Deep Hash Distillation for Image Retrieval

Supplementary Material

Contents

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 ($\mathrm{b}_i = \text{sign}(\mathrm{h}_i)$, $\mathrm{b}_j = \text{sign}(\mathrm{h}_j)$), respectively. For retrieval, Hamming distance $\mathcal{D}_{\mathrm{H}}$ is computed with the binary codes as:

$$\mathcal{D}_{\mathrm{H}}(\mathrm{b}_i, \mathrm{b}_j) = \text{XOR}\left(\mathrm{b}_i, \mathrm{b}_j\right), \tag{i}$$

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:

$$\begin{aligned} \text{XOR}\left(\mathrm{b}_i, \mathrm{b}_j\right) &= \frac{1}{2}\left(K - \mathrm{b}_i^T \cdot \mathrm{b}_j\right) \\ &= \frac{1}{2}\left(K - \left\|\mathrm{b}_i\right\|_2 \left\|\mathrm{b}_j\right\|_2 \mathcal{S}\left(\mathrm{b}_i, \mathrm{b}_j\right)\right) \\ &= \frac{K}{2}\left(1 - \mathcal{S}\left(\mathrm{b}_i, \mathrm{b}_j\right)\right), \end{aligned} \tag{ii}$$

where $\|\mathrm{b}_i\|_2 = \|\mathrm{b}_j\|_2 = \sqrt{K}$, and $\mathcal{S}(\cdot,\cdot)$ denotes cosine similarity which also can be notated as:

$$\begin{aligned} \mathcal{S}\left(\mathrm{b}_i, \mathrm{b}_j\right) &= \cos \alpha_{ij} \\ &= \frac{\mathrm{b}_i^T \cdot \mathrm{b}_j}{\left\|\mathrm{b}_i\right\|_2 \left\|\mathrm{b}_j\right\|_2}, \end{aligned} \tag{iii}$$

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

$$1 - \mathcal{S}\left(\mathrm{b}_i, \mathrm{b}_j\right) \simeq 1 - \mathcal{S}\left(\mathrm{h}_i, \mathrm{h}_j\right) \tag{iv}$$

where $1 - \mathcal{S}(\mathrm{h}_i, \mathrm{h}_j)$ 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 and 3 for reproducibility of our Deep Hash Distillation (DHD) framework.

Algorithm 1 Data augmentation and loss functions.

```
# Import pytorch
import torch as T
# Import kornia library for augmentation
# https://https://github.com/kornia/kornia
import kornia.augmentation as Kg

# Augmentation class to separate teacher/student.
class Augmentation():

    # Transformation defined in kornia
    # Apply augmentation sequentially
    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))

    def forward(self, x):
        return self.Aug(x)

# Hash Proxy-based learning in Eqn 4 of paper.
class HP_Loss():

    # Employ trainable hash proxies
    def init(self, N_cls, N_bit):
        self.HP = Parameter(shape=(N_cls, N_bit))

    def forward(self, h, tau, L):
        # Compute cosine similarity
        # and Temperature scaling
        D = cos_sim(h, self.HP) / tau
        # Label normalization
        L = normalize(L)
        return (-L * log_softmax(D)).sum()

# Reducing quantization error in Eqn 6 of paper.
class BCE-Q_Loss():

    # Utilize two Gaussian estimators
    def init(self, std):
        self.gp = T.exp(-0.5*((x-1)/std)**2)
        self.gn = T.exp(-0.5*((x+1)/std)**2)
        self.BCE = T.nn.BCELoss()

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

Algorithm 2 Self-distilled Hashing training scheme.

