A Lower Bound of Hash Codes' Performance

Anonymous Author(s)
Affiliation
Address
email

Abstract

As a crucial approach for compact representation learning, hashing has achieved great success in effectiveness and efficiency. Numerous heuristic Hamming space metric learning objectives are designed to obtain high quality hash codes. Nevertheless, a theoretical analysis on criteria of learning good hash codes remains largely unexploited. In this paper, we prove that inter-class distinctiveness and intra-class compactness among hash codes determine the lower bound of hash codes' performance. Promoting these two characteristics could lift the bound and improve hash learning. We propose a surrogate model to fully exploit such objective by estimating posterior of hash codes and further control it, which results in low-bias optimization. Extensive experiments reveal effectiveness of the proposed method. By testing on a series of hashing methods, we obtain performance improvements among all of them, with an up to 26.5% increase in mean Average Precision and an up to 20.5% increase in accuracy.

1 Introduction

Explosive increase of multimedia digital contents requires people to develop large-scale processing techniques for effective and efficient multimedia understanding [36, 49, 52, 11, 32, 29]. Meanwhile, growing focus on carbon neutrality initiatives urge researchers to handle above issues with low power and memory consumption. Fortunately, hashing, as a key role in compact representation learning, has been demonstrating its efficiency and effectiveness over past few decades towards above demands [18, 22, 43, 15, 27, 25, 21]. By digesting multimedia contents into short binary codes, hashing is able to significantly reduce storage, boost speed, and preserve key information.

Recent years, compared to hand-crafted hashing functions, learning to hash shows its advantages to alleviate information loss by performing data-dependent Hamming space projection with a non-linear model [33, 48, 46, 47, 27, 25, 21, 24]. The purpose of optimizing hash models is to generate semantically meaningful hash codes for downstream tasks. A practical convention is to increase similarities among codes of same category, otherwise the opposite. Such convention is validated to be effective in many applications [8, 42, 26, 41, 50, 34].

Practices in current advances still leave two questions which remain largely unexploited: How could we determine a hash model could produce *good* codes, and by what means to achieve this goal? Taking comprehensive study on above questions is essential, since it not only guides us to develop high performance algorithms in typical hash-based applications, but also provides potential possibilities to inspire future works in hash learning.

Although some works [48, 1] try to answer above questions by building connections between hash codes and common descriptors with domain knowledge, we still seeks for theoretical guarantees for hash codes' performance. Similar situations occur in related works [47, 46, 31], where objectives to optimize hashing models are designed by intuitions or heuristics.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

By formulating correlations among hash codes as inter-class distinctiveness and intra-class compactness, we try to address above issues and provide a lower bound of hash codes' performance. Based on this, we could further derive from them as objectives to lift lower bound. Moreover, to effectively train model towards such targets, estimating and further controlling posterior of hash codes is necessary, since bit-level dependence exists in posterior distribution. If we neglect this, bias will be introduced during hashing model optimization and therefore hinders performance. Summarizing both of them, our contribution is presented below:

1) To the best of our knowledge, we are the first to formulate lower bound of hash codes' performance, which is directly proportional to inter-class distinctiveness and intra-class compactness among codes. Base on this, an objective is further proposed to lift this lower bound and thus enhance performance.

2) To perform low-bias optimization towards above objective, a novel posterior estimation over hash codes is proposed. As a non-linear function, it could be utilized for optimizing hashing models via gradient descent. Such plug-and-play component is able to integrate into current hashing models to boost performance.

3) Extensive experiments on three benchmark datasets reveal effectiveness of proposed method in two typical tasks. Specifically, we obtain an up to 26.5% increase on mean Average Precision for fast retrieval. We also observe an up to 20.5% increase on accuracy for hash-based recognition.

Rest of paper is organized as follows: We first give a short review on current hashing methods and preliminaries (Secs. 2 and 3). Then, lower bound on hash codes' performance by inter-class distinctiveness and intra-class compactness is proposed (Sec. 4). To fully exploit such guidance, posterior estimation over multivariate Bernoulli distribution is designed for code control (Sec. 5). Experiments (Sec. 6) are then conducted for evaluation.

