i where $S_{jk} = 0$. Therefore, we have:

$$d^+_{j,i} = \frac{1}{\|S_{j*}\|_1 - 1} \sum_{S_{jk}=1, k\neq j} \mathcal{H}_i(x_j, x_k), \tag{14}$$

$$d^-_{j,i} = \frac{1}{n - \|S_{j*}\|_1} \sum_{S_{jk}=0} \mathcal{H}_i(x_j, x_k). \tag{15}$$

The evaluation standard of the adaptability of one hash table to the current data environment is defined as $d^-_{j,i} - d^+_{j,i}$, which has a large value only if F_i adapts well to the current data environment. Therefore, the adaptability value of F_i is computed as follows:

$$v_i = \frac{1}{n} \sum_{j=1}^{n} (d^-_{j,i} - d^+_{j,i}). \tag{16}$$

According to Eq.16, adaptability value of all existing hash tables in the hash table pool are computed.

Due to the time and space efficiency, only K hash tables are stored in the hash table pool of DIH. Thus, the hash table with the smallest weight is discarded from the current hash table pool. Then, softmax normalization is devised to attain the final weight of each hash table:

$$v_i = \frac{e^{v_i}}{\sum_{j=1}^{K} e^{v_j}}. \tag{17}$$

For the multi-table hashing system in DIH, the weighted Hamming distance is computed and ranked. According to Eq.17 and Eq.12, the weighted Hamming distance between the query image x_q and the candidate image x_j is computed as follows:

$$\mathcal{H}_w(x_j, x_q) = \sum_{i=1}^{K} v_i \times \mathcal{H}_i(x_j, x_q). \tag{18}$$

Similar database images with the smallest weighted Hamming distances are returned.

We summarize the proposed DIH in Algorithm 1.

Algorithm 1 Deep Incremental Hashing at $t > K$

Input: newly arriving data chunk $D^t = \{x^t_i, y^t_i\}^n_{i=1}$, hash table pool Ω^{t-1}, set of all seen classes $L^{t-1} = \{l_i\}^{c^{t-1}}_{i=1}$, set of Gaussian vectors generated for each seen class $\Gamma^{t-1} = \{\gamma_i\}^{c^{t-1}}_{i=1}$.
Output: Ω^t, L^t, Γ^t.

1. **for** each newly appearing class l in D^t **do**
2. Generate γ from standard Gaussian distribution;
3. Append l, γ to L^{t-1}, Γ^{t-1}, respectively;
4. **end for**
5. $\Gamma^t = \Gamma^{t-1}$; $L^t = L^{t-1}$;
6. Compute goals codes for D^t according to Eq.2;
7. Train a new hash table F^t_{K+1} using Eq.10;
8. Append F^t_{K+1} to Ω^{t-1};
9. **for** $i = 1$ to $K + 1$ **do**
10. Compute v^t_i according to Eq.16;
11. **end for**
12. Find the index of the hash table with the smallest adaptability value: $j = \arg\min_{1\leq j\leq K} v^t_j$;
13. Pop (F^t_j, v^t_j) from Ω^{t-1};
14. Normalize $\{v^t_i\}^K_{i=1}$ according to Eq.17;
15. $\Omega^t = \Omega^{t-1}$;

IV. Experimental Studies

To evaluate DIH in handling concept drift problems, the non-stationary data environments are simulated utilizing existing stationary datasets. We mainly focus on the new classes emerging problem and the distribution drifting problem. As shown in Table I and Table II, 7 scenarios are designed for the new classes emerging problem and 3 scenarios are designed for the distribution drifting problem. As shown in Table III, another 3 scenarios are designed for non-stationary scenarios involving both two problems.

A. Experiment Settings

1) Datasets and Simulation Settings: MNIST [45], CIFAR10 [46], and NUSWIDE [47] are three common stationary datasets, which are utilized to simulate the non-stationary data environments involving the new classes emerging problem. The MNIST dataset has 70,000 28×28 handwritten digit images belonging to 10 classes. Each class has 7,000 images. The CIFAR10 dataset has 60,000 $32 \times 32 \times 3$ RGB images. Each image belongs to one of ten classes. Each class has 6,000 images. As shown in Table I, 6 scenarios, i.e. (1) MNIST_6_4, (2) MNIST_5_5, (3) MNIST_4_6, (4) CIFAR10_6_4, (5) CIFAR10_5_5, and (6) CIFAR10_4_6, are designed for the new classes emerging problem. In these simulations, the size of each data chunk is 2,000 and there are 20 data chunks in total. The NUSWIDE dataset is a multi-label dataset with 269,648 RGB images in total. Each image belongs to some of 81 classes and has different image size. As shown in Table I, NUSWIDE_20_61 is designed for the new classes emerging problem in multi-label data stream. Images sampled from time step $t = 1$ to $t = 10$ are guaranteed not to be annotated with those 61 new classes appearing later. The size of each data chunk is 3,000 and there are 20 data chunks in total.

CIFAR100 dataset [46] is utilized to simulate the distribution drifting problem. The CIFAR100 dataset has 60,000 $32 \times 32 \times 3$ RGB images, each of which belongs to one of 100 sub-classes, i.e. fine label, and one of 20 superclasses, i.e. coarse label. Each superclass includes 5 sub-classes. Each superclass has 6,000 images in total. As shown in Table II, 3 scenarios, i.e. (1) CIFAR100_20_20, (2) CIFAR100_20_15, and (3) CIFAR100_20_10, are designed for the distribution drifting problem. There are 20 data chunks in total, each of which has 2,000 images. The appearing ratio of each sub-class inside a superclass is sampled from the Gaussian distribution.

