Neurons Merging Layer: Towards Progressive Redundancy Reduction for Deep Supervised Hashing

Chaoyou Fu[1234*] Liangchen Song[4*] Xiang Wu[12] Guoli Wang[4] Ran He[123†]
[1] Center for Research on Intelligent Perception and Computing, CASIA
[2] National Laboratory of Pattern Recognition, CASIA
[3] University of Chinese Academy of Sciences
[4] Horizon Robotics
{chaoyou.fu, rhe}@nlpr.ia.ac.cn, alfredxiangwu@gmail.com
{liangchen.song, guoli.wang}@horizon.ai

Abstract

Deep supervised hashing has become an active topic in information retrieval. It generates hashing bits by the output neurons of a deep hashing network. During binary discretization, there often exists much redundancy between hashing bits that degenerates retrieval performance in terms of both storage and accuracy. This paper proposes a simple yet effective Neurons Merging Layer (NMLayer) for deep supervised hashing. A graph is constructed to represent the redundancy relationship between hashing bits that is used to guide the learning of a hashing network. Specifically, it is dynamically learned by a novel mechanism defined in our active and frozen phases. According to the learned relationship, the NMLayer merges the redundant neurons together to balance the importance of each output neuron. Moreover, multiple NMLayers are progressively trained for a deep hashing network to learn a more compact hashing code from a long redundant code. Extensive experiments on four datasets demonstrate that our proposed method outperforms state-of-the-art hashing methods.

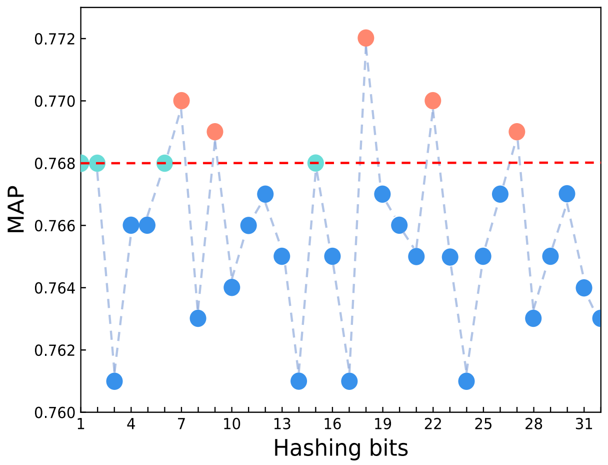


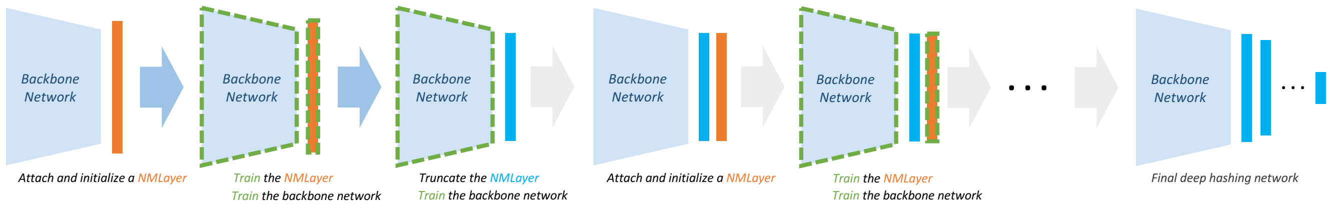
Figure 1: Illustration of the redundancy in hashing bits generated by a common CNN-F network [5]. The horizontal red dotted line represents the Mean Average Precision (MAP) calculated using all bits. The vertical axis represents the MAP calculated after removing corresponding bit. For example, removing the 1-st bit does not affect the MAP, while removing the 3-rd bit leads to a remarkable drop of MAP. Even more, the MAP increases after removing the 18-th bit.

1. Introduction

With the explosive growth of data, hashing has been one of the most efficient indexing techniques and drawn substantial attention [17]. Hashing aims to map high-dimensional data into a binary low-dimensional Hamming space. Equipped with the binary representation, hashing can be performed with constant or sub-linear computation complexity, as well as the markedly reduced space complexity [11]. Traditionally, the binary hashing codes can be generated by random projection [10] or learned from data distribution [11].

Over the last few years, inspired by the remarkable success of deep learning, researchers have paid much attention to combining hashing with deep learning [35, 4]. Particularly, by utilizing the similarity information for supervised learning, deep supervised hashing has greatly improved the performance of hashing retrieval [21, 14]. In general, the last layer of a neural network is modified as the output

*Equal contribution
†Corresponding author



Backbone Network
Attach and initialize a NMLayer
Train the NMLayer
Train the backbone network
Truncate the NMLayer
Train the backbone network
Attach and initialize a NMLayer
Train the NMLayer
Train the backbone network
Final deep hashing network

Figure 2: Illustration of our progressive optimization strategy. For a standard backbone network, i.e., deep hashing network, we first attach and initialize a NMLayer after the hashing layer. Then, we train the NMLayer as well as the backbone network for a certain number of epochs. Next, based on the learned adjacency relationship among neurons, the NMLayer is truncated to determine which neurons to merge. After that, according to the truncation results, we continue to train the backbone network. At this point, the training process of the first NMLayer is completed. By iterating the above process, which is attaching NMLayer and optimizing the whole network, we finally get the required hashing bits. Note that although many NMLayers are attached, total weights of the network change little. Because each time completing the training of a NMLayer, we just determine which neurons in the hashing layer to merge, without adding extra weights.

layer of hashing bits. Then, both features and hashing bits are learned from the neural network during optimizing the hashing loss function, which is elaborately designed to keep the similarities between the input data.

Despite the effectiveness of the existing deep supervised hashing methods, the redundancy of hashing bits remains a problem that has not been well studied [17, 8]. As shown in Figure 1, we can see that the redundancy has a significant impact on the retrieval performance. Because of the redundancy, the importance of different hashing bits varies greatly. However, a straightforward intuition is that all hashing bits should be equally important. In order to address the redundancy problem, we propose a simple yet effective method to balance the importance of each bit in the hashing codes. In details, we propose a new layer named Neurons Merging Layer (NMLayer) for deep hashing networks. It constructs a graph to represent the adjacency relationship between different neurons. During the training process, the NMLayer learns the relationship by a novel scheme defined in our active and frozen phases, as shown in Figure 3. Through the learned relationship, the NMLayer dynamically merges the redundant neurons together to balance the importance of each neuron. In addition, by training multiple NMLayers, we propose a progressive optimization strategy to gradually reduce the redundancy. The full process of our progressive optimization strategy is illustrated in Figure 2. Extensive experimental results on the CIFAR-10, NUS-WIDE, MS-COCO and Clothing1M datasets verify the effectiveness of our method. In short, our main contributions are summed up as follows:

1. We construct a graph to represent the redundancy relationship between hashing bits, and propose a mechanism that consists of the active and frozen phases to effectively update the relationship. This graph results in a new layer named NMLayer, which reduces the redundancy of hashing bits by balancing the importance of each bit. The NMLayer can be easily integrated into a standard deep neural network.

2. We design a progressive optimization strategy for training deep hashing networks. A deep hashing network is initialized with more hashing bits than the required bits, then the redundancy is progressively reduced by multiple NMLayers that form neurons merging. Compared with other hashing methods of fixed code length, NMLayers obtain a more compact code from a redundant long code.

3. Extensive experimental results on four challenging datasets show that our proposed method achieves significant improvements especially on large-scale datasets, when compared with state-of-the-art hashing methods.

2. Related Work

2.1. Hashing for Image Retrieval

Existing hashing methods can be grouped into two categories, i.e., data-independent hashing methods and data-dependent hashing methods. In data-independent hashing methods, hashing functions are mostly defined by random projection or manually constructed, e.g., locality sensitive hashing (LSH) [10]. Compared with data-dependent hashing methods, data-independent hashing methods are usually incapable of generating compact hashing codes on the short code length [14]. In recent years, most hashing algorithms are designed in data-dependent manner.

Generally, the training data is utilized in two different aspects in data-dependent methods, including unsupervised and supervised ways. Representative unsupervised hashing methods include iterative quantization hashing (ITQ) [11] and ordinal embedding hashing (OEH) [25]. Both of them explore the metric structure between the training data and thus retrieve the neighbors. Though learning in an unsupervised manner avoids the demand of the annotated data, exploiting the available supervisory information usually implies a better hashing code. For instance, supervised hashing with kernels (KSH) [27] employs a kernel function to optimize the Hamming distance of data pairs. Other supervised hashing methods based on hand-crafted features, including latent factor hashing (LFH) [40] and column-sampling based discrete supervised hashing (COSDISH) [15], also exhibit impressive results.

Moreover, benefit from deep neural networks, deep supervised hashing methods have made great progress [17, 4, 6, 37, 12, 32, 29, 1, 3]. Convolutional neural network hashing (CNNH) [35] is one of the early deep supervised hashing methods, which learns features and hashing codes in two separate stages. On the contrary, deep pairwise-supervised hashing (DPSH) [21] integrates the feature learning stage and hashing optimization stage in an end-to-end framework. Recently, adversarial networks [8, 2, 19, 9, 41, 34] and reinforcement learning [39] are also applied to hashing learning. However, as far as we know, no work has been done to specifically study and address the redundancy problem in deep supervised hashing.

2.2. Network Redundancy Reducing

Reducing the redundancy of neural networks is a well studied topic. Some early works can date back to [28, 18], in which they remove the least relevant units in a network. More recently, [24] removes entire neurons based on the idea that neurons have little influence on the output of the network should be removed. Similarly, in [38], they measure the importance of neurons in the final response layer and propose Neuron Importance Score Propagation to propagate the importance to every neuron in the network. Although these methods take the saliency of individual parameters or neurons into consideration, they zero out parameters or neurons in the network to reduce the number of parameters complexity. However, by and large, our problem setting is finding out the redundancy in output hashing bits and thus merging the output neurons. It can achieve a better performance when compared with the network that contains same hashing bits.

3. Preliminaries and Notations

3.1. Notation

We use uppercase letters like A to denote matrices and use a_{ij} to denote the (i, j)-th element in matrix A. The transpose of A is denoted by $A^\top$. $\text{sgn}(\cdot)$ is used to denote the element wise sign function, which returns 1 if the element is positive and returns -1 otherwise.