2 Related Works

This paper will focus on data-dependent learning-to-hash methods. Recent advances on building these models are validated to be effective on hash-based recognition and fast retrieval [18, 1, 26, 41, 17]. The typical approach utilizes a projection function to convert raw inputs into compact binary codes and conduct classification or approximate nearest neighbor search on codes for downstream tasks [8, 42, 50]. As aforementioned in introduction, to obtain such projector, current practices suggest us to formulate it as a Hamming space metric learning problem under binary constraints. Pointwise [51, 21, 30], pairwise [26, 55, 4, 25, 5], triplet [28, 10], listwise [54, 45] and prototype-based [6, 53, 17, 2] objectives are all studied for model training, where the core idea is to adjust similarities among samples from same / different categories.

Besides above objectives, we should also notice that hashing model optimization is NP-hard [1, 48, 30, 41]. This is caused by the discrete, binarized hash codes formulation. To tackle this, previous works propose a lot of methods that target on following purposes: **a)** Surrogate functions that approximate hashing by continuous functions, such as $tanh$ [5], Stochastic neuron [9], $affine$ [39] or Straight-Through [3]. By adopting these, the discrete hashing model optimization is transformed into approximated but easier one via gradient descent. **b)** Non-differentiable optimization algorithm design, which focus on solutions that directly handle discrete inputs, including Coordinate descent [38], Discrete gradient estimation [13], *etc*. There are also bag of tricks which help for stable training. For example, minimizing quantization error reduces gap between raw outputs of model and final hash codes [26], and code-balancing alleviates the problem of producing trivial codes [12].

Please refer to literature reviews on above works [47, 46, 31] for comprehensive studies on learning to hash. We could see two issues exist and remain largely unexploited in above works. The first is objective design, which is commonly proposed by intuitive and with few theoretical guarantees. The second is model optimization, where bit-level dependence is important and involved in code control but most of works neglect it. To tackle these, we would firstly give a lower bound of hash codes' performance and utilize it as a guidance for hashing model training. Then, a novel model optimization approach is proposed in a multivariate perspective for low-bias code control.

3 Preliminaries

Learning to hash performs data-dependent compact representation learning. Inputs $\boldsymbol{x} \in \boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^d$ are firstly transformed into h-dim real-valued vector $\boldsymbol{\ell}$ by $\mathcal{F}_{\boldsymbol{\theta}}$, a projection parameterized by $\boldsymbol{\theta}$:

$\mathbb{R}^d \xrightarrow{\mathcal{F}_{\boldsymbol{\theta}}} \mathbb{R}^h$. And $\boldsymbol{\ell}$ is then binarized to h bits binary code $\boldsymbol{b} \in \boldsymbol{\mathcal{B}} \subseteq \mathbb{H}^h$:

$$\boldsymbol{x} \xrightarrow{\mathcal{F}_{\boldsymbol{\theta}}} \boldsymbol{\ell} \xrightarrow{\text{bin}} \boldsymbol{b}, \ \textit{where} \ \text{bin}\left(\cdot\right) = \begin{cases} +1, & (\cdot) \geq 0, \\ -1, & (\cdot) < 0. \end{cases}$$

Typical usage of hash codes. Common hashing-based tasks include fast retrieval [19, 40, 27, 2] and recognition [26, 38, 41, 33]. For fast retrieval, given a query under same distribution of $\boldsymbol{\mathcal{X}}$, we convert it into query hash code $\boldsymbol{q}$ and conduct fast approximate nearest neighbor search in $\boldsymbol{\mathcal{B}}$ to produce rank list $\mathcal{R}$. In $\mathcal{R}$, samples are organized from nearest to furthest to query. Correspondingly among all results in $\mathcal{R}$, true positives $\boldsymbol{tp}^i \in \boldsymbol{TP}$ indicate samples that match with query while others are false positives $\boldsymbol{fp}^i \in \boldsymbol{FP}$ (i indicates rank). As a criterion, Average Precision (AP) is commonly adopted to determine how well the retrieved results are. It encourages true positives to have higher ranks than false positives: $AP = \frac{1}{|\boldsymbol{TP}|} \sum_{\forall \boldsymbol{tp}^{i,m} \in \boldsymbol{TP}} P@i$, where $P@i$ is the rank-i precision. As for hash-based recognition, since current works formulate it as a variant of retrieval, we could still adopt AP as a criterion to indicate performance. Detailed explanation is placed in Supp. Sec. C.

4 Lower Bound by Inter-Class Distinctiveness and Intra-Class Compactness

We start at an arbitrary rank list for studying characteristics of codes in above scenarios. To calculate AP easily, we first introduce *mis-rank* that indicates how many false positives take higher places than true positives. We refer readers to Supp. Secs. A and B for detailed description and proof.

Definition. Mis-rank m indicates how many false positives have higher ranks than a true positive *i.e.*, $\boldsymbol{tp}^{i,m}$ is at rank i, meanwhile $m = \left|\left\{d\left(\boldsymbol{q}, \boldsymbol{fp}\right) < d\left(\boldsymbol{q}, \boldsymbol{tp}^{i,m}\right)\right\}\right|$, where $d\left(\boldsymbol{q}, \cdot\right)$ is distance between $\boldsymbol{q}$ and a sample, and $|\cdot|$ is number of elements in set.

Then, average precision could be derived as:

$$AP = \frac{1}{|\boldsymbol{TP}|} \sum_{\forall \boldsymbol{tp}^{i,m} \in \boldsymbol{TP}} P@i = \frac{1}{|\boldsymbol{TP}|} \sum_{\forall \boldsymbol{tp}^{i,m} \in \boldsymbol{TP}} \frac{i-m}{i},$$

for all true positives in rank list. This is because the rank-i precision $P@i$ for any true positive $\boldsymbol{tp}^{i,m}$ equals to $(i-m)/i$.

From the derivation, we could immediately obtain that AP increases *iff* m decreases. Therefore, now we could focus on how to reduce m to thereby increase average precision. Noticed that value of m is highly related to two distances, $d\left(\boldsymbol{q}, \boldsymbol{fp}\right)$ and $d\left(\boldsymbol{q}, \boldsymbol{tp}\right)$, following proposition is raised:

Proposition.

$$\overline{m} \propto \frac{\max d\left(\boldsymbol{q}, \boldsymbol{tp}\right)}{\min d\left(\boldsymbol{q}, \boldsymbol{fp}\right)} \ \forall \boldsymbol{tp} \in \boldsymbol{TP}, \ \boldsymbol{fp} \in \boldsymbol{FP}$$

where $\overline{\cdot}$ denotes upper bound. Correspondingly,

$$\underline{AP} \propto \frac{\min d\left(\boldsymbol{q}, \boldsymbol{fp}\right)}{\max d\left(\boldsymbol{q}, \boldsymbol{tp}\right)} \ \forall \boldsymbol{tp} \in \boldsymbol{TP}, \ \boldsymbol{fp} \in \boldsymbol{FP}$$

where $\underline{\cdot}$ denotes lower bound.

If inputs $\boldsymbol{x}^j$ have associated labels y^j, above studies are able to be further generalized. Class-wise centers, inter-class distinctiveness and intra-class compactness are introduced to formulate our final lower-bound of hash codes' performance.

Definition. Center $\boldsymbol{c}^c \in \boldsymbol{C}$ is a representative hash code of a specific class c:

$$\boldsymbol{c}^c = \arg\min_{\boldsymbol{b}} \sum_j d\left(\boldsymbol{b}, \boldsymbol{b}^j\right), \ \forall \left\{\boldsymbol{b}^j \mid y^j = c\right\}$$

where $\boldsymbol{b}^j$ is the jth sample in set $\boldsymbol{\mathcal{B}}$ and y^j is the label of $\boldsymbol{b}^j$.

Definition. *Inter-class distinctiveness* is $\min \mathcal{D}_{inter}$ where $\mathcal{D}_{inter}$ is a set measures distances between all centers over $\boldsymbol{C}$: $\left\{d\left(\boldsymbol{c}^j, \boldsymbol{c}^k\right) \ \mid \ \forall \boldsymbol{c}^j, \boldsymbol{c}^k \in \boldsymbol{C}\right\}$. In contrast, *intra-class compactness* is $1/\max \mathcal{D}_{intra}$ where $\mathcal{D}_{intra} = \left\{d\left(\boldsymbol{b}^j, \boldsymbol{c}^c\right) \ \mid \ \forall y^j = c\right\}$ measures distances among samples of same class to their corresponding center.

Combining above propositions and definitions, we could reach:

Proposition.

$$\underline{AP} \propto \frac{\min d\left(\boldsymbol{q}, \boldsymbol{fp}\right)}{\max d\left(\boldsymbol{q}, \boldsymbol{tp}\right)} \geq const \cdot \frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}. \tag{1}$$

This proposition reveals how AP is affected by above two inter- and intra- factors. Actually, it could cover common scenarios as a criterion of hash codes' performance, *e.g.*, fast retrieval and recognition, which will be explained in supplementary materials. Intuitively, lifting this lower bound would enhance performance, formulating the following objective:

$$\begin{aligned} &maximize \min \mathcal{D}_{inter}, && (2)\\ &minimize \max \mathcal{D}_{intra}, && (3)\\ &s.t.\ \boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^d \xrightarrow{\mathcal{F}_{\boldsymbol{\theta}},\ \mathrm{bin}} \boldsymbol{\mathcal{B}} \subseteq \mathbb{H}^h. \end{aligned}$$

Inspired by [53], we give a specific solution to implement Eqs. (2) and (3) under supervised hashing by firstly defining distance maximized class-specific centers and then shrinking hash codes to their corresponding centers. We will validate effectiveness with such solution in experiments. It is worth noting that achieving above goal is an open topic, including unsupervised or other complex scenarios, which we leave for future study. Next, to optimize hashing models with above supervision, we introduce posterior estimation over hash codes and then control it.

5 Posterior Estimation for Code Control

It is essentially hard to minimize distance between a hash code and a target to achieve Eq. (3) due to two reasons. Firstly, posterior distribution of $\boldsymbol{\mathcal{B}}$ given inputs $\boldsymbol{\mathcal{X}}$ is formulated to be under multivariate Bernoulli distribution: $p\left(\boldsymbol{\mathcal{B}} \mid \boldsymbol{\mathcal{X}}\right) \sim \mathrm{MVB}$. If we do not take correlations among different variables into account, optimization on $\boldsymbol{\mathcal{B}}$ is biased. Secondly, dealing with hash codes is a $\{-1, +1\}$ discrete optimization, which is not trivial to handle. Therefore, we try to organize Eq. (3) as a Maximum Likelihood Estimation (MLE) to tackle above issues. Considering definition of Hamming distance between an arbitrary hash code $\boldsymbol{b}$ and its target $\boldsymbol{t}$:

$$d\left(\boldsymbol{b}, \boldsymbol{t}\right) = \sum_{i}^{h} \mathbb{1}\left\{\boldsymbol{b}_i \neq \boldsymbol{t}_i\right\} = \sum_{i}^{h} \mathbb{1}\left\{\boldsymbol{\ell}_i \boldsymbol{t}_i < 0\right\}$$

where $\mathbb{1}$ is the characteristic function and $\left(\cdot\right)_i$ indicates value on the ith dimension. Since Hamming distance measures how many bits are different between two codes, probability of $d\left(\boldsymbol{b}, \boldsymbol{t}\right) = \delta$ can be formulated as:

$$p\left(d\left(\boldsymbol{b}, \boldsymbol{t}\right) = \delta\right) = \sum_{\forall i \in \binom{h}{\delta},\ j \notin \binom{h}{\delta}} \left\{p\left(\boldsymbol{b}_i \neq \boldsymbol{t}_i,\ \boldsymbol{b}_j = \boldsymbol{t}_j\right)\right\}.$$

Therefore, $\boldsymbol{b}$ and $\boldsymbol{t}$ have distance δ *iff* $p\left(d\left(\boldsymbol{b}, \boldsymbol{t}\right) = \delta\right)$ is maximized. However, to precisely calculate it is difficult, since it involves joint probability of $\boldsymbol{b}$. Therefore, we try to estimate all joint probabilