Deep reinforcement hashing with redundancy elimination for effective image retrieval

Juexu Yang\textsuperscript{a}, Yuejie Zhang\textsuperscript{a,\textdagger}, Rui Feng\textsuperscript{a}, Tao Zhang\textsuperscript{b,\dagger}, Weiguo Fan\textsuperscript{c}

\textsuperscript{a} School of Computer Science, Shanghai Key Laboratory of Intelligent Information Processing, Fudan University, Shanghai, China
\textsuperscript{b} School of Information Management and Engineering, Shanghai Key Laboratory of Financial Information Technology, Shanghai University of Finance and Economics, Shanghai, China
\textsuperscript{c} Department of Business Analytics, Tippie College of Business, University of Iowa, USA

\section{1. Introduction}

With the explosive development of social media, enormous amount of data including texts, images, and videos is produced every day. To retrieve them efficiently, multiple methods have been proposed. Recently, Approximate Nearest Neighbor (ANN) search has attracted increasing attention for its high retrieval accuracy and low computational cost. Among various ANN search methods, hashing is one of the most promising methods that generate compact hash codes for high-dimensional data points and perform retrieval in Hamming Space. The focus for this paper is on learning to hash methods\cite{1}, which are data-dependent and can utilize supervised information to generate high-quality hash codes for efficient image retrieval. Usually, data-dependent hashing methods outperform data-independent methods (e.g., Locality Sensitive Hashing (LSH)\cite{2}) with a big margin.

For decades, many hashing methods have been proposed and studied. Recently, with the great success of deep learning, deep hashing methods are attracting increased research interests. The high fitting ability of deep neural network makes it possible to fit any non-linear hash function. Moreover, deep hashing enables an end-to-end learning style that performs feature learning and hash code learning simultaneously. On many public benchmarks, deep hashing methods yield the state-of-the-art performance with much more compact hash codes. As a notable example, robust discrete code modeling for supervised hashing\cite{3} proposed a hashing method that leveraged discrete optimization to get rid of the quantization error and could handle both noisy hash codes and noisy semantic labels.

However, there are two major disadvantages for current deep hashing methods. First, due to the constraint of computational resources, most deep neural network methods have to be trained in a mini-batch way, which makes them very inefficient at data
sampling. Suppose there are $n$ images, for pair-wise hashing methods there are $C_n^2 = \frac{n(n-1)}{2}$ image pairs which is $O(n^2)$, and for triplet hashing methods there are total $C_n^3 = \frac{n(n-1)(n-2)}{6}$ image triplets which is $O(n^3)$. It would take enormous amount of time to collect enough samples for training. Without enough samples, hashing methods can only preserve the local similarity and fail to preserve the global similarity, which may hurt the retrieval accuracy. Second, the hash codes generated by most existing methods contain some degree of redundancy. Some bits of the generated hash codes can be thrown away without harming the retrieval accuracy, and the existence of these bits may even decrease the retrieval accuracy. There are two sources for such redundant or even harmful bits. One is the noisy data in the dataset, and the other is the commonly used mini-batch based training strategy, which makes the generated hash codes only able to preserve local similarity relationships.

Based on the above observations, a novel deep reinforcement hashing model with redundancy elimination method called Deep Reinforcement De-Redundancy Hashing (DRDH) is proposed in this paper, which can fully exploit the similarity information in a global way. Our scheme adopts deep reinforcement learning to eliminate the redundancy in generated hash codes. When performing hash code inference, label information is utilized to build a similarity matrix, and a set of hash codes that can reconstruct this similarity matrix are learned. The similarity matrix is calculated in an on-demand block-wise way, so that an arbitrarily large similarity matrix can be handled. When performing hash code mapping, Deep Neural Network is exploited to map raw images to the previously inferred hash codes. This mapping can be formulated as a multi-label binary classification problem, in which the hash bit mapping can be processed in $O(n)$ and the sampling from $O(n^2)$ pairs or $O(n^3)$ triplets will no longer be needed. After hash code mapping, Deep Reinforcement Learning is particularly exploited to eliminate redundant hash bits from the hash codes. More specifically, Deep Q Network (DQN) [4] is leveraged to learn a mask that can mask out those redundant and harmful hash bits. Extensive experiments demonstrate that DRDH can generate compact and de-redundant hash codes, and yield better retrieval performance than those of state-of-the-art methods on four public datasets of CIFAR-10, NUS-WIDE, MS-COCO, and Open-Images-V4.

Our contributions to the literature are mainly three folds: (1) We design a new block-wise similarity calculation manner which is used to infer a set of hash codes that can preserve the global similarity relationship. Existing related methods are either trained in a mini-batch style that can only preserve local similarity relationships or conducted to achieve the hash code inference column by column, which is intractable when the dataset is really large. Different from these methods, our proposed block-wise calculation can not only fully exploit the global similarity information but also able to handle an arbitrarily large similarity matrix. (2) We leverage Deep Q Network in DRDH to eliminate redundant hash bits, which makes DRDH able to acquire compact hash codes with less redundant bits. Some existing works [5–7] adopt regularization terms in loss function to avoid redundant hash bits. Although this strategy is effective, it is at the cost of hurting the overall expressiveness of the generated hash codes, and may reduce the retrieval accuracy. Different from such a redundancy-avoidance strategy, our method aims at a redundancy-elimination mechanism which does not sacrifice the overall retrieval accuracy. (3) We perform extensive experiments on four standard benchmark image datasets to show that DRDH is better than many state-of-the-art methods in a real image retrieval environment.

The rest of the paper is organized as follows. Section 2 briefly reviews some related works. In Section 3, we describe in detail our new framework of Deep Reinforcement Hashing with Redundancy Elimination. Section 4 gives our experimental results and analyses on the algorithm evaluation, and we conclude the paper in Section 5.

2. Related works

Existing hashing methods can be roughly divided into two categories, i.e., unsupervised and supervised.

Unsupervised hashing methods such as LSH [2] and its variants aim at learning compact hash codes for unlabeled data. LSH adopts the random projection as a hash function, which usually needs long hash codes ($\geq$128 bits) to achieve sufficient accuracy. SADH [8] established an unsupervised hashing framework, named Similarity-Adaptive Deep Hashing, which alternately proceeded over three training modules of deep hash model training, similarity graph updating, and binary code optimization. In addition, a discrete optimization algorithm was devised to directly handle the binary constraints with a general hashing loss. While unsupervised hashing methods can be trained on unlabeled data, they may suffer from the semantic gap problem, i.e., high-level semantic information differs from low-level feature information. Supervised hashing methods fully exploit supervised label information, and thus can generate discriminative hash codes while keeping them short at the same time. Some notable supervised hashing methods are KSH [9], LHF [10], FastH [11], and SDH [12]. Besides, GCNHI [13] introduced a semi-supervised graph convolutional network based hashing framework, which directly carried out spectral convolution operations on both an image set and an affinity graph built over the set, naturally yielding similarity-preserving binary embedding.

In recent years, deep neural network has been applied to learn hashing methods, and shown great breakthroughs in image retrieval accuracy. DHN [14] was the first end-to-end deep hashing method that preserved the pair-wise similarity and reduced the quantization error simultaneously. Beyond pairwise similarity preserving loss, PRDH [15] added constraints to further keep the independence of each bit and the equal possibility of being 1 and 0 for each bit. HashNet [16] added different weights for positive pairs and negative pairs to balance the training data, and by using continuation technique it could achieve lower quantization error with a series of increasingly cliffy activate function. DSH [17] incorporated a margin loss which forced the hamming distance between dissimilar image pairs to be at least some margin. Therefore, it could ensure similar images pairs within a certain hamming sphere while dissimilar image pairs were out of that hamming sphere. MiHash [18] proposed a new Mutual Information metric for both training more discriminative hashing models and evaluating the performance of existing hashing methods. DAPH [19] exploited the power of asymmetry which used two sets of hash codes, that is, one for database images and the other one for the query images. SCFSH [5] introduced a rotation matrix to further reduce the quantization error. gDRH [6] generated multiple hash codes for different regions in an image, and the query process considered both global and local information. DSH-GAN [7] leveraged Generative Adversarial Network to synthesize more images and used those images to conduct a semi-supervised learning. DCWH [20] proposed a loss function based on the Gaussian model to handle the deep hashing learning problem at the class level. DCH [21] proposed a Cauchy Loss to force similar images cluster within small hamming radius and to better reduce the quantization error. TALKR [22] was the first tie-aware hashing framework that could handle tie match during hamming distance sorting. DRLIH [23] proposed a deep reinforcement learning hashing network, using RNN as agents to take previous hash function’s error into account. HashGAN [24] proposed a Pair Conditional Wasserstein GAN (PC-WGAN) that could exploit
pairwise similarity information to generate synthetic images. DSAH [25] proposed a saliency loss of image quadruples to guide the attention network for automatic discriminative regions mining. DPH [26] introduced the priority mechanism into the hashing literature, and with that it could deal with both easy and hard images, easy and hard quantization. KRH [27] explored normalizing a Gaussian Kernel and proposed two unsupervised hashing methods that were desirable for k-nearest search. Bayesian de-noising hashing for robust image retrieval [28] proposed a post-processing de-noising layer that could correct corrupted bits (due to image noises) in generated hash codes. CMDH [29] exploited a mechanism that could jointly learn a unified binary code through discrete optimization without any continuous relaxation that might hurt the retrieval accuracy. In DDCMH [30] graph regularizations were introduced to preserve both intra-modality similarity and inter-modality similarity. MFKH [31] incorporated multiple features into hashing methods and could preserve similarity relationships better. OLSH [32] learned the hash function in an online style that not only utilized the newly arrived data to retain the hash function, but also preserved the similarities among old data. SDDH [33] proposed a robust similarity metric to make the learned discrete hash codes more suitable for classification. The hash functions were learned directly from the learned discrete hash codes to achieve optimal approximation. MSDH [34] presented a discrete hashing method with multiple supervision hashing, which supervised the hash code learning with both class-wise and instance-class similarity matrices. Besides, an iterative optimization algorithm was proposed to directly learn the discrete hash codes. RODH [35] proposed a ranking optimization discrete hashing method, which directly generated discrete hash codes from raw images by balancing the category-level information of discretization and the discrimination of ranking information.

However, there exists two major disadvantages of current deep hashing methods. First, most of them are trained in a mini-batch based style which makes them only able to preserve local similarity relationship. Moreover, this mini-batch based training strategy has to sample from the collection of all image pairs, which has a magnitude of $O(n^2)$. This inefficiency in data sampling increases training time significantly. Some hashing methods use an optimization strategy that learns hash codes column by column. During each epoch, only one column is learned while all the other columns are fixed. Since this strategy can ease the sampling problems and preserve the global similarity, there are two pitfalls in this strategy. When the dataset is very large, the matrix involved in the optimization process could be huge and cannot fit into the main memory. Meanwhile, such a column-based method can only optimize some simple loss functions and cannot handle more complex loss functions. To ease this problem, we especially propose a novel block-wise calculation style, which preserves similarity information block by block and accumulates the gradient of all blocks together in the end to update the hash codes. Second, most deep hashing methods generate hash codes with redundant or even harmful bits. Guo et al. [36] proved that the commonly used pair-wise loss and triplet loss were prone to generate highly correlated hash bits due to the mini-batch based training strategy. They adopted a strategy that split a dataset into different parts and trained different models on each part. While this strategy could potentially reduce the hash bit correlation significantly, it might harm the overall retrieval accuracy because high-correlation did not mean redundancy. We want to eliminate redundancy, not high-correlation, if this correlation is required. To tackle such a problem, we particularly design a block-wise calculation mechanism that can fully exploit the global similarity information. To further eliminate redundant bits from generated hash codes, we use Deep Q Network (DQN) to learn a mask that can mask out redundant or harmful bits.

3. A new framework of deep reinforcement hashing with redundancy elimination

3.1. Formulation

In a learning to hash task, a set of training images are represented as feature vectors $X = \{x_i\}_{i=1}^n \in \mathbb{R}^{n \times D}$, where $x_i$ can be a shallow feature, deep feature or raw pixels. For supervised hashing, each image is annotated with a semantic label $Y = \{y_i\}_{i=1}^n \in \{0, 1\}^{1 \times m}$, where $m$ is the total number of semantic categories. For $y_i$ at $i$th image, it means the $i$th image belongs to the $j$th category, and an image can belong to multiple categories. To express the similarity relationship between two images, a similarity matrix $S \in \{-1, +1\}^{n \times n}$ is constructed, where $S_{ij} = +1$ means the $i$th image and the $j$th image are similar, otherwise $S_{ij} = -1$. Two images are considered similar if they share at least one semantic category. There is an inner relationship between the similarity matrix $S$ and the semantic label $Y$:

$$ S = \min \{YY^T, \mathbf{1}\} \times 2 - 1 $$

(1)

where $\min \{., .\}$ is an element-wise minimum function, and $\mathbf{1}$ is a matrix whose elements are all ones. The task of deep hashing is to learn a non-linear hash function $h(\cdot)$ that maps the images from the original feature space to a low-dimensional binary space, while preserving the similarity relationships: $h(x) = B$ where $B = \{b_i\}_{i=1}^n \in \{-1, +1\}^{n \times k}$ and $k$ is the length of hash code ($k$ bits). We then use the Hamming distance to measure the distance between two data points in a binary space, which is the number of bits where two hash codes are different.

3.2. Architecture

The architecture of our proposed Deep Reinforcement Hashing with Redundancy Elimination model, i.e., Deep Reinforcement De-Redundancy Hashing (DRDH), is shown in Fig. 1, which contains three parts of Hash Code Inference, Hash Code Mapping, and Hash Code De-Redundancy. First we perform the hash code inference to find a set of hash codes that can best preserve the similarity relationship between every two images. After that, Convolutional Neural Network (CNN) is utilized to map raw images to the inferred hash codes. Here, we conduct experiments using both AlexNet [37] and ResNet18 [38] as our deep neural networks. We set the output dimension of the last layer to the length of the inferred hash codes, while keeping all other layers unchanged. After the mapping, we utilize Deep Q Network (DQN) to learn a mask that can be used to mask out redundant bits of the mapped hash codes.

3.3. Block-wise hash code inference

First we need to infer a set of hash codes that can best preserve the original similarity relationships globally. There is a natural relationship between the Hamming distance and the inner product of two hash codes:

$$ H_{\text{dis}}(b_i, b_j) = \frac{1}{2}(k - b_i \cdot b_j^T) $$

(2)

where $k$ is the length of a hash code. As can be seen, the larger the inner product is, the smaller the Hamming distance is. Thus we can view the inner product of two hash codes as a factor of affinity, and then use it to reconstruct the original similarity relationship as:

$$ \min_B \mathcal{L}_1 = \|BB^T - KS\|_2^2 $$

(3)

Since $B$ is in a binary space, it makes Eq. (3) become a Mixed Integer Programming (MIP) problem, which is usually NP-hard and
would take exponential time to optimize. Following the common practice, we remove the binary constraint, and make $B$ in real space denoted as $\tilde{B}$:

$$
\min_{\tilde{B}} \mathcal{L}_1 = \left\| \tilde{B} \tilde{B}^T - kS \right\|^2_2
$$

(4)

After the optimization, we can get $B$ by simply applying the sign function $\text{sign}(\cdot)$ to $\tilde{B}$:

$$
B = \text{sign}(\tilde{B}) \quad \text{where} \quad \text{sign}(x) = \begin{cases} -1, & x < 0 \\ +1, & x \geq 0 \end{cases}
$$

(5)

However, there may exist a large gap between $B$ and $\tilde{B}$. This gap is called quantization error, and is harmful to the retrieval accuracy. A regularization term is added to reduce the quantization error:

$$
\min_{\tilde{B}} \mathcal{L}_1 = \left\| \tilde{B} \tilde{B}^T - kS \right\|^2_2 + \left| \tilde{B} - 1 \right|
$$

(6)

where $1$ is a matrix whose elements are all ones. The similarity matrix $S$ is an $n \times n$ matrix, where $n$ is the number of training images. While $S$ can make a full description of the original similarity relationships, it is too big to be stored in memory. Since we already know how to construct $S$ through the semantic label $Y$ according to Eq. (1), we then substitute $S$ with $Y$:

$$
\min_{\tilde{B}} \mathcal{L}_1 = \left\| \tilde{B} \tilde{B}^T - k(\min(Y Y^T, 1) \times 2 - 1) \right\|^2_2 + \left| \tilde{B} - 1 \right|
$$

(7)

Now we can calculate $S$ in a block-wise style. Suppose the block size is $h \times w$, and $n$ is divisible by $h$ and $w$, Eq. (7) can be reformulated as:

$$
\min_{\tilde{B}} \mathcal{L}_1 = \sum_{r=0}^{h-1} \sum_{c=0}^{w-1} \left\| \tilde{B}_{r+h-c, c+w-1} \tilde{B}_{r+h-c, c+w-1}^T - k(\min(Y_{r+h-c, c+w-1} Y_{r+h-c, c+w-1}^T, 1) \times 2 - 1) \right\|^2_2 + \left| \tilde{B} - 1 \right|
$$

(8)

where $Y_{r+h-1}$ is the $r$th row to the $(r+h-1)th$ row of $Y$, the same goes for $\tilde{B}$. Now we can perform the optimization in a block-wise way, which is shown as the solid dark blue block on the similarity matrix in Fig. 1. This solid dark blue block will slide over the whole similarity matrix $S$. Here, the gradient descent method is utilized to optimize Eq. (8), in which the gradient will be calculated for each block and then added together. This accumulated gradient will back-propagate only once, and thus the similarity relationships in each block is preserved.

For the convenience of the subsequent hash code mapping, we transform the inferred hash codes $B$ from $\{-1, +1\}^{n \times k}$ to $\{0, 1\}^n \times k$, which is denoted as $Z$:

$$
Z = \{z_i\}_{i=1}^n = \frac{1}{2}(B + 1)
$$

(9)

While the block-wise similarity calculation looks similar to the mini-batch based method, there are two differences between them. Firstly, the block-wise calculation only involves semantic labels, and thus is faster than the mini-batch based hashing method which needs to involve image features. Another difference is that the gradient is accumulated for the block-wise calculation and thus can preserve the global similarity information, while the mini-batch based method usually updates parameters based on each single batch. Meanwhile, in the hash code inference stage, no image feature is involved, and that is why this stage can be really fast compared to the other two stages (as shown in Table 4). In this stage, we only need to know the labels (i.e., semantic annotations), and from which we can establish the similarity relationship between each image pair. In general, “image” can be substituted to any modality like text, audio, video and etc. With the similarity relationship established, we can decide what the best hash code for each label (and the image it represents) is, in order to best preserve the similarity relationship. In addition, it is worth noting that Anchor Graph [39] is another classic method to reduce the size of the similarity matrix. It first performs clustering to get cluster centers (i.e., anchor points), and then uses these anchor points to approximate the similarity matrix. The final performance depends on how well the clustering is performed and the following approximation. Different from Anchor Graph, our motivation is to tackle the huge similarity matrix directly in a simple and evident style.

The process of the block-wise hash code inference is shown in Algorithm 1.

3.4. Hash code mapping

Now we try to map raw images to the previously inferred hash codes. We use Convolutional Neural Network (CNN) to do this
Algorithm 1
Block-wise Hash Code Inference

Input: Training image labels: \( Y = \{y_i\}_{i=1}^n \), Hash code length: \( k \), Number of epoch: \( t_1 \), Window height: \( h \), Window width: \( w \).
Output: Inferred hash codes for training set: \( Z = \{z_i\}_{i=1}^n \)

for \( t = 1: t_1 \)
    for \( r = 0: \frac{h}{w} - 1 \)
        for \( c = 0: \frac{w}{w} - 1 \)
            Calculate the block loss and block gradient according to Eq. (8).
            Add the gradient to the overall gradient.
        end
    end
end
Update \( \mathcal{B} \) according to the overall gradient.

Get binary hash codes. \( \mathcal{B} = \text{sign}(\mathcal{B}) \)

Transform \( \mathcal{B} \) from \([−1, +1]^d \) to \([0, 1]^d \): \( Z = \{z_i\}_{i=1}^n = \frac{1}{2} (\mathcal{B} + 1) \) according to Eq. (9).

mapping. The output of the network is denoted as \( F(x; \theta) \in \mathbb{R}^k \), where \( \theta \) is the parameter of the CNN and \( k \) is the length of the hash code. We view this mapping as a \( k \)-time binary classification problem, which is shown as:

\[
\min_{\theta} \mathcal{L}_2 = -\sum_{i=1}^n z_i \log (\sigma (F(x; \theta))) + (1 - z_i) \log (1 - \sigma (F(x; \theta)))
\]

(10)

where \( \sigma(\cdot) \) is the sigmoid function. This time we no longer need to sample from \( n^2 \) image pairs, instead we only need to classify each image correctly. \( F(x; \theta) \) is denoted as \( F \), and then the derivative of Eq. (10) is:

\[
\frac{d\mathcal{L}_2}{dF} = \sum_{i=1}^n (1 - z_i)\sigma(F - z_i(1 - \sigma(F))
\]

(11)

Eq. (10) can then be optimized via standard Back Propagation (BP). The hash function is denoted as \( h(x_i; \theta) \), and the mapped hash codes are denoted as \( C = \{c_i\}_{i=1}^n \in \{0, 1\}^{n \times k} \). We can get them by simply applying a threshold:

\[
h(x_i; \theta) = c_i = I(\sigma(F(x_i; \theta)) \geq 0.5)
\]

(12)

where \( I(\text{bool}) \) is an element-wise indicator function.

Technically, the hash code inference stage is aimed at that if knowing the similarity relationships among a set of images (through their labels), we attach each label with a hash code so that the hash code can best preserve the original similarity relationship. While the hash code mapping stage focuses on that if having a new unseen image without knowing its semantic label, we map it to a hash code so that this hash code gets along well with those hash codes that are acquired by the inference stage.

The process of classification based hash code mapping is shown in Algorithm 2.

Algorithm 2
Classification-based Hash Code Mapping

Input: Inferred hash codes: \( Z = \{z_i\}_{i=1}^n \), Number of epoch: \( t_2 \).
Output: Hash function: \( h(x; \theta) \)

for \( t = 1: t_2 \)
    for \( i = 1: n \)
        Feed the image \( x_i \) and its inferred hash code \( z_i \), into CNN, calculate multi-binary cross-entropy loss according to Eq. (10), and back-propagate.
    end
end

Output hash function \( h(x; \theta) \).

3.5. DQN-based hash code de-redundancy

After mapping each image with its inferred hash code, we try to eliminate redundant bits from the hash codes. Specifically we exploit Deep Q Network (DQN) to learn a mask \( m \in \{0, 1\}^k \) that has the same length as the hash code, and then use that mask to mask out those redundant bits. The de-redundant hash code can be denoted as \( D = \{d_i\}_{i=1}^n \in \{0, 1\}^{n \times k} \), in which \( d_i \) is defined as:

\[
d_i = c_i \odot m
\]

(13)

where \( \odot \) is an element-wise multiplication. The Q value is used to estimate how preferable it is to take the action \( a \) when in the state \( s \). The Q value is comprised of two parts, that is, the current reward and the max Q value of its neighbor:

\[
Q(s, a) = r(s) + \max_{a'} Q(s', a')
\]

(14)

where \( r(s) \) is the reward in the state \( s \), and \( s' \) is the new state after taking the action \( a \) in the state \( s \). The current state is defined as the current mask \( s = m \). Suppose we want to keep \( p \) bits and mask out \( k - p \) bits, then there must be \( p \) ones and \( k - p \) zeros in the mask. We define the action as swapping two bits, with one from those \( p \) ones, and the other one from those \( k - p \) zeros. Thus there are \( p(k - p) \) different actions in total. More specifically, the \( i \)th action is to swap the \( i \)th one and the \( i \)th zero. The relationship among them is shown as:

\[
u = [i/p] \odot v = i \mod p
\]

(15)

Since many loss functions (e.g., pair-wise, triplet) try to preserve the retrieval accuracy indirectly (using the local relationship), and may fail to preserve the global retrieval accuracy, we try to optimize the Mean Average Precision (MAP) directly. The reward is defined as:

\[
r(s) = \text{MAP}(s) - 1
\]

(16)

where \( \text{MAP}(s) \) is the mean average precision for the hash codes that are masked out by the current mask. Since \( \text{MAP}(s) \) is always smaller than 1, the reward would be negative and makes the
Algorithm 3
DQN-based Hash Code De-redundancy

Input: Hash function: h(x; v), Hash code length: k, Number of bits to keep: p, Number of epoch: t,
Output: Mask: m

Initialize the mask (state s) with p ones and k − p zeros.

for t = 1: t1 do
  if replay buffer is not full
    Take the action a on the state s according to epsilon-greedy probability, calculate reward r of the
    new state s′, store (s, a, r, s′) into replay buffer.
  else
    Sample experience from replay buffer and train the DQN with Eq. (18). Take the action a on
    the sampled state s. If new reward is larger than the
    smallest reward in replay buffer, replace that smallest reward experience with the new
    experience.
  end

Output the optimal mask m.

agent end the search quickly. The training procedure is as follows. The mask is fed into a Deep Q-Network which has three fully-connected layers and outputs the predicted Q value for each action. Here, we adopt Double Q-Learning [40] to accelerate training. There are two networks, i.e., a prediction network and a ground-truth network. The parameter of the ground-truth network will be synchronized with the prediction network periodically. The predicted Q value of the $a^j$th action is denoted as $Q_{gt}(s; a; \beta_{gt})$, where $\beta_{gt}$ is the parameter of the prediction network. We then calculate the ground-truth Q value $Q_{gt}(s; a; \beta_{gt})$ according to Eq. (14).

$$Q_{gt}(s; a; \beta_{gt}) = r(s) + \max_{s′} Q_{gt}(s′; a′; \beta_{gt})$$

(17)

Thus the loss function can be simply defined as the difference between $Q_{pred}(s; a; \beta_{pred})$ and $Q_{gt}(s; a; \beta_{gt})$:

$$\min_{\beta_{pred}} L_3 = \left\| Q_{pred}(s; a; \beta_{pred}) - Q_{gt}(s; a; \beta_{gt}) \right\|_2^2$$

(18)

where $\beta_{gt}$ is the parameter of the ground-truth network. Eq. (18) can then be optimized via the standard back-propagation. After training DQN, it can be utilized to predict the Q value of a mask and perform the action with the maximum Q value until we get a good mask.

The process of DQN based De-redundancy is summarized in Algorithm 3.

4. Experiments

We evaluate our deep reinforcement hashing with redundancy elimination approach, DRDH, against several state-of-the-art hashing methods on four standard benchmark datasets. All the related experiments are implemented with deep learning library PyTorch on a single NVIDIA GTX 1080-ti GPU.

4.1. Experiment setup

**NUS-WIDE** [41] is a public image dataset collected from Flickr.com. It contains 269,648 images in 81 ground truth categories. By keeping the top 21 categories, 195,834 images can be obtained. Following the experimental protocols in HashGAN [24], we randomly sample 5,000 images as the query images and treat the remaining as the database. We further randomly sample 10,000 images from the database as the training set.

**CIFAR-10**\(^1\) is a public image dataset with 60,000 small 32×32 images in 10 categories. Following the protocol in Deep Quantization Network [42], we randomly select 100 images per category as the query set, 500 images per category as the training set, and view the remaining images as the database.

**MS-COCO** [43] is a widely-used image dataset for Image Recognition, Segmentation, and Captioning. The current release contains 82,783 training images and 40,504 validation images, where each image is labeled by some of 80 semantic concepts. By merging the training set and validation set and removing the images that do not have any semantic label, 122,218 images can be obtained in total. We randomly sample 5,000 images as query images and treat the remaining as the database. We further randomly sample 10,000 images as the training images.

**Open-Images-V4**\(^2\) is an unprecedented large-scale dataset for the challenges on Object Detection and Visual Relationship Detection. In our experiment, we use its Object Detection subset which contains 1,910,098 images in total. After filtering invalid images, we obtain 1,903,392 images. There are 601 semantic labels in these images. We select top 27 semantic classes, in which each class has at least 50,000 images. In this way we obtain 1,709,658 images. We then randomly select 90,000 images (about 5%) as the query dataset, and view the rest of them as the database. From the database we further randomly select 180,000 images (about 10%) for training.

Following the standard evaluation protocol in previous works [16,17,44,45], two images are considered similar if they share at least one semantic label. Although we use the ground truth image labels to construct the similarity information, our DRDH model can learn compact binary hash codes when only the similarity information is available, which is more general than many label-information based hashing methods [14, 46].

We compare the retrieval performance of DRDH with several state-of-the-art hashing methods, including supervised shallow hashing methods BRE [47], ITQ-CCA [48], KSH [9], and SDH [12], and supervised deep hashing methods CNNH [49], DNNH [44], DHH [14], HashNet [16], and HashGAN [24]. We evaluate the retrieval quality based on four standard evaluation metrics, i.e., Mean Average Precision (MAP), Precision-Recall curves (PR), Precision curves within Hamming distance 2 ($|\Phi|H \leq 2$), and Precision curves with respect to the numbers of top returned samples ($|\Phi|N$). For direct comparison to published results, all methods adopt the identical training and testing sets. Following HashNet and DHH, we adopt MAP@5,000 for NUS-WIDE, MAP@5,000 for MS-COCO, MAP@54,000 for CIFAR-10, and MAP@50,000 for OpenImage-V4 respectively. For a given query, the Average Precision (AP) is defined as:

$$AP@T = \frac{\sum_{t=1}^{T} P(t) \delta(t)}{\sum_{t=1}^{T} \delta(t')},$$

(19)

\(^1\) http://www.cs.toronto.edu/~kriz/cifar.html

\(^2\) https://storage.googleapis.com/openimages/web/index.html
where $T$ is the top returned data points; $P(t)$ denotes the precision of top retrieved results; and $\delta(t) = 1$ if the $t^{th}$ retrieved result is a true neighbor of the query, otherwise $\delta(t) = 0$. MAP is the mean value of AP over many queries. Since the calculation of AP involves sorting a large index array, it is believed to be slow. However, with the help of GPU, all MAP calculations can be accelerated through parallelism.

For shallow hashing methods we adopt the 4,096-dimensional DeCaF [50] features as image features. For deep hashing methods, we use the original images as input. When performing the blockwise hash code inference, we use the standard gradient descent method, adopt Adam [51] as the optimizer, and set the learning rate to 1.0. For the hash code mapping, we try both AlexNet and ResNet18 as the hash encoder, fine-tune all layers but train the last layer from scratch. Note that ResNet18 is used on Open-Image-V4 only. Since we train the last layer from scratch, we set its learning rate 10 times larger as the one in previous layers. The learning rate for previous layers is set to $10^{-5}$. The learning rate for the last layer is set to $10^{-4}$. The mini-batch size is set to 32. For the hash code redundancy elimination, a three-layer forward network is used as the DQN.

To accelerate the training of DQN, we leverage Dueling Network [52] and Prioritized Experience Replay [53]. In Dueling Network, the Q value is divided into two parts, Value and Advantage, as can be seen from Figure 1. The Value part is the estimation of the value of a state, while the Advantage part is the estimation of the value that can be gained by taking actions in that state. This division makes the Q value estimation much easier.

To fully utilize training samples, Prioritized Experience Replay is proposed. A replay buffer is maintained and contains important samples. Important samples mean high MAP samples in the replay buffer. The DQN will then not only be trained through random training samples, but also samples from the replay buffer. This forces the DQN to review important samples, and thus can accelerate the training process.

### 4.2. Experimental results

The MAP results of all methods using AlexNet are shown in Table 1(a), in which all performance data except that of DRDH are cited from HashGAN [24]. For fair comparison we compare with other deep hashing methods using AlexNet. However, for the larger dataset like Open-Image-V4 the fitting ability of AlexNet is not enough, thus the results using ResNet18 are included. The MAP results of DRDH using ResNet18 are shown in Table 1(b). As can be seen from Table 1(a), our DRDH model substantially outperforms all the comparison methods by large margins. Specifically, compared to SDH, the best shallow hashing method with deep features as input, DRDH can achieve the absolute increases of 18.8%, 27.4%, and 17.8% in average MAP (i.e., average growth for 16 bits to 64 bits) on NUS-WIDE, CIFAR-10, and MS-COCO respectively. It is worth noting that DRDH also outperforms HashGAN, the state-of-the-art deep hashing method, by large margins of 7.8%, 7.9%, and 1.9% in average MAP on three datasets respectively. There is an existing method called DRLH [23] that leverages the power of reinforcement learning as well. We make a comparison with it in Table 1(c), following the same setting in DRLH and using VGG19 as the CNN. We also compare our DRDH with two most recent hashing methods, i.e., MSDH [34] and RODH [35], using AlexNet. The related comparison results on the CIFAR-10 dataset are shown in Tables 1(d) and 1(e) respectively. In Table 1(d), we follow the same setting as MSDH, which randomly samples 1,000 images as queries and uses the rest 59,000 images as training images. In Table 1(e), we follow the same setting as RODH, which randomly samples 10,000 images as queries and uses the remaining 50,000 images as training images. It can be observed from Table 1(d) and 1(e), our DRDH is able to trim the hash code to only 8 bits without the significant loss in the retrieval accuracy. Another observation is that the redundancy elimination is effective even with more training data, that is, in Table 1(a) only 5,000 images are used for training while there are 59,000 images and 50,000 images used

<table>
<thead>
<tr>
<th>Table 1(a)</th>
<th>The comparison results for MAP of Hamming ranking for different numbers of bits on three benchmark image datasets using AlexNet.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>NUS-WIDE 48 bits</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>FTQ-CCA [48]</td>
<td>0.400</td>
</tr>
<tr>
<td>BRE [47]</td>
<td>0.503</td>
</tr>
<tr>
<td>KSH [9]</td>
<td>0.521</td>
</tr>
<tr>
<td>SDH [12]</td>
<td>0.588</td>
</tr>
<tr>
<td>CNHH [49]</td>
<td>0.570</td>
</tr>
<tr>
<td>DNNH [44]</td>
<td>0.598</td>
</tr>
<tr>
<td>DHN [14]</td>
<td>0.637</td>
</tr>
<tr>
<td>HashNet [16]</td>
<td>0.662</td>
</tr>
<tr>
<td>HashGAN [24]</td>
<td>0.715</td>
</tr>
<tr>
<td>DRDH</td>
<td>0.805</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 1(b)</th>
<th>The experimental results for MAP of DRDH using ResNet18, in which 16 to 48 bits are acquired by eliminating redundancy in 64 bits.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>NUS-WIDE 48 bits</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>DRDH</td>
<td>0.844</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 1(c)</th>
<th>The comparison results for MAP of Hamming ranking for different numbers of bits on NUS-WIDE and CIFAR-10 using VGG19.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>NUS-WIDE 12 bits</td>
</tr>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>DRLH [23]</td>
<td>0.823</td>
</tr>
<tr>
<td>DRDH</td>
<td>0.839</td>
</tr>
</tbody>
</table>
for training in Table 1(d) and 1(e) respectively. This implies that even with enough training data, it is still possible for the generated hash codes to contain redundant bits.

The MAP results demonstrate two insights. (1) Shallow hashing methods cannot learn discriminative deep representations and compact hash codes through the end-to-end framework, which explains the fact that they are surpassed by deep hashing methods. (2) Deep hashing methods DHN and HashGAN learn less lossy hash codes by jointly preserving similarity information and controlling the quantization error, which significantly outperform pioneering methods CNNH and DNNH without reducing the quantization error.

As can be seen from Table 1(b), DRDH also works well with deeper and stronger network like ResNet18. Another interesting observation is that, DRDH achieves the best performance on the largest dataset Open-Image-V4. This is because for all these four datasets, there are only about 10% images used for training, and the rest 90% images are used as the database. This lack of sufficient training samples decreases the MAP heavily. The motivation to sample only 1/10 images for training is to simulate the real image retrieval environment, where the servers collect new images every day, and most of them are never seen by the hashing network. While this motivation is quite reasonable, we must guarantee that the dataset is large enough, so that even 1/10 of it is still representative for all semantic categories.

There are two main reasons for why DRDH improves substantially from the state-of-the-art HashGAN. (1) DRDH tries to preserve the global similarity relationships by using the block-wise calculation. Thus it can always preserve the whole similarity matrix instead of a small similarity matrix constructed from a mini-batch. Therefore, while it is hard for mini-batch based methods to sample enough pairs from training data, DRDH can fully preserve the global similarity relationships through the block-wise hash code inference. (2) DRDH can obtain 16 bits, 32 bits, and 48 bits hash codes by eliminating redundant bits from 64 bits hash codes. This brings DRDH two advantages. First, it can be trained only once, and obtain different lengths of hash codes through different de-redundancy settings, which is a huge acceleration of training for hashing methods. Second, by eliminating redundant bits from longer hash codes, DRDH can generate more compact and valuable hash codes. As can be viewed from Table 1(a), the improvements for shorter codes (16 to 48 bits) are greater than those for longer code (64 bits).

The performance in Precision within Hamming radius 2 (P@H ≤ 2) is very important for efficient image retrieval, since the Hamming ranking only requires O(1) time cost for each query, which enables really fast pruning. As shown in Figs. 2(a), 3(a), and 4(a), DRDH can achieve the highest P@H ≤ 2 on all three benchmark datasets using different numbers of bits. This validates that DRDH can learn more compact hash codes than all comparison methods to establish more efficient and accurate Hamming ranking. This improvement will be more significant for shorter hash codes, because each hamming distance will be assigned more images than longer codes.

The retrieval performance in terms of Precision-Recall curves (PR) and Precision curves with respect to different numbers of top returned samples (P@N) are presented in Figs. 2(b), 3(b), 4(b) and 2(c), 3(c), 4(c), respectively. Our proposed DRDH significantly outperforms all comparison methods by large margins under these two evaluation metrics. In particular, DRDH can achieve much higher precision at lower recall levels or with smaller number of top samples. This is very desirable for precision-first retrieval in practical image retrieval systems.

4.3. Ablation study

We investigate three variants of DRDH: (1) **DRDH-BT** is a DRDH variant that is trained in a mini-batch style without the block-wise hash code inference and without the redundant bits elimination; (2) **DRDH-B** is a DRDH variant that is trained in mini-batch...
and without the block-wise hash code inference; (3) DRDH-T is a DRDH variant without redundant hash bits elimination. The ablation study experiments are conducted on NUS-WIDE, and both AlexNet version and ResNet18 version of DRDH are verified. The related results are shown in Table 2(a) and 2(b). For DRDH and DRDH-B, 16-48 bits are acquired by eliminating the redundant bits from 64 bits.

**Block-wise Calculation** Table 2(a) and 2(b) show that DRDH significantly outperforms DRDH-B by a large margin of 1.9% in average MAP (i.e., average growth for 16bits to 64bits), DRDH-T outperforms DRDH-BT by a margin of 2.1% in average MAP. This demonstrates the efficacy of our proposed block-wise similarity calculation that can preserve global similarity relationship, especially when applied to the large-scale multi-label dataset.

---

**Table 2(a)**
The experimental results for MAP of DRDH and its three variants (AlexNet version).

<table>
<thead>
<tr>
<th>Method</th>
<th>NUS-WIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>DRDH-B</td>
<td>0.822</td>
</tr>
<tr>
<td>DRDH</td>
<td>0.836</td>
</tr>
<tr>
<td>DRDH-T</td>
<td>0.844</td>
</tr>
</tbody>
</table>

**Table 2(b)**
The experimental results for MAP of DRDH and its three variants (ResNet18 version).

<table>
<thead>
<tr>
<th>Method</th>
<th>NUS-WIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>DRDH-B</td>
<td>0.774</td>
</tr>
<tr>
<td>DRDH</td>
<td>0.794</td>
</tr>
<tr>
<td>DRDH-T</td>
<td>0.805</td>
</tr>
</tbody>
</table>

**Table 3(a)**
The experimental results for MAP of de-redundant hash codes in different lengths using AlexNet.

<table>
<thead>
<tr>
<th>k bits</th>
<th>NUS-WIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>DRDH-64 bits</td>
<td>0.8182</td>
</tr>
<tr>
<td>DRDH-48 bits</td>
<td>0.8134</td>
</tr>
<tr>
<td>DRDH-32 bits</td>
<td>0.8091</td>
</tr>
<tr>
<td>DRDH-16 bits</td>
<td>0.7991</td>
</tr>
</tbody>
</table>

**Redundant Bits Elimination** Table 2(a) and 2(b) show that DRDH outperforms DRDH-T by a margin of 0.6% in average MAP. DRDH-B outperforms DRDH-BT by a margin of 0.8% in average MAP. This indicates that with the help of the redundant bits elimination, we can get more compact hash codes by generating longer hash codes first, and then eliminate redundant bits from them, which can produce better results than generating shorter hash codes directly. From Table 2(a) and 2(b), the block-wise hash code inference and the hash bits de-redundancy can work together in harmony to improve the overall retrieval accuracy.

To further demonstrate the efficacy of the redundancy elimination, we try to eliminate just a few bits (1-4 bits) from different hash code lengths. The related results are shown in Table 3(a). We conduct the evaluation on four different settings, i.e., 16-12 bits, 32-28 bits, 48-44 bits, and 64-60 bits. For most settings, the de-redundancy can eliminate redundant hash bits while preserving the retrieval accuracy or even improving the retrieval accuracy a little bit. For 16-12 bits the MAP goes down, because the total hash code length is too short and each bit is of great importance. We also try different initial hash code lengths other than 64 bits. Table 3(b) shows the results of eliminating redundant bits from 128 bits hash codes, and Table 3(c) shows the results of eliminating redundant bits from 48 bits hash codes. It is worth noting that the MAP for 128 bits is almost the same as that for 64 bits. This is because when the hash code is longer enough, the main obstacle for retrieval accuracy is the noise from the dataset.
instead of the hash code’s capacity. From Table 3(b), we can see that the redundancy elimination stage works well with longer hash codes. Table 3(c) demonstrates that the redundancy elimination stage works well with fewer initial bits (i.e., 48 bits instead of 64 bits).

**Visualization.** Fig. 5 shows the t-SNE [54] visualization of the hash codes generated by our DRDH and HashGAN [24] on CIFAR-10 respectively. As shown in the figure, there exists more discriminative margins among the hash code clusters generated by DRDH. Fig. 6 shows some good and bad examples of the top-5 retrieval results on three benchmark datasets. Both good and bad examples are included. There are a couple of reasons for these bad examples, some of them apply to all three datasets, and some of them apply to specific datasets. For all three datasets, the main source of these bad retrieval examples is that the training set is too small compared to the database. Only 1/10 of the database images are selected for training. For NUS-WIDE a serious problem is the noisy label, more specifically, the missing label. There are many semantic objects appear in the images, while their labels are not included in the image labels. For CIFAR-10, the low resolution of images is the largest obstacle for retrieval. We also compare the retrieval results by our DRDH with those by the classic deep hashing baseline model DSH [17]. In Fig. 7, we have exhibited some examples of the top-10 retrieval results for both our DRDH and DSH. As can be seen, DRDH yields better Precision@10 (P@10) than the widely adopted baseline model DSH.

**Time cost.** To further analyze the time cost for each stage, we record the time cost for our DRDH and its three variants. In Table 4, we separate the logging time cost from the total time cost. For the hash code inference stage, the logging time is the time cost to calculate the MAP. Every time when the sliding window has slides through the whole similarity matrix, we calculate the MAP for current hash code. For the hash code mapping stage, the logging time is the time cost for hash code generation that is oriented to the whole dataset and from raw pixel after each epoch (in order to calculate the MAP), and the time cost to calculate the MAP. For the redundancy elimination stage, there is no logging time, since the MAP itself is the reward required by reinforcement learning.

As can be seen from Table 4, the inference stage is very fast, since it only involves label information. Another observation is that, the logging time is a big overhead and it is inevitable for the redundancy elimination stage. The time complexity for the MAP calculation is \(O(N\log N)\), where \(N\) is the size of the whole dataset. This is because the calculation requires sorting, and \(O(N\log N)\) is the lower bound for in-memory comparison based sorting. However, the MAP calculation can be accelerated by GPU, and its time cost is still within a tolerable range.

![Fig. 5. The t-SNE visualizations of the hash codes on CIFAR-10.](image-url)
Fig. 6. Some good and bad examples of the top-5 retrieval results on three benchmark datasets.

Fig. 7. Some examples of the top-10 retrieval results by our DRDH and DSH on NUS-WIDE.

It is worth noting that we have included the time cost of DSH [17] in Table 4. DSH is a widely used baseline model in hashing literatures. It is a one-stage based hashing method that conducts the hash code inference and hash code mapping in the same stage. As can be seen from Table 4, the time costs for the mapping stage and the redundancy elimination stage of DSH are both a little bit higher than those of DRDH. This is because DSH is trained in a mini-batch based style and needs enough training epochs to grasp the global similarity. At the same time, in each of these epochs the raw image pixels are fed into the whole CNN, and then loss is calculated and back-propagated. Moreover, since the mapping stage can only preserve local similarities and causes the reward signal of the following redundancy elimination stage become sparser, the overall search time will be increased.

5. Conclusion and future work

This paper tackles two key problems that exist in most deep learning to hash methods. The first problem is that they are usually trained in a mini-batch style, which makes them inefficient at data sampling and cannot preserve the global similarity relationship. We solve this problem by proposing a block-wise hash code inference, which can directly infer optimal hash codes from the large similarity information. The second is that most hashing methods generate hash codes with redundant bits. We tackle this issue by leveraging deep reinforcement learning to learn a mask that can mask out redundant hash bits, and then improve the retrieval accuracy. For future work, we will explore more on how to map visual information to hash codes effectively.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No. 61976057, No. 61572140), Shanghai Municipal R&D Foundation (No. 17DZ1100504, No. 16JC1420401), Shanghai Natural Science Foundation (No. 19ZR1417200), and Humanities and Social Sciences Planning Fund of Ministry of Education of