TABLE I: SETTINGS OF EXPERIMENTS FOR NEW CLASSES EMERGING

Dataset	$1 \leq t \leq 5$ ($1 \leq t \leq 10$ for NUSWIDE)	$6 \leq t \leq 20$ ($11 \leq t \leq 20$ for NUSWIDE)	Scenario name
MNIST	Images randomly selected from 6 original classes	Images randomly selected from 10 classes	MNIST_6_4
	Images randomly selected from 5 original classes	Same as above	MNIST_5_5
	Images randomly selected from 4 original classes	Same as above	MNIST_4_6
CIFAR10	Images randomly selected from 6 original classes	Images randomly selected from 10 classes	CIFAR10_6_4
	Images randomly selected from 5 original classes	Same as above	CIFAR10_5_5
	Images randomly selected from 4 original classes	Same as above	CIFAR10_4_6
NUSWIDE	Images randomly selected from 20 original classes	Images randomly selected from 81 classes	NUSWIDE_20_61

TABLE II: SETTINGS OF EXPERIMENTS FOR DISTRIBUTION DRIFTING

Dataset	$1 \leq t \leq 20$	Scenario name
CIFAR100	20 drifting superclasses	CIFAR100_20_20
	15 drifting superclasses, 5 static superclasses	CIFAR100_20_15
	10 drifting superclasses, 10 static superclasses	CIFAR100_20_10

TABLE III: SETTINGS OF EXPERIMENTS FOR COMBINED NON-STATIONARY ENVIRONMENT

Dataset	$1 \leq t \leq 20$	$21 \leq t \leq 40$	Scenario name
CIFAR100	5 static superclasses, 5 drifting superclasses	10 new superclasses appear	CIFAR100_DN
	15 static superclasses, 5 new static superclasses appear since $t = 6$	10 out of 15 original static superclasses start drifting	CIFAR100_ND
	5 static superclasses, 5 drifting superclasses, 10 new static superclasses appear since $t = 6$	Experiment ends	CIFAR100_D&N

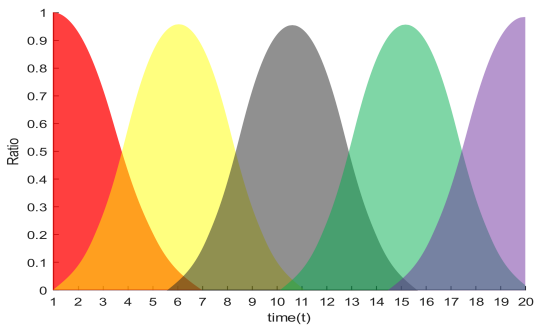


Fig. 4: Appearing ratio of 5 sub-classes for a drifting superclass

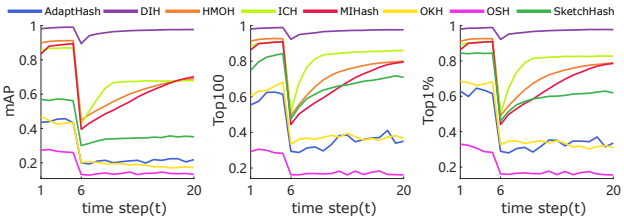
For instance, when superclass "large natural outdoor scenes" is chosen to be a drifting superclass, the appearing ratio of 5 sub-classes, i.e. Sea, Cloud, Forest, Mountain, and Field, at each time step, will change dynamically as shown in Fig. 4. For a static superclass, the appearing ratio of each 5 sub-class at each time step is equal.

We also designed 3 scenarios to evaluate DIH and the other compared methods in more complicated non-stationary data environments involving both two concept drift problems. As shown in Table III, 3 scenarios, i.e. (1) CIFAR100_DN, (2) CIFAR100_ND, and (3) CIFAR100_D&N, are designed. For CIFAR100_DN and CIFAR100_ND, there are 40 data chunks, each of which has 1,000 images. For CIFAR100_D&N, there are 20 data chunks, each of which has 1,500 images.

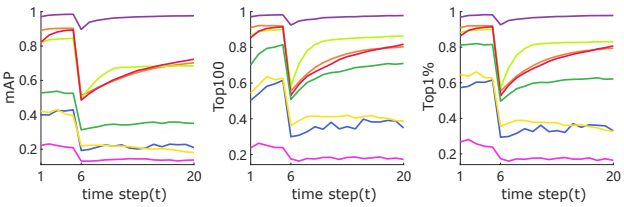
To evaluate all methods at each time step, a test dataset is firstly sampled before the simulation runs. For MNIST and CIFAR10, 100 images of each class are sampled from the whole dataset to form the test dataset. For CIFAR100, 50 images of each sub-class are sampled to form the test dataset. The test chunk at each time step is generated according to the appearing ratio of each class (sub-class) of the corresponding newly arriving data chunk. Due to the complicated inter-class relationship in NUSWIDE, 1,000 images are sampled only involving the original 20 classes as the test chunk for the evaluation at $t \leq 10$ and another 3,000 images are sampled involving all 81 classes for the evaluation at $t \geq 11$.

