Deep Incremental Hashing Network for Efficient Image Retrieval

Dayan Wu[1,2]*, Qi Dai[3], Jing Liu[1,2], Bo Li[1]†, Weiping Wang[1]
[1]Institute of Information Engineering, Chinese Academy of Sciences
[2]School of Cyber Security, University of Chinese Academy of Sciences
[3]Microsoft Research Asia
{wudayan,liujing,libo,wangweiping}@iie.ac.cn qid@microsoft.com

Abstract

Hashing has shown great potential in large-scale image retrieval due to its storage and computation efficiency, especially the recent deep supervised hashing methods. To achieve promising performance, deep supervised hashing methods require a large amount of training data from different classes. However, when images of new categories emerge, existing deep hashing methods have to retrain the CNN model and generate hash codes for all the database images again, which is impractical for large-scale retrieval system. In this paper, we propose a novel deep hashing framework, called ***D****eep* ***I****ncremental* ***H****ashing* ***N****etwork (****DIHN****), for learning hash codes in an incremental manner. DIHN learns the hash codes for the new coming images directly, while keeping the old ones unchanged. Simultaneously, a deep hash function for query set is learned by preserving the similarities between training points. Extensive experiments on two widely used image retrieval benchmarks demonstrate that the proposed DIHN framework can significantly decrease the training time while keeping the state-of-the-art retrieval accuracy.*

1. Introduction

Learning to hash has achieved great successes in various computer vision tasks, e.g., image retrieval [5, 8, 27, 17, 30, 20, 21, 33], classification [24], and person re-identification [35, 31]. It aims to encode documents, images, videos or other sorts of multimedia data into short binary hash codes, while preserving the similarity of the original data. With the compact hash codes, the storage and computation costs are significantly reduced. In this paper, we focus on the recent deep hashing methods which incorporate deep neural networks into the learning of hash codes. Such approaches [13, 16] have shown superior improvements over the conventional hashing methods like Locality Sensitive Hashing (LSH) [6], Spectral Hashing (SH) [28] and Iterative Quantization [7].

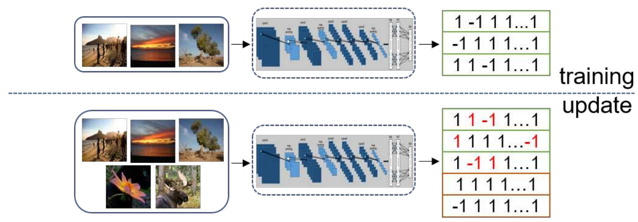


Figure 1. Framework of the traditional deep hashing methods for model updates. When new semantic concepts emerge, traditional methods have to retrain the models using both the previous training images and the new coming images. The database images are required to be fed into the new models to generate hash codes again.

To date in the literature, the state-of-the-art deep hashing methods typically exploit the powerful convolutional neural networks (CNN) to capture the underlying semantic structures of images. By simultaneously learning the feature representations and hash functions, these methods take full advantage of the pre-trained CNN and show satisfactory performance. Numerous approaches have been proposed, including both the single-label image hashing [13, 16] and multi-label image hashing [29, 36] methods.

One practical challenge in modern image retrieval system is how to keep the model up to date. With the explosive growth of various web images, lots of the new semantic concepts constantly emerge, while the up to date model is not always timely available. In other words, we have to update the retrieval model once new concepts appear. Such requirement is highly difficult for existing hashing methods, as illustrated in Figure 1. On the one hand, the learned hash codes of the same images from the updated model would change. One has to feed all the database images into the model to generate the codes again, which is very time-consuming, especially for large database. On the other hand, retraining the model with both original and additional data further increases the time cost of update.

*This work was done during an internship at Microsoft Research Asia.
†Corresponding author.

In this paper, we introduce a novel deep hashing framework for learning binary hash codes in an incremental way, named Deep Incremental Hashing Network (DIHN). An overview of the proposed framework is illustrated in Figure 2. Given the incremental images, the query images, and the hash codes of original images, we learn the hash functions for query images and hash codes of incremental images simultaneously. Specifically, a deep convolutional neural network is utilized as the hash function only for query images, while the hash codes of incremental images are directly learned. With such asymmetric design, the hash codes of original images are kept unchanged. We further devise an incremental hashing loss function for model training, which elaborately involve the similarity preservation between training points. The main contribution of this work can be summarized as follows:

- We propose a novel deep hashing framework, named *Deep Incremental Hashing Network* (DIHN), for learning binary hash codes in an incremental way. To the best of our knowledge, DIHN is the first deep hashing approach which can incrementally learn the hash codes for training images from new categories while holding the original ones invariant. The proposed method provides a flexible way for updating modern image retrieval system.
- An incremental hashing loss function is devised by preserving the similarities between training images. It incorporates the existing binary hash codes for original images to train the hash function for query images. Meanwhile, the binary hash codes for incremental images can be directly obtained during the optimization as well.
- Extensive experiments demonstrate that the proposed approach can significantly decrease the training time[1] with almost no loss on retrieval accuracy, compared with the state-of-the-art methods.

2. Related Work

We summarize recent works related to our proposed approach into two categories: deep hash learning and incremental learning. The former emphasizes encoding data points into compact binary codes for efficient data retrieval, while the later focuses on the class incremental learning that learns about more and more concepts over time.

Deep hash learning has recently shown very strong performance improvements over the shallow learning methods. Convolutional neural network hashing (CNNH) [32] learns the model to fit the binary codes computed from

[1]Please note that the training time includes both model training time and hash codes generation time for database images.