```
# 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()
```

Algorithm 3 Entire 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: $\mathcal{X}_B = \{(x_1, \mathrm{y}_1), ..., (x_{N_B}, \mathrm{y}_{N_B})\}$
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:     $\mathrm{h}_{2n-1} \leftarrow \tanh(H_{\theta_H}(E_{\theta_E}(\tilde{x}_{2n-1})))$
10:    $\mathrm{h}_{2n} \leftarrow \tanh((H_{\theta_H}(E_{\theta_E}(\tilde{x}_{2n}))))$
11:    $\mathrm{p}_{2n-1} \leftarrow P_{\theta_P}(\mathrm{h}_{2n-1})$
12: end for
13: $\ell_{SdH} \leftarrow \mathcal{L}_{SdH}$ with $\{\mathrm{h}_{2n-1}, \mathrm{h}_{2n}\}_{n=1}^{N_B}$
14: $\ell_{HP} \leftarrow \mathcal{L}_{HP}$ with $\{\mathrm{p}_{2n-1}, y_n\}_{n=1}^{N_B}$
15: $\ell_{bce\text{-}Q} \leftarrow \mathcal{L}_{bce\text{-}Q}$ with $g^+, g^-, \{\mathrm{h}_{2n-1}\}_{n=1}^{N_B}$
16: $\theta_{E,H} \leftarrow \theta_{E,H} - \gamma\left(\frac{\partial \ell_{SdH}}{\partial \theta_{E,H}} + \frac{\partial \ell_{HP}}{\partial \theta_{E,H}} + \frac{\partial \ell_{BCE\text{-}Q}}{\partial \theta_{E,H}}\right)$
17: $\theta_P \leftarrow \theta_P - \gamma \frac{\partial \ell_{HP}}{\partial \theta_P}$
Output: Updated $\theta_E, \theta_H, \theta_P$
```

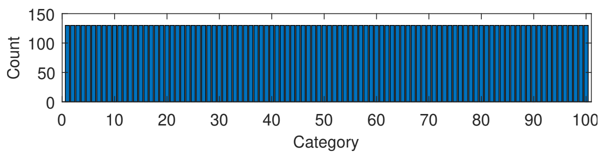
B. Experimental Setting

B.a. More Details on Implementation

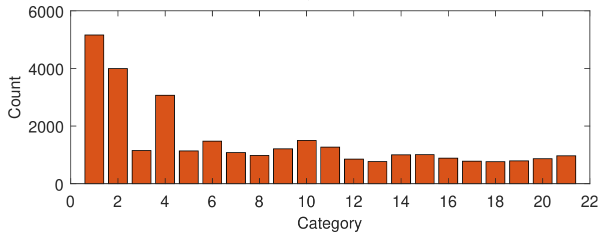
The pretrained model weights of AlexNet [8] and ResNet [6] are from Torchvision[1]. Especially for transformer backbones, we adopt base pretrained model weights as: ViT [3] with `vit_base_patch16_224`, DeiT [12] with `deit_base_distilled_patch16_224` and SwinT [10] with `swin_base_patch4_window7_224`, from timm [2] open source library respectively. In order to fit in to deep encoders, we crop the center of the dataset images to the size of 224×224 after augmentation for training, while cropping without augmentation for test.

There may be performance differences in the retrieval results of existing methods [1,2,4,9,13–15] depending on the implementation setups, but we set the same training strategy as: Adam optimizer [7], learning rate schedule [11] 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 [1,2] 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.

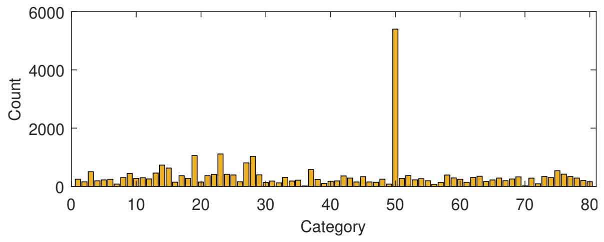
B.b. Dataset Configuration and Statistics



(a) ImageNet



(b) NUS-WIDE



(c) MS COCO

Figure A. Number of images per category in the training set.

During dataset preparation, we resize all images to 256×256. 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 *imgaug*[3] 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×100% of pixels are dropped from each image where $p = \{t|0 < t < 0.01\}$.
- **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 $d = \{t| -30° < t < 30°\}$.
- **Shearing**: each input image is sheared at a randomly sampled degree d where $d = \{t| -30° < t < 30°\}$.
- **Gaussian Noise**: noise is sampled once per pixel from a normal distribution with std s where $s = \{t|0 < t < 25.5\}$ and added to each image.

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

	ImageNet $s_T = 0.5$			NUS-WIDE $s_T = 0.5$			MS COCO $s_T = 0.5$		
Hyper-parameter	16-bit	32-bit	64-bit	16-bit	32-bit	64-bit	16-bit	32-bit	64-bit
$\tau = 0.2$	**0.864**	**0.891**	**0.901**	0.791	0.825	0.840	0.818	0.868	0.888
$\tau = 0.4$	0.858	0.884	0.892	0.812	0.839	0.850	**0.839**	**0.873**	**0.889**
$\tau = 0.6$	0.858	0.881	0.887	**0.820**	**0.839**	**0.850**	0.829	0.866	0.878
$\tau = 0.8$	0.850	0.874	0.886	0.813	0.837	0.846	0.815	0.854	0.876
$\tau = 1.0$	0.856	0.871	0.882	0.817	0.836	0.845	0.809	0.854	0.872

Table A. mAP scores by varying τ on ResNet backbone DHD, where the setup utilized in the main paper is shown in bold.