2) Metrics: Three different metrics are devised to evaluate each method. Mean Average Precision (mAP) is the most comprehensive metric to measure the performance of a hashing retrieval system. It is computed from the view of the retrieval performance on the whole database. Top 100 Precision (Top10) is to evaluate the retrieval precision of the first 100 returned images. It is useful when applications are expected to return a fixed number of images from the database. However, for incremental setting, Top 100 Precision tends to increase due to the increasing size of the database during retrieval. Therefore, the Top 1% Precision (Top1%) [6] is used to count the retrieval precision of the first 1% of the database size returned images.

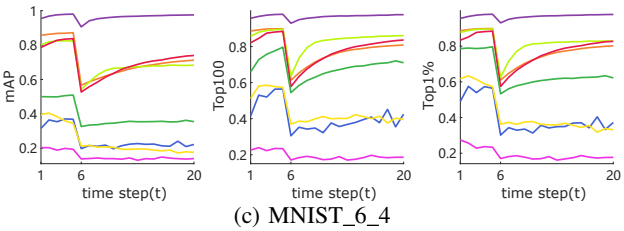
3) Comparative Methods and Parameter Settings: DIH is compared with existing dynamic hashing methods. They are ICH [6], SketchHash [7], HMOH [8], AdaptHash [34], OKH [35], OSH [36], and MIHash [37]. For MNIST, the original 28×28 image and the flattened 784-dim vector are used as input for DIH and other methods, respectively. A simple network architecture was found to perform better on MNIST than very deep network architectures. Therefore, the network architecture used in scenarios based on MNIST is the one proposed in DSH [24]. For CIFAR10, CIFAR100, and NUSWIDE, DIH uses the pretrained VGG16 [42] network architecture and resizes all images to $224 \times 224 \times 3$ as input. Considering that existing dynamic hashing methods are not based on deep hash neural networks and inputs of these methods are feature vectors, extracted 4096-dim features from the fc7 layer of VGG16 pretrained on ImageNet [48] are



(a) MNIST_4_6

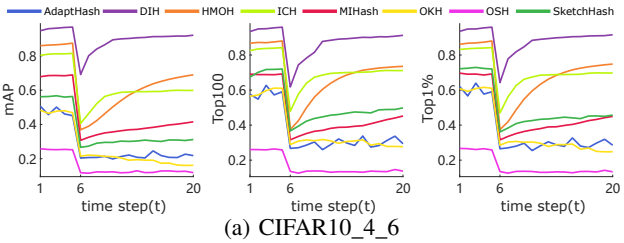


(b) MNIST_5_5

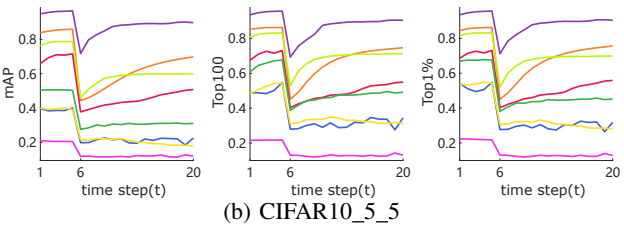


(c) MNIST_6_4

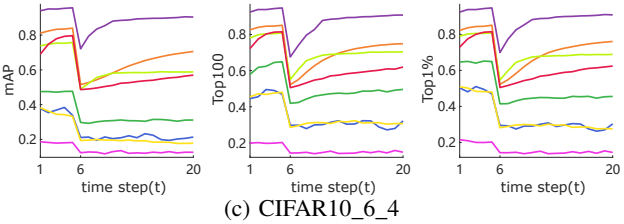
Fig. 5: Results on MNIST



(a) CIFAR10_4_6

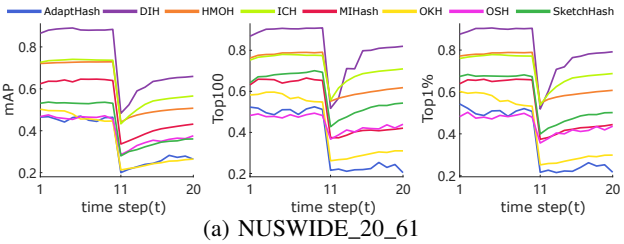


(b) CIFAR10_5_5



(c) CIFAR10_6_4

Fig. 6: Results on CIFAR10



(a) NUSWIDE_20_61

Fig. 7: Results on NUSWIDE

employed as their inputs.

Parameters of other compared methods are adjusted according to the corresponding papers or officially released MATLAB codes. The hash code length is set to 32 bits for all methods. For DIH, the SGD optimizer is used, where the weight decay is set to $1e^{-2}$ and the momentum is set to 0.9. The learning rate is set to $1e^{-5}$, $1e^{-4}$, $5e^{-4}$, and $5e^{-4}$ respectively for MNIST, CIFAR10, CIFAR100, and NUSWIDE. The mini-batch size is set to 50 for all datasets. The number of hash tables in the hash table pool is set to 5 in all experiments. Data augmentation is used to avoid the effect of overfitting.

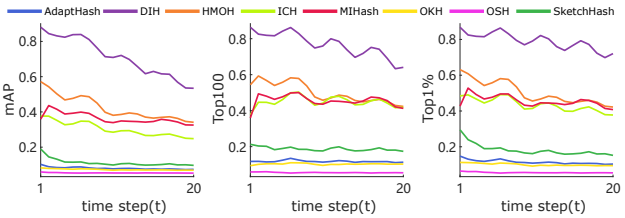
All experiments are repeated 5 times and results are averaged. In Sectioin IV-B, all results on 13 scenarios are demonstrated and analyzed. The averaged value of mAP, Top 100 precision and Top 1% precision on all time steps of all scenarios are shown in Table IV, Table V, and Table VI, respectively. In Section IV-C, ablation study of DIH is described to show the effectiveness of the proposed weighting scheme and multi-table system. In Section IV-D, multiple experiments using different values of the parameter K and r are carried out. The results are shown and the sensitivity of parameters in DIH is analyzed.

B. Results and Analysis

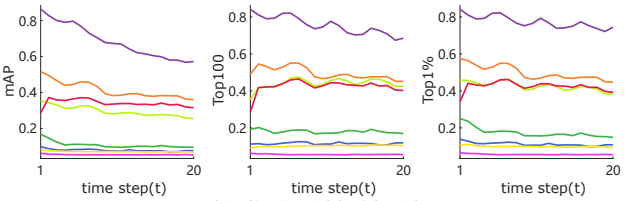
1) Results on New Classes Emerging Scenarios: All methods are run on seven scenarios to evaluate their performance of handling the new classes emerging problem. We assume that the more new classes appear, the more difficult will the scenario be for all methods. The mAP, Top100 precision, and Top1% precision along with time steps on all scenarios are plotted. Experimental results based on MNIST, CIFAR10, and NUSWIDE are shown in Fig. 5, Fig. 6, and Fig. 7, respectively.

On all 7 data scenarios, DIH attains better retrieval accuracy than the other compared methods by a large margin. According to Table IV, in terms of the mAP metric, compared with other laudable methods including HMOH, MIHash, and ICH, DIH outperforms them by 37.4%, 53.9%, and 37.6%, respectively. In terms of the Top100 metric, DIH outperforms them by 23.6%, 43.6%, and 18.6%, respectively. In terms of the Top1% metric, DIH outperforms them by 24.7%, 44.2%, and 20.9%, respectively. On CIFAR10_6_4, CIFAR10_5_5, CIFAR10_4_6, and NUSWIDE_20_61, although the other traditional methods use extracted deep features as their inputs, the classifier part of DIH can generate more effective hash codes in an end-to-end manner through finetuning a pretrained network in non-stationary environments. On MNIST_6_4, MNIST_5_5 and MNIST_4_6, all metrics are close to the upper bound, which shows that the network can efficiently adapt to new classes emerging scenarios on MNIST.

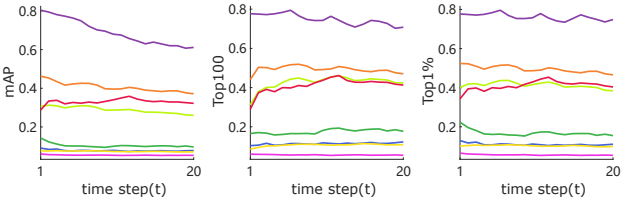
When new classes first emerge at the time step $t = 6$ (11 for the NUSWIDE_20_61 scenario), retrieval performances of all methods drop by a large value. Hash functions should be updated to learn an effective partition for the newly emerging semantic classes. For other single-table hashing methods, hash codes of images in the database are updated whenever



(a) CIFAR100_20_20

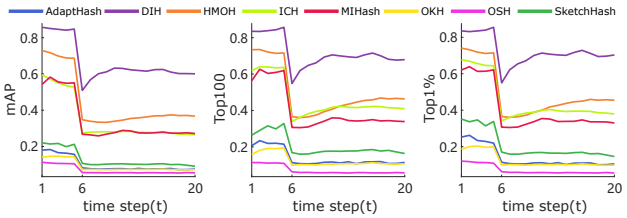


(b) CIFAR100_20_15

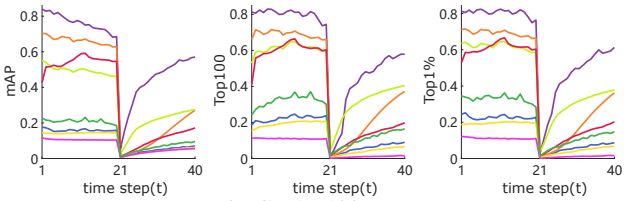


(c) CIFAR100_20_10

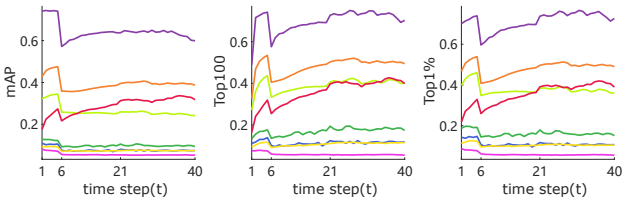
Fig. 8: Results of the distribution drifting problem on CIFAR100



(a) CIFAR100_D&N



(b) CIFAR100_DN



(c) CIFAR100_ND

Fig. 9: Results of combined scenarios on CIFAR100

the projection matrix changes. However, the newly learned distribution may not accord with former data distributions. Though the new distribution is captured, they may forget the former distribution and cannot generate effective hash codes for the former data. By contrast, a multi-table hashing system with the adaptive weighting scheme is devised in DIH to store the former knowledge. DIH can achieve better performance when new classes first emerge.