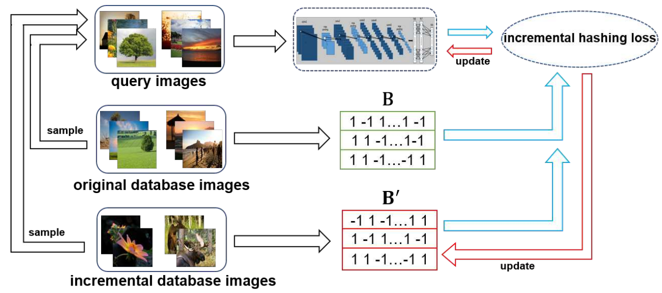


Figure 2. Framework of the proposed method for model updates. A CNN model is utilized as the hash function only for query images, while the hash codes of incremental database images are directly learned. With such asymmetric design, the hash codes of original database images are kept unchanged.

the pairwise similarity matrix. Network In Network Hashing (NINH) [11] and Deep Regularized Similarity Comparison Hashing (DRSCH) [35] try to preserve the similarities between image triplets. In addition, pairwise labels are utilized for hash function learning in Deep Supervised Hashing (DSH) [16] and Deep Pairwise Supervised Hashing (DPSH) [14]. Deep Supervised Discrete Hashing (DSDH) [13] further incorporate classification information within a single stream. Some methods also consider the multi-level semantic similarity between multi-label images [29, 36]. More recently, several asymmetric deep hashing methods are proposed, including Asymmetric Deep Supervised Hashing (ADSH) [8] and Deep Asymmetric Pairwise Hashing (DAPH) [23]. Specifically, ADSH only learns hash function for query points, while DAPH jointly trains two different models to learn pairwise similarity-preserving codes. The advantages of asymmetric hashing scheme are thoroughly discussed in [19]. The above methods are all batch learning methods which learn hash functions on a given dataset. To handle data which comes sequentially, online hashing methods [1, 12] are proposed. They have demonstrated good performance-complexity trade-offs by learning hash functions from streaming data.

While all the aforementioned methods achieve impressive results, it is difficult and time-consuming for them to perform model updates. To solve this problem, our work mainly focuses on how to incrementally learn binary codes for the images from new categories. For our DIHN, we adopt the asymmetric idea from ADSH. However, they are fundamentally different in the way that DIHN simply treats the asymmetry as a mean of keeping the original hash codes unchanged, and concentrates on the incremental framework. In contrast, ADSH explores the asymmetric method for hash function learning. Besides, our DIHN is different from online hashing methods since the latter still require updating the hash codes for all database images during training.

Incremental learning aims to extend the existing model's knowledge with new training data. Developing incremental algorithms could actually help address the issues of training cost and data availability. One direction of incremental learning is to utilize fixed data representation. Nearest Class Mean (NCM) [18] represents each class as a prototype vector that is the average vector of all examples for the class so far. A new example is classified to its nearest prototype class. Multiclass Transfer Incremental Learning [10] shows that a loss of accuracy can be avoided when incorporating new classes to the classifier, as long as the classifier can be retrained from a small set of data. On the contrary, recent methods try to learn data representation incrementally. ICaRL [22] learns the representation by exploiting the distillation and exemplar images, which can reduces the requirement of storing old data. Different from above methods, in this paper we present a novel incremental learning framework for hashing, which guarantees both the fixed original hash codes and the representation learning for query data.

3. Deep Incremental Hashing Network

3.1. Problem Definition

Assume we are given m database images $D = \{d_i\}_{i=1}^{m}$ and the set of class labels $L = \{L_1, L_2, ..., L_c\}$, where each image d_i is associated with a class label $L_j \in L (1 \leq j \leq c)$. So far we have learned the binary hash codes $B = \{b_i\}_{i=1}^{m} \in \{-1, +1\}^{k \times m}$ for the database images utilizing existing deep hashing methods, where k denotes the length of binary codes. Now, a set of n images $D' = \{d_i\}_{i=m+1}^{m+n}$ each of which is associated with a new class label $L'_j \in L' = \{L'_1, L'_2, ..., L'_{c'}\} (1 \leq j \leq c')$ emerge. Our goal is to learn the binary hash codes $B' = \{b_i\}_{i=m+1}^{m+n} \in \{-1, +1\}^{k \times n}$ for the images in new categories while leaving the ones in old categories unchanged. In addition, for the query set $Q = \{a_i\}_{i=1}^{q}$ of size q, which consists of images from both old and new categories, a CNN model is learned as the hash functions for generating their hash codes $B_Q = \{b_{Q_i}\}_{i=1}^{q} \in \{-1, +1\}^{k \times q}$. The pairwise supervised information, denoted as $S \in \{-1, +1\}^{(m+n) \times q}$, is available during training. The first m rows of S denote the semantic similarities between D and Q, and the remaining n rows denote that between D' and Q. $S_{ij} = 1$ indicates that d_i and a_j are semantically similar, while $S_{ij} = -1$ is the opposite.

3.2. Framework Overview

As illustrated in Figure 2, the proposed Deep Incremental Hashing Network (DIHN) has three inputs, i.e., the original database images, the incremental database images, and the query images, respectively. In practical scenarios, the query images are generally unavailable. Therefore, we sample both the original and incremental databases to form the set of query images. DIHN contains two important parts: *hash function learning part* and *incremental hash code learning part*. The hash function learning part exploits a deep CNN model to extract appropriate feature representations for query images. Subsequently, deep hash functions are learned on top of them to generate the binary hash codes B_Q. Note that this module is only employed for query set. The incremental hash code learning part directly learns the hash codes B' for incremental database images. To achieve the above two goals, an incremental hashing loss function is proposed to preserve the similarities between query and database points. While the binary hash codes for the original database images (B) are already given, they remain unchanged during the whole training process.