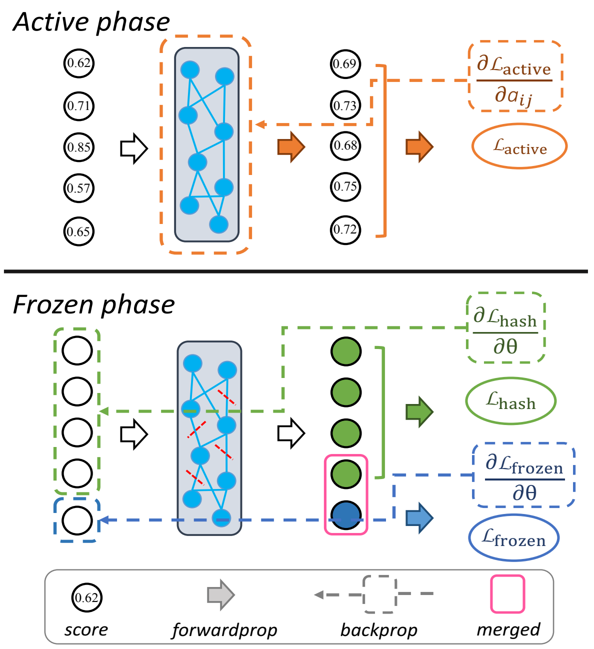

Active phase
0.62
0.71
0.85
0.57
0.65
0.69
0.73
0.68
0.75
0.72
$\frac{\partial \mathcal{L}_{\text{active}}}{\partial a_{ij}}$
$\mathcal{L}_{\text{active}}$
Frozen phase
$\frac{\partial \mathcal{L}_{\text{hash}}}{\partial \theta}$
$\mathcal{L}_{\text{hash}}$
$\frac{\partial \mathcal{L}_{\text{frozen}}}{\partial \theta}$
$\mathcal{L}_{\text{frozen}}$
0.62
score
forwardprop
backprop
merged

Figure 3: Illustration of the two phases of our NMLayer. In the active phase, we calculate the score of each neuron and then utilize the active loss (Eq. (3)) to update the redundancy relationship, i.e., the adjacency matrix. In the frozen phase, after truncating the adjacency matrix, the hashing loss (Eq. (7)) and the frozen loss (Eq. (5)) are used to update corresponding neurons.

3.2. Problem Definition

Suppose we have n images denoted as $X = \{x_i\}_{i=1}^n$, where x_i denotes the i-th image. Furthermore, the pairwise supervisory similarity is denoted as $S = \{s_{ij}\}$. $s_{ij} \in \{-1, +1\}$, where $s_{ij} = -1$ means x_i and x_j are dissimilar images and $s_{ij} = +1$ means x_i and x_j are similar images.

Deep supervised hashing aims at learning a binary code $b_i \in \{-1, +1\}^K$ for each image x_i, where K is the length of binary codes. $B = \{b_i\}_{i=1}^n$ denotes the set of all hashing codes. The Hamming distance of the learned binary codes of image x_i and x_j should keep consistent with the similarity attribute s_{ij}. That is, similar images should have shorter Hamming distances, while dissimilar images should have longer Hamming distances.

4. Neurons Merging Layer

In this section, we describe the details of NMLayer, which aims at balancing the importance of each hashing output neuron. A NMLayer has two phases during the training process, namely the active phase and the frozen phase, as shown in Figure 3. Basically, when a NMLayer is initially attached after a hashing output layer, it is set in the active phase to learn the redundancy relationship, i.e., the adjacency matrix, between different hashing bits. After enough updating on the weights through backpropagation, we truncate the adjacency matrix to determine which neurons to merge. Then, the NMLayer is set to frozen phase to learn hashing bits. The above process can be iterated for several times until the final output of the network reaches the required hashing bits. Actually, the learning process of a NMLayer is constructing a graph G. The neurons of a NMLayer are nodes, while the adjacency matrix A denotes the set of all edges. In the remainder of this section, we begin with presenting the basic structure of the NMLayer and then introduce the different policies of forward and backward in the active and frozen phases. Next, we define the behavior of the NMLayer when the neural network is in the evaluation mode. Finally, we introduce our progressive optimization strategy in detail.

4.1. Structure of the NMLayer

As mentioned above, a NMLayer is basically a graph G with learnable adjacency matrix A. Note that G is an undirected graph, i.e., $a_{ij} = a_{ji}$. The nodes of G are denoted by $\mathcal{V}$, which is a set of hashing bits. Specifically, the value type of A differs in two phases. During the active phase, A is learned through backpropagation and $A \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, where $|\mathcal{V}|$ means the number of nodes. Each element a_{ij} in A denotes the necessity whether the two nodes v_i and v_j should be merged as one single node. After entering frozen phase, the graph structure is fixed, that is A becomes fixed and now $A \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{V}|}$, where $a_{ij} = 1$ means that the i-th and j-th neurons are merged, while $a_{ij} = 0$ means the opposite.

4.2. Active Phase

When a NMLayer is first attached and initialized, all the elements in A are set to 0, which indicates that no nodes are currently merged or inclined to be merged. In the active phase, our target is to find out which nodes should be merged together, based on a simple intuition that all nodes, i.e., all hashing bits, should carry equal information about the input data. In our NMLayer, the principle is restated in a practical way that eliminating any single hashing bit should lead to an equal decline of performance, thus no redundancy in the final hashing bits. Next, we elaborate on how to evaluate the importance of neurons in a typical forward pass of neural networks.