With the number of newly emerging classes increasing, performances of most methods tend to drop more. However, no prominent drop is shown for DIH. This shows that DIH can adapt better to complicated scenarios than the other methods. Also devising a multi-table system, performance of ICH drops less than most methods but bigger than DIH. This shows that the proposed adaptive weighting scheme in DIH can generate a more effective ensemble of multiple hash tables. DIH is more adaptive than the other comparative methods to non-stationary environments.

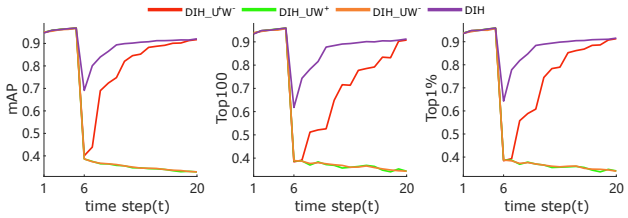
When it comes to the rate of recovery of different methods, DIH is visually the fastest method to recover after new classes first appear and has satisfactory retrieval performance again. For HMOH and MIHash, they recover much slower than DIH and cannot have the same or competitive retrieval performance as the one before new classes emerge. This shows that DIH can quickly adapt to the new data environment.

2) Experiments with Distribution Drifting: In experiments simulating the distribution drifting problem using CIFAR100, the coarse label of each data is used as its truth label to train and evaluate all methods. Experimental results on CIFAR100_20_20, CIFAR100_20_15, and CIFAR100_20_10 are shown in Fig. 8a, Fig. 8b, and Fig. 8c, respectively.

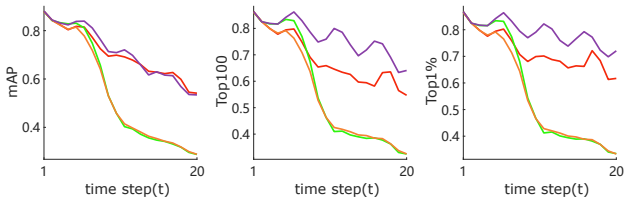
In all three experiments, performances of DIH, MIHash, HMOH, and ICH show much better results than the other four compared methods. The other four methods may fail to learn the complicated semantic information inside each data chunk and cannot handle the distribution drifting problem.

When the number of drifting classes increases, there is visually more fluctuation along with time steps on three metric curves. In terms of the mAP metric, performances of DIH, HMOH, and ICH tend to drop slowly along with time. The performance of MIHash first goes up and then drops slowly. Performances of the best four methods increase around time steps such as $t = 5$, $t = 10$, and $t = 15$. This is because that the ratio of the newly appearing sub-class achieves maximum inside a superclass and large amounts of images of this sub-class are added to the database. After that time, the major appearing sub-class gradually change to another unseen one. Thus the performance of the hashing system drops.

3) Experiments for Combined Non-stationary Scenarios: Results on three combined scenarios are shown in Fig. 9a, Fig. 9b, and Fig. 9c, respectively. On all three scenarios, DIH shows much better performance than the other comparative methods. On CIFAR100_DN, there are 5 drifting superclasses at time steps $t < 21$, performances of most methods fluctuate due to the distribution drifting. At $t = 21$, performances of all methods drop sharply due to the vanishing of old classes and the emerging of new classes. At time steps $t > 21$, DIH recovers faster than any other methods and finally attains satisfactory performance. On CIFAR100_D&N, performances of all methods drop at $t = 6$ due to the appearance of new classes. Only the performance of DIH shows the tendency to increase after new classes appear. On CIFAR100_ND, though performances of DIH, MIHash, and HMOH all show tendency



(a) CIFAR10_4_6



(b) CIFAR100_20_20

Fig. 10: Results of ablation study

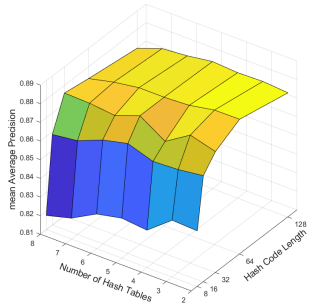
to increase after new classes appear, DIH attains the best retrieval performance. These results demonstrate that DIH can efficiently adapt to non-stationary environments and attain satisfactory retrieval performance.

C. Ablation Study

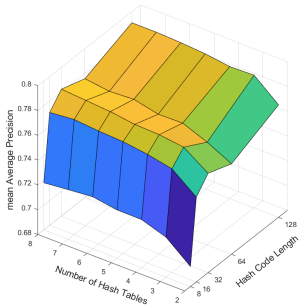
To further show the effectiveness of the proposed multi-table hashing system, DIH is compared with other three versions. For DIH_U$^+$W$^-$, a new hash table is trained at each time step. Equal weights are assigned to all hash tables in the hash table pool during retrieval. For DIH_U$^-$W$^+$, a new hash table is trained at each time step when $t \leq K$. At $t > K$, these K hash tables are not updated and used for retrieval. Throughout all time steps, the proposed weighting scheme is devised to treat existing hash tables adaptively. For DIH_U$^-$W$^-$, the updating procedure is the same to DIH_U$^-$W$^+$. However, the adaptive weight scheme is not devised. Equal weights are assigned to the existing hash tables in the hash table pool.