3.3. Deep Hash Functions

To exploit the recent advances in image representation with deep convolutional neural networks, we construct the hash functions by incorporating a CNN model, forcing the output of its last fully-connected layer to be length k, i.e., the length of binary hash codes. Accordingly, our deep hash function is defined as

$$b_{Q_i} = h\left(a_i; \theta\right) = sign\left(f(a_i; \theta)\right) \tag{1}$$

where θ denotes the parameters of CNN model, and $f(\cdot)$ denotes the output of the last fully-connected layer.

3.4. Incremental Hashing Loss

The incremental hashing loss is devised to preserve the semantic similarities between training points. With the pairwise label S, the target is to reduce/enlarge the Hamming distances between the binary codes of similar/dissimilar pairs. We thus adopt the L_2-norm loss to minimize the difference between the inner product of binary code pairs and the similarity, which can be formulated as

$$\begin{aligned} \min_{B', B_Q} J(B', B_Q) = & \sum_{i=1}^{m} \sum_{j=1}^{q} (b_i^T b_{Q_j} - kS_{ij})^2 \\ & + \sum_{i=m+1}^{m+n} \sum_{j=1}^{q} (b_i^T b_{Q_j} - kS_{ij})^2, \end{aligned} \tag{2}$$

$$\begin{aligned} s.t. \quad & B' = \{b_{m+1}, b_{m+2}, ..., b_{m+n}\} \in \{-1, +1\}^{k \times n}, \\ & B_Q = \{b_{Q_1}, b_{Q_2}, ..., b_{Q_q}\} \in \{-1, +1\}^{k \times q}. \end{aligned}$$

Since the binary codes for the original database images $B = \{b_1, b_2, ..., b_m\}$ are fixed, we only need to learn the B' and B_Q. By integrating the CNN learning model into the above loss, we combine Eqn. (1) and Eqn. (2), obtaining the loss function as follows

$$\begin{aligned}\min_{B',\theta} J(B',\theta) &= \sum_{i=1}^{m}\sum_{j=1}^{q}[b_i^T sign\left(f(a_j;\theta)\right) - kS_{ij}]^2 \\ &+ \sum_{i=m+1}^{m+n}\sum_{j=1}^{q}[b_i^T sign\left(f(a_j;\theta)\right) - kS_{ij}]^2, \\ s.t.\quad & B' = \{b_{m+1}, b_{m+2}, ..., b_{m+n}\} \in \{-1,+1\}^{k\times n}.\end{aligned} \tag{3}$$

Note that the problem in Eqn. (3) is in general NP-hard because of the sign function $sign\left(\cdot\right)$. Hence, a common continuous relaxation that replace it with the function $tanh\left(\cdot\right)$ is adopted, bringing the new formulation as

$$\begin{aligned}\min_{B',\theta} J(B',\theta) &= \sum_{i=1}^{m}\sum_{j=1}^{q}[b_i^T tanh\left(f(a_j;\theta)\right) - kS_{ij}]^2 \\ &+ \sum_{i=m+1}^{m+n}\sum_{j=1}^{q}[b_i^T tanh\left(f(a_j;\theta)\right) - kS_{ij}]^2, \\ s.t.\quad & B' = \{b_{m+1}, b_{m+2}, ..., b_{m+n}\} \in \{-1,+1\}^{k\times n}.\end{aligned} \tag{4}$$

As aforementioned, the query images are sampled from both original and incremental database images. Let $\Psi = \{1,2,3,...,m\}$ denotes the indices of the original database images, and $\Phi = \{m+1, m+2, m+3, ..., m+n\}$ be the indices of the incremental database images. Specifically, we denote the $\psi = \{i_1, i_2, i_3, ..., i_{q*}\} \subset \Psi$ and $\phi = \{j_1, j_2, j_3, ..., j_{q'}\} \subset \Phi$ as the indices of the query images sampled from the original and incremental database sets respectively, where q^* and q' are the number of their images, with $q^* + q' = q$. We then rewrite the above objective function $J(B',\theta)$ as

$$\begin{aligned}\min_{B',\theta} J(B',\theta) &= \sum_{i\in\Psi}\sum_{j\in(\psi\cup\phi)}[b_i^T tanh\left(f(d_j;\theta)\right) - kS_{ij}]^2 \\ &+ \sum_{i\in\Phi}\sum_{j\in(\psi\cup\phi)}[b_i^T tanh\left(f(d_j;\theta)\right) - kS_{ij}]^2 \\ &+ \lambda \sum_{j\in(\psi\cup\phi)}[b_j - tanh\left(f(d_j;\theta)\right)]^2 \\ &+ \mu \sum_{j\in(\psi\cup\phi)}(tanh\left(f(d_j;\theta)\right)^T \mathbf{1})^2, \\ s.t.\quad & B' = \{b_i | i\in\Phi\} \in \{-1,+1\}^{k\times n},\end{aligned} \tag{5}$$

where additional terms $\lambda\sum_{j\in(\psi\cup\phi)}[b_j - tanh\left(f(d_j;\theta)\right)]^2$ and $\mu\sum_{j\in(\psi\cup\phi)}[tanh\left(f(d_j;\theta)\right)^T \mathbf{1}]^2$ are further added, and λ, μ are two hyper-parameters. For the first term, the reason is that we have two hash code representations for each image d_j in query set: one is the fixed or learned binary hash codes b_j from database set, and the other is the deep CNN representation $tanh\left(f(d_j;\theta)\right)$. By exploiting this regularization, we hope to minimize the gap between them. While the second term makes a balance for each bit, which encourages the numbers of -1 and $+1$ to be approximately equal among all the query images.