	ImageNet $\tau = 0.2$			NUS-WIDE $\tau = 0.6$			MS COCO $\tau = 0.4$		
Hyper-parameter	16-bit	32-bit	64-bit	16-bit	32-bit	64-bit	16-bit	32-bit	64-bit
$s_T = 0.2$	0.866	0.888	0.897	0.812	0.838	0.849	0.839	0.871	0.889
$s_T = 0.4$	0.866	0.889	0.896	0.810	0.839	0.848	0.837	0.873	0.888
$s_T = 0.5$	**0.864**	**0.891**	**0.901**	**0.820**	**0.839**	**0.850**	**0.839**	**0.873**	**0.889**
$s_T = 0.6$	0.860	0.885	0.895	0.817	0.839	0.849	0.834	0.871	0.886
$s_T = 0.8$	0.858	0.881	0.890	0.819	0.837	0.843	0.828	0.866	0.883
$s_T = 1.0$	0.845	0.874	0.884	0.815	0.833	0.841	0.825	0.860	0.878

Table B. mAP scores by varying s_T on ResNet backbone DHD, where the setup utilized in the main paper is shown in bold.

	λ_1	λ_2	σ	16-bit	64-bit
(1)	0.1	0.1	0.5	**0.657**	**0.721**
(2)	0.05	0.1	0.5	0.652	0.716
(3)	0.5	0.1	0.5	0.654	0.715
(4)	0.1	0.05	0.5	0.658	0.717
(5)	0.1	0.5	0.5	0.652	0.713
(6)	0.1	0.1	0.25	0.654	0.716
(7)	0.1	0.1	1.0	0.656	0.717

Table C. Ablation study on hyper-parameters of DHD on ImageNet. The setup utilized in the main paper is shown in bold.

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 τ. Following the results and the observation in Sec B.b, we set the optimal τ differently to each dataset, since cross entropy-based $\mathcal{L}_{HP}$ should be handled according to how the labels are distributed. The more categories to classify, the lower τ 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 λ_1 of $\mathcal{L}_{SdH}$ and λ_2 of $\mathcal{L}_{bce\text{-}Q}$ 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 $\mathcal{L}_{bce\text{-}Q}$. 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

Trainable Hash Proxies. We additionally visualized the pairwise cosine similarity of trainable hash proxies for all classes in each dataset in Figures B, C, D, with ResNet backbone @ 64-bit. For single label ImageNet dataset, semantically similar classes show higher similarity such as *European fire salamander-spotted salamander* and *electric locomotive-passenger car*. For multi-label datasets, the dependency between labels stands out. Classes that frequency appear together in a single image show higher similarity such as *flowers-plants* and *ocean-beach* in NUS-WIDE, or *baseball bat-baseball glove* and *truck-bus-car* in MS COCO. Particularly, we observed that discriminability can still be preserved when two classes have high cosine similarity as shown in the *buildings-window-vehicle* case of NUS-WIDE. Here the cosine similarity of *buildings-window* (0.48) and *window-vehicle* (0.55) is high, but the cosine similarity of *buildings-vehicle* (0.08) is low.

Qualitative Results. In order to see whether the difficulty of transformation is really different according to s_T, we illustrate Figure E. For $\mathcal{T}_T$, images do not significantly deviated from the original. However, for $\mathcal{T}_S$, some images are distorted to the point of being hard to recognize. Figure F 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.