Forward. Suppose the size of a mini-batch in a forward pass is N, the number of neurons is K, and the neurons are $\{v_1, \dots, v_K\}$. In each forward pass, scores that evaluating the importance of each neuron are computed for the next backward pass. More precisely, for each neuron we compute the retrieval precision, i.e., Mean Average Precision (MAP), after eliminating it. We denote the input of the

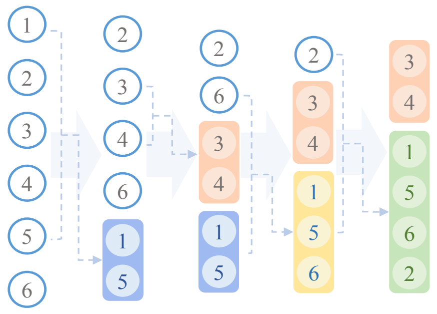


Figure 4: A demonstration of the merging process from six neurons to two neurons. Two neurons are merged at each step.

mini-batch as $\{X_n\}_{n=1}^{N}$ and the validation set as $\mathcal{Y}$, then the score s_k of the k-th neuron is computed as

$$s_k = \text{Prec}_k(X_n, \mathcal{Y}), \tag{1}$$

where the function $\text{Prec}(X_n, \mathcal{Y})$ means computing the precision with query $\mathcal{Y}$ and gallery X_n, and the subscript k means computing precision without the k-th hashing bit. Recall that in the active phase, elements in A imply the necessity of whether two nodes in the graph should be merged. In the forward pass, we take A into consideration to calculate new scores $\{s'_k\}_{k=1}^{K}$, that is

$$s'_i = s_i + \frac{1}{2}\sum_{i\neq j} a_{ij}(s_j - s_i). \tag{2}$$

The intuition of the above equation is that if s_i and s_j are merged together, i.e., $a_{ij} = 1$, then s'_i is equal to s'_j. Next, we update A according to the $\{s'_k\}_{k=1}^{K}$ in the following backward pass.

Backward. In order to update A through backpropagation, a loss function $\mathcal{L}_{\text{active}}$ is defined on $\{s'_k\}_{k=1}^{K}$. The principle of the loss function is to determine the inequality between neurons. Therefore, a feasible and straightforward loss function is

$$\mathcal{L}_{\text{active}} = \sum_{i\neq j} |s'_i - s'_j|. \tag{3}$$

In fact, by Eq. (3), the derivative of $\mathcal{L}_{\text{active}}$ with respect to a_{ij} is

$$\frac{\partial \mathcal{L}_{\text{active}}}{\partial a_{ij}} = \text{sgn}(s'_i - s'_j)\cdot(s_j - s_i). \tag{4}$$

Observing that the value of derivative depends on $s_j - s_i$. It can be interpreted that the more different the two nodes are, the higher necessity the two nodes should be merged.

Algorithm 1: Progressively training multiple Neurons Merging Layers.

```
input : Training set $X$, validation set $\mathcal{Y}$, initial
        hashing bits $B_{\text{in}}$, truncation parameter $m$,
        iteration number $N_0$, $N_1$.
output: The network with hashing bits $B_{\text{out}}$.
1  Initialize the backbone network $F^{(1)}$;
2  for $t = 1, \dots, T$ do
3      Attach NMLayer $G^{(t)}$ after $F^{(t)}$:
4          $F^{(t)} \leftarrow F^{(t)} + G^{(t)}$;
5      Set $G^{(t)}$ in the active phase;
6      for $i = 1, \dots, N_0$ do
7          Train $F^{(t)}$ by Eq. (3), Eq. (7);
8      end
9      Truncate $G^{(t)}$;
10     Set $G^{(t)}$ in the frozen phase;
11     for $j = 1, \dots, N_1$ do
12         Train $F^{(t)}$ by Eq. (5), Eq. (7);
13     end
14 end
```

4.3. Truncation of the Adjacency Matrix

With A being updated for several epochs, we then perform a truncation on A to merge neurons. After truncating A, all the elements in A are either 0 or 1. Nodes with an adjacency value of 1 will be merged to reduce redundancy. Note that, the strategy of truncation is various and we just use a straightforward one. We turn the maximum m values in A to 1, and the others to 0.

4.4. Frozen Phase

If all the values in matrix A are 0, that is the NMLayer neither trained nor truncated, both of the forward pass and backward pass are same as a normal deep supervised hashing network. When some elements in A are 1, it means the corresponding nodes are merged together. The new merged node that consists of several child nodes has new forward and backward strategies. Here, we illustrate our strategy with a simple case. Suppose that two nodes $\{v_1, v_2\}$ have been merged together after truncation, i.e., $a_{12} = 1$. Therefore, the length of the output hashing bits is now $K-1$, and we denote the new node as v_{12}, then the new output hashing bits are $\{v_{12}, v_3, \dots, v_K\}$.

Forward. We randomly choose one child node from the new merged node as the output in the forward pass. Note that, choosing a random child node or a fixed order child node as the output has little impact on the retrieval precision, which is demonstrated in Table 2. In our simple example, suppose v_1 is randomly chosen, so the output of v_{12} is equal to v_1.

Backward. For the child node chosen as output in the forward pass, the gradient in the backward pass is simply calculated by the loss of hashing networks, such as a pairwise hashing loss like Eq. (7). As for those child nodes not chosen in the new merged node, we set a target according to the sign of the output of the chosen child node. In our simple example, the gradient of v_1 is calculated according to the pairwise hashing loss, while the gradient of v_2 is computed by $||v_2 - \text{sgn}(v_1)||^2$. The intuition that not directly using the same gradient as v_1 is to reduce the correlation between the neurons. More generally, for all of the child nodes in the new merged node expect v_j chosen in the forward pass, the loss function is defined as

