Maximum-Margin Hamming Hashing

Rong Kang, Yue Cao, Mingsheng Long (✉), Jianmin Wang, and Philip S. Yu
School of Software, BNRist, Tsinghua University
Research Center for Big Data, Tsinghua University
National Engineering Laboratory for Big Data Software
kangr15@mails.tsinghua.edu.cn, {mingsheng,jimwang}@tsinghua.edu.cn

Abstract

Deep hashing enables computation and memory efficient image search through end-to-end learning of feature representations and binary codes. While linear scan over binary hash codes is more efficient than over the high-dimensional representations, its linear-time complexity is still unacceptable for very large databases. Hamming space retrieval enables constant-time search through hash lookups, where for each query, there is a Hamming ball centered at the query and the data points within the ball are returned as relevant. Since inside the Hamming ball implies retrievable while outside irretrievable, it is crucial to explicitly characterize the Hamming ball. The main idea of this work is to directly embody the Hamming radius into the loss functions, leading to Maximum-Margin Hamming Hashing (MMHH), a new model specifically optimized for Hamming space retrieval. We introduce a max-margin t-distribution loss, where the t-distribution concentrates more similar data points to be within the Hamming ball, and the margin characterizes the Hamming radius such that less penalization is applied to similar data points within the Hamming ball. The loss function also introduces robustness to data noise, where the similarity supervision may be inaccurate in practical problems. The model is trained end-to-end using a new semi-batch optimization algorithm tailored to extremely imbalanced data. Our method yields state-of-the-art results on four datasets and shows superior performance on noisy data.

1. Introduction

Recent years we have witnessed the significant growth in computer vision applications that generate large-scale and high-dimensional visual data online. Thus, developing multimedia retrieval method of high efficiency and accuracy is a popular topic in both academic and industrial communities. Approximate Nearest Neighbor (ANN) search, which can well compromise the retrieval accuracy and computational efficiency, has attracted increasing attention. There are two solutions to ANN search: indexing [22] and hashing [46]. Hashing methods aim to convert high-dimensional visual data into compact binary codes while preserving their similarity relationship in the Hamming space. This work focuses on learning to hash [46], a data-dependent hashing scheme for efficient image retrieval that has shown better performance than data-independent methods. Seminal work of learning to hash includes Local Sensitive Hash (LSH) [11] and Spectral Hashing (SH) [47], to name a few.

A rich line of learning to hash methods have been proposed for efficient ANN search, which enables Hamming ranking over compact binary hash codes of database points and user queries. These methods mainly fall into unsupervised methods [20, 13, 37, 33, 51, 34] and supervised methods [30, 32, 52]. Recently, deep learning to hash methods [49, 21, 41, 10, 7, 4, 29, 43, 16, 17, 27] have employed deep networks for end-to-end learning of feature representations and binary codes, through which non-linear hash functions are readily supported. These methods have achieved state-of-the-art retrieval performance, validating the importance of jointly learning similarity-preserving representations and controlling quantization error of converting the continuous representations into binary codes [53, 25, 28, 4].

Most previous deep hashing methods focus on maximizing the retrieval performance for a linear scan over the whole database of N data points. The time complexity of linear scan over K-bit hash codes is $O(NK/8)$, much more efficient than $O(ND)$, the time complexity of linear scan over D-dimensional continuous representations ($D \gg K$). However, the linear complexity is still unacceptable for very large databases. In this paper, we focus on Hamming space retrieval [37, 2] that enables constant-time search through *hash lookups*, where for each query, it returns data points within a Hamming ball of radius H. As is widely known, a remarkable advantage of hashing is that any bucket in the hash table can be queried in $O(1)$ by table lookups. For binary codes of K bits, the number of distinct hash buckets to be examined increases exponentially w.r.t. H [37]. And empirically, we can find all H-neighbors of a given query in

constant time only when $H \leq 2$. Hence, we concentrate on optimizing the search efficiency and accuracy of Hamming space retrieval based on Hamming radius 2.

The key observation of Hamming space retrieval is that, for each query, only points inside the Hamming ball will be returned as relevant, while those outside are pruned directly. Thus, it requires explicit discrimination between inside and outside of the Hamming ball, by explicitly characterizing the *Hamming ball* in loss functions. Take the state-of-the-art Deep Cauchy Hashing (DCH) [2] as an example, without characterizing the Hamming ball, the loss of similar pairs within the Hamming ball takes a large fraction of the total loss but is not beneficial to candidate pruning, resulting in a biased model for Hamming space retrieval.

The robustness to noise in data labels is also important for supervised hashing. In real applications, the similarity information is collected from the semantic labels or implicit feedback from click-through data. Due to human subjectivity or the lack of expertise, the similarity supervision may be inaccurate. In this setting, similar images might be wrongly labeled as dissimilar, or vice versa. Unreliable training data may result in very deficient hashing models. Previous methods [53, 4, 2] typically assign large loss on image pairs with "dissimilar" labels but with similar hash codes, regardless that the large loss may stem from label noise. Consequently, these models are vulnerable to noisy data.

Towards the above problems, this paper proposes a max-margin t-distribution loss, where the t-distribution concentrates more similar points to be within the Hamming ball, and the margin characterizes the Hamming radius such that less penalization is imposed to similar points falling inside the Hamming ball. With the margin idea, the loss function also introduces robustness to data noise in that the image pairs with wrong labels will not be overly penalized to deteriorate the model. We further propose a new semi-batch optimization algorithm tailored to extremely imbalanced data, a common scenario in practice that similarity information is much sparser than dissimilarity information. Comprehensive experiments demonstrate that MMHH can generate compact and noise-robust hash codes and yield state-of-the-art image retrieval performance on four benchmark datasets.

2. Related Work

Existing hashing methods can be roughly put into two categories: unsupervised hashing and supervised hashing. A comprehensive survey is provided in [46].

Unsupervised hashing methods encode data to binary codes by learning hash functions from unlabeled data based on reconstruction error minimization [40, 13] and graph learning [47, 20, 31, 54]. Supervised hashing methods incorporate supervised information (e.g., pairwise similarity) to mitigate the semantic gap and improve hashing quality. KSH [30] and SDH [41] generate nonlinear or discrete hash codes by minimizing the Hamming distances over similar pairs while maximizing the distances over dissimilar ones.

Deep hashing has attracted extensive attention recently due to the superior performance. CNNH [49] follows a two-stage strategy in which the first stage learns hash codes and the second stage learns a deep-network hash function to fit the codes. DNNH [21] improves CNNH with a simultaneous feature learning and hash coding framework such that deep representations and hash codes are optimized jointly. DSRH [28] learns deep hash functions from semantic ranking supervision. DHN [53] and DSDH [24] simultaneously preserve pairwise similarity and control quantization error. HashNet [4] improves DHN by balancing training pairs and achieving nearly zero quantization error using continuation. ADSH [17] treats queries and database points in an asymmetric way for more efficient training. DSEH [23] learns a deep joint semantic-embedding for hashing. BGDH [50] captures the underlying structure of data by constructing a bipartite graph. WDH [6] leverages social tag semantics to enhance hash codes quality. In addition, adversary training mechanism has been introduced to fully capture semantic information from unlabeled data [35, 42, 45, 8].

Unlike previous hashing methods [4, 24, 17] designed for linear scan performance, we focus on Hamming space retrieval that prunes all buckets outside the Hamming ball for each query. Thus it is crucial to concentrate more similar points to be within the Hamming ball. The only work in this line is Deep Cauchy Hashing (DCH) [2]. However, it treats all data pairs uniformly, no matter they are inside or outside the Hamming ball, contradicting the mechanism of candidate pruning using the Hamming ball as an absolute boundary. Different from these previous works, our work explicitly characterizes the Hamming ball by a new max-margin t-distribution loss, which is also robust to noisy data.

3. Maximum-Margin Hamming Hashing

3.1. Hamming Space Retrieval

In supervised image retrieval, we are given an image database $\mathcal{X}_D = \{\boldsymbol{x}_i\}_{i=1}^N$ and a subset $\mathcal{X}_T = \{\boldsymbol{x}_i\}_{i=1}^n$ of $\mathcal{X}_D$ as the training set. Image pairs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ ($\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}_T$) are provided with the pairwise similarity label s_{ij}, where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are similar if $s_{ij} = 1$, and dissimilar if $s_{ij} = 0$. In deep supervised hashing, we build a model by learning a nonlinear hash function $f : \boldsymbol{x} \mapsto \boldsymbol{h} \in \{-1, 1\}^K$ via deep networks such that the similarity relationship in $\mathcal{S} = \{s_{ij}\}$ can be preserved in the binary hash codes.

This paper focuses on Hamming space retrieval, which is a constant-time retrieval scenario for practical applications.

Definition 1. ***Hamming Space Retrieval*** *[2], also known as hash lookup search, refers to the retrieval scenario that directly returns data points within Hamming radius H to each query by hash lookups instead of linear scan.*

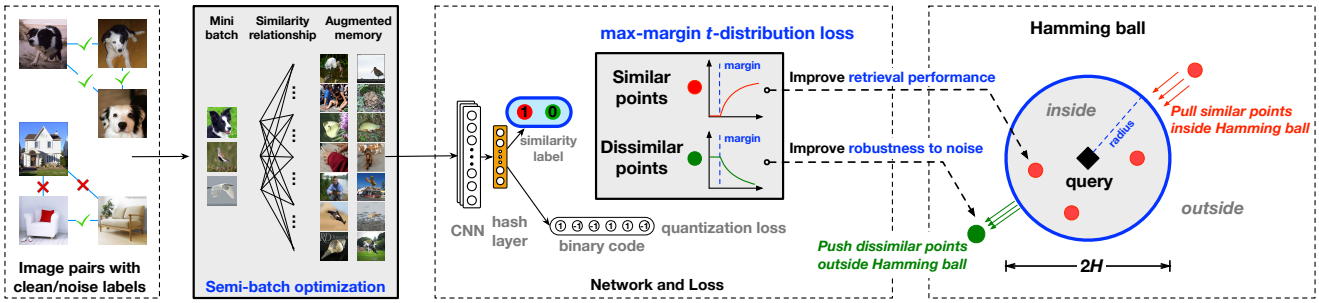


Figure 1. The architecture of MMHH consists of four components: (1) a convolutional network for learning image representations; (2) a fully-connected hash layer for converting image representations into K-bit hash codes; (3) a max-margin t-distribution loss that explicitly characterizes the Hamming ball by the margin for preserving similarity relationship in the Hamming space, and for introducing robustness to data noise; (4) a quantization loss for controlling the binarization error. The model is trained by a semi-batch optimization method for imbalanced data. The max-margin t-distribution loss differs significantly for similar and dissimilar pairs inside and outside Hamming ball.

For binary codes of K bits, the number of distinct hash buckets to be retrieved is $N(K,H) = \sum_{k=0}^{H} \binom{K}{k}$, where H is the **Hamming radius**. When the Hamming radius is small enough, typically $H \leq 2$, we can build a hash lookup table to find out all H-neighbors in constant time. However, $N(K,H)$ increases exponentially with H and it is unaffordable to retrieve $N(K,H)$ buckets when $H > 2$ [2]. For Hamming space retrieval, the critical challenge is to learn hash codes enabling effective and efficient pruning through Hamming ball of radius $H \leq 2$. We present Maximum-Margin Hamming Hashing (MMHH) towards this goal.

3.2. Network Architecture

The architecture of Maximum-Margin Hamming Hashing (MMHH) is illustrated in Figure 1. Similar to previous work [2], we replace the classifier layer in convolutional neural network (CNN) [19, 14] with a fully-connected *hash layer* together with a hyperbolic tangent ($\tanh$) function to transform the feature representation of each image $\boldsymbol{x}_i$ into K-dimensional continuous code $\boldsymbol{z}_i \in \mathbb{R}^K$. Finally, we obtain binary hash codes by $\boldsymbol{h}_i = \operatorname{sgn}(\boldsymbol{z}_i)$ where $\operatorname{sgn}(\boldsymbol{z})$ is the sign function. A novel max-margin t-distribution loss is designed for learning compact and noise-robust hash codes.