References

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

[2] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. Hashnet: Deep learning to hash by continuation. In *CVPR*, pages 5608–5617, 2017. iii

[3] 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

[4] Lixin Fan, KamWoh 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

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

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

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. iii

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

[9] 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

[10] 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

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

[12] 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

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

[14] 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. iii

[15] Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *AAAI*, 2016. iii

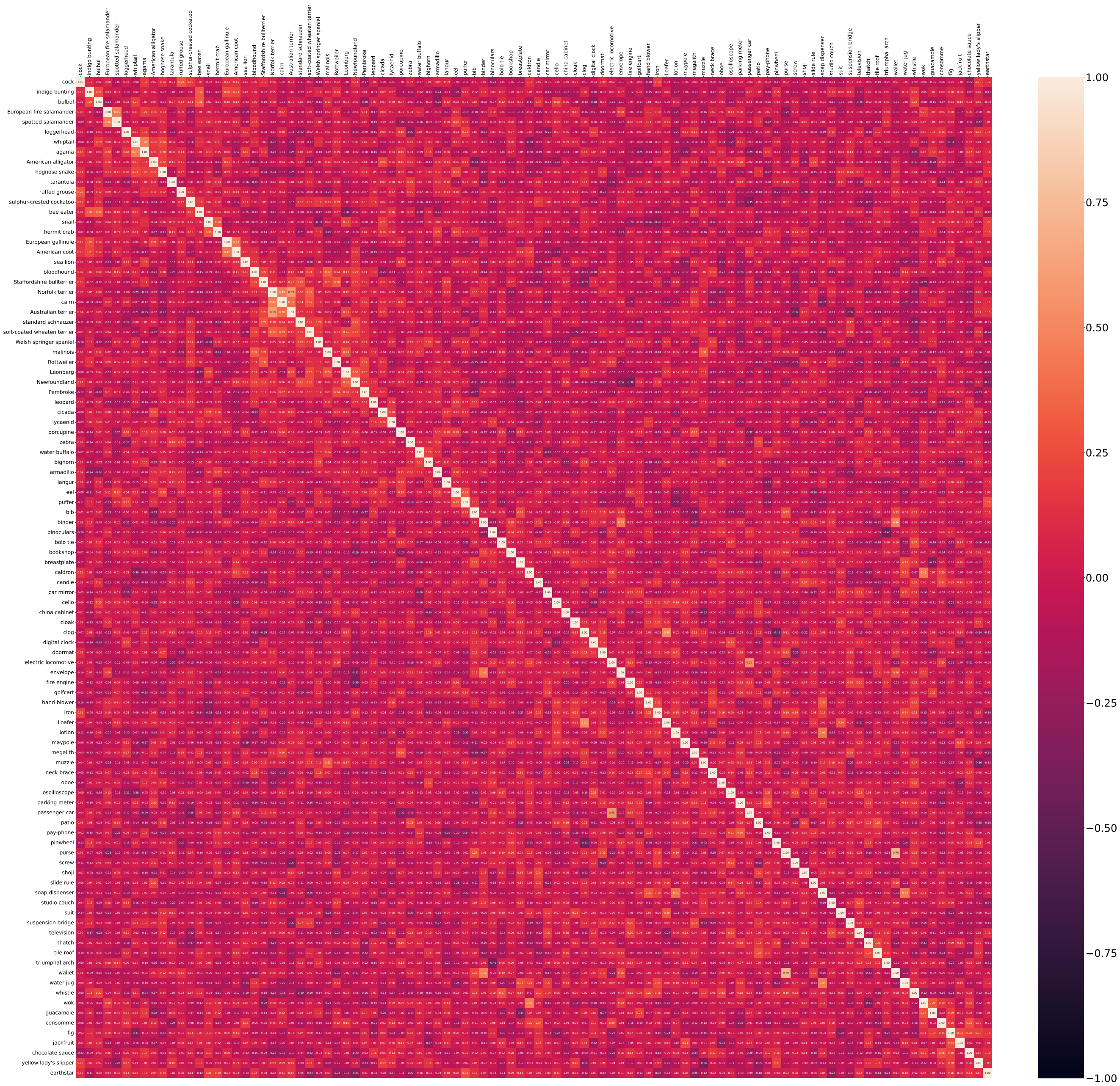


Figure B. Pairwise cosine similarities of trainable hash proxies on ImageNet.

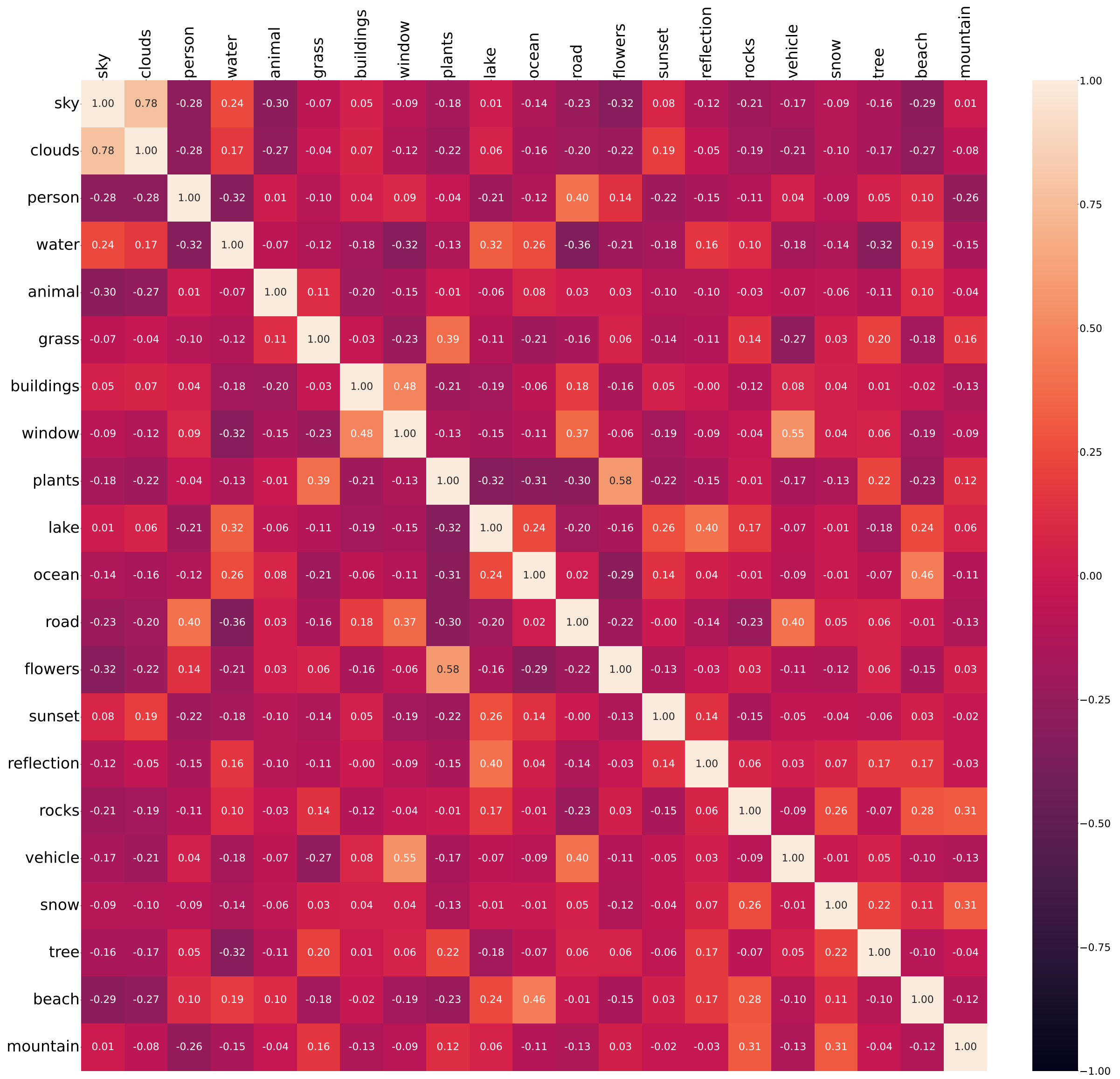


Figure C. Pairwise cosine similarities of trainable hash proxies on NUS-WIDE.