These three versions and DIH are evaluated on two representative concept drift scenarios, i.e. CIFAR10_4_6 and CIFAR10_20_20. $K = 5$ and $r = 32$ are set for all methods. Retrieval performances along with time steps are shown in Fig. 10a and Fig. 10b. Averaged metric values on 20 time steps are shown in Table VII.

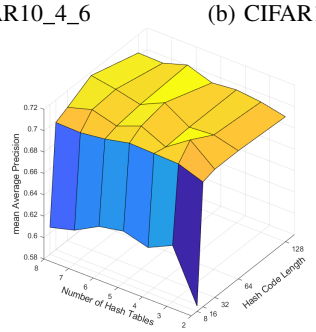
According to Fig. 10 and Table VII, DIH_U$^-$W$^-$ and DIH_U$^-$W$^+$ attain much worse performance than the other two methods. This is because that hash tables are not updated in the simulated non-stationary environments. The static retrieval system cannot handle queries from the newly appearing concepts. Compared with DIH, DIH_U$^+$W$^-$ attains much worse performance on both new classes emerging scenario and the distribution drifting scenario. This shows that the proposed weighting scheme can effectively assess the adaptability of a hash table to the current data environment. Multiple hash tables trained at different time steps can be combined in an adaptive manner to achieve better retrieval performance in non-stationary environments.



(a) CIFAR10_4_6



(b) CIFAR100_20_20



(c) CIFAR100_DN

Fig. 11: Results of experiments with different setting of number of employed hash tables and hash code length

D. Parameter Selection of K And r

With different combinations of number of employed hash tables K and hash code length r, 7×5 experiments on CIFAR10_4_6, CIFAR100_20_20, and CIFAR100_D&N are carried out. Other parameters are the same among all 35 experiments. Results on three scenarios are shown in Fig. 11a, Fig. 11b, and Fig. 11c, where the z-axis represents the averaged value of mAP on all 20 time steps.

As shown in Fig. 11, the retrieval performance of DIH drops prominently when a small hash codes length is used, such as $r = 8$. According to Eq.6, with a small number of binomial variables, the Hamming distance between two goal codes would have a large difference from the expectation value. By contrast, the distance could well approximate around the expectation value when there are more binomial variables, such as $r = 128$. Besides, the performances of $r = 16$, $r = 32$, and $r = 64$ have relatively small differences. Considering the storage and computation efficiency, the choice of 16 or 32bits is satisfactory.

According to Fig. 11b, the retrieval performance would increase when more hash tables are deployed. However, the case is not the same for scenarios involving the new classes emerging problem. No prominent growing trend is shown on CIFAR10_4_6 and CIFAR100_D&N. One reason is that, though having small values, much non-negative weights would be assigned to old hash tables in the hash table pool when K is large. The retrieval system may not satisfactorily handle queries belonging to the newly emerging classes. For this problem, a threshold weight value could be devised. Those out-of-date hash tables having weights smaller than the threshold value could be dropped promptly at early time steps. Empirically, considering scenarios involving both two concept

TABLE IV: MEAN AVERAGE PRECISION RESULTS ON 13 SCENARIOS

Methods	MNIST			CIFAR10			NUSWIDE	CIFAR100					
	6_4	5_5	4_6	6_4	5_5	4_6	20_61	20_20	20_15	20_10	ND	DN	D&N
AdaptHash	0.252	0.265	0.269	0.248	0.257	0.279	0.353	0.080	0.079	0.078	0.079	0.102	0.099
HMOH	0.710	0.697	0.676	0.676	0.659	0.640	0.606	0.423	0.414	0.406	0.394	0.393	0.443
ICH	0.697	0.698	0.695	0.616	0.628	0.626	0.631	0.303	0.295	0.289	0.264	0.348	0.346
MIHash	0.704	0.694	0.647	0.587	0.510	0.449	0.517	0.363	0.340	0.330	0.289	0.322	0.344
OKH	0.241	0.258	0.250	0.230	0.251	0.263	0.357	0.073	0.072	0.074	0.076	0.091	0.091
OSH	0.155	0.158	0.169	0.140	0.141	0.155	0.403	0.050	0.054	0.054	0.057	0.072	0.068
SketchHash	0.388	0.390	0.397	0.349	0.354	0.365	0.432	0.112	0.108	0.104	0.101	0.135	0.129
DIH	**0.964**	**0.966**	**0.969**	**0.890**	**0.889**	**0.899**	**0.745**	**0.712**	**0.683**	**0.689**	**0.644**	**0.597**	**0.666**

TABLE V: TOP 100 PRECISION RESULTS ON 13 SCENARIOS

Methods	MNIST			CIFAR10			NUSWIDE	CIFAR100					
	6_4	5_5	4_6	6_4	5_5	4_6	20_61	20_20	20_15	20_10	ND	DN	D&N
AdaptHash	0.397	0.409	0.407	0.344	0.352	0.367	0.366	0.118	0.116	0.114	0.114	0.139	0.139
HMOH	0.788	0.780	0.767	0.720	0.710	0.694	0.687	0.500	0.495	0.493	0.481	0.435	0.498
ICH	0.838	0.835	0.832	0.707	0.718	0.714	0.721	0.449	0.440	0.427	0.391	0.457	0.461
MIHash	0.785	0.765	0.720	0.626	0.543	0.464	0.527	0.455	0.424	0.411	0.356	0.361	0.404
OKH	0.440	0.454	0.437	0.349	0.374	0.372	0.431	0.103	0.103	0.107	0.109	0.117	0.124
OSH	0.195	0.195	0.200	0.160	0.150	0.167	0.447	0.056	0.057	0.057	0.060	0.059	0.071
SketchHash	0.684	0.686	0.688	0.508	0.511	0.518	0.592	0.188	0.181	0.175	0.170	0.214	0.205
DIH	**0.967**	**0.969**	**0.970**	**0.879**	**0.882**	**0.878**	**0.818**	**0.767**	**0.754**	**0.747**	**0.699**	**0.606**	**0.716**