3.3. Stochastic Neighbor Embedding

Preserving the similarity relationship over image pairs is a typical criterion for deep learning to hash. In this respect, Stochastic Neighbor Embedding (**SNE**) [15] is well fit to hashing since it also converts the high-dimensional data into low-dimensional representations while preserving the similarity relationship of the high-dimensional data into low-dimensional codes.

Similar to SNE [15, 36], we first convert the distance between each image pair into a probability that quantifies the similarity relationship. Given a pair of images together with their pairwise similarity label as $(\boldsymbol{x}_i, \boldsymbol{x}_j, s_{ij})$, we use probability p_{ij} and q_{ij} to denote the similarity between $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and between their hash codes $(\boldsymbol{h}_i, \boldsymbol{h}_j)$ respectively. To minimize the mismatch between p_{ij} and q_{ij}, a natural measure is the Kullback-Leibler (KL) divergence:

$$L = \mathrm{KL}(P \| Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{1}$$

where P and Q are the joint probability distributions in the high-dimensional and low-dimensional spaces respectively. Since P quantifies the similarity information of image pairs, we can naturally set $p_{ij} = s_{ij}$ in supervised hashing.

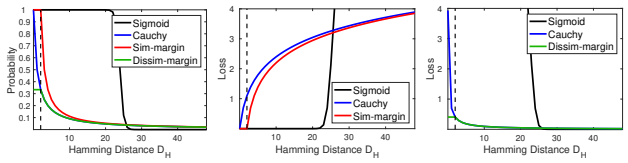
3.4. Max-Margin t-Distribution Loss

A disadvantage of Equation (1) is that it ignores dissimilar image pairs. When p_{ij} is extremely small, indicating $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are dissimilar, the divergence is very small and pair $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ has negligible influence on the loss function. As a result, Equation (1) fails to completely retain the dissimilarity relationship of the high-dimensional data. To fully capture both similarity and dissimilarity relationships, we propose a weighted KL divergence as

$$L = \sum_i \sum_j w_{ij} \cdot \begin{cases} p_{ij} \log \dfrac{p_{ij}}{q_{ij}}, & s_{ij} = 1 \\ (1 - p_{ij}) \log \dfrac{1 - p_{ij}}{1 - q_{ij}}, & s_{ij} = 0 \end{cases} \tag{2}$$

where w_{ij} is the weight for each training pair $(\boldsymbol{x}_i, \boldsymbol{x}_j, s_{ij})$ to mitigate the data imbalance problem. In each batch, we enhance the weights of similar pairs ($s_{ij} = 1$) by setting w_{ij} as the ratio of dissimilar pairs over similar pairs [4]. We drop the constant terms and rewrite Equation (2) as follows:

$$L = \sum_{s_{ij} \in \mathcal{S}} w_{ij} \left(s_{ij} \log \frac{1}{q_{ij}} + (1 - s_{ij}) \log \frac{1}{1 - q_{ij}} \right). \tag{3}$$



(a) Probability (b) Sim-Loss (c) Dissim-Loss

Figure 2. (a) Probability function, (b) loss for similar pairs, and (c) loss for dissimilar pairs. The X-axis denotes Hamming Distance.

Based on Equation (3), any valid probability function q_{ij} can be used to instantiate a specific hashing model. Figure 2(a) shows several valid probability functions w.r.t. Hamming distance D_H between hash codes of data pairs. HashNet [4] chooses Sigmoid function: $q = \frac{1}{1+\exp(-D_{ip})}$ (black line), where D_{ip} is the inner product between data pairs calculated by their Hamming distance: $D_{ip} = K - 2D_H$. DCH [2] adopts Cauchy distribution: $q = \frac{1}{1+D_H}$ (blue line). Figure 2(b) shows the loss for similar pairs taking as $\log \frac{1}{q_{ij}}$, and Figure 2(c) shows the loss for dissimilar pairs taking as $\log \frac{1}{1-q_{ij}}$.

However, all above probability functions fail to explicitly characterize the **Hamming ball** which is an absolute boundary between relevant and irrelevant. To address this disadvantage, we propose a novel probability function based on max-margin t-distribution, which can both concentrate similar pairs outside the Hamming ball (Figure 1, the right plot) and improve the model's robustness to noisy data:

$$q_{ij} = \begin{cases} \dfrac{1}{1+\max(0, D_H(i,j) - H)}, & s_{ij} = 1 \\ \dfrac{1}{1+\max(H, D_H(i,j))}, & s_{ij} = 0 \end{cases} \tag{4}$$

where H is the Hamming ball radius pre-specified in Hamming space retrieval, e.g. $H = 2$ for most scenarios. Next, we elaborate the two cases of Equation (4) respectively.

Loss for Similar Pairs. As shown in Figure 2(a), the probability generated by the Sigmoid function stays nearly 1 when the Hamming distance between hash codes is much larger than 2. It starts to decrease very fast only when the Hamming distance gets close to $K/2$, and keeps nearly 0 when the Hamming distance is larger than $K/2$. It means that the Sigmoid function is only sensitive to Hamming distance near $K/2$, but not discriminative enough in the rest. The Cauchy distribution decreases sharply from 1 to 0.33 when the Hamming distance increases from 0 to 2, which over-emphasizes the pairs within the Hamming ball and has a limited benefit on the pruning effectiveness.

In Hamming space retrieval, data points inside the Hamming ball are retrievable while those outside are irretrievable. For similar pairs, the loss function should focus more on the similar pairs outside the Hamming ball of radius H, since the ones inside the ball are guaranteed to be returned. Our max-margin t-distribution loss (Figure 2(b), red line) vanishes for similar pairs within Hamming radius H but increases fast when the Hamming distance is larger than H, which can effectively focus the model on the similar pairs outside the Hamming ball to reduce *false negatives*.

Loss for Dissimilar Pairs. For dissimilar pairs with small Hamming distance, previous methods [53, 4, 2] make them far apart by improperly large penalization regardless of the potential noise. For clean data, dissimilar images tend to have codes with large distance. For noisy data, however, there are many "dissimilar" pairs with very small Hamming distance, possibly due to erroneous similarity labels commonly seen in real applications. In this scenario, assigning overly large loss to those feature-close but label-dissimilar pairs will make the model vulnerable to noisy data.

As shown in Figure 2(c), the Sigmoid loss increases quickly for dissimilar pairs when the distance is less than $K/2$, and the Cauchy loss approaches infinity for dissimilar pairs with distance less than 2. Hence, both of them are not robust to noisy data. Our max-margin t-distribution loss for dissimilar pairs (Figure 2(c), green line) truncates the loss to a constant when the Hamming distance $D_H \leq H$, yielding a smooth loss curve which is more robust to noisy data.

3.5. Objective Function

Combining the probability proposed in Equation (4) into Equation (3), we obtain the max-margin t-distribution loss:

$$\begin{aligned} L = &\sum_{s_{ij} \in \mathcal{S}} w_{ij}(s_{ij}) \log\left(1 + \max(0, D_H(i,j) - H)\right) \\ &+ \sum_{s_{ij} \in \mathcal{S}} w_{ij}(1 - s_{ij}) \log\left(1 + \frac{1}{\max(H, D_H(i,j))}\right). \end{aligned} \tag{5}$$

Equation (5) is difficult to optimize with binary constraints $\boldsymbol{h}_i \in \{-1, 1\}^K$. Many previous works [53, 24, 2] use continuous relaxation to replace the discrete Hamming distance between a pair of codes $(\boldsymbol{z}_i, \boldsymbol{z}_j)$, employing the relationship between the Hamming distance and their cosine distance:

$$D_H(i,j) = \frac{K}{2}\left(1 - \cos(\boldsymbol{z}_i, \boldsymbol{z}_j)\right). \tag{6}$$

Equation (6) is more convenient to be applied in the deep network than the Hamming distance. To control the quantization error of converting the continuous representations to binary codes, we adopt the standard quantization loss:

$$Q = \sum_{i=1}^{n} \| \operatorname{sgn}(\boldsymbol{z}_i) - \boldsymbol{z}_i \|_2^2. \tag{7}$$

By integrating the novel max-margin t-distribution loss in Equation (5) and the quantization loss in Equation (7), we obtain the overall objective function for the MMHH model:

$$\min_{\Theta} L + \lambda Q, \tag{8}$$

where Θ denotes the network parameters, and λ is a hyper-parameter to trade-off L and Q selected by cross-validation. After convergence, we obtain K-bit binary codes by $\boldsymbol{h} \leftarrow \mathrm{sgn}(\boldsymbol{z})$. Since the quantization error is explicitly controlled, the final binarization step is sufficiently accurate.

3.6. Semi-Batch Optimization

A critical challenge for learning to hash from similarity data is that the similarity information is often very sparse in real retrieval systems, namely, similar pairs are often much fewer than dissimilar pairs [4]. Most methods alleviate data imbalance by up-weighting the similar pairs. This may fail on extremely imbalanced data as there are nearly none similar pairs in each mini-batch, and up-weighting cannot cover sufficient similarity information for mini-batch training.

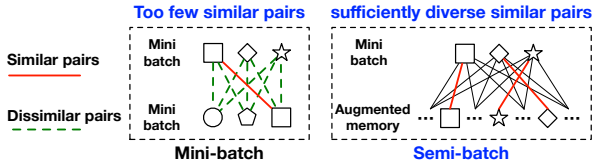


Figure 3. Semi-batch optimization for training on imbalance data.

Motivated by augmented memory [44, 48], we propose semi-batch optimization to address this problem. We store the codes of the training set as the augmented memory. For the $(t+1)$-th epoch, the network parameters are $\Theta^{(t)}$ and the augmented memory is $\{\boldsymbol{z}_1^{(t)}, \cdots, \boldsymbol{z}_n^{(t)}\}$. Given an input $\boldsymbol{x}_i$, we compute the new code by $\boldsymbol{z}_i^{(t+1)} = f(\boldsymbol{x}_i; \Theta^{(t)})$ and update the corresponding memory. The pairwise loss is calculated by $\boldsymbol{z}_i^{(t+1)}$ with all its similar and dissimilar pairs in the augmented memory. Finally, we back-propagate the loss to update the network parameters. As a result, the semi-batch optimization guarantees that the network is always trained with sufficiently diverse similar pairs, as shown in Figure 3.

4. Experiments

We evaluate the performance of our proposed approach with the state-of-art methods on four benchmark datasets: NUS-WIDE, CIFAR-10, MS-COCO, and ImageNet.

4.1. Evaluation Setup

CIFAR-10 [18] is a dataset consisting of 10 object categories each with 6000 images. We sample 100 images per class as query set and take the rest as database. We sample 500 images per class in the database as the training set.

MS-COCO [26] contains 82,783 training images and 40,504 validation images, each annotated by some of the 80 categories. We first prune images without category information and obtain 122,218 images by combining the training and validation images. We sample 5,000 images as query set and take the rest as the database. We sample 10,000 images from the database as the training set.

NUS-WIDE [5] contains 269,648 images within 81 concepts. We sample 5,000 images as query set and take the rest as database. We sample 10,000 images from the database as the training set.

ImageNet [38] contains over 1.2M images in the training set and 50K images in the validation set, where each image is labeled by one of 1,000 categories. We use the images in the training and validation sets as the database and query set respectively, and randomly select 100 images per category from the database as the training set.

Following the evaluation protocol of previous work [21, 53, 4, 2], the similarity information for hash function learning and ground truth evaluation is constructed from image labels. If two images $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ share at least one label, they are similar ($s_{ij} = 1$); otherwise, they are dissimilar ($s_{ij} = 0$). Although we construct similarity information by image labels, our approach can learn hash codes in the case where only similarity information is available.