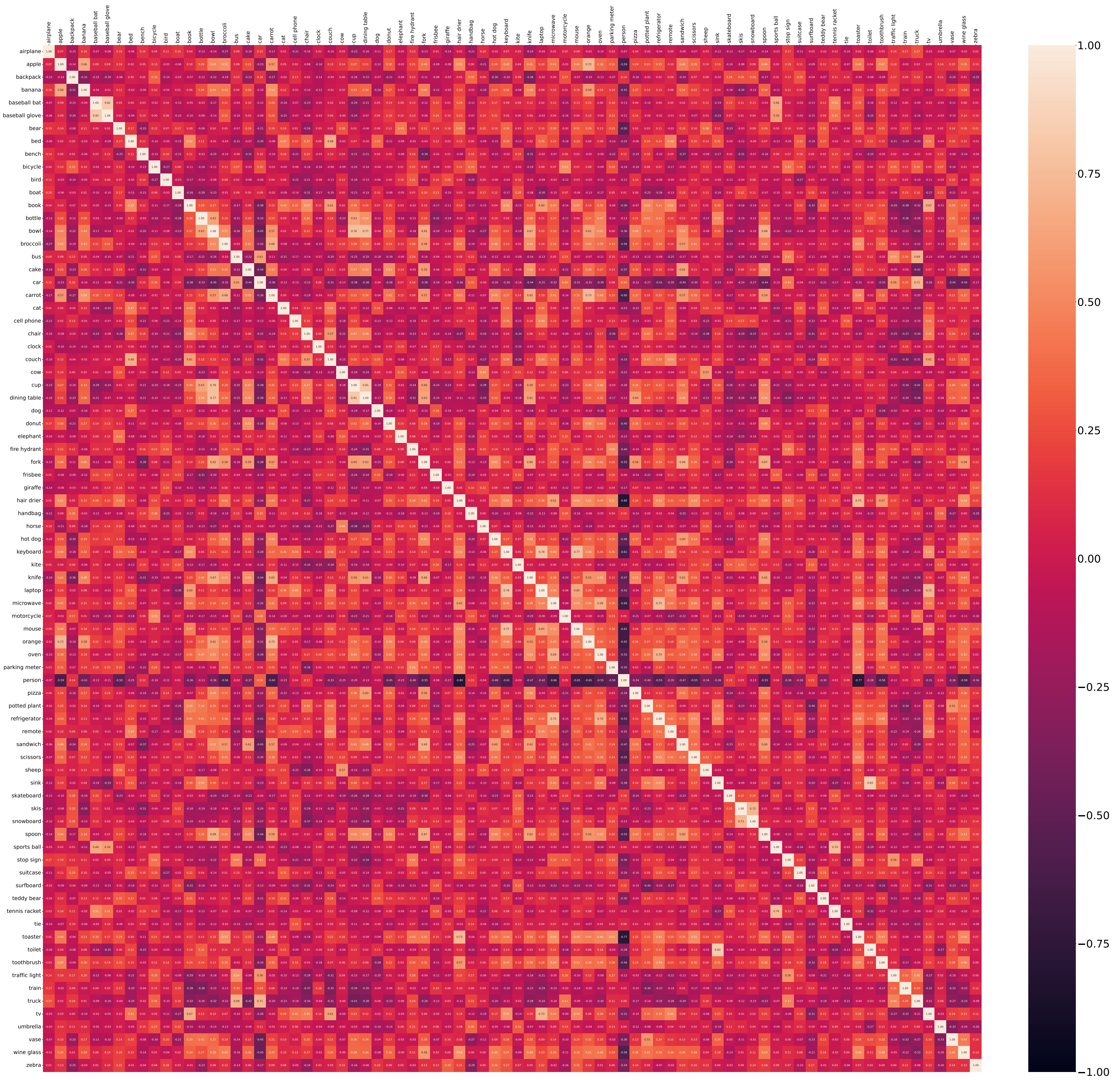


Figure D. Pairwise cosine similarities of trainable hash proxies on MS COCO.

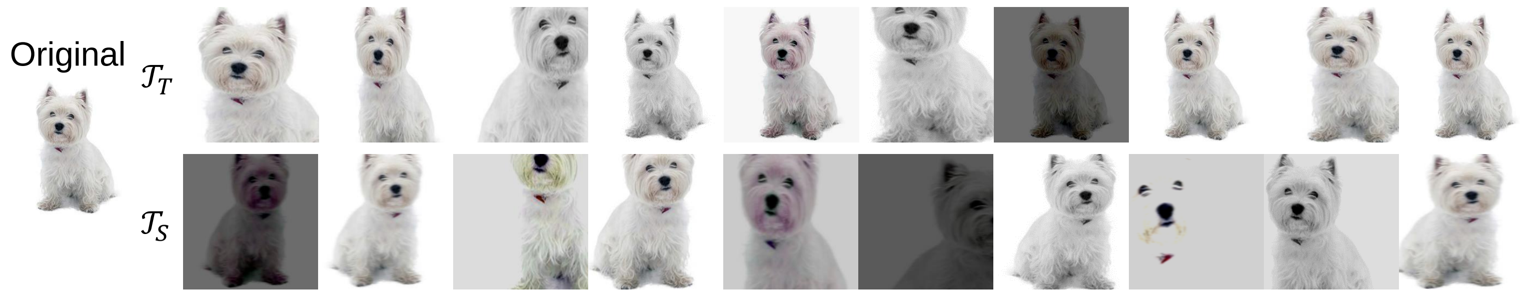


Figure E. Visualized augmentation results of different groups: weakly-transformed teacher $\mathcal{T}_T$ and strongly-transformed student $\mathcal{T}_S$.