3.5. Optimization

We learn the parameters θ and B' in problem (5) with an alternating strategy. More specifically, in θ-step, we learn θ with B' fixed. While in B'-step, we learn B' with θ fixed. The two steps are repeated for several iterations.

3.5.1 θ-step.

With B' fixed, it is easy to optimize the parameters of the CNN model with the standard back-propagation algorithm. For simplicity, we denote $u_j = tanh(f(d_j;\theta))$ and $z_j = f(d_j;\theta)$. The partial derivative of $J(B',\theta)$ with respect to z_j could be calculated by

$$\begin{aligned}\frac{\partial J}{\partial z_j} &= 2(\sum_{i\in\Psi}[(b_i^T u_j - kS_{ij})b_i] + \sum_{i\in\Phi}[(b_i^T u_j - kS_{ij})b_i] \\ &+ \lambda(u_j - b_j) + \mu u_j) \cdot (\mathbf{1} - u_j \cdot u_j),\end{aligned} \tag{6}$$

where $\mathbf{1}$ is the all-ones vector, and the operator $(\cdot)$ means the element-wise multiplication between two vectors. This gradient is then utilized to update θ.

3.5.2 B'-step.

Once θ is fixed, updating B' is not straightforward. To solve this problem, we first rewrite Eqn. (5) into the following matrix form,

$$\begin{aligned}\min_{B'} J(B') &= ||B'^T U - kS||_F^2 + \lambda||B'_\phi - U_\phi||_F^2 \\ &= ||B'^T U||_F^2 - 2ktr(B'SU^T) \\ &\quad - 2\lambda tr(B'_\phi U_\phi^T) + const, \\ & s.t.\quad B' = \{b_i | i\in\Phi\} \in \{-1,+1\}^{k\times n},\end{aligned} \tag{7}$$

where $S = \{-1,+1\}^{n\times q}$ is the similarity matrix between the query set and the incremental database set, $U = \{u_i | i \in (\psi\cup\phi)\} \in [-1,+1]^{k\times q}$ and $U_\phi = \{u_i | i\in\phi\} \in [-1,+1]^{k\times q'}$ are the relaxed binary-like representations for all the query images and query images sampled from incremental database set respectively, and B'_ϕ is the binary hash codes of the incremental database images indexed by ϕ, i.e., $B'_\phi = \{b_i | i\in\phi\} \in \{-1,+1\}^{k\times q'}$.

To further simplify (7), we define $\widetilde{U} = \{\widetilde{u}_i | i\in\Phi\} \in [-1,+1]^{k\times n}$, where $\widetilde{u}_i$ is given by

$$\widetilde{u}_i = \begin{cases} u_i, & i\in\phi, \\ 0, & otherwise, \end{cases} \tag{8}$$

and rewrite the equation (7) as follows

$$\begin{aligned}\min_{B'} J(B') &= ||B'^T U||_F^2 - 2tr(B'^T[kUS^T + \lambda\widetilde{U}]) \\ &= ||B'^T U||_F^2 + tr(B'^T P), \\ &s.t. \quad B' = \{b_i | i \in \Phi\} \in \{-1,+1\}^{k\times n},\end{aligned} \tag{9}$$

where $P = -2kUS^T - 2\lambda\widetilde{U}$. Note that the constant term in (7) is omitted for simplicity.

When optimizing Eqn. (9), we adopt the discrete cyclic coordinate descent (DCC) algorithm proposed in [25]. Specifically, we update B' bit by bit, which means each time we update one row of B' with other rows fixed. We denote B'_{l*} as the l-th row of B', $l = 1, ..., k$ and $\hat{B}'_l$ as the matrix of B' excluding B'_{l*}. Similarly, let U_{l*} be the l-th row of U and $\hat{U}_l$ be the matrix of U excluding U_{l*}. For P, let P_{l*} denote the l-th row of P and $\hat{P}_l$ denote the matrix of P excluding P_{l*}. Therefore, Eqn. (9) can be transformed to

$$\begin{aligned}\min_{B'_{l*}} J(B'_{l*}) &= ||B'^T U||_F^2 + tr(B'^T P) \\ &= tr(B'_{l*}(2\hat{B}'^T_l \hat{U}_l U^T_{l*} + P^T_{l*})), \\ &s.t. \quad B'_{l*} \in \{-1,+1\}^{1\times n}.\end{aligned} \tag{10}$$

It is obvious that when the sign of each bit in B'_{l*} is different from that of the corresponding bit in $2\hat{B}'^T_l \hat{U}_l U^T_{l*} + P^T_{l*}$, the loss $J(B'_{l*})$ can reach its minimum value. Therefore, the solution to problem (10) is as follows

$$B'_{l*} = -sign(2\hat{B}'^T_l \hat{U}_l U^T_{l*} + P^T_{l*}). \tag{11}$$

We then replace the l-th row of B' with B'_{l*} to update B'. All rows are updated sequentially by repeating (11).

The learning algorithm for DIHN is summarized in Algorithm 1. It is worth noting that if the size of B is large, the computational complexity of the learning algorithm will be high. To handle the problem, we can replace B with a subset sampled from it.

3.6. Original Hash Codes Learning

The DIHN learning algorithm only focuses on the incremental hash codes learning. Note that DIHN puts no limits on the methods for learning the original hash codes B, which means that all the existing hashing methods can be used. There is no doubt that the quality of original hash codes directly influence the performance of DIHN. Therefore, it is better to adopt more powerful deep hashing methods, e.g., ADSH and DSDH, for original code generation.

