Self-Distilled Hashing for Deep Image Retrieval

Young Kyun Jang\textsuperscript{1}, Geonmo Gu\textsuperscript{2}, Byungsoo Ko\textsuperscript{2}, and Nam Ik Cho\textsuperscript{1}

Department of ECE, INMC, Seoul National University, Seoul Korea\textsuperscript{1}, NAVER/LINE Vision\textsuperscript{2}

\{kyun0914, korgm403, kobiso62\}@gmail.com, nicho@snu.ac.kr

Abstract

In hash-based image retrieval systems, the transformed input from the original usually generates different codes, deteriorating the retrieval accuracy. To mitigate this issue, data augmentation can be applied during training. However, even if the augmented samples of one content are similar in real space, the quantization can scatter them far away in Hamming space. This results in representation discrepancies that can impede training and degrade performance. In this work, we propose a novel self-distilled hashing scheme to minimize the discrepancy while exploiting the potential of augmented data. By transferring the hash knowledge of the weakly-transformed samples to the strong ones, we make the hash code insensitive to various transformations. We also introduce hash proxy-based similarity learning and binary cross entropy-based quantization loss to provide fine quality hash codes. Ultimately, we construct a deep hashing framework that generates discriminative hash codes. Extensive experiments on benchmarks verify that our self-distillation improves the existing deep hashing approaches, and our framework achieves state-of-the-art retrieval results. The code will be released soon.

1. Introduction

Learning to hash has been one of the most important tasks in the image retrieval community [32]. Especially for retrieval from large-scale databases, hashing is essential due to its high search speed and low storage cost. By converting high-dimensional data points into compact binary codes with a hash function, the retrieval system can utilize a simple bit-wise XOR operation to define a distance between the images. A wide variety of works have been studied for this purpose [13,26,30,33], and are still being actively pursued.

In recent years, techniques for hash learning have been significantly advanced by deep learning, which is called deep hashing. By integrating the hash function into the deep framework, deep encoder and hash function are simultaneously learned to generate image hash codes. Regarding the training of deep hashing, the leading techniques are pairwise similarity approaches that use sets of similar or dissimilar image pairs [3,4,17,34,38], and global similarity in company with classification approaches that use class labels assigned to images [18,19,36].

Since hash-based retrieval systems compute the distance between images with binary codes, corresponding codes need to be quantized with sign operation, from the continuous real space to the discrete Hamming space of $\{-1, 1\}$. In this process, the continuously optimized image representation is altered, and quantization error occurs, which in turn degrades the discriminative capability of the hash code. This becomes even more problematic when an input image is transformed and deviated from the original distribution.

To avoid performance degradation due to transformations, the most common solution is to generalize the deep model by training it with augmented data having various transformations. However, it is challenging to apply this augmentation strategy to deep hashing training since discrepancy in the representation may occur. Figure 1 shows an example case that may appear: 1) The sign of the hash code can be shifted with a slight change. Specifically, the last element of the weakly-transformed image’s hash code...
differs by 0.2 ($-0.1 \rightarrow 0.1$) from the original, but it results in $-1 \rightarrow 1$ shift in the Hamming space. 2) The sign of the quantized hash code does not shift even with the big change in the hash code. The last element of a strongly transformed image’s hash code differs by 1.0 ($-0.1 \rightarrow -1.1$) from the original, resulting in no shifts in the Hamming space. Namely, the direct use of data augmentation in deep hashing increases the discrepancy between Hamming and real space, which hinders finding the optimal binary code.

To resolve this issue, we introduce a novel concept dubbed Self-distilled Hashing, which customizes self-distillation [5, 31, 35, 37] to prevent severe discrepancy in deep hashing training. Specifically, based on the understanding of the relation between cosine distance and Hamming agreement, we minimize the cosine distance between the hash codes of two different views (transformed results) of one image to maximize the Hamming agreement between their binary outcomes. Further for stable learning, we separate the difficulties of transformations as easy and difficult, and transfer the hash knowledge from easy to difficult.

Moreover, we propose two additional training objectives that optimize hash codes to enhance the self-distilled hashing: 1) a hash proxy-based similarity learning, and 2) a binary cross entropy-based quantization loss. The first term allows the deep hashing model to learn global (inter-class) discriminative hash codes with temperature-scaled cosine similarity. The second term contributes to making the hash code naturally move away from the binary threshold in a classification manner with likelihood estimators.

By combining all of our proposals, we construct a Deep-Hash Distillation framework (DHD), which yields discriminative hash codes for fast image retrieval. We conduct extensive experiments on single and multi-labeled benchmark datasets for image hashing evaluation. In addition, we validate the effectiveness of self-distilled hashing using data augmentation on the existing methods [3, 4, 12, 36] and show the performance improvements. Furthermore, we establish that DHD is applicable with a variety of deep backbone architectures including vision transformers [10, 27, 31]. Experimental results confirm that DHD significantly improves the retrieval performance with the state-of-the-art scores.

We can summarize our contributions as follows:

- To the best of our knowledge, this is the first work to address the discrepancy between real and Hamming space provoked by data augmentation in deep hashing.
- With the introduction of self-distilled hashing scheme and training loss functions, we successfully embed the power of augmentations into the hash codes.
- Comprehensive experimental studies demonstrate the superiority of our work, which is also feasible to various deep hashing methods and backbones.

2. Related Works

For a better understanding, we present a brief introduction to the deep hashing methods and the research that inspired our proposal. Refer to a survey [32] to see more details of the early works in non-deep hashing approaches (ITQ [13], SH [33], KSH [26], SDH [30]).

Deep hashing methods. Hashing algorithms using deep learning techniques such as Convolutional Neural Network (CNN) are leading the mainstream with striking results. For example, CNNH [34] utilizes a CNN to generate compact hash codes by training a network with given pairwise label information. DHN [38] learns hash codes by approximating discrete values with relaxation and trains them with supervised signals. HashNet [4] adopts the inner product to measure pairwise similarity between hash codes and tackles the data imbalance problem by employing weighted maximum likelihood estimation. DCH [3] employs Cauchy distribution to minimize the Hamming distance of the images with the same class label.