We evaluate the retrieval performance of **MMHH** with eight state-of-the-art methods, including three supervised shallow methods **ITQ** [13], **KSH** [30], and **SDH** [41], along with five supervised deep methods **CNNH** [49], **DNNH** [21], **DHN** [53], **HashNet** [4] and **DCH** [2]. For shallow hashing methods, we use as input the 4096-dimensional DeCAF7 features [9]. For deep hashing methods, we use raw images as the input.

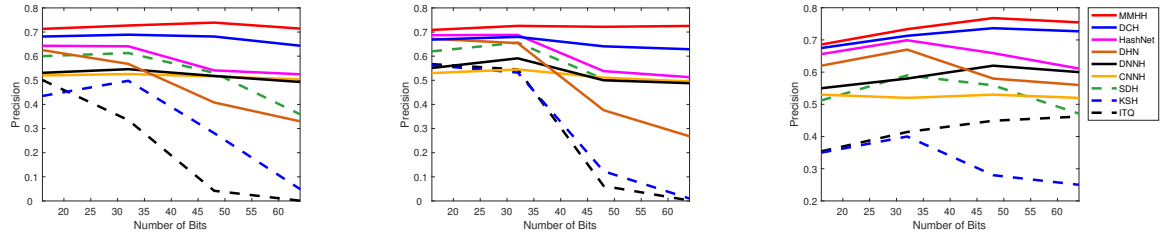
The evaluation protocol for Hamming spatial retrieval [37, 2] constitutes two phases: (1) *pruning*, which searches the hash lookup table and returns points within Hamming radius H to each query; (2) *re-ranking*, which re-ranks the continuous codes of the returned points in ascending order of their distances to the query. For the best efficiency, the search based on pruning and re-ranking is widely-deployed in large-scale retrieval systems, and the continuous features before binarization are adopted for the re-ranking step.

To measure the retrieval quality, we use three standard evaluation metrics for Hamming space retrieval with Hamming radius $H = 2$ [2, 3]: Mean Average Precision within Hamming radius 2 (**MAP@H$\leq$ 2**), Precision within Hamming radius 2 (**P@H$\leq$2**), and Recall within Hamming radius 2 (**R@H$\leq$2**). More specifically, we first calculate the Average Precision (AP) [2] for all queries, then we compute MAP@H$\leq$2 as the mean of Average Precision of all queries. The larger the MAP@H$\leq$2, the better the quality of retrieved data within Hamming radius 2.

The implementations are based on TensorFlow [1]. We adopt AlexNet [19] as the main backbone for all deep hashing methods. We further validate the efficacy of semi-batch optimization on ImageNet by comparing MMHH, DCH and HashNet with ResNet50 [14] as the backbone. We fine-tune the ImageNet-pretrained backbones and train the hash

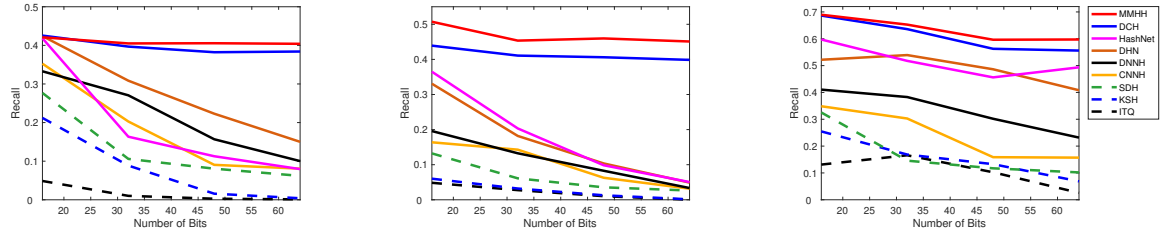
Table 1. MAP Results of Re-ranking within Hamming Radius 2 for Different Bits on Three Benchmark Datasets

Method	NUS-WIDE				MS-COCO				CIFAR-10			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
MMHH	**0.7719**	**0.7992**	**0.7888**	**0.7547**	**0.7358**	**0.7832**	**0.7896**	**0.8074**	**0.7923**	**0.8178**	**0.8246**	**0.8189**
DCH	0.7401	0.7720	0.7685	0.7124	0.7010	0.7576	0.7251	0.7013	0.7901	0.7979	0.8071	0.7936
HashNet	0.6944	0.7147	0.6736	0.6190	0.6851	0.6900	0.5589	0.5344	0.7476	0.7776	0.6399	0.6259
DHN	0.6901	0.7021	0.6685	0.5664	0.6749	0.6680	0.5151	0.4186	0.6929	0.6445	0.5835	0.5883
DNNH	0.6191	0.6216	0.5902	0.5626	0.5771	0.6023	0.5235	0.5013	0.5703	0.5985	0.6421	0.6118
CNNH	0.5843	0.5989	0.5734	0.5729	0.5602	0.5685	0.5376	0.5058	0.5512	0.5468	0.5454	0.5364
SDH	0.6681	0.6824	0.5979	0.4679	0.6449	0.6766	0.5226	0.5108	0.5620	0.6428	0.6069	0.5012
KSH	0.5185	0.5659	0.4102	0.0608	0.5797	0.5532	0.2338	0.0216	0.4368	0.4585	0.4012	0.3819
ITQ	0.5706	0.4397	0.0825	0.0051	0.5949	0.5612	0.0585	0.0105	0.4258	0.4652	0.4774	0.4932



(a) NUS-WIDE (b) MS-COCO (c) CIFAR-10

Figure 4. The Precision curves within Hamming radius 2 of MMHH and comparison methods on the three benchmark datasets.



(a) NUS-WIDE (b) MS-COCO (c) CIFAR-10

Figure 5. The Recall curves within Hamming radius 2 of MMHH and comparison methods on the three benchmark datasets.

layer by back-propagation. As the hash layer is trained from scratch, its learning rate is 10 times of the other layers'. We use SGD with 0.9 momentum and a weight decay of 5×10^{-4}. The default batch size of images is 48. All hyper-parameters of all methods are selected by cross-validation.

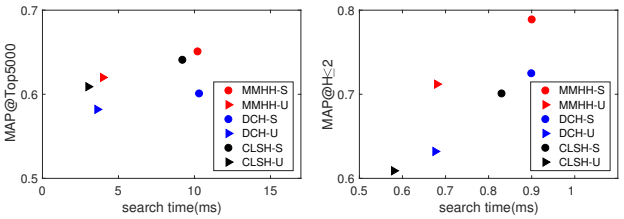
4.2. Retrieval Performance

MAP@H≤2. These results are shown in Table 1. MMHH substantially outperforms all comparison methods, indicating that MMHH performs higher-quality pruning in the first retrieval phase and more effective re-ranking in the second phase. Compared with SDH, the best shallow hashing method with deep features as input, MMHH achieves substantial boosts of 17.5%, 19.0% and 23.5% in average MAP@H≤2 on NUS-WIDE, MS-COCO and CIFAR-10 respectively. MMHH outperforms HashNet, the method designed for linear scan with best MAP@H≤2. HashNet does not impose strong constraints on similar data pairs and cannot concentrate similar points into the Hamming ball. MMHH further outperforms DCH, the state-of-the-art hashing method for Hamming space retrieval. The max-margin t-distribution loss enables MMHH to reduce the penalization to similar pairs within the Hamming ball and penalize the other pairs more significantly.

P@H≤2. Figure 4 shows the precision curves within Hamming radius 2. MMHH substantially outperforms the comparison methods on all three datasets. When using longer hash codes, the Hamming space becomes sparser and fewer data points fall in the Hamming ball of radius 2 [37]. The precision of the other methods on MS-COCO decreases sharply using longer codes, while MMHH performs stably across different code lengths. This validates the efficacy of the max-margin t-distribution loss, which enables MMHH to squeeze more similar data points into the Hamming ball.

R@H≤2. Figure 5 illustrates the recall curves within Hamming radius 2 (R@H≤2). MMHH achieves the high-

est R@H≤2 on MS-COCO. It outperforms DCH on NUS-WIDE and CIFAR-10 when the hash bit is longer than 32. Such impressive precision and competitive recall performance prove that MMHH can handle different retrieval scenarios varying from precision-first to recall-first settings.



(a) MAP@Top5000 (b) MAP@H≤ 2

Figure 6. MAP vs. search time on MS-COCO under protocol [39].

Unseen Classes Retrieval Protocol [39]. The evaluation protocols and datasets used in the above results are consistent with most previous hashing methods [49, 24, 2, 39]. In this setting, the datasets are labeled according to categories and all test classes are known in the training phase. Hence, the hashing methods only need to discriminate between known classes, which is easier than the scenario of image retrieval in the wild.

We further test our method following a more difficult **unseen classes retrieval protocol** [39]: On MS-COCO, we randomly select 20 classes as query/database, and the rest 60 classes as training set. We compare MMHH with DCH and **Classifier + LSH (CLSH)** [39] under the seen (S) (used as above) and unseen (U) protocols.

Figure 6 shows MAP@Top5000 (for linear scan) and MAP@H≤2 (for lookup search) w.r.t the search time. CLSH is top-performant in efficiency, closely followed by MMHH and DCH. Despite the performance degradations in the unseen protocol, MMHH is competitive with CLSH on linear scan and outperforms DCH and CLSH on lookup search, validating its effectiveness on a variety of scenarios.

Table 2. Retrieved Entries of MMHH and HashNet, COCO, 48bits

Method	All Sim	$H \leq 2$	Sim@ $H \leq 2$	Return None
MMHH	198.4M	208.0M	144.2M	13%
HashNet	198.4M	23.5M	22.7M	44%

Linear Scan Retrieval Protocol [4]. To highlight the difference between lookup search and linear scan, we compare MMHH and HashNet on MS-COCO. In terms of efficiency, the search time of hash lookup (0.9 ms/query) is much shorter than that of linear scan (10.2 ms/query). In terms of effectiveness, as shown in Table 2, there are a total of 198.4M ground truth similar pairs for 5,000 queries. MMHH retrieves 208.0M images within Hamming Distance 2, of which 144.2M are correct, while HashNet only retrieves 23.5M entities. Note that in hash lookup search, it is possible to retrieve **zero** images for many queries. For HashNet, though 22.7M of 23.5M retrieved images are correct, 44% queries return no images, dramatically lowering P@H≤2 and R@H≤2. In contrast, MMHH achieves much higher recalls, where only 13% queries return no images.

4.3. Results on Noisy Data

We evaluate the robustness of state-of-the-art deep hashing methods on training data with noisy labels by comparing MMHH, DCH and HashNet. We generate noisy data from clean data by stochastically changing some labels as [12]. For each image in the training set we change its label to another one with probability p (denote the label noise rate). The database and test set remain unchanged.

Figure 7 shows the MAP@H≤2 results of three methods on NUS-WIDE with label noise rate from 0% (clean data) to 50%. MMHH is the robustest method in which MAP drops only 0.02 when the label noise rate increases to 50%, while under the same noise rate, the MAPs of DCH and HashNet decrease by 0.06 and 0.17 respectively.

Table 3. Loss Fraction of Noisy Labels on NUS-WIDE ($p = 50\%$)

Methods	MAP drop	Error Sim	Error Dissim
MMHH	0.02	0.7%	11.3%
DCH	0.06	0.7%	14.3%
HashNet	0.17	0.8%	30.7%

We further analyze the impact of noisy labels on the loss in Table 3. For the noisy version of NUS-WIDE ($p = 50\%$), the losses of erroneous similar pairs (which are actually dissimilar) are relatively small for three methods (0.7∼0.8%). However, the loss of erroneous dissimilar pairs (which are actually similar) of HashNet is 30.7%, significantly larger than MMHH and DCH. The HashNet loss gets overly large when the distance of dissimilar pairs is small, validating that HashNet is vulnerable to noisy labels. DCH mitigates the impact of erroneous dissimilar pairs, and the max-margin t-distribution loss of MMHH further addresses the issue of loss explosion of the Cauchy loss in DCH (see Figure 2(c)). As a result, MMHH shows the best robustness to noisy data.