TABLE VI: TOP 1% PRECISION RESULTS ON 13 SCENARIOS

Methods	MNIST			CIFAR10			NUSWIDE	CIFAR100					
	6_4	5_5	4_6	6_4	5_5	4_6	20_61	20_20	20_15	20_10	ND	DN	D&N
AdaptHash	0.398	0.401	0.399	0.340	0.357	0.364	0.370	0.115	0.113	0.110	0.111	0.141	0.140
HMOH	0.781	0.770	0.756	0.721	0.708	0.693	0.684	0.504	0.498	0.495	0.482	0.428	0.494
ICH	0.821	0.818	0.813	0.699	0.710	0.704	0.709	0.435	0.426	0.413	0.379	0.449	0.453
MIHash	0.778	0.758	0.713	0.627	0.545	0.466	0.536	0.455	0.424	0.410	0.355	0.365	0.405
OKH	0.418	0.433	0.416	0.339	0.364	0.363	0.424	0.100	0.099	0.103	0.106	0.119	0.125
OSH	0.193	0.193	0.199	0.159	0.149	0.166	0.446	0.051	0.046	0.041	0.060	0.061	0.071
SketchHash	0.651	0.642	0.637	0.495	0.497	0.503	0.571	0.182	0.174	0.168	0.162	0.213	0.206
DIH	**0.967**	**0.970**	**0.971**	**0.886**	**0.888**	**0.887**	**0.810**	**0.790**	**0.773**	**0.762**	**0.713**	**0.623**	**0.723**

TABLE VII: RETRIEVAL PERFORMANCE OF DIFFERENT VERSIONS OF DEEP INCREMENTAL HASHING

Methods	CIFAR10_4_6			CIFAR100_20_20		
	mAP	Top100	Top1%	mAP	Top100	Top1%
DIH_U$^-$W$^-$	0.502	0.510	0.507	0.539	0.549	0.553
DIH_U$^-$W$^+$	0.502	0.511	0.507	0.543	0.560	0.564
DIH_U$^+$W$^-$	0.831	0.749	0.788	0.705	0.683	0.720
DIH	**0.899**	**0.878**	**0.887**	**0.712**	**0.767**	**0.790**

drifting problems, the choice $K = 5$ can give sound results.

V. Conclusions

In this paper, considering concept drift problems in real-world applications, we propose the Deep Incremental Hashing (DIH) to better adapt to non-stationary data environments. Parameters of the deep hash neural network are updated using a point-wise loss function guided by similarity-preserving goal codes of the current data chunk. A multi-table system with an adaptive weighting scheme is proposed to simultaneously retain the old knowledge and learn the new knowledge effectively.

DIH is a supervised algorithm and goal codes must be computed using the annotated label information, which is costly in reality. One of the future works is how to combine labelled data and unlabelled data to get a semi-supervised algorithm which is more suitable for reality. Moreover, the computation of goal codes should also take the inter-class similarity relationship into consideration, which can be taken as the prior knowledge for learning. Hash codes of two similar classes should have a smaller Hamming distance compared to that between two dissimilar classes. An approach to adaptively handling the complicated inter-class semantic relationship is further required.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant 61876066, in part by the 67th China Postdoctoral Science Foundation under Grant 2020M672631, and in part by Guangdong Province Science and Technology Plan Project (Collaborative Innovation and Platform Environment Construction) 2019A050510006.

References

[1] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[2] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 97–104.

[3] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*, 2004, pp. 253–262.

[4] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[5] J. He, R. Mao, Z. Shao, and F. Zhu, "Incremental learning in online scenario," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[6] W. W. Ng, X. Tian, Y. Lv, D. S. Yeung, and W. Pedrycz, "Incremental hashing for semantic image retrieval in nonstationary environments," *IEEE transactions on cybernetics*, vol. 47, no. 11, pp. 3814–3826, 2016.

[7] C. Leng, J. Wu, J. Cheng, X. Bai, and H. Lu, "Online sketching hashing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[8] M. Lin, R. Ji, H. Liu, X. Sun, S. Chen, and Q. Tian, "Hadamard matrix guided online hashing," *International Journal of Computer Vision*, vol. 128, no. 8, pp. 2279–2306, 2020.

[9] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng, "Query-aware locality-sensitive hashing for approximate nearest neighbor search," *Proceedings of the VLDB Endowment*, vol. 9, no. 1, pp. 1–12, 2015.

[10] S. Kim, H. Yang, and M. Kim, "Boosted locality sensitive hashing: Discriminative binary codes for source separation," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 106–110.

[11] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 12, pp. 2916–2929, 2012.

[12] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Advances in Neural Information Processing Systems*, vol. 21. Curran Associates, Inc., 2009.

[13] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *ICML*, 2011, pp. 1–8.