Hash center-based methods. There have been several approaches to find out class-wise hash representatives (centers), which can provide global similarity to hash codes by including the process of predicting image class labels with hash codes during training [17–19, 36]. CSQ [36] uses predefined orthogonal binary hash targets to guarantee distant Hamming distance between classes and makes hash codes to follow the targets. DPN [12] employs randomly assigned target vectors with maximal inter-class similarity and utilizes bit-wise hinge-like loss. Unlike DPN and CSQ, which use a hash target that is not trainable, in our DHD, the hash center is set as a trainable proxy which jointly learns the similarity with the hash codes during training.

Self-distillation. Inspired by knowledge distillation [16], self-distillation emerges as a concept that employs a single network to generalize itself in a self-taught fashion, and plenty of works demonstrate its benefits in improving deep model performance [5, 31, 35, 37]. Many of them utilize a simple Siamese architecture [7] to explore and learn the visual representation with data augmentation, by contrasting two different augmented results of one image. Similarly, we conduct the self-distillation with augmentations in deep hashing to see the hash codes of two different views of one image simultaneously. Additionally, in accordance with the characteristics of hashing, we consider a method of minimizing the cosine distance that behaves similarly to the distance in the Hamming space to reduce the representation discrepancy during model training.
3. Method

The goal of a deep hashing model \( H \) of Deep Hash Distillation (DHD) is to map an input image \( x \) to a \( K \)-dimensional binary code \( b \in \{-1, 1\}^K \) in Hamming space. For this purpose, \( H \) is optimized to find a high quality real-valued hash code \( h \), and then sign operation is utilized to quantize \( h \) as \( b \). Instead of including non-differentiable quantization process in model training, we learn \( H \) in the real space to estimate optimal \( b \) with continuously relaxed \( h \) while fully exploiting the power of data augmentation. We notate trainable component with \( \theta \) as a subscript. In the following, \( h \) becomes robust to various transformations in 3.1, and becomes discriminative and binary-like in 3.2.

3.1. Self-distilled Hashing

In general, \( H \) is trained in the real space to obtain discriminative \( h \), which should maintain its property in the Hamming space even if quantized to \( b \). Therefore, it is important to align \( h \) and \( b \) to carry a similar representation during training. However, when data augmentation is applied to the training input and the following change occurs in \( h \), there can be misalignment between \( h \) and \( b \), as shown in Figure. 1. Thus, direct use of augmentations can cause discrepancies in the representation between \( h \) and \( b \), which degrades retrieval performance as we observed in Sec 4.4.

Hamming distance as cosine distance. It is noteworthy that the Hamming distance between the binary codes can be interpreted as cosine distance (1-cosine similarity) \(^1\). That is, for deep hashing, the cosine similarity between hash codes \( h_i \) and \( h_j \) can be utilized to approximate the Hamming distance between the binary codes \( b_i \) and \( b_j \) as:

\[
D_H(b_i, b_j) \approx \frac{K}{2}(1 - S(h_i, h_j))
\]

where \( b_i = \text{sign}(h_i) \), \( b_j = \text{sign}(h_j) \), \( D_H(\cdot, \cdot) \) denotes Hamming distance, \( S(\cdot, \cdot) \) denotes cosine similarity. That is, the minimized cosine distance between the hash codes minimizes the Hamming distance between the binary codes.

Easy-teacher and difficult-student. As shown in Figure 2a, we propose a self-distilled hashing scheme, which supports the training of deep hashing models with augmentations. We employ weight-sharing Siamese structure [21] to contrast hash codes of two different views (augmentation results) of one image at once. According to the observations in self-distillation works [5,7], keeping the output representation of one side steady has a significant impact on performance gain. Therefore, we configure two separate augmentation groups to provide input views with different difficulties of transformation: one is weakly-transformed easy

---

\(^1\)Refer supplementary material for proof.


**Algorithm 1** Self-distilled Hashing PyTorch pseudo-code.

```python
# H: deep hashing model
# sT: transformation occurrence scale
# augT = augmentation(sT): # teacher group
# augS = augmentation(1.0): # student group
for x in loader: # load a minibatch of n-samples
  xT, xS = augT(x), augS(x) # random views
  hT, hS = H(xT), H(xS) # hashing, n-by-K
  L = SdH(hT, hS) # loss
  L.backward() # back-propagate
  update(H(xS)) # Adam update

def SdH(hT, hS): #Self-distilled Hashing
  hT = hT.detach() # stop gradient
  hT = normalize(hT, dim=1) # l2-normalize
  hS = normalize(hS, dim=1) # l2-normalize
  return 1- (hT * hS).sum(dim=1).mean()
```

teacher $T_T$, and the other is strongly-transformed difficult student $T_S$. Here, we control the difficulty in a stochastic sampling manner as: employing the same hyper-parameter $s_T$ to all transformations in the group, and make them occur less (weakly) or more (strongly) by scaling their own probability of occurrence. While this manner makes the teacher representation stable, it has the advantage that few extreme examples that produce unstable results are not completely ruled out and contribute to learning. Besides, we stop the gradient of the teacher view’s corresponding hash codes to avoid collapsing into trivial solutions [5, 7].

**Loss computation.** For a given image $x$, self-distillation is conducted with image views as: $\tilde{x}_T = t_T(x)$ and $\tilde{x}_S = t_S(x)$, where $t_T, t_S$ are randomly sampled transformations from $T_T, T_S$, respectively. The deep encoder $E_g$ and the hash function $H_g$ take $\tilde{x}_T$ and $\tilde{x}_S$ as inputs and produce corresponding hash code $h_T$ and $h_S$. Then, the proposed Self-distilled Hashing (SdH) loss is computed as:

$$L_{SdH}(h_T, h_S) = 1 - S(h_T, h_S)$$

(2)

Optimizing $H$ with $L_{SdH}$ results in the alignment of $h_T$ and $h_S$, and thus $b_T$ and $b_S$ as follows Eqn. 1, which in turn reduces the discrepancy in representation between two differently transformed output binary codes. We provide a pseudo-code implementation in Algorithm 1.

**Flexibility.** Note that self-distilled hashing is applicable to the other common deep hashing models [3, 4, 12, 36] with regard to exploiting data augmentation during training. Furthermore, various backbones [10, 15, 22, 27, 31] can be utilized as deep encoder, and any hash function $H_g$ configuration is compatible. For simplicity, we employ a single FC layer with a layer normalization [1] to address overly dominant binary bits and balance the intra-binary representation, and apply $\tanh$ operation at the end to be bound in $[-1, 1]$.