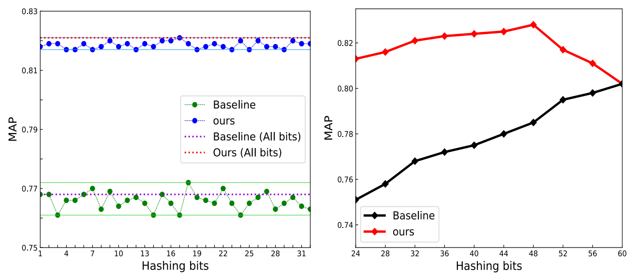
$$\mathcal{L}_{\text{frozen}} = \sum_{i \neq j} ||v_i - \text{sgn}(v_j)||^2. \quad (5)$$

4.5. Evaluation Mode

When the whole network is set in evaluation mode, we no longer choose the output of a merged node in a random manner. Instead, we compute the output of the merged node by majority-voting. Again, using the simple example above, the output of v_{12} depends on $\text{sgn}(v_1)$ and $\text{sgn}(v_2)$. That is, if $\text{sgn}(v_1) = \text{sgn}(v_2) = +1$, then $v_{12} = +1$. Note that when $\text{sgn}(v_1) = +1$ and $\text{sgn}(v_2) = -1$, then $v_{12} = 0$, which implies that the output of v_{12} is uncertain. In this paper, we directly calculate the Hamming distance without considering this particular case and leave this study for our future pursuit.

4.6. Progressive Optimization Strategy

By progressively training multiple NMLayers, we merge the output neurons of a deep hashing network as shown in Figure 2. Meanwhile, the detailed algorithm is shown in Algorithm 1. It should be emphasized that in the training process, we only update the graph in a limited number of iterations. In addition, during evaluation, the graph is fixed and no more calculations are required. Therefore, the calculation of graph has little influence on the running time of the whole algorithm. Note that we use multiple NMLayers instead of one because merging too many neurons at once will degrade algorithm performance, which is reported in Figure 7b. By performing the algorithm, we aim to get a network with B_{out} hashing bits from a backbone network F with B_{in} hashing bits. Hyper-parameters in the algorithm are shown as follow: m means turning the maximum m values of the adjacency matrix to 1 and the others to 0, which is defined in the truncation operation; the active phase and frozen phase are trained by N_0 and N_1 epochs respectively. For better understanding, we show a simple example in Figure 4, in which the hashing network has 6 output neurons at the beginning and merges 2 neurons per step.



(a) (b)

Figure 5: Analyses of redundancy in hashing bits. (a) Redundancy comparison between our method and baseline; (b) Bit reduction process. The red line denotes the MAP results during progressively reducing hashing bits from 60 to 24 (see from right to left). The black line denotes the MAP results of baseline when training the same fixed length hashing bits.

5. Experiments

5.1. Experimental Details

Pairwise Hashing Loss. Following the optimization method in [27], we keep the similarity s_{ij} between images x_i and x_j by optimizing the inner product of b_i and b_j:

$$\begin{aligned} \min_{B} \quad & \mathcal{L}_{\text{hash}} = \sum_{i=1}^{m}\sum_{j=1}^{n}(b_i^\top b_j - K s_{ij})^2 \\ \text{s.t.} \quad & b_i, b_j \in \{-1,+1\}^K \end{aligned} \tag{6}$$

where K denotes the length of hashing bits. m and n are the numbers of query images and retrieval images, respectively. Obviously, the problem in Eq. (6) is a discrete optimization problem, which is difficult to solve. Note that for the input image x_i, the output of our neural network is denoted by $u_i = F(x_i, \theta)$ (θ is the parameter of our neural network), and the binary hashing code b_i is equal to sgn(u_i). In order to solve the discrete optimization problem, we replace the binary b_i with continuous u_i, and add a L_2 regularization term as [21]. Then, the reformulated loss function can be written as

$$\begin{aligned} \min_{U,\Theta} \quad & \mathcal{L}_{\text{hash}} = \sum_{i=1}^{m}\sum_{j=1}^{n}(u_i^\top u_j - K s_{ij})^2 + \eta \sum_{i=1}^{n} \|b_i - u_i\|_2^2 \\ \text{s.t.} \quad & u_i, u_j \in \mathbb{R}^{K\times 1}, b_i = \text{sgn}(u_i) \end{aligned} \tag{7}$$

where η is a hyper-parameter and Eq. (7) is used as our basic pairwise hashing loss.

Parameter Settings. In order to make a fair comparison with previous deep supervised hashing methods [21, 20, 13], we adopt CNN-F network [5] pre-trained on ImageNet dataset [30] as the backbone of our method. The last fully connected layer of the CNN-F network is modified to hashing layer to output binary hashing bits. The parameters in our algorithm are experimentally set as follows. The number of neurons B_{in} in hashing layer is set to 60. In addition, the number of truncating edges in per step, i.e. m, is set to 4. During training, we set the batch size to 128 and use Stochastic Gradient Descent (SGD) with 10^{-4} learning rate and 10^{-5} weight decay to optimize the backbone network. Then, the learning rate of NMLayer and the hyper-parameter η in Eq. (7) are set to 10^{-2} and 1200 respectively. Moreover, the parameters N_0 and N_1 are set to 5 and 40 respectively.

Datasets. Following paper [21, 3, 13], we evaluate our method on four datasets, including CIFAR-10 [16], NUS-WIDE [7], MS-COCO [23] and Clothing1M [36].

CIFAR-10 contains 60,000 32 $\times$ 32 color images. It is a single-label dataset. The division of our query set and database set are same with [21]. In addition, since a validation set is needed to calculate neuron scores in the active phase, we randomly select 200 images from the training set as our validation set and use the rest 4,800 images as our training set.