[14] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon, "Spherical hashing: Binary code embedding with hyperspheres," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 11, pp. 2304–2316, 2015.

[15] H. Liu, R. Ji, J. Wang, and C. Shen, "Ordinal constraint binary coding for approximate nearest neighbor search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 4, pp. 941–955, 2018.

[16] H. Liu, R. Ji, Y. Wu, and W. Liu, "Towards optimal binary code learning via ordinal embedding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[17] Z. Zhang, X. Zhu, G. Lu, and Y. Zhang, "Probability ordinal-preserving semantic hashing for large-scale image retrieval," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 15, no. 3, pp. 1–22, 2021.

[18] C. Ma, I. W. Tsang, F. Shen, and C. Liu, "Error correcting input and output hashing," *IEEE Transactions on Cybernetics*, vol. 49, no. 3, pp. 781–791, 2019.

[19] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for large-scale search," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 12, pp. 2393–2406, 2012.

[20] Q.-Y. Jiang and W.-J. Li, "Asymmetric deep supervised hashing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[21] G. Wang, Q. Hu, Y. Yang, J. Cheng, and Z.-G. Hou, "Adversarial binary mutual learning for semi-supervised deep hashing," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021.

[22] Z. Cao, M. Long, J. Wang, and P. S. Yu, "Hashnet: Deep learning to hash by continuation," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5608–5617.

[23] W.-J. Li, S. Wang, and W.-C. Kang, "Feature learning based deep supervised hashing with pairwise labels," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. IJCAI/AAAI Press, 2016, pp. 1711–1717.

[24] H. Liu, R. Wang, S. Shan, and X. Chen, "Deep supervised hashing for fast image retrieval," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2064–2072.

[25] H. Zhu, M. Long, J. Wang, and Y. Cao, "Deep hashing network for efficient similarity retrieval," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[26] X. Wang, Y. Shi, and K. M. Kitani, "Deep supervised hashing with triplet labels," in *Asian conference on computer vision*. Springer, 2016, pp. 70–84.

[27] H. Lai, Y. Pan, Y. Liu, and S. Yan, "Simultaneous feature learning and hash coding with deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3270–3278.

[28] X. Zhou, F. Shen, L. Liu, W. Liu, L. Nie, Y. Yang, and H. T. Shen, "Graph convolutional network hashing," *IEEE Transactions on Cybernetics*, vol. 50, no. 4, pp. 1460–1472, 2020.

[29] F. Cakir, K. He, S. A. Bargal, and S. Sclaroff, "Hashing with mutual information," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 10, pp. 2424–2437, 2019.

[30] C. Yan, B. Gong, Y. Wei, and Y. Gao, "Deep multi-view enhancement hashing for image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 4, pp. 1445–1451, 2020.

[31] Y. Peng, J. Zhang, and Z. Ye, "Deep reinforcement learning for image hashing," *IEEE Transactions on Multimedia*, vol. 22, no. 8, pp. 2061–2073, 2019.

[32] Y. Liu, J. Song, K. Zhou, L. Yan, L. Liu, F. Zou, and L. Shao, "Deep self-taught hashing for image retrieval," *IEEE Transactions on Cybernetics*, vol. 49, no. 6, pp. 2229–2241, 2019.

[33] D. Wu, Q. Dai, J. Liu, B. Li, and W. Wang, "Deep incremental hashing network for efficient image retrieval," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9069–9077.

[34] F. Cakir and S. Sclaroff, "Adaptive hashing for fast similarity search," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1044–1052.

[35] L.-K. Huang, Q. Yang, and W.-S. Zheng, "Online hashing," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2309–2322, 2017.

[36] F. Cakir, S. A. Bargal, and S. Sclaroff, "Online supervised hashing," *Computer Vision and Image Understanding*, vol. 156, pp. 162–173, 2017.

[37] F. Cakir, K. He, S. Adel Bargal, and S. Sclaroff, "Mihash: Online hashing with mutual information," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 437–445.

[38] W. W. Ng, X. Tian, W. Pedrycz, X. Wang, and D. S. Yeung, "Incremental hash-bit learning for semantic image retrieval in nonstationary environments," *IEEE transactions on cybernetics*, vol. 49, no. 11, pp. 3844–3858, 2018.

[39] Z. Weng and Y. Zhu, "Online hashing with bit selection for image retrieval," *IEEE Transactions on Multimedia*, vol. 23, pp. 1868–1881, 2021.

[40] M. Lin, R. Ji, S. Chen, X. Sun, and C.-W. Lin, "Similarity-preserving linkage hashing for online image retrieval," *IEEE Transactions on Image Processing*, vol. 29, pp. 5289–5300, 2020.

[41] Z. Weng and Y. Zhu, "Online hashing with efficient updating of binary codes," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 12 354–12 361.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[43] B. Liu, Y. Cao, M. Long, J. Wang, and J. Wang, "Deep triplet quantization," in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, pp. 755–763.

[44] X. Tian, W. W. Y. Ng, and H. Wang, "Concept preserving hashing for semantic image retrieval with concept drift," *IEEE Transactions on Cybernetics*, vol. 51, no. 10, pp. 5184–5197, 2021.

[45] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[46] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Princeton, NJ, USA, Tech. Rep., 2009.

[47] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng, "Nus-wide: A real-world web image database from national university of singapore," in *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece., July 8-10, 2009.

[48] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 248–255.