### 3.2. Teacher loss

Besides self-distilled hashing, additional training signals such as supervised learning loss, and quantization loss are required to obtain the discriminative hash codes. We only employ teacher hash codes to compute the losses, in order to transfer the learned hash knowledge to the student’s codes.

**Proxy-based similarity learning.** Supervised hash similarity learning with pre-defined non-trainable binary hash targets has shown great performance [12, 36]. However, the hash target has limitations in that it requires a complex initialization process, and cannot contain semantic similarity by itself. Therefore, as shown in Figure 2b, we introduce a collection of trainable hash proxies $p_θ$ that can simply be initialized along with deep parameters of $H$, and use this to compute class-wise prediction $p_T$ with $h_T$ as:

$$p_T = [S(p_{θ1}, h_T), S(p_{θ2}, h_T), ..., S(p_{θN_{cls}}, h_T)]$$

(3)

where $p_θ$ is a hash proxy assigned to each of the $i$ category and $N_{cls}$ denotes the number of categories. Then, we use $p_T$ to learn the similarity with class label $y$ by computing Hash Proxy (HP) loss as:

$$L_{HP}(y, p_T, τ) = H(y, \text{Softmax}(p_T/τ))$$

(4)

where $τ$ is a temperature scaling hyper-parameter, $H(u, v) = -\sum_k u_k \log v_k$ is a cross entropy, and Softmax operation is applied along the dimension of $p_T$. Although other hashing losses [3, 4, 12, 36] are available, we employ $L_{HP}$ since it shares the property of cosine similarity with the self-distilled hashing. Note that, similar to Eqn 1, $L_{HP}$ can learn Hamming agreement if scaled with $τ$.

**Reducing quantization error.** To make continuous hash code elements act like binary bits, the deep hashing methods [3, 4, 18, 36] aim to reduce the quantization error by minimizing the distance (e.g. Euclidean) between the hash code bit and its closest binary goal ($+1$ or $-1$) in a regression manner. However, since the purpose of hashing is to classify the sign of each bit, it is a more natural choice to view it as a binary classification: maximum likelihood problem. Hence, we adopt a pre-defined Gaussian distribution estimator $g(h)$ of mean $m$ and standard deviation $σ$ as:

$$g(h) = \exp \left( -\frac{(h - m)^2}{2σ^2} \right)$$

(5)

to evaluate the binary likelihood of hash code element $h$. By employing two estimators: $g^+$ of $m = 1$, and $g^-$ of $m = -1$ with the same $σ$, we compute the likelihoods and a Binary Cross Entropy-based (BCE) quantization loss as:

where \( H_b(u, v) = -u \log v + (1 - u) \log(1 - v) \) is a binary cross entropy, \( g_k^+ \), \( g_k^- \) denotes \( k \)-th hash code element’s estimated likelihood: \( g_k^+ = g^+(h_k) \), \( g_k^- = g^-(h_k) \), and \( b_k^+ \), \( b_k^- \) denotes binary likelihood labels which are obtained as:

\[
b_k^+ = \frac{1}{2} (\text{sign}(h_k) + 1), b_k^- = 1 - b_k^+
\]

As a result, quantization error is reduced by a binary classification loss with the given estimators, allowing to use the merits of cross entropy presented in [2].

### 3.3. Training

**Total training loss.** Suppose we are given a training mini-batch of \( N_B \) data points: \( \mathcal{X}_B = \{(x_1, y_1), \ldots, (x_{N_B}, y_{N_B})\} \) where each image \( x_i \) is assigned a label \( y_i \in \{0, 1\}^N \). Training views are obtained as \( x_{T_i} = t_{T_i}(x_i) \) and \( x_{S_i} = t_{S_i}(x_i) \) for all data points, where \( T_i \sim T_T \) and \( S_i \sim S_S \). Total loss \( \mathcal{L}_T \) for DHD is computed with \( \mathcal{X}_B \) as:

\[
\mathcal{L}_T(\mathcal{X}_B) = \frac{1}{N_B} \sum_{n=1}^{N_B} (\mathcal{L}_{HP} + \lambda_1 \mathcal{L}_{SDH} + \lambda_2 \mathcal{L}_{bce-Q})
\]

where \( \lambda_1 \) and \( \lambda_2 \) are hyper-parameters that balance the influence of the training objectives. The entire DHD framework is trained in an end-to-end fashion.

**Multi-label case.** In the case of determining semantic similarity between multi-hot labeled images, the previous works [34,36,38] simply checked whether the images share at least one positive label or not. However, learning with the above similarity has limitations in that the label dependency [8] is ignored. Thus, we aim to capture the intelligence that appears in label dependency by utilizing the Softmax cross entropy with the normalized multi-hot label \( y \). Specifically, \( y \) is converted as \( y = y/\|y\|_1 \) to balance the contribution of each label, and the same \( \mathcal{L}_{HP} \) is computed to optimize the deep hashing model for multi-label image retrieval.

### 4. Experiments

To evaluate our DHD, we conduct image retrieval experiments against several conventional and modern methods. Three most popular hashing based image retrieval benchmark datasets are explored, and we explain the composition of each dataset in Table 1. Detailed explanations are described in the supplementary material.

| Dataset       | # Database | # Train | # Query | \( N_c \) |
|---------------|------------|---------|---------|-----------|
| ImageNet [29] | 128,503    | 13,000  | 5,000   | 100       |
| NUS-WIDE [9]  | 149,736    | 10,500  | 2,100   | 21        |
| MS COCO [25]  | 117,218    | 10,000  | 5,000   | 80        |

Table 1. Description of the image retrieval datasets.

### 4.1. Setup

Following the protocol utilized in deep hashing methods [3,4,36], we adopt three benchmark datasets: single-labeled ImageNet [29], and multi-labeled NUS-WIDE [9], and MS COCO [25] with the same compositions.

**Evaluation metrics.** To evaluate retrieval quality, we employ three metrics: 1) mean average precision (mAP), 2) precision-recall curves (PR curves), and 3) precision with respect to top-M returned images (P@Top-M). Regarding mAP score computation, we select the top-M images from the retrieval ranked-list results. The returned images and the query image are considered relevant whether one or more category labels are the same. We set binary code length: hash code dimensionality \( K \) as 16, 32, and 64, to examine the performance according to the code size.

