Supplementary Material

Kam Woh Ng[1,2]
kamwoh.ng@surrey.ac.uk
Xiatian Zhu[1,3]
xiatian.zhu@surrey.ac.uk
Jiun Tian Hoe[4]
jiuntian001@e.ntu.edu.sg
Chee Seng Chan[5]
cs.chan@um.edu.my
Tianyu Zhang[6]
macwish@hotmail.com
Yi-Zhe Song[1,2]
y.song@surrey.ac.uk
Tao Xiang[1,2]
t.xiang@surrey.ac.uk

[1] CVSSP, University of Surrey
[2] iFlyTek-Surrey Joint Research Centre on Artificial Intelligence, University of Surrey
[3] Surrey Institute for People-Centred Artificial Intelligence, University of Surrey
[4] Nanyang Technological University
[5] CISiP, Universiti Malaya
[6] GAC R&D Center

A Training setup

Environment. Our machine is with 16 core Intel i7-5960X CPU, 64GB RAM, and 4 Nvidia Titan XP GPU and 1 GTX 3070 GPU. Each experiment is run by using only 1 GPU. We are using PyTorch [14] for all of our experiments.

Code Implementations. For LsH [6, 9], SH [21], ITQ [7], SSDH [23], TBH [19] Greedy-Hash [20], CIBHash [17] and Bihalf [12] methods, we referred from open-source repository (some are from authors) at [1], [2], [3], [4], [5], [6], [7], and [8] respectively. We implemented all the methods with PyTorch [14]. We follow authors' original setting for the hyperparameters of the methods (either from the paper or the released codes).

Hyperparameters for Category Datasets. For all datasets, we train for 100 epochs on all methods using *Adam* Optimizer [10] with initial learning rate of 0.0001, weight decay of 0.0005, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We lowered the learning rate to 0.00001 after 80 epochs of training. We train all methods with batch size of 64 on a single GPU. For our **SDP**, the quantization error term $\lambda_{\text{quan}} = 1$.

[1] https://github.com/TreezzZ/LSH_PyTorch
[2] https://github.com/TreezzZ/SH_PyTorch
[3] https://github.com/TreezzZ/ITQ_PyTorch
[4] https://github.com/yangerkun/IJCAI2018_SSDH
[5] https://github.com/ymcidence/TBH
[6] https://github.com/ssppp/GreedyHash
[7] https://github.com/qiuzx2/CIBHash
[8] https://github.com/liyunqianggyn/Deep-Unsupervised-Image-Hashing

Hyperparameters for Instance Datasets. For GLDv2, we train for 20 epochs on selected methods (*i.e.*, ITQ [7], GreedyHash [20] and Bihalf [12]) using *Adam* Optimizer [10] with initial learning rate of 0.0001, weight decay of 0.0005, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We lowered the learning rate to 0.00001 after 10 epochs of training. We train all methods with batch size of 256 on a single GPU. We notice that using the default scale of quantization loss term is not efficient for instance level datasets, we suspect that because instance level features are very sensitive to the similarity, thus high quantization loss term would easily lead to trivial solution (*i.e.*, more severe similarity collapsing). Hence, we use a much lower quantization term, *i.e.*, 0.01 for GreedyHash, 0.1 for Bihalf, and 0.01 for our SDP.

Hash function h. Following the recent unsupervised hashing works [17, 19], we employ a non-linear hash function with the architecture `[nn.Linear(d,4096), nn.GELU(), nn.Linear(4096,k), nn.BatchNorm1d(k)]` (with default PyTorch's weight initialization) for category-level datasets. For instance-level datasets, we follow the recent work [8] and employ a linear hash function `[nn.Linear(d,k), nn.BatchNorm1d(k)]`.

A.1 Datasets

By following the protocol setup by previous works [6, 12, 17, 19, 20], we have chosen 4 category-level datasets, *i.e.*, i) **CIFAR-10**, ii) **NUS-WIDE**, iii) **MS-COCO** and iv) **ImageNet100**. For instance level, we follow the previous work [8] and choose **GLDv2**, $\mathcal{R}$**Oxf** and $\mathcal{R}$**Paris** for evaluation. Please note that we follow the dataset separation settings from previous works for all datasets.

CIFAR-10 [11]. We combine the training and testing images which have a total of 60K images. We then randomly pick 100 images from each class as queries (1K in total), and the remaining 59K as database images. Finally, 5K images are sampled from the database as training images.

ImageNet100 [4]. We are using a subset of ImageNet [4] as used by [2] and followed by many supervised deep hashing works. It consists of 128K images from 100 classes. All validation images from the 100 classes are used as queries (1K in total), and the remaining 128K as database images. Finally, 13K images are sampled from the database as training images.

NUS-WIDE [3]. It consists of 81 concepts with 269K multi-labeled images. 21 of the most frequent concepts are selected which contains 195K images. 100 images are selected randomly per concept as queries (2.1K in total) while the remaining 193K as database images. Finally, 500 images per concept are randomly sampled from the database as training images (10.5K in total).

MS-COCO [13]. We are using a public released dataset from [2][9] where images with no category information have been removed. Both training and validation images are then combined which have a total of 122K images. 5K images are selected randomly as queries while the remaining 117K as database images. Finally, 10K images are randomly sampled from the database as training images.

Google Landmark Datasets V2 (GLDv2) [22]. To understand the effectiveness of unsupervised hashing methods in large-scale instance-level retrieval task (*i.e.*, large number of classes), we choose GLDv2 for large-scale experiments. Specifically, we choose a cleaned version of GLDv2, namely, GLDv2-train-clean as training set. It has 1.2M training images

[9] https://github.com/thuml/HashNet/tree/master/pytorch/data/coco

with 81K classes, 1129 queries and 762K database images. Due to the expensive cost of training from scratch, we use their released pre-trained model[10] (**R50-DELG-GLDv2-clean** [1]) to compute the global features (average-pooled feature map) for the training, queries and database images with all the images scaled to 512×512. The features are 2048-dimension vectors. We then train the hash function with the computed features (*i.e.*, GLDv2-trained), then use it to compute hash codes for queries and database images for evaluations.

$\mathcal{R}$**Oxf** and $\mathcal{R}$**Paris** [18] are revisited annotated datasets of Oxford [15] and Paris [16]. Both $\mathcal{R}$Oxf/$\mathcal{R}$Par contain 70 images as query and 4993/6322 database images. Since there are no training set, we are also using the pre-trained **R50-DELG-GLDv2-clean** to compute the global features, but follow DELG[1] settings with 3 scales $\{\frac{1}{\sqrt{2}}, 1, \sqrt{2}\}$ to produce multi-scale image representations, and the 3 features are first l_2 normalized, average-pooled to obtain a single feature vector, and l_2 normalized again. Images are scaled from 1024 with 3 scales and the aspect ratio was remained. We then use the GLDv2-trained hash function to compute hash codes for evaluations.

[10] https://github.com/tensorflow/models/tree/master/research/delf

B Ablation study

Calibration distribution. We evaluate the effect of calibration distribution. Here we set the contrastive loss $\mathcal{L}_{\text{cl}} = 0$. We further test normalized `Gaussian` distribution (bounded within $[-1,1]$) as well as a variety of `Beta` distributions. We observe in Table 2 that (1) The performance `Beta` calibration is generally stable in the range of $[2,5]$; (2) `Gaussian` calibration ($N(\mu = 0, \sigma = 0.3)$) is similarly effective suggesting the flexibility of our SDC in distribution selection.

Distributions	CIFAR10	ImageNet100	GLDv2
`Beta`$(0.2,0.2)$	19.3	70.3	11.6
`Beta`$(0.5,0.5)$	54.8	76.3	11.9
`Beta`$(0.7,0.7)$	55.9	77.2	11.8
`Beta`$(1,1)$	58.4	79.0	12.0
`Beta`$(2,2)$	62.5	80.2	**12.1**
`Beta`$(3,3)$	62.5	80.3	12.0
`Beta`$(5,5)$	**63.0**	80.4	**12.1**
`Beta`$(7,7)$	62.6	**81.3**	11.8
`Beta`$(10,10)$	61.3	80.7	11.8
`Gaussian`	62.6	80.2	11.5

Table 1: Effect of the calibration distribution. *Setting*: 64-bits hash codes on CIFAR10 and ImageNet and 512-bits on GLDv2.

Contrastive loss. We evaluate the effect of contrastive loss. Here we set $\alpha = \beta = 5$. By incorporating contrastive loss ($\mathcal{L}_{cl}$), we observe notable performance enhancements. This improvement can be attributed to the self-supervised loss's ability to better adapt to the dataset's domain. Consequently, we observe a substantial performance gap on CIFAR10, while a minimal gap is observed on ImageNet100.

	CIFAR10	ImageNet100
w/ $\mathcal{L}_{cl}$	**66.3**	**80.6**
wo/ $\mathcal{L}_{cl}$	63.0	80.4

Table 2: Effect of the contrastive loss $\mathcal{L}_{cl}$. *Setting*: 64-bits hash codes on CIFAR10 and ImageNet.

Algorithm 1: SDC Loss Function.

```
# phi: non-linear hash layer
# N: number of pairs (half of the batch)
# x: features (2N, d), x2: features of randomly augmented input (2N, d)

# compute continuous codes
f = phi(x) # (2N, k)

# construct pairs
xi, xj = x[:N], x[N:] # (N, d)
fi, fj = f[:N], f[N:] # (N, k)
t, t_idx = sort(cossim(xi, xj)) # (N,)
s = cossim(fi, fj) # (N,)

# sort with t's index
s = s[t_idx] # (N,)

# inverse CDF of beta distribution
C = beta_ppf(N, alpha=5, beta=5) # (N,)

# contrastive loss with simclr
loss_cl = SimCLR(x, x2)

loss_sdc = (s - C).abs().mean()
loss_q = (1 - cossim(f, f.sign())).mean()
loss = loss_sdc + lambda_q * loss_q + lambda_cl * loss_cl
```

C Algorithm

The algorithm of our method is summarized in Algorithm 1.

References

[1] Bingyi Cao, André Araujo, and Jack Sim. Unifying deep local and global features for image search. In *European Conference on Computer Vision*, 2020.

[2] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S. Yu. Hashnet: Deep learning to hash by continuation. In *International Conference on Computer Vision*, 2017.

[3] 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 *Conference on Image and Video Retrieval*, 2009.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, 2009.

[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 *International Joint Conference on Artificial Intelligence*, 2020.

[6] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *International Conference on Very Large Data Bases*, 1999.

[7] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012.

[8] Jiun Tian Hoe, Kam Woh Ng, Tianyu Zhang, Chee Seng Chan, Yi-Zhe Song, and Tao Xiang. One loss for all: Deep hashing with a single cosine similarity based learning objective. In *Advances in Neural Information Processing Systems*, 2021.

[9] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Annual ACM Symposium on Theory of Computing*, 1998.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[11] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[12] Yunqiang Li and Jan van Gemert. Deep unsupervised image hashing by maximizing bit entropy. In *AAAI Conference on Artificial Intelligence*, 2021.

[13] 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 *European Conference on Computer Vision*, 2014.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

[15] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Computer Vision and Pattern Recognition*, 2007.

[16] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Computer Vision and Pattern Recognition*, 2008.

[17] Zexuan Qiu, Qinliang Su, Zijing Ou, Jianxing Yu, and Changyou Chen. Unsupervised hashing with contrastive information bottleneck. In *International Joint Conference on Artificial Intelligence*, 2021.

[18] Filip Radenovic, Ahmet Iscen, Giorgos Tolias, Yannis Avrithis, and Ondrej Chum. Revisiting oxford and paris: Large-scale image retrieval benchmarking. In *Computer Vision and Pattern Recognition*, 2018.

[19] Yuming Shen, Jie Qin, Jiaxin Chen, Mengyang Yu, Li Liu, Fan Zhu, Fumin Shen, and Ling Shao. Auto-encoding twin-bottleneck hashing. In *Computer Vision and Pattern Recognition*, 2020.

[20] Shupeng Su, Chao Zhang, Kai Han, and Yonghong Tian. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *Advances in Neural Information Processing Systems*, 2018.

[21] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, 2009.

[22] Tobias Weyand, Andre Araujo, Bingyi Cao, and Jack Sim. Google landmarks dataset v2-a large-scale benchmark for instance-level recognition and retrieval. In *Computer Vision and Pattern Recognition*, 2020.

[23] Erkun Yang, Cheng Deng, Tongliang Liu, Wei Liu, and Dacheng Tao. Semantic structure-based unsupervised deep hashing. In *International Joint Conference on Artificial Intelligence*, 2018.