3.7. Computational Complexity

The computational complexity for training DIHN includes the one for optimizing θ and the other one for optimizing B'. For the former part, the complexity is $\mathcal{O}((m+n)qk)$. For the later part, the complexity is $\mathcal{O}(nqk^2)$. In practice, q and k are much smaller than m and n. Therefore, the computational complexities become $\mathcal{O}(m+n)$ and $\mathcal{O}(n)$ respectively. For traditional symmetric deep hashing methods, when all database images are used for training, the complexity is at least $\mathcal{O}(m^2)$. For triplet loss functions, the complexity will reach $\mathcal{O}(m^3)$. Even for the asymmetric hashing method ADSH, its complexity for optimizing B' is $\mathcal{O}((m+n)qk^2)$, which is much higher than DIHN. Compared with existing deep hashing methods, the complexity of training DIHN is much smaller, proving that DIHN is an efficient incremental hashing method.

Algorithm 1: The learning algorithm for DIHN

Input: original database set D; incremental database set D'; similarity matrix S; original hash codes B; code length k; iteration number T.
Output: incremental hash codes B' and neural network parameter θ.
Initialize $B' \in \{-1,+1\}^{k\times n}$;
Initialize neural network parameter θ.
repeat
for *i=1→T* **do**
  1.Forward computation to compute $f(d;\theta)$ from the raw images in mini-batch;
  2.Compute derivation according to (6);
  3.Update the neural network θ by utilizing back propagation.
end
Update B' according to (11)
until convergence or reach maximum iterations

Table 1. Split details of two datasets. # Original denotes the number of original classes, while # Incremental denotes the number of incremental classes.

Datasets	# Original / # Incremental			
CIFAR-10	7/3	8/2	9/1	4/6
NUS-WIDE	18/3	19/2	20/1	10/11

4. Experiments

4.1. Evaluation Setup

We conduct extensive evaluations of our proposed method on two widely used datasets, **CIFAR-10** [9] and **NUS-WIDE** [4].

- **CIFAR-10**[2] consists of 60,000 color images in 10 classes, and each class has 6,000 images of size 32 × 32.

[2]http://www.cs.toronto.edu/ kriz/cifar.html

Table 2. Comparison of MAP w.r.t. different number of bits on two datasets. Note that the MAP performance is calculated on the top 5,000 returned images for NUS-WIDE dataset. The best results for MAP are shown in bold.

Methods	CIFAR-10				NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
SH	0.1830	0.1640	0.1610	0.1610	0.6210	0.6160	0.6150	0.6120
ITQ	0.2619	0.2754	0.2861	0.2941	0.7143	0.7361	0.7457	0.7553
FastH	0.5971	0.6632	0.6847	0.7020	0.7267	0.7692	0.7817	0.8037
LFH	0.4178	0.5738	0.6414	0.6927	0.7116	0.7681	0.7949	0.8135
SDH	0.4539	0.6334	0.6514	0.6603	0.7646	0.7998	0.8017	0.8142
DSH	0.6441	0.7421	0.7703	0.7992	0.7125	0.7313	0.7401	0.7485
DHN	0.6805	0.7213	0.7233	0.7332	0.7719	0.8013	0.8051	0.8146
DPSH	0.6818	0.7204	0.7341	0.7464	0.7941	0.8249	0.8351	0.8442
DQN	0.5540	0.5580	0.5640	0.5800	0.7680	0.7760	0.7830	0.7920
DSDH	0.7400	0.7860	0.8010	0.8200	0.7760	0.8080	0.8200	0.8290
ADSH	0.8898	<u>0.9280</u>	0.9310	0.9390	<u>0.8400</u>	0.8784	0.8951	0.9055
DIHN_1+ADSH	<u>0.8933</u>	0.9279	**0.9386**	0.9346	0.8357	<u>0.8862</u>	<u>0.8987</u>	0.9070
DIHN_2+ADSH	**0.8975**	**0.9294**	0.9293	0.9385	**0.8470**	**0.8926**	0.8964	<u>0.9103</u>
DIHN_3+ADSH	0.8916	0.9273	<u>0.9330</u>	**0.9456**	0.8353	0.8821	**0.9004**	**0.9115**
DIHN_6+ADSH	0.8784	0.9172	0.9293	<u>0.9401</u>	-	-	-	-
DIHN_{11}+ADSH	-	-	-	-	0.8120	0.8727	0.8880	0.8972

- **NUS-WIDE**[3] contains 269,648 images collected from Flickr. The association between images and 81 concepts are manually annotated. Following [16, 37], we use the images associated with the 21 most frequent concepts, where each of these concepts associates with at least 5,000 images, resulting in a total of 195,834 images.

For CIFAR-10, we randomly select 1,000 images (100 images per class) as query set, with the remaining images as database images. Similarly, for NUS-WIDE, we randomly choose 2,100 images (also 100 images per class) as query set, leaving the rest as database. Note that the "query" set here is different from the query set in previous sections. The former set is the testing set while the later set is an input for training, which is sampled from the database.

For incremental learning setting, we split the database into two parts, i.e., the original and incremental database sets. Detailed splitting strategies are listed in Table 1. For each dataset, we have 4 split settings. In CIFAR-10, "7/3" means the original set contains images from 7 classes while the incremental set includes images from the other 3 classes. In NUS-WIDE, which is a multi-label dataset, "18/3" means the images in original set are associated with at most 18 concepts, while the images in incremental set are associated with at least one concept of the remaining 3 concepts. Note that we also design two challenging settings, in which the number of the incremental classes is larger than that of the original classes, i.e. 4/6 for CIFAR-10 and 10/11 for NUS-WIDE.

[3]http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm

We compare DIHN with several competitors, including both traditional and deep hashing methods. All competitors are trained on the whole database set, without the incremental setting. For traditional approaches, we compare with SH [28], ITQ [7] from unsupervised hashing, and FastH [15], LFH [34], SDH [25] from supervised hashing. The deep hashing methods include DSDH [13], DQN [2], DHN [37], DPSH [14], DSH [16], and ADSH [8]. To reduce the computational complexity of DIHN when B is very large, we design a variant of DIHN, denoted as **DIHN***, which replaces the large B with its subset B^*. We report the results of DIHN* with different sample rates. In implementation, following [8], we adopt the CNN-F model [3] as the basic network architecture for both DIHN and all the other deep hashing approaches. This CNN architecture has 5 convolutional layers and 2 FC layers. For traditional (non-deep) methods, we utilize the 4,096-dim deep features extracted from the CNN-F model pre-trained on ImageNet. For DIHN, we set $\lambda = 200, q = 2000$ and $\mu = 50$ by cross-validation for both datasets.

We report the Mean Average Precision (MAP) to evaluate the retrieval accuracy of the proposed DIHN and baselines. For NUS-WIDE dataset, the MAP performance is calculated on the top 5,000 returned images. To verify the effectiveness of DIHN, we report the MAP results of DIHN with different dataset split strategies, denoted as $\textbf{DIHN}_1$, $\textbf{DIHN}_2$, $\textbf{DIHN}_3$, $\textbf{DIHN}_6$ and $\textbf{DIHN}_{11}$, in which DIHN_i ($i = 1, 2, 3, 6, 11$) means that the number of incremental classes is i. In order to perform a fair comparison, most of the MAP results are directly reported from previous works.

All of our proposed approaches are implemented on

Table 3. Comparison of MAP w.r.t. different number of bits on two datasets. $\text{DIHN}^*_{0.2}$ denotes DIHN^* with sample rate 0.2. Note that the MAP performance is calculated on the top 5,000 returned images for NUS-WIDE dataset.

Methods	CIFAR-10				NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
DIHN_3+ADSH	**0.8916**	**0.9273**	**0.9330**	**0.9456**	**0.8353**	**0.8821**	**0.9004**	**0.9115**
$\text{DIHN}^*_{0.2}$+ADSH	0.7823	0.8770	0.8901	0.9079	0.8113	0.8680	0.8952	0.9075
$\text{DIHN}^*_{0.4}$+ADSH	0.8539	0.9175	0.9260	0.9319	0.8300	0.8749	0.8978	0.9078
$\text{DIHN}^*_{0.6}$+ADSH	<u>0.8910</u>	<u>0.9262</u>	0.9264	<u>0.9372</u>	<u>0.8321</u>	0.8776	0.8977	<u>0.9111</u>
$\text{DIHN}^*_{0.8}$+ADSH	0.8815	0.9221	<u>0.9306</u>	0.9303	0.8291	<u>0.8820</u>	<u>0.8993</u>	0.9103

MatConvNet [26] framework. We carry out the experiments on a PC with 4 AMD Opteron cores (8 threads), 64GB RAM, and NVIDIA Tesla P100.

4.2. Accuracy Comparison

Table 2 shows the MAP performance comparisons on two datasets. In most cases, the supervised methods outperform the unsupervised methods, and the deep hashing methods further exceed the traditional hashing methods. Among all the deep hashing methods, two discrete hashing methods ADSH and DSDH achieve the best performance. ADSH further shows a large improvement over DSDH, indicating the advantages of asymmetric hashing scheme. To verify the effectiveness of DIHN, we adopt the asymmetric deep hashing method ADSH to generate original binary hash codes B. For normal incremental settings, i.e. $i = 1, 2, 3$, DIHN_i+ADSH can achieve competitive retrieval performance. For challenging incremental settings, i.e. $i = 6, 11$, the performance of DIHN_i+ADSH is a little inferior to that of DIHN_i+ADSH under the normal settings. This is mainly because the original database images under the challenging incremental settings are less than the ones under the normal settings, resulting in less discriminative original binary hash codes B.

We further verify the effectiveness of DIHN^* with three different sample rates: 0.2, 0.4, 0.6 and 0.8. The MAP results are listed in Table 3. Note that the number of incremental classes for DIHN^* is 3. In general, compared to DIHN_3, DIHN^* can achieve competitive retrieval performance for sample rates 0.6 and 0.8. For lower sample rates 0.2 and 0.4, the performance of DIHN^* becomes poor, and $\text{DIHN}^*_{0.4}$ achieves much better performance than $\text{DIHN}^*_{0.2}$, indicating the importance of the original binary hash codes B. As can be seen in optimization process, B is highly related to θ, and θ can affect B'.

To deal with the situation when only parts of the incremental database images are labeled, we investigate a DIHN variant, called **DIHN-S**. DIHN-S directly learns binary hash codes for the labeled incremental database images while generating hash codes for unlabeled ones through the learned CNN model. For example, $\text{DIHN}_3\text{-S}_{0.2}$ denotes DIHN_3 with 20%/80% labeled/unlabeled incremental database images. The MAP results are shown in Table 4. In general, DIHN-S is inferior to DIHN mainly beacause of the quantization error and less sampled labeled incremental images. However, due to the advantages of the asymmetric hashing scheme, DIHN-S can efficiently utilize the learned hash codes and still achieve competitive retrieval performance. As expected, the MAP results increase with the increasing of the number of the labeled incremental database images.