### 4.2. Implementation Details

**Data augmentation.** Following the works presented in [6], we choose family \( T \) of five image transformations: 1) resized crop, 2) horizontal flip, 3) color jitter, 4) grayscale, and 5) blur, where all of each are sampled uniformly with a given probability and sequentially applied to the inputs. We keep the internal parameters of each transformation equal to [6]. For self-distilled hashing, we configure two groups with \( T \), where the difficult student group is \( T_S = T \), and the easy teacher group \( T_T \) is configured by scaling all transform occurrence with \( s_T \), which is in the range of \( (0, 1) \). We set \( T_T \) as the default for the methods trained without SdH.

**Experiments.** Retrieval experiments are conducted by dividing backbones as: Non-deep, AlexNet [22], ResNet (ResNet50) [15], and vision transformers [10,27,31]. For non-deep hashing approaches: ITQ [13], SH [33], KSH [26] and SDH [30], we report the results directly from the latest works [3,4,36] for comparison. We set up the same training environment by leveraging PyTorch framework and the image transformation functions of korina [11] library for augmentation. We employ Adam optimizer [20] and decay the learning rate with cosine scheduling [28] for training deep hashing methods. Especially for DHD hyper-parameters, \( s_T \) is set to 0.2 for AlexNet, and 0.5 for other backbones. \( \tau \) is set by considering \( N_{cls} \) as \{0.2, 0.6, 0.4\} for \{ImageNet, NUS-WIDE, MS COCO\}, respectively. \( \lambda_1 \) and \( \lambda_2 \) are set equal to 0.1 for a balanced contributions each training objective, and \( \sigma \) in \( \mathcal{L}_{bce-Q} \) is set to 0.5 as default.

---

\(^2\)More details can be found in the supplementary material.
4.3. Results

The mAP scores are calculated by varying the top-M for each dataset as: ImageNet@1000, NUS-WIDE@5000 and MS COCO@5000 to make a fair comparison with previous works [3, 4, 36]. The results are listed in Table 2, where the highest score for each backbone is shown in bold, and we highlight our DHD method. Among the non-deep hashing methods, SDH shows the best retrieval results by employing supervised label signals in hash function learning. Deep hashing methods generally outperform non-deep hashing ones, since elaborately labeled annotations are fully utilized during training. For ImageNet, NUS-WIDE, and MS COCO, averaging the mAP scores of all bit lengths yields 36.3%p, 33.7%p, and 25.0%p differences between the non-deep and deep methods, respectively.

Notably, our DHD shows the best mAP scores for all datasets in every bit length with every deep backbone architecture. In particular for AlexNet backbone hashing approaches, DHD shows performance improvement of 16.3%p, 7.9%p, and 9.2%p by averaging the mAP scores of all bit lengths in three dataset results orderly, compared to others. In comparison with ResNet backbone methods, DHD also achieves 2.7%p, 1.8%p, and 4.7%p higher retrieval scores on average. In line with the trend of other computer vision tasks, we introduce transformer-based image representation learning architectures: ViT [10], DeiT [31], and SwinT [27] to the hashing community and perform retrieval experiments. As reported, when the transformer is integrated into the DHD framework, it delivers outstanding results for the benchmark image datasets with the increase of 5.8%p, 2.2%p, and 4.8%p, in the same as above, compared to ResNet backbone DHD.

![Table 2. mean Average Precision (mAP) scores for different bits on three benchmark image datasets.](image)

| Method     | Backbone | ImageNet | NUS-WIDE | MS COCO |
|------------|----------|----------|----------|---------|
|            |          | 16-bit   | 32-bit   | 64-bit  | 16-bit | 32-bit | 64-bit | 16-bit | 32-bit | 64-bit |
| ITQ [13]   | Non-deep | 0.266    | 0.436    | 0.576   | 0.435  | 0.396  | 0.365  | 0.586  | 0.562  | 0.502  |
| SH [33]    |          | 0.210    | 0.329    | 0.418   | 0.401  | 0.421  | 0.423  | 0.495  | 0.507  | 0.510  |
| KSH [33]   |          | 0.160    | 0.298    | 0.394   | 0.394  | 0.407  | 0.399  | 0.521  | 0.534  | 0.536  |
| SDH [30]   |          | 0.299    | 0.455    | 0.585   | 0.575  | 0.590  | 0.613  | 0.554  | 0.564  | 0.580  |
| CNNH [34]  | AlexNet  | 0.315    | 0.473    | 0.596   | 0.655  | 0.659  | 0.647  | 0.599  | 0.617  | 0.620  |
| DNNH [23]  |          | 0.353    | 0.522    | 0.610   | 0.703  | 0.738  | 0.754  | 0.644  | 0.651  | 0.647  |
| DHN [38]   |          | 0.367    | 0.522    | 0.627   | 0.712  | 0.739  | 0.751  | 0.701  | 0.710  | 0.735  |
| HashNet [4] |          | 0.425    | 0.559    | 0.649   | 0.720  | 0.745  | 0.758  | 0.685  | 0.714  | 0.742  |
| DCH [3]    |          | 0.636    | 0.645    | 0.656   | 0.740  | 0.752  | 0.763  | 0.695  | 0.721  | 0.748  |
| DHD(Ours)  |          | 0.657    | 0.701    | 0.721   | 0.780  | 0.805  | 0.820  | 0.749  | 0.781  | 0.792  |

| Method     | Backbone | ImageNet | NUS-WIDE | MS COCO |
|------------|----------|----------|----------|---------|
| DPN [12]   | ResNet   | 0.828    | 0.863    | 0.872   | 0.783  | 0.816  | 0.838  | 0.796  | 0.838  | 0.861  |
| CSQ [36]   |          | 0.851    | 0.865    | 0.873   | 0.810  | 0.825  | 0.839  | 0.750  | 0.824  | 0.852  |
| DHD(Ours)  |          | 0.864    | 0.891    | 0.901   | 0.820  | 0.839  | 0.850  | 0.839  | 0.873  | 0.889  |
| DHD(Ours)  | ViT [10] | 0.927    | 0.938    | 0.944   | 0.837  | 0.862  | 0.870  | 0.886  | 0.919  | 0.939  |
| DHD(Ours)  | DeiT [31]| 0.932    | 0.943    | 0.948   | 0.839  | 0.861  | 0.867  | 0.883  | 0.913  | 0.925  |
| DHD(Ours)  | SwinT [27]| 0.944  | 0.955    | 0.956   | 0.848  | 0.867  | 0.875  | 0.894  | 0.930  | 0.945  |

Table 3. Ablation study on our work with ImageNet. ✓ indicates that the corresponding element is used. T denotes types of augmentation, and Eud. is the abbreviation of Euclidean distance.