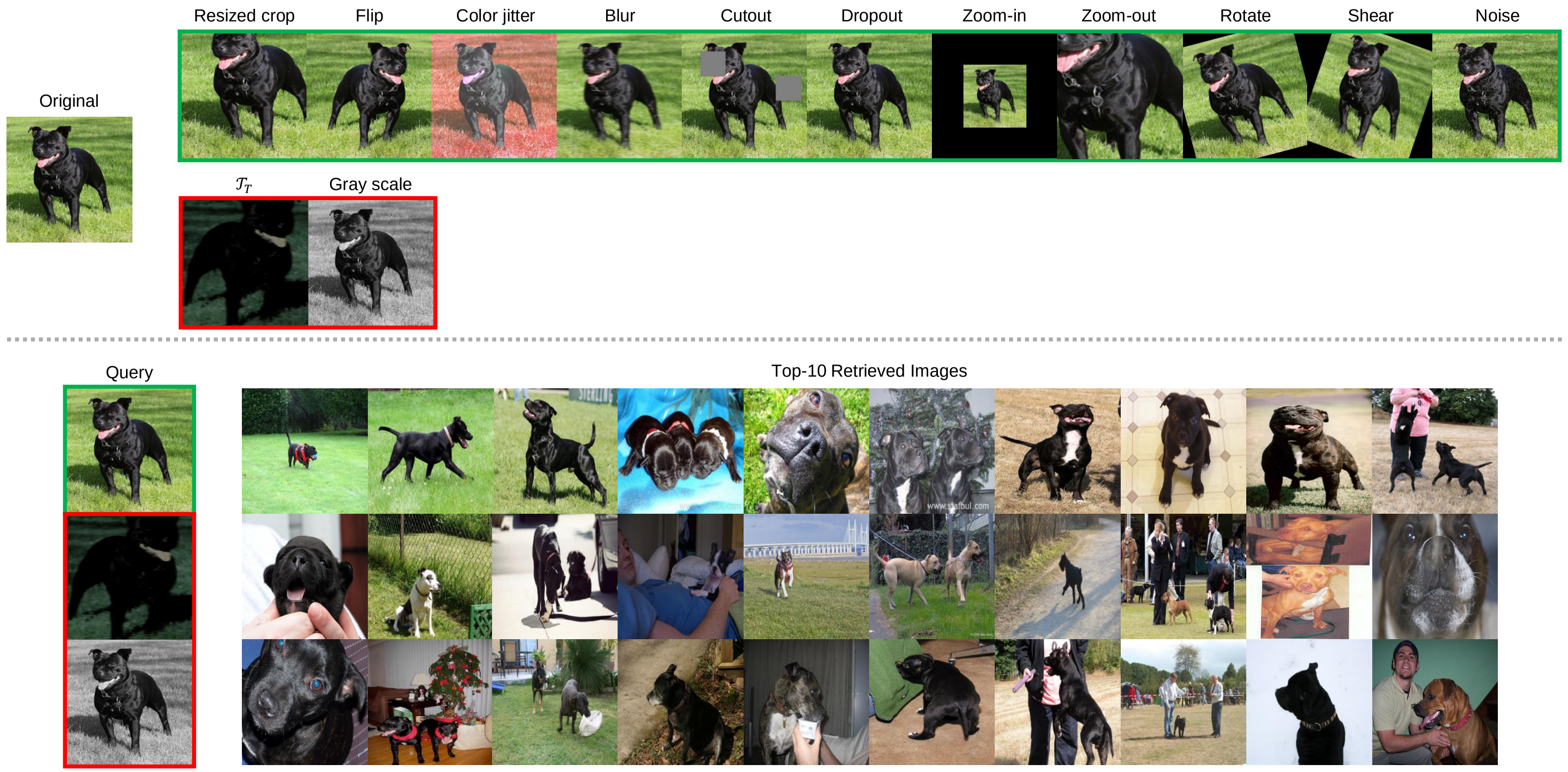


Figure F. Above: Examples of various transformations applied to the original image. The green box indicates the same search result as the original, and the red box indicates otherwise. Below: Retrieved images on ImageNet dataset. The image output form the strongly transformed $\mathcal{T}_T$ and the 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.