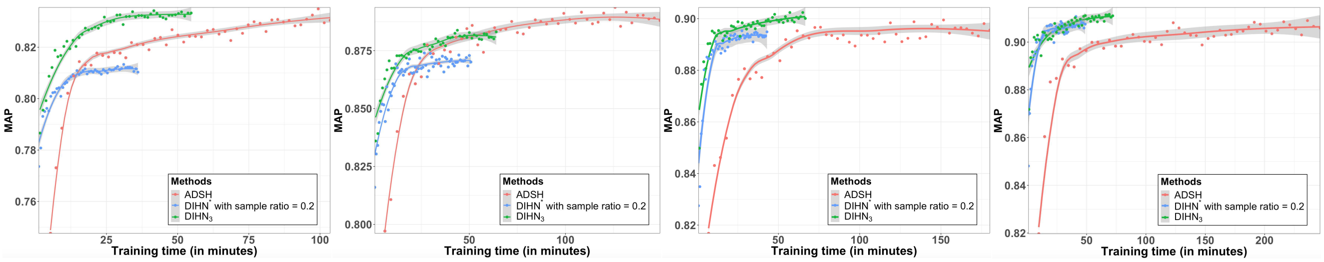
4.3. Time Complexity

We further compare the training costs of DIHN_3+ADSH and $\text{DIHN}^*_{0.2}$+ADSH with other approaches. The number of incremental classes for $\text{DIHN}^*_{0.2}$+ADSH is 3. For the competitor, we only choose the ADSH, since it has been shown in [8] that ADSH already performs much faster than other deep hashing methods. We report the MAP results at different training time points.

Figure 3 details the performances and time costs on NUS-WIDE dataset. In all cases, DIHN_3+ADSH shows much better training efficiency than ADSH, as expected in complexity analysis section. The reason is that DIHN does not need to take the original database into account. For example, it takes only about 28 minutes for DIHN_3+ADSH to achieve promising performance when the code length is 12. In contrast, ADSH has to cost three times (83 minutes) as much as DIHN_3+ADSH to reach the similar retrieval accuracy. This somewhat reveals the weakness of ADSH that it is not suitable for the incremental scenario. ADSH has to retrain the model when updating, even the previous hash codes and model are already available. While our DIHN is much more flexible to handle the newly-emerging data. Moreover, compared to DIHN_3+ADSH, our $\text{DIHN}^*_{0.2}$+ADSH could further reduce the training time to get convergence. Specifically, when the code length is 48, $\text{DIHN}^*_{0.2}$+ADSH takes about 20 minutes to get convergence, while DIHN_3+ADSH costs 40 minutes. Table 5 further shows the comparison of training time spent in one iteration (θ-step+B'-step) on NUS-WIDE dataset. As expected, ADSH takes much more time than DIHN_3 and $\text{DIHN}^*_{0.2}$ in one iteration, which is consistent with complexity analysis.

Table 4. Comparison of MAP w.r.t. different number of bits on two datasets. DIHN$_3$-S$_{0.2}$ denotes DIHN$_3$ with 20%/80% labeled/unlabeled incremental database images. Note that the MAP performance is calculated on the top 5,000 returned images for NUS-WIDE dataset.

Methods	CIFAR-10				NUS-WIDE			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
DIHN$_3$+ADSH	**0.8916**	**0.9273**	**0.9330**	**0.9456**	**0.8353**	**0.8821**	**0.9004**	**0.9115**
DIHN$_3$-S$_{0.2}$+ADSH	0.7594	0.8423	0.8714	0.8774	0.8151	0.8337	0.8529	0.8640
DIHN$_3$-S$_{0.4}$+ADSH	0.8374	0.8983	0.9156	0.9205	0.8097	0.8490	0.8630	0.8692
DIHN$_3$-S$_{0.6}$+ADSH	0.8799	0.9091	0.9249	0.9278	0.8179	0.8525	0.8681	0.8733
DIHN$_3$-S$_{0.8}$+ADSH	0.8769	0.9269	0.9379	0.9352	0.8171	0.8652	0.8811	0.8901

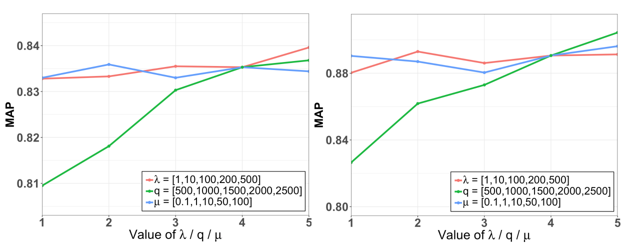


(a) 12 bits (b) 24 bits (c) 32 bits (d) 48 bits

Figure 3. The MAP results vs. training time costs curves of DIHN$_3$+ADSH, DIHN$^*_{0.2}$+ADSH, and ADSH on NUS-WIDE dataset. DIHN shows much better training efficiency.

Table 5. Comparison of the training time (in seconds) spent in one iteration w.r.t. different number of bits on NUS-WIDE dataset.

Methods	12 bits	24 bits	32 bits	48 bits
ADSH	122.78	178.97	216.26	296.41
DIHN$_3$	65.06	76.83	79.88	86.49
DIHN$^*_{0.2}$	42.66	48.06	51.60	58.60



(a) NUS-WIDE (b) CIFAR-10

Figure 4. The MAP with the change of hyper-parameters q, λ and μ on two datasets. The code length is 12.

4.4. Parameter Sensitivity

The MAP results of DIHN$_3$+ADSH under different values of the hyper-parameters λ, q, and μ are shown in Figure 4. We tune one parameter with the other two fixed. For instance, we tune λ in the range of $[1, 10, 100, 200, 500]$ by fixing $q = 2000$ and $\mu = 50$, respectively. Similarly, we set $\lambda = 200$, $q = 2000$ when tuning μ, and set $\lambda = 200$, $\mu = 50$ when tuning q. As demonstrated in Figure 4, the performance varies with different values of the three hyper-parameters. Specifically, λ, q and μ have a wide range $[1, 500]$, $[500, 2500]$ and $[0.1, 100]$, respectively. Note that the MAP value increases with the increasing of q in the range $[500, 2500]$, indicating that sampling more images can boost the retrieval performance. However, as discussed before, the computational complexity increases with the increasing of q. Hence, we set $q = 2000$ in consideration of both computational complexity and retrieval performance.