To further demonstrate that DHD genuinely provides quality search outcomes, we plot graph of the PR curve and the precision for the top 1,000 retrieved images at 64 bits in Figures 3 and 4. As shown in the experimental results, we can confirm that the proposed DHD establishes the state-of-the-art retrieval performance.

Ablation study. In Table 3, we show the experimental results according to the presence or absence of our contributions. To minimize the dependency on the network architecture and explore our contribution correctly, we use AlexNet as the backbone. In (1), even when only $L_{HP}$ is applied, it shows good results compared to the AlexNet backbone existing methods in Table 2. In (2, 6, 7), search accuracy is improved in all cases where $L_{hr-Q}$ is applied. In (3, 4) the results with strong student augmentation $T_s$ show that simply applying it to training degrades the performance due to the emerged discrepancy in representation between Hamming and real space, while SdH mitigates this problem and
improves performance by properly exploiting the power of data augmentation, as shown in (5, 6, 7). In order to check whether the cosine distance is the best choice for SdH or not, we replace the cosine distance in Eqn 2 with the Euclidean distance and report the results in (6), but the results in (7) demonstrates that the cosine distance is better. To summarize the ablation results, each of our proposals affects the performance improvement, and the best is achieved when all of them are combined as in (7).

4.4. Analysis

Insensitivity to transformations. To investigate the sensitivity to transformations, we examine how the binary code shifts when transformed images are fed to ResNet backbone methods, by using ImageNet query set. We measure the average Hamming distance between the untransformed ($s_T = 0$) binary codes and the transformed ($s_T$ in (0, 1)), binary codes. As observed in Figure 5, CSQ DPN, and a model learned with $\mathcal{L}_{HP}$ are trained with $T_T$, showing sensitivity to transformations due to barely used augmentation. When the augmentation is applied ($\mathcal{L}_{HP} + T_S$), model is improved to be more robust to transformations, however, the map score decreases due to the discrepancy in representation between Hamming and Real space during training. On the other hand, the combined $\mathcal{L}_{HP} + \mathcal{L}_{SDH}$ exhibits the highest robustness while achieving the best mAP score, by minimizing discrepancy and successfully exploring the potential of strong augmentation.

Self-distilled Hashing with other methods. In order to prove that SdH can be applied to other deep hashing baselines [3, 4, 12, 36], we perform retrieval experiments with AlexNet backbone and show the results in Table 4. With SdH setup, we employ $T_T$ and $T_S$ groups to produce input views, and for without SdH setup, we only use $T_S$ to generate input views. By comparing the reported results of [3, 4]...
Table 4. mean Average Precision (mAP) scores with or without Self-distilled Hashing (SdH).

| Method        | ImageNet with SdH | ImageNet without SdH | NUS-WIDE with SdH | NUS-WIDE without SdH | MS COCO with SdH | MS COCO without SdH |
|---------------|-------------------|----------------------|-------------------|----------------------|------------------|---------------------|
|               | 16-bit | 64-bit | 16-bit | 64-bit | 16-bit | 64-bit | 16-bit | 64-bit | 16-bit | 64-bit |
| HashNet [4]   | 0.501  | 0.661  | 0.337  | 0.502  | 0.745  | 0.769  | 0.705  | 0.762  | 0.695  | 0.753  | 0.655  | 0.727  |
| DCH [3]       | 0.640  | 0.673  | 0.571  | 0.597  | 0.754  | 0.771  | 0.748  | 0.767  | 0.703  | 0.746  | 0.669  | 0.697  |
| DPN [12]      | 0.630  | 0.708  | 0.562  | 0.656  | 0.757  | 0.801  | 0.753  | 0.787  | 0.710  | 0.772  | 0.672  | 0.760  |
| CSQ [36]      | 0.634  | 0.711  | 0.570  | 0.662  | 0.759  | 0.804  | 0.757  | 0.793  | 0.707  | 0.765  | 0.670  | 0.752  |
| DHD (Ours)    | 0.657  | 0.721  | 0.583  | 0.671  | 0.780  | 0.820  | 0.775  | 0.806  | 0.749  | 0.792  | 0.731  | 0.766  |

Table 5. mAP scores on unseen deformations.

| Deformation       | with SdH | without SdH |
|-------------------|----------|-------------|
| None              | 0.891(2.3%↑) | 0.871       |
| Cutout            | 0.862(3.7%↑) | 0.827       |
| Dropout           | 0.810(7.9%↑) | 0.765       |
| Zoom in           | 0.658(19.0%↑) | 0.552       |
| Zoom out          | 0.816(1.4%↑) | 0.805       |
| Rotation          | 0.856(2.4%↑) | 0.836       |
| Shearing          | 0.842(2.7%↑) | 0.815       |
| Gaussian noise    | 0.768(10.5%↑) | 0.673       |

Robustness to unseen deformations. To further examine the generalization capacity of DHD, we conduct experiments with unseen (not seen during training) transformations to inputs following the evaluation protocol utilized in [14]. As reported in Table 5, deep hashing model with SdH significantly outperforms the model without SdH at all deformations, showing a performance difference of up to 19% (zoom in). In particular, SdH makes deep hashing model robust to per-pixel deformations such as dropout and Gaussian noise, even though SdH has not included any pixel-level transformations.

5. Conclusion

In this paper, we have proposed a novel Self-distilled Hashing (SdH) scheme for deep hashing learning that can generate robust hash codes from transformations. By maximizing the cosine similarity between hash codes of different views of one image, SdH minimizes the discrepancy in the representation due to augmentation and exploits its power in training. Besides, we optimized deep hashing model with proxy-based hash similarity learning and quantization loss of maximum likelihood manner. With all these proposals, we configured Deep Hash Distillation (DHD) framework that yields discriminative hash codes for hashing-based image retrieval systems. Experimental results on popular benchmarks validate the effectiveness of our approach with the state-of-the-art performance.

---

3Detailed deformation setup is listed in the supplementary material.
References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

[2] Malik Boudiaf, Jérôme Rony, Istiaq Masud Ziko, Eric Granger, Marco Pedersoli, Pablo Piantanida, and Ismail Ben Ayed. A unifying mutual information view of metric learning: cross-entropy vs. pairwise losses. In ECCV, pages 548–564. Springer, 2020.

[3] Yue Cao, Mingsheng Long, Bin Liu, and Jianmin Wang. Deep cauchy hashing for hamming space retrieval. In CVPR, pages 1229–1237, 2018.

[4] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. Layer normalization.

[5] Mathilde Caron, Hugo Touvron, Ishan Misra, Herve Jegou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In ICCV, pages 9650–9660, 2021.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. arXiv preprint arXiv:2002.05709, 2020.

[7] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In CVPR, pages 15750–15758, 2021.

[8] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, and Yanwen Guo. Multi-label image recognition with graph convolutional networks. In CVPR, pages 5177–5186, 2019.

[9] Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In ACM ICMR, pages 1–9, 2009.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In CVPR, 2020.

[11] et al. E. Riba. Kornia: an open source differentiable computer vision library for pytorch. In WACV, 2020.

[12] Lixin Fan, KamWoh Ng, Ce Ju, Tianyu Zhang, and Chee Seng Chan. Deep polarized network for supervised discrete hashing. In IJCV, pages 740–755. European conference on computer vision, 2015.

[13] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. PAMI, 35(12):2916–2929, 2012.

[14] Geonmo Gu, Byungsoo Ko, and Han-Gyu Kim. Proxy synthesis: Learning with synthetic classes for deep metric learning. In AAAI, 2021.

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

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[17] Young Kyun Jang and Nam Ik Cho. Generalized product quantization network for semi-supervised image retrieval. In CVPR, pages 3420–3429, 2020.

[18] Young Kyun Jang, Dong-ju Jeong, Seok Hee Lee, and Nam Ik Cho. Deep clustering and block hashing network for face image retrieval. In ACCV, pages 325–339. Springer, 2018.

[19] Dong-ju Jeong, Sung-Kwon Choo, Wonkyo Seo, and Nam Ik Cho. Classification-based supervised hashing with complementary networks for image search. In BMVC, page 74, 2018.

[20] Diederik P.Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2015.

[21] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In ICML deep learning workshop, volume 2. Lille, 2015.

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

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

[24] Yunqiang Li and Jan van Gemert. Deep unsupervised image hashing by maximizing bit entropy. In AAAI, 2020.

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.

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

[27] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixin Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In ICCV, 2021.

[28] Ilya Loshchilov and Frank Hutter. Sigma: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983, 2016.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. IJCV, 115(3):211–252, 2015.

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

[31] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983, 2016.

[32] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. IJCV, 115(3):211–252, 2015.

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

[34] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In ICML, pages 10347–10357. PMLR, 2021.

[35] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. PAMI, 40(4):769–790, 2017.
[33] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In NeurIPS, pages 1753–1760, 2009. 1, 2, 5, 6
[34] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In AAAI, 2014. 1, 2, 5, 6
[35] Ting-Bing Xu and Cheng-Lin Liu. Data-distortion guided self-distillation for deep neural networks. In AAAI, volume 33, pages 5565–5572, 2019. 2
[36] Li Yuan, Tao Wang, Xiaopeng Zhang, Francis EH Tay, Zequn Jie, Wei Liu, and Jiashi Feng. Central similarity quantization for efficient image and video retrieval. In CVPR, pages 3083–3092, 2020. 1, 2, 4, 5, 6, 7, 8
[37] Sukmin Yun, Jongjin Park, Kimin Lee, and Jinwoo Shin. Regularizing class-wise predictions via self-knowledge distillation. In CVPR, pages 13876–13885, 2020. 2
[38] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In AAAI, number 1, 2016. 1, 2, 5, 6
Introduction

In this supplementary material, we present a proof in A.a, detailed explanations in B.a and B.c, statistics in B.b, additional ablations in C.a, and additional visualized results in C.b to better understand the main paper. To reproduce our work, we provide a pseudo-code and training scheme in A.b, and an anonymous Github link which contains an executable code and a list of data.

A. Details on Deep Hash Distillation

A.a. Hamming Distance Analysis

For a given input image \( x_i \) and \( x_j \), a deep hashing model \( \mathcal{H} \) produces corresponding hash codes \( h_i \) and \( h_j \), which are quantized to binary codes \( b_i \) and \( b_j \) in \( \{-1, 1\}^K \) with sign operation \( b_i = \text{sign}(h_i) \), \( b_j = \text{sign}(h_j) \), respectively. For retrieval, Hamming distance \( D_H \) is computed with the binary codes as:

\[
D_H(b_i, b_j) = \text{XOR}(b_i, b_j),
\]

where XOR is a bit-wise count operation that outputs in the range \([0, K]\). From a mathematical point of view, XOR can be interpreted as:

\[
\text{XOR}(b_i, b_j) = \frac{1}{2} (K - b_i^T \cdot b_j)
\]

\[
= \frac{1}{2} (K - \|b_i\|_2 \|b_j\|_2 \, \mathcal{S}(b_i, b_j))
\]

\[
= \frac{K}{2} (1 - \mathcal{S}(b_i, b_j)),
\]

where \( \|b_i\|_2 = \|b_j\|_2 = \sqrt{K} \), and \( \mathcal{S}(\cdot, \cdot) \) denotes cosine similarity which also can be notated as:

\[
\mathcal{S}(b_i, b_j) = \cos \alpha_{ij} = \frac{b_i^T \cdot b_j}{\|b_i\|_2 \|b_j\|_2},
\]

where \( \alpha_{ij} \) is the angle between \( b_i \) and \( b_j \). It should be noted that \( 1 - \mathcal{S}(b_i, b_j) \) presents a cosine distance between \( b_i \) and \( b_j \) and can be approximated with \( h_i \) and \( h_j \) as:

\[
1 - \mathcal{S}(h_i, h_j) \simeq 1 - \mathcal{S}(b_i, b_j)
\]

where \( 1 - \mathcal{S}(h_i, h_j) \) is a cosine distance between \( h_i \) and \( h_j \). Therefore, minimizing the cosine distance between hash codes during the deep hashing training allows to reduce the Hamming distance between their binary codes.
A.b. Pseudo-code and Learning Algorithm

We provide a PyTorch-like pseudo-code in Algorithm 1 and detailed training process in Algorithm 2 for reproducibility of our Deep Hash Distillation (DHD) framework.

Algorithm 1 PyTorch-like pseudo-code details.

```python
# Import pytorch
import torch as T
# Import kornia library for augmentation
import kornia.augmentation as Kg
# https://github.com/kornia/kornia
# Import kornia library for augmentation
# Augmentation class to separate teacher/student.

class Augmentation():
    # Transformation defined in kornia
    def init(self, sT):
        self.Aug = Sequential(
            Kg.RandomResizedCrop(p=1.0*sT),
            Kg.RandomHorizontalFlip(p=0.5*sT),
            Kg.ColorJitter(p=0.8*sT),
            Kg.RandomGrayscale(p=0.2*sT),
            Kg.RandomGaussianBlur(p=0.5*sT))

    # Hash Proxy-based learning in Eqn 4 of paper.
    def HP_Loss():
        # Employ trainable hash proxies
        self.HP = Parameter(shape=(N_cls, N_bit))
        # Likelihood estimate
        self.BCE = T.nn.BCELoss()
        self.gp = T.exp(-0.5*((x-1)/std)**2)
        self.gn = T.exp(-0.5*((x+1)/std)**2)

        def forward(self, x):
            y = (x.sign().detach() + 1.0) / 2.0
            # Compute Binary Cross Entropy
            ln = self.BCE(hn, 1-y)
            lp = self.BCE(hp, y)
            return lp + ln
```

Algorithm 2 DHD training for batch size \( N_B \)

1. Initialize \( \theta_E \) with pretrained model weights.
2. Initialize \( \theta_H \) and \( \theta_P \) with Xavier initialization.
3. \( g^+(h) = \exp \left( -\frac{(h-1)^2}{2\sigma^2} \right) \)
4. \( g^-(h) = \exp \left( -\frac{(h+1)^2}{2\sigma^2} \right) \)

Input: Parameters of each component: \( \theta_E, \theta_H, \theta_P \)

Input: \( \mathcal{X}_B = \{(x_1, y_1), \ldots, (x_{N_B}, y_{N_B})\} \)

5. for \( n \) in \( \{1, \ldots, N_B\} \) do
6.   draw two transformations \( t_{2n-1} \sim \mathcal{T}_T, t_{2n} \sim \mathcal{T}_S \)
7.   \( \tilde{x}_{2n-1} \leftarrow t_{2n-1}(x_n) \)
8.   \( \tilde{x}_{2n} \leftarrow t_{2n}(x_n) \)
9.   \( h_{2n-1} \leftarrow \tanh(\theta_H(E_{\theta_E}(\tilde{x}_{2n-1}))) \)
10. \( h_{2n} \leftarrow \tanh((\theta_H(E_{\theta_E}(\tilde{x}_{2n})))) \)
11. \( p_{2n-1} \leftarrow P_{\theta_P}(h_{2n-1}) \)
12. end for
13. \( \ell_{SDH} \leftarrow \mathcal{L}_{SDH} \) with \( \{h_{2n-1}, h_{2n}\}_{n=1}^{N_B} \)
14. \( \ell_{HP} \leftarrow \mathcal{L}_{HP} \) with \( \{p_{2n-1}, y_n\}_{n=1}^{N_B} \)
15. \( \ell_{BCE-Q} \leftarrow \mathcal{L}_{BCE-Q} \) with \( g^+, g^- \), \( \{h_{2n-1}\}_{n=1}^{N_B} \)
16. \( \theta_{E,H} \leftarrow \theta_{E,H} - \gamma \left( \frac{\partial \ell_{SDH}}{\partial \theta_{E,H}} + \frac{\partial \ell_{HP}}{\partial \theta_{E,H}} + \frac{\partial \ell_{BCE-Q}}{\partial \theta_{E,H}} \right) \)
17. \( \theta_P \leftarrow \theta_P - \gamma \frac{\partial \ell_{HP}}{\partial \theta_P} \)

Output: Updated \( \theta_E, \theta_H, \theta_P \)
B. Experimental Setting

B.a. More Details on Implementation

The pretrained model weights of AlexNet [9] and ResNet [7] are from Torchvision1. Especially for transformer backbones, we adopt base pretrained model weights as: ViT [4] with vit_base_patch16_224, DeiT [13] with deit_base_distilled_patch16_224 and SwinT [11] with swin_base_patch4_window7_224, from timm2 open source library respectively. In order to fit in deep encoders, we crop the center of the dataset images to the size of $224 \times 224$ after augmentation for training, while cropping without augmentation for test.

There may be performance differences in the retrieval results of existing methods [2,3,5,10,14–16] depending on the implementation setups, but we set the same training strategy as: Adam optimizer [8], learning rate schedule [12] with warm-up for the first 10 epochs, and reducing the learning rate of the deep encoder by 1/20, for a fair comparison. Batch size is set to 128 as default, however, we adjust it as a half for pair-wise learning approaches [2,3] for stable learning, and as a quarter when training transformers, in order to fit in NVIDIA 3090 RTX 24GB GPUs. Other conditions for training are adopted with defaults proposed in each method.

B.b. Dataset Configuration and Statistics

During dataset preparation, we resize all images to $256 \times 256$. Figure A shows how distributed each category is in the training sets of retrieval datasets.

ImageNet is a single-labeled dataset to evaluate the classification performance, which consists of over 1.2M images in the training set and 50K images in the validation set. There are 1,000 categories to be classified. However, we employ a subset of ImageNet with 100 categories, where the training set and the query set are configured by assigning single label ($1/100 = 1\%$ of total) to each image. The training set is composed of the same 130 images per category.

NUS-WIDE is a multi-labeled dataset which contains 269,648 images crawled from the Web, all annotated with at least one concept out of a total of 81. We collect images with 21 most frequent concepts to perform experiments, where the training set and the query set are arranged to have at least 500 and 100 images for each category, respectively. To be specific with training set, one image has up to 10 labels ($10/21 = 48\%$ of total), with an average of 2.94 labels ($2.93/21 = 14\%$ of total).