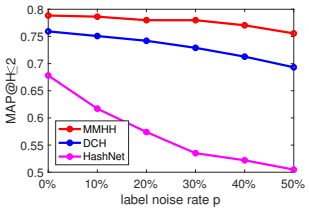


Figure 7. Noisy Training Data

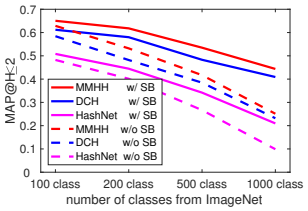


Figure 8. Semi-Batch Results

Table 4. MAP Results of Re-ranking within Hamming Radius 2 for MMHH and Its Variants on Three Benchmarks

Method	NUS-WIDE				MS-COCO				CIFAR-10			
	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits	16 bits	32 bits	48 bits	64 bits
MMHH	**0.7719**	**0.7992**	**0.7888**	**0.7547**	**0.7358**	**0.7832**	**0.7896**	**0.8074**	**0.7923**	**0.8178**	**0.8246**	**0.8189**
MMHH-Q	0.7352	0.7891	0.7235	0.7066	0.7022	0.7292	0.7613	0.7979	0.7601	0.7741	0.7634	0.7898
MMHH-E	0.5966	0.6365	0.6868	0.5478	0.6478	0.6294	0.6625	0.6901	0.5550	0.6208	0.5874	0.5770

4.4. Effectiveness Analyses

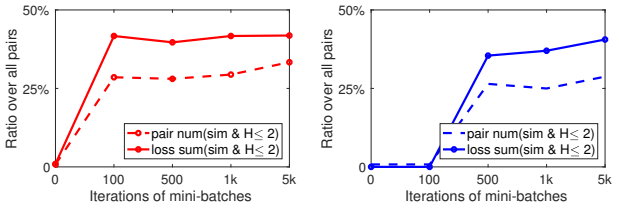
Semi-Batch Optimization. We justify semi-batch optimization by comparing MAP@H$\leq$2 of MMHH, DCH and HashNet with/without semi-batch training on the ImageNet subsets of 100, 200, 500, 1,000 classes in Figure 8.

First, semi-batch is generally helpful for MMHH, DCH and HashNet to achieve better accuracies. Second, no matter with/without semi-batch, MMHH outperforms the others by large margins. Third and more interestingly, semi-batch brings larger improvement on the subsets with more classes (e.g., ImageNet-1000 vs. ImageNet-100), confirming our motivation of using semi-batch optimization to deal with similarity sparsity in extremely imbalanced data.

Table 5. Ratio of Similar Pairs in a Batch on ImageNet-1K

sim-pairs	=0	≤ 1	≤ 2	≤ 3	≤ 4	≥ 5	> 100
w/o SB (%)	10.1	32.9	59.5	80.9	93.8	6.2	0

We delve into semi-batch optimization by comparing the numbers of similar pairs appearing in each batch (batch-size = 48) for ImageNet-1K. Without semi-batch training, there are (48×48) pairs in each batch. With semi-batch training, there are (48×10^5) pairs in each batch. Further, as shown in Table 5, without semi-batch training, only 6.2% batches have more than 5 similar pairs, and 10.1% batches have no similar pair. For batches with too few similar pairs, up-weighting does not solve the similarity sparsity. In contrast, with semi-batch training, every batch has 4,800 similar pairs. It explains why semi-batch optimization brings large performance boost on ImageNet-1K.



(a) 48 bits (b) 512 bits

Figure 9. Loss and number of similar pairs over all pairs (COCO)

Max-Margin t-Distribution Loss. Section 4.3 has shown the robustness of our max-margin t-distribution loss to noisy data. Considering similar pairs, though the Cauchy loss in DCH and our loss appear similar (Figure 2(b)), their difference is vital to Hamming space retrieval. Figure 9 shows the ratio of the number of similar pairs (and their loss) inside the Hamming ball ($H \leq 2$) over the number of all pairs (and their loss), using Cauchy loss on MS-COCO. Both ratios increase through the training process. Although similar pairs with small Hamming distance should have low losses, their actual loss ratio increases more. One reason is up-weighting similar pairs by w_{ij} in Equation (2). For 48 bits in Figure 9(a), it is unreasonable that these retrievable ($H \leq 2$) similar pairs occupy $\sim$40% of the overall loss. Thus, DCH is a biased model. MMHH with max-margin t-distribution loss reduces the bias by focusing on more difficult pairs outside the Hamming ball. Further, our loss keeps different from Cauchy loss for longer codes. As shown in Figure 9(b), for 512 bits, there are 28% similar pairs inside the Hamming ball occupying 35% of overall loss.

Ablation Study. We investigate two variants of MMHH: **MMHH-Q** is an MMHH variant without the quantization loss ($\lambda = 0$); **MMHH-E** is an MMHH variant replacing our cosine distance (Equation (6)) with the widely-used Euclidean distance as $D_H(i,j) = \|\boldsymbol{z}_i - \boldsymbol{z}_j\|_2^2$ in our loss. The MAP@H$\leq$2 results on all three datasets are summarized in Table 4. Without the quantization loss, MMHH-Q incurs average MAP decreases of 4.0%, 3.1% and 4.2% on three datasets respectively, testifying the necessity of controlling the quantization error. MMHH-E performs worse than MMHH by 16.2%, 12.2% and 22.8%, proving that the cosine distance is a better proxy to the Hamming distance when the quantization error is controlled simultaneously.

5. Conclusion

We propose Max-Margin Hamming Hashing (MMHH) to enable constant-time Hamming space retrieval. The approach explicitly characterizes the Hamming ball by a max-margin t-distribution loss. Further, the loss enhances robustness to noisy data by preventing the model from trapped by wrongly-labeled image pairs. We also introduce a semi-batch optimization algorithm for training hashing models on extremely imbalanced data. Our approach yields state-of-the-art empirical results on both clean and noisy datasets.