NUS-WIDE contains 269,648 images collected from web. It is a multi-label dataset, in which each image belongs to multiple classes. Our query set and database set are same with [21]. Moreover, we randomly divide the training set into 420 images and 10,080 images as our validation set and our training set respectively.

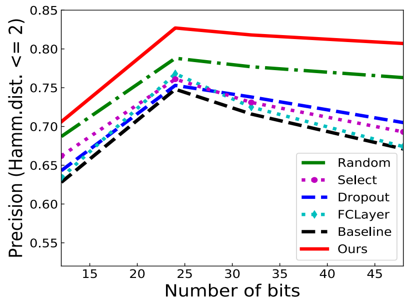
MS-COCO contains 82,783 training images. It is a multi-label dataset. The division of our query set and database set are same with [14]. Then, 400 images are randomly selected from the training set as our validation set and the rest 9,600 images are used as our training set.

Clothing1M is a million-level large-scale dataset that contains 1,037,497 images. It is a single-label dataset. The division of our query set (7,000 images) and database set (about 1,020,000 images) are same with [13]. In addition, 280 images are randomly selected from the training set as our validation set and the rest 13,720 images are used as our training set.

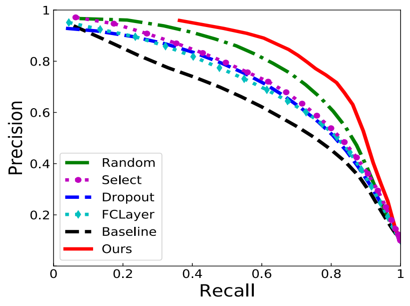
Evaluation Methodology. We use Mean Average Precision (MAP) to evaluate retrieval performance. For the single-label CIFAR-10 and Clothing1M datasets, images with the same label are considered to be similar ($s_{ij} = 1$). For the multi-label NUS-WIDE and MS-COCO datasets, two images are considered to be similar ($s_{ij} = 1$) if they share at least one common label. Specially, the MAP of the NUS-WIDE dataset is calculated based on the top 5,000 returned samples [21, 20].

Table 1: MAP of different methods on CIFAR-10, NUS-WIDE, MS-COCO and Clothing1M datasets. *Ours* denotes the results when B_{in} is equal to 60, while *Ours** denotes the results when B_{in} is equal to compared methods (12, 24, 32 and 48 respectively).

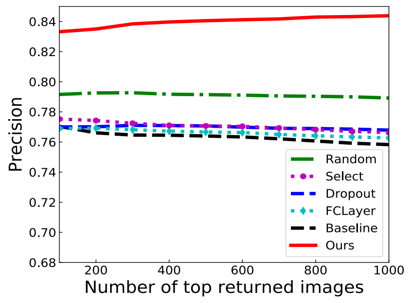
Method	CIFAR-10				NUS-WIDE				MS-COCO				Clothing1M			
	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bit
Ours	**0.786**	**0.813**	**0.821**	**0.828**	**0.801**	**0.824**	**0.832**	**0.840**	**0.754**	**0.772**	**0.777**	**0.782**	**0.311**	**0.372**	**0.389**	**0.401**
*Ours**	0.750	0.797	0.813	0.825	0.774	0.812	0.827	0.832	0.744	0.769	0.775	0.780	0.268	0.343	0.377	0.396
DDSH	0.753	0.776	0.803	0.811	0.776	0.803	0.810	0.817	0.745	0.765	0.771	0.774	0.271	0.332	0.343	0.346
DSDH	0.740	0.774	0.792	0.813	0.774	0.801	0.813	0.819	0.743	0.762	0.765	0.769	0.278	0.302	0.311	0.319
DPSH	0.712	0.725	0.742	0.752	0.768	0.793	0.807	0.812	0.741	0.759	0.763	0.771	0.193	0.204	0.213	0.215
DSH	0.644	0.742	0.770	0.799	0.712	0.731	0.740	0.748	0.696	0.717	0.715	0.722	0.173	0.187	0.191	0.202
DHN	0.680	0.721	0.723	0.733	0.771	0.801	0.805	0.814	0.744	0.765	0.769	0.774	0.190	0.224	0.212	0.248
COSDISH	0.583	0.661	0.680	0.701	0.642	0.740	0.784	0.796	0.689	0.692	0.731	0.758	0.187	0.235	0.256	0.275
SDH	0.453	0.633	0.651	0.660	0.764	0.799	0.801	0.812	0.695	0.707	0.711	0.716	0.151	0.186	0.194	0.197
FastH	0.597	0.663	0.684	0.702	0.726	0.769	0.781	0.803	0.719	0.747	0.754	0.760	0.173	0.206	0.216	0.244
LFH	0.417	0.573	0.641	0.692	0.711	0.768	0.794	0.813	0.708	0.738	0.758	0.772	0.154	0.159	0.212	0.257
ITQ	0.261	0.275	0.286	0.294	0.714	0.736	0.745	0.755	0.633	0.632	0.630	0.633	0.115	0.121	0.122	0.125



(a)



(b)



(c)

Figure 6: The comparison results on CIFAR-10 dataset. (a) Precision curves within Hamming distance 2; (b) Precision-recall curves of Hamming ranking with 32 bits; (c) Precision curves with 32 bits w.r.t. different numbers of top returned samples.

5.2. Experimental Results

We compare our method with several state-of-the-art hashing methods, including one unsupervised method ITQ [11]; four non-deep supervised methods, COSDISH [15], SDH [31], FastH [22] and LFH [40]; five deep supervised methods, DDSH [20], DSDH [20], DPSH [21], DSH [26], and DHN [42].

In Table 1, the MAP results of all methods on CIFAR-10, NUS-WIDE, MS-COCO and Clothing1M datasets are reported. For fair comparison, the results of DDSH, DSDH and DPSH come from rerunning the released codes under the same experimental setting, while other results are directly reported from previous works [14, 13]. As we can see from Table 1, compared with the unsupervised hashing method, the supervised methods achieve better results in most cases. Meanwhile, in all supervised hashing methods, the deep hashing methods outperform the non-deep hashing methods. Furthermore, our method outperforms all other methods on all datasets, which validates the effectiveness of our method. Specifically, compared to the state-of-the-art deep supervised hashing method DDSH, our method achieves increases of **3.36%**, **2.83%**, **0.98%** and **13.97%** in average performance with different code lengths on CIFAR-10, NUS-WIDE, MS-COCO and Clothing1M, respectively. It is obviously that our improvement is significant especially on large-scale Clothing1M.

Besides, in Table 1, we also report the results of the setting that B_{in} is equal to compared hashing methods, which are denoted as *ours**. Specifically, we set the number of initial neurons of hashing layer to 12, 24, 32 and 48 respectively, then the redundancy of these hashing bits is reduced by our method. From Table 1, we can observe that our method is superior to all other hashing methods with the setting of 24, 32 and 48 bits, except 12 bits. The reason behind this phenomenon is that the redundancy of short code is essentially low.

5.3. Experimental Analyses

Analyses of Redundancy in Hashing Bits. On CIFAR-10 dataset, we train a 32-bits hashing network without NM-Layer as a *Baseline* based on Eq. (7). Then, in order to show the redundancy of hashing bits, we remove a bit per time and report the final MAP with our method and the baseline method in Figure 5a. It is clearly observed that compared to baseline, the variance of MAP of our algorithm is much lower, thus we can come to the conclusion that the redundancy in hashing bits has been reduced. In addition, as the redundancy is reduced, each bit of hashing codes can be fully utilized. Therefore, the retrieval precision of our method is greatly improved.

As shown in Figure 5b, compared with the baseline results trained on the fixed length hashing bits, we record the changes of MAP during progressively reducing hashing bits from 60 to 24. As we can see from Figure 5b, the MAP value of our method increases from 60 to 48 bits. At the same time, the curve of our method is more stable, while the baseline curve drops rapidly. Both of these phenomenons are due to the effective redundancy reduction of our approach. Finally, the MAP curve of our method reaches its maximum value at 48 bits. Therefore, we consider 48 as the most appropriate code length on CIFAR-10 dataset. Inspired by this insight, our approach can also be conducive to finding the most appropriate code length while reducing the redundancy.

Comparisons with Other Variants. In order to further verify the effectiveness of our method, we elaborately design several variants of our method. Firstly, *Random* is a variant of our method without active phase. It replaces the dynamic learning adjacency matrix in the active phase with a random matrix. Secondly, *Select* is a variant of our method without frozen phase. It directly selects the most important bits as the final output instead of merging them. Thirdly, considering that the dropout technique [33] is widely adopted in neural networks to reduce the correlation between neurons, we add a dropout layer before the hashing layer to reduce the correlation of hashing bits and denote it as *Dropout*. Finally, since the process of our neurons merging can be viewed as a process of dimension reduction, we design a variant *FCLayer* to compare the differences between our NMLayer and the fully connected layer. It replaces the NMLayer with a fully connected layer, which is optimized by loss function Eq. (7).

The above variants are compared using three widely used evaluation metrics as [35]: Precision curves within Hamming distance 2, Precision-recall curves and Precision curves with different numbers of top returned samples. The results of above variants are reported in Figure 6. From Figure 6 we can see that compared to our method, the performance of both *Random* and *Select* has declined. It demonstrates the validity of our active and frozen phases. In addition, the improvements of *Dropout* and *FCLayer* over *Baseline* are small, which proves the effects of the dropout technique and the fully connected layer are limited to the hashing retrieval.

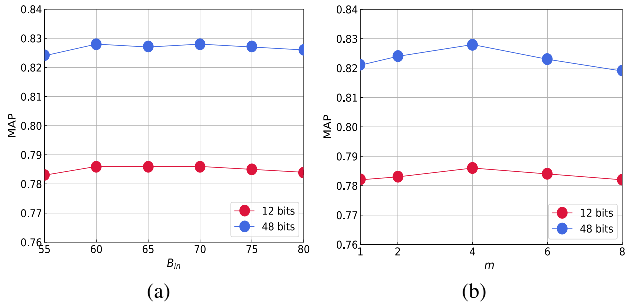


Figure 7: Sensitivity study on CIFAR-10 dataset.

Table 2: Different node selection strategy comparison.

Method	CIFAR-10			
	12 bits	24 bits	32 bits	48 bits
1-st	0.786	0.812	0.818	0.826
2-nd	0.785	0.810	0.820	0.828
Random one	0.786	0.813	0.821	0.828

Sensitivity to Parameters. Table 2 presents the results of different node selection strategies in the frozen phase. In the new merged node, we respectively select the first, second and random child nodes as our output in the forward pass. It is obvious that choosing which child node as the output has little impact on the MAP. In addition, Figure 7a and Figure 7b present the effects of hyper-parameters B_{in} and m respectively. Note that increasing the number of B_{in} dose not obviously improve the retrieval accuracy. It is due to that 60 bits already have enough expression capacity and extra neurons are saturated. Moreover, the retrieval results decrease when m is too large, which demonstrates that merging too many neurons at once will degrade the performance of our algorithm. It also explains the necessity of our progressive optimization strategy.

6. Conclusion

In this paper, we analyze the redundancy of hashing bits in deep supervised hashing. To address this, we construct a graph to represent the redundancy relationship and propose a novel layer named NMLayer. The NMLayer merges the redundant neurons together to balance the importance of each hashing bit. Moreover, based on the NMLayer, we propose a progressive optimization strategy. A deep hashing network is initialized with more hashing bits than the required bits, and then multiple NMLayers are progressively trained to learn a more compact hashing code from a redundant long code. Our improvement is significant especially on large-scale datasets, which is verified by comprehensive experimental results.

References

[1] F. Cakir, K. He, and S. Sclaroff. Hashing with binary matrix pursuit. In *ECCV*, 2018.

[2] Y. Cao, B. Liu, M. Long, J. Wang, and M. KLiss. Hashgan: Deep learning to hash with pair conditional wasserstein gan. In *CVPR*, 2018.

[3] Y. Cao, M. Long, B. Liu, J. Wang, and M. KLiss. Deep cauchy hashing for hamming space retrieval. In *CVPR*, 2018.

[4] Y. Cao, M. Long, J. Wang, H. Zhu, and Q. Wen. Deep quantization network for efficient image retrieval. In *AAAI*, 2016.

[5] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014.

[6] Z. Chena, X. Yuana, J. Lua, Q. Tiand, and J. Zhoua. Deep hashing via discrepancy minimization. In *CVPR*, 2018.

[7] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. Nus-wide: a real-world web image database from national university of singapore. In *CIVR*, 2009.

[8] C. Du, X. Xie, C. Du, and H. Wang. Redundancy-resistant generative hashing for image retrieval. In *IJCAI*, 2018.

[9] K. Ghasedi Dizaji, F. Zheng, N. Sadoughi, Y. Yang, C. Deng, and H. Huang. Unsupervised deep generative adversarial hashing network. In *CVPR*, 2018.

[10] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[11] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[12] K. He, F. Cakir, S. A. Bargal, and S. Sclaroff. Hashing as tie-aware learning to rank. In *CVPR*, 2018.

[13] Q.-Y. Jiang, X. Cui, and W.-J. Li. Deep discrete supervised hashing. *TIP*, 27(12):5996–6009, 2018.

[14] Q.-Y. Jiang and W.-J. Li. Asymmetric deep supervised hashing. In *AAAI*, 2018.

[15] W.-C. Kang, W.-J. Li, and Z.-H. Zhou. Column sampling based discrete supervised hashing. In *AAAI*, 2016.

[16] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, University of Toronto*, 2009.

[17] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, 2015.

[18] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *NIPS*, 1990.

[19] C. Li, C. Deng, N. Li, W. Liu, X. Gao, and D. Tao. Self-supervised adversarial hashing networks for cross-modal retrieval. In *CVPR*, 2018.

[20] Q. Li, Z. Sun, R. He, and T. Tan. Deep supervised discrete hashing. In *NIPS*, 2017.

[21] W.-J. Li, S. Wang, and W.-C. Kang. Feature learning based deep supervised hashing with pairwise labels. In *IJCAI*, 2016.

[22] G. Lin, C. Shen, Q. Shi, A. Van den Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *CVPR*, 2014.

[23] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[24] B. Liu, M. Wang, H. Foroosh, M. F. Tappen, and M. Pensky. Sparse convolutional neural networks. In *CVPR*, 2015.

[25] H. Liu, R. Ji, Y. Wu, and W. Liu. Towards optimal binary code learning via ordinal embedding. In *AAAI*, 2016.

[26] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *CVPR*, 2016.

[27] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, 2012.

[28] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *NIPS*, 1989.

[29] Q. Qiu, J. Lezama, A. Bronstein, and G. Sapiro. Foresthash: Semantic hashing with shallow random forests and tiny convolutional networks. In *ECCV*, 2018.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

[31] F. Shen, C. Shen, W. Liu, and H. Tao Shen. Supervised discrete hashing. In *CVPR*, 2015.

[32] Y. Shen, L. Liu, F. Shen, and L. Shao. Zero-shot sketch-image hashing. In *CVPR*, 2018.

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

[34] G. Wang, Q. Hu, J. Cheng, and Z. Hou. Semi-supervised generative adversarial hashing for image retrieval. In *ECCV*, 2018.

[35] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, 2014.

[36] T. Xiao, T. Xia, Y. Yang, C. Huang, and X. Wang. Learning from massive noisy labeled data for image classification. In *CVPR*, 2015.

[37] P. Xu, Y. Huang, T. Yuan, K. Pang, Y.-Z. Song, T. Xiang, T. M. Hospedales, Z. Ma, and J. Guo. Sketchmate: Deep hashing for million-scale human sketch retrieval. In *CVPR*, 2018.

[38] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

[39] X. Yuan, L. Ren, J. Lu, and J. Zhou. Relaxation-free deep hashing via policy gradient. In *ECCV*, 2018.

[40] P. Zhang, W. Zhang, W.-J. Li, and M. Guo. Supervised hashing with latent factor models. In *SIGIR*, 2014.

[41] X. Zhang, H. Lai, and J. Feng. Attention-aware deep adversarial hashing for cross-modal retrieval. In *ECCV*, 2018.

[42] H. Zhu, M. Long, J. Wang, and Y. Cao. Deep hashing network for efficient similarity retrieval. In *AAAI*, 2016.