MS COCO is a multi-labeled dataset containing 132,218 images of 80 categories, where each image is assigned one or more semantic labels. We adopt randomly sampled 10,000 and 5,000 images for training and test, respectively. In particular for training set, one image has up to 15 labels ($15/80 = 19\%$ of total), with an average of 2.94 labels ($2.94/80 = 4\%$ of total).

B.c. Deformation Setup

Following the experimental setup in [5] to investigate the quality of image embeddings, we use the imgaug3 library to evaluate the deep hashing model performance when unseen transformation (deformation) is given. Experimental results are reported in Table 5 of the main paper.

- **Cutout**: each input image is randomly filled by two grayish pixels that are 20% of the image size.
- **Dropout**: $p \times 100\%$ of pixels are dropped from each image where $p = \{t \mid 0 < t < 0.01\}$.
- **Zoom-in**: each input image is transformed by zoom-in at scale of 50%.
- **Zoom-out**: each input image is transformed by zoom-out at scale of 200%.
- **Rotation**: each input image is rotated at a randomly sampled degree $d$ where $d = \{t \mid -30^\circ < t < 30^\circ\}$.
- **Shearing**: each input image is sheared at a randomly sampled degree $d$ where $d = \{t \mid -30^\circ < t < 30^\circ\}$.
- **Gaussian Noise**: noise is sampled once per pixel from a normal distribution with std $s$ where $s = \{t \mid 0 < t < 25.5\}$ and added to each image.

---

1. https://pytorch.org/vision
2. https://github.com/rwightman/pytorch-image-models
3. https://github.com/aleju/imgaug

---

Figure A. Number of images per category in the training set.
C. Additional Experimental Results

C.a. Ablation Study on Hyper-parameters

To investigate the influences of hyper-parameters utilized in DHD, we perform retrieval experiments and show the results in Table B, A and C. Specifically, the mean Average Precision (mAP) scores are measured by varying the value of hyper-parameter to be investigated, while fixing others as defaults.

In Table A, we examine the temperature scaling parameter $\tau$. Following the results and the observation in Sec B.b, we set the optimal $\tau$ differently to each dataset, since cross entropy-based $\mathcal{L}_{H,P}$ should be handled according to how the labels are distributed. The more categories to classify, the lower $\tau$ is required to get more distinct predictions.

In Table B, we explore the impact of transformation occurrence $s_T$. It can be seen that the performance is almost similar when $s_T$ is less than 0.5, and the performance starts to degrade when $s_T$ is larger than 0.5. This indicates that dividing the augmentation groups as easy teacher and difficult student, giving teacher group to low $s_T$, is effective.

In Table C (2-5), we change the balancing parameter $\lambda_1$ of $\mathcal{L}_{SdH}$ and $\lambda_2$ of $\mathcal{L}_{bce-Q}$ to figure out the impact of training objectives in retrieval performance. From the results with little performance difference, it can be confirmed that the proposed loss functions are not sensitive to the hyper-parameters. In Table C (6, 7), we vary the standard deviation value of the Gaussian estimators of $\mathcal{L}_{bce-Q}$. The mAP scores show that providing a stricter estimator ($\sigma = 0.25$) tends to degrade performance, but that also did not seem to have a significant effect. From the results in Table C (8), we find out that applying LN does not change the results much.

C.b. Additional Visualized Results

Quantization. To further clarify the visualized results of $\mathcal{L}_{bce-Q}$, we plot histograms in Figure B with retrieval database set of more than 100,000 images, in line with Figure 6 (drawn with query set) of the main paper. From the quantization results for more images, the outstanding effect of $\mathcal{L}_{bce-Q}$ is remarkably observed.

Qualitative Results. In order to see whether the difficulty of transformation is really different according to $s_T$, we illustrate Figure C. For $T_r$, images do not significantly deviated from the original. However, for $T_s$, some images are distorted to the point of being hard to recognize. Figure D shows what images are actually retrieved as results when transformation is applied. We can confirm that our DHD is robust to transformation and achieves high quality results.
Figure B. 3D visualized histograms of output hash code elements from (i) ImageNet, (ii) NUS-WIDE, and (iii) MS COCO. x-axis presents value of each hash code element $h$, y-axis presents bit position, and z-axis presents frequency counts.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016. iv
[2] Yue Cao, Mingsheng Long, Bin Liu, and Jianmin Wang. Deep cauchy hashing for hamming space retrieval. In CVPR, pages 1229–1237, 2018. iii
[3] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. Hashnet: Deep learning to hash by continuation. In CVPR, pages 5608–5617, 2017. iii
[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In ICLR, 2020. iii
[5] Lixin Fan, Kam Woh Ng, Ce Ju, Tianyu Zhang, and Chee Seng Chan. Deep polarized network for supervised learning of accurate binary hashing codes. In IJCAI, pages 825–831, 2020. iii
[6] Geonmo Gu, Byungsoo Ko, and Han-Gyu Kim. Proxy synthesis: Learning with synthetic classes for deep metric learning. In AAAI, 2021. iii
[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, pages 770–778, 2016. iii
[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2015. iii
[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In NeurIPS, pages 1097–1105, 2012. iii
[10] Hanjiang Lai, Yan Pan, Ye Liu, and Shuicheng Yan. Simultaneous feature learning and hash coding with deep neural networks. In CVPR, pages 3270–3278, 2015. iii
[11] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In ICCV, 2021. iii
[12] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983, 2016. iii
[13] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In ICML, pages 10347–10357. PMLR, 2021. iii
[14] Rongkai Xia, Yan Pan, Hanjiang Lai, Cong Liu, and Shuicheng Yan. Supervised hashing for image retrieval via image representation learning. In AAAI, 2014. iii
[15] Li Yuan, Tao Wang, Xiaopeng Zhang, Francis EH Tay, Ze-quin Jie, Wei Liu, and Jiashi Feng. Central similarity quantization for efficient image and video retrieval. In CVPR, pages 3083–3092, 2020. iii
[16] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In AAAI, number 1, 2016. iii
Figure C. Visualized augmentation results of different groups: weakly-transformed teacher $T_T$ and strongly-transformed student $T_S$.

Figure D. Above: Examples of various transformations applied to the original image. The green box indicates the same search result as the original, and the red box indicates otherwise. Below: Retrieved images on ImageNet dataset. The image output from the strongly transformed $T_T$ and the image transformed to gray scale show different retrieval results. Nevertheless, it can be seen that visually similar images of the same content are retrieved well.