5. Conclusion

In this paper, we have proposed a deep incremental hash learning architecture for large-scale image retrieval, called DIHN. The DIHN has the following three features: capability of learning new data, preservation of old codes, and efficiency of training. To the best of our knowledge, DIHN is the first deep incremental hashing approach. We carefully design an incremental hashing loss function that aims at simultaneously generating hash codes for incremental database images and learning a CNN model for producing hash codes for query images. Extensive experiments demonstrate that the proposed deep incremental hashing method DIHN can significantly decrease the training time with almost no loss of accuracy.

Acknowledgments

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (XDC02050200) and Beijing Municipal Science and Technology Project (Z181100002718004).

References

[1] Fatih Cakir, Kun He, Sarah Adel Bargal, and Stan Sclaroff. Mihash: Online hashing with mutual information. In *ICCV*, pages 437–445, 2017.

[2] Yue Cao, Mingsheng Long, Jianmin Wang, Han Zhu, and Qingfu Wen. Deep quantization network for efficient image retrieval. In *AAAI*, pages 3457–3463, 2016.

[3] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014.

[4] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *CIVR*, 2009.

[5] Qi Dai, Jianguo Li, Jingdong Wang, and Yu-Gang Jiang. Binary optimized hashing. In *MM*, pages 1247–1256. ACM, 2016.

[6] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[7] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, pages 817–824, 2011.

[8] Qing-Yuan Jiang and Wu-Jun Li. Asymmetric deep supervised hashing. In *AAAI*, 2018.

[9] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

[10] Ilja Kuzborskij, Francesco Orabona, and Barbara Caputo. From n to n+1: Multiclass transfer incremental learning. In *CVPR*, pages 3358–3365, 2013.

[11] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, pages 3270–3278, 2015.

[12] Cong Leng, Jiaxiang Wu, Jian Cheng, Xiao Bai, and Hanqing Lu. Online sketching hashing. In *CVPR*, pages 2503–2511, 2015.

[13] Qi Li, Zhenan Sun, Ran He, and Tieniu Tan. Deep supervised discrete hashing. In *NIPS*, pages 2482–2491, 2017.

[14] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. Feature learning based deep supervised hashing with pairwise labels. In *IJCAI*, pages 1711–1717, 2016.

[15] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton Van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*, pages 1963–1970, 2014.

[16] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. Deep supervised hashing for fast image retrieval. In *CVPR*, pages 2064–2072, 2016.

[17] Fuchen Long, Ting Yao, Qi Dai, Xinmei Tian, Jiebo Luo, and Tao Mei. Deep domain adaptation hashing with adversarial learning. In *SIGIR*, pages 725–734. ACM, 2018.

[18] Thomas Mensink, Jakob Verbeek, Florent Perronnin, and Gabriela Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *TPAMI*, pages 2624–2637, 2013.

[19] Behnam Neyshabur, Nati Srebro, Ruslan R Salakhutdinov, Yury Makarychev, and Payman Yadollahpour. The power of asymmetry in binary hashing. In *NIPS*, pages 2823–2831, 2013.

[20] Yingwei Pan, Ting Yao, Houqiang Li, Chong-Wah Ngo, and Tao Mei. Semi-supervised hashing with semantic confidence for large scale visual search. In *SIGIR*, 2015.

[21] Zhaofan Qiu, Yingwei Pan, Ting Yao, and Tao Mei. Deep semantic hashing with generative adversarial networks. In *SIGIR*, 2017.

[22] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, pages 5533–5542, 2017.

[23] Fumin Shen, Xin Gao, Li Liu, Yang Yang, and Heng Tao Shen. Deep asymmetric pairwise hashing. In *MM*, 2017.

[24] Fumin Shen, Yadong Mu, Yang Yang, Wei Liu, Li Liu, Jingkuan Song, and Heng Tao Shen. Classification by retrieval: Binarizing data and classifiers. In *SIGIR*, pages 595–604, 2017.

[25] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. In *CVPR*, pages 37–45, 2015.

[26] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *MM*, pages 689–692, 2015.

[27] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A survey on learning to hash. *TPAMI*, 40(4):769–790, 2018.

[28] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2009.

[29] Dayan Wu, Zheng Lin, Bo Li, Mingzhen Ye, and Weiping Wang. Deep supervised hashing for multi-label and large-scale image retrieval. In *ICMR*, pages 150–158, 2017.

[30] Dayan Wu, Jing Liu, Bo Li, and Weiping Wang. Deep index-compatible hashing for fast image retrieval. In *ICME*, pages 1–6. IEEE, 2018.

[31] Lin Wu, Yang Wang, Zongyuan Ge, Qichang Hu, and Xue Li. Structured deep hashing with convolutional neural networks for fast person re-identification. *CVIU*, 167:63–73, 2018.

[32] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, 2014.

[33] Ting Yao, Fuchen Long, Tao Mei, and Yong Rui. Deep semantic-preserving and ranking-based hashing for image retrieval. In *IJCAI*, 2016.

[34] Peichao Zhang, Wei Zhang, Wu-Jun Li, and Minyi Guo. Supervised hashing with latent factor models. In *SIGIR*, pages 173–182, 2014.

[35] Ruimao Zhang, Liang Lin, Rui Zhang, Wangmeng Zuo, and Lei Zhang. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. *TIP*, pages 4766–4779, 2015.

[36] Fang Zhao, Yongzhen Huang, Liang Wang, and Tieniu Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, pages 1556–1564, 2015.

[37] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *AAAI*, pages 2415–2421, 2016.