6. Acknowledgments

This work is supported by National Key R&D Program of China (2017YFC1502003) and National Natural Science Foundation of China (61772299, 71690231, 61672313).

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, pages 265–283, 2016.

[2] Yue Cao, Mingsheng Long, Bin Liu, and Jianmin Wang. Deep Cauchy Hashing for Hamming Space Retrieval. In *CVPR*, pages 1229–1237, 2018.

[3] Zhangjie Cao, Mingsheng Long, Chao Huang, and Jianmin Wang. Transfer adversarial hashing for hamming space retrieval. In *AAAI*, pages 6698–6705, 2018.

[4] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. Hashnet: Deep learning to hash by continuation. In *ICCV*, pages 5609–5618, 2017.

[5] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yan-Tao Zheng. Nus-wide: A real-world web image database from national university of singapore. In *ICMR*. ACM, 2009.

[6] Hui Cui, Lei Zhu, Chaoran Cui, Xiushan Nie, and Huaxiang Zhang. Efficient weakly-supervised discrete hashing for large-scale social image retrieval. *Pattern Recognition Letters*, 2018.

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

[8] Cheng Deng, Erkun Yang, Tongliang Liu, Jie Li, Wei Liu, and Dacheng Tao. Unsupervised semantic-preserving adversarial hashing for image search. *IEEE Transactions Image Processing*, 28(8):4032–4044, 2019.

[9] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, pages 647–655, 2014.

[10] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *CVPR*, pages 2475–2483. IEEE, 2015.

[11] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529. ACM, 1999.

[12] Jacob Goldberger and Ehud Ben-Reuven. Training deep neural-networks using a noise adaptation layer. In *ICLR*, 2017.

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

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, pages 770–778, 2016.

[15] Geoffrey E. Hinton and Sam T. Roweis. Stochastic neighbor embedding. In *NeurIPS*, pages 833–840, 2002.

[16] Qing-Yuan Jiang, Xue Cui, and Wu-Jun Li. Deep discrete supervised hashing. *IEEE Transactions Image Processing*, 27(12):5996–6009, 2018.

[17] Qing-Yuan Jiang and Wu-Jun Li. Asymmetric deep supervised hashing. In *AAAI*, pages 3342–3349, 2018.

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[20] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. In *NeurIPS*, pages 1042–1050, 2009.

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

[22] Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *TOMM*, 2(1):1–19, Feb. 2006.

[23] Ning Li, Chao Li, Cheng Deng, Xianglong Liu, and Xinbo Gao. Deep joint semantic-embedding hashing. In *IJCAI*, pages 2397–2403, 2018.

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

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

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755. Springer, 2014.

[27] Venice Erin Liong, Jiwen Lu, Ling-Yu Duan, and Yappeng Tan. Deep variational and structural hashing. *TPAMI*, 2018.

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

[29] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. Learning multifunctional binary codes for both category and attribute oriented retrieval tasks. In *CVPR*, pages 6259–6268, 2017.

[30] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081. IEEE, 2012.

[31] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *ICML*, pages 1–8. ACM, 2011.

[32] Xianglong Liu, Junfeng He, Cheng Deng, and Bo Lang. Collaborative hashing. In *CVPR*, pages 2139–2146, 2014.

[33] Xianglong Liu, Junfeng He, Bo Lang, and Shih-Fu Chang. Hash bit selection: a unified solution for selection problems in hashing. In *CVPR*, pages 1570–1577. IEEE, 2013.

[34] Xiaoqiang Lu, Xiangtao Zheng, and Xuelong Li. Latent semantic minimal hashing for image retrieval. *IEEE Transactions on Image Processing*, 26(1):355–368, 2016.

[35] Yuqing Ma, Yue He, Fan Ding, Sheng Hu, Jun Li, and Xianglong Liu. Progressive generative hashing for image retrieval. In *IJCAI*, pages 871–877, 2018.

[36] Laurens Van Der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 9(2605):2579–2605, 2008.

[37] Mohammad Norouzi, Ali Punjani, and David J Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*, pages 3108–3115. IEEE, 2012.

[38] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015.

[39] Alexandre Sablayrolles, Matthijs Douze, Nicolas Usunier, and Herve Jegou. How should we evaluate supervised hashing? In *ICASSP*, pages 1732–1736, 2017.

[40] Ruslan Salakhutdinov and Geoffrey E Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *AISTATS*, pages 412–419, 2007.

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

[42] Jingkuan Song, Tao He, Lianli Gao, Xing Xu, Alan Hanjalic, and Heng Tao Shen. Binary generative adversarial networks for image retrieval. In *AAAI*, pages 394–401, 2018.

[43] Shupeng Su, Chao Zhang, Kai Han, and Yonghong Tian. Greedy hash: Towards fast optimization for accurate hash coding in CNN. In *NeurIPS*, pages 806–815, 2018.

[44] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NeurIPS*, pages 3630–3638, 2016.

[45] Guan'an Wang, Qinghao Hu, Jian Cheng, and Zengguang Hou. Semi-supervised generative adversarial hashing for image retrieval. In *ECCV*, pages 491–507, 2018.

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

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

[48] Zhirong Wu, Alexei A. Efros, and Stella X. Yu. Improving generalization via scalable neighborhood component analysis. In *ECCV*, pages 712–728, 2018.

[49] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In *AAAI*, pages 2156–2162. AAAI, 2014.

[50] Xinyu Yan, Lijun Zhang, and Wu-Jun Li. Semi-supervised deep hashing with a bipartite graph. In *IJCAI*, pages 3238–3244, 2017.

[51] Felix X Yu, Sanjiv Kumar, Yunchao Gong, and Shih-Fu Chang. Circulant binary embedding. In *ICML*, pages 353–360. ACM, 2014.

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

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

[54] Xiaofeng Zhu, Xuelong Li, Shichao Zhang, Zongben Xu, Litao Yu, and Can Wang. Graph pca hashing for similarity search. *IEEE Transactions on Multimedia*, 19(9):2033–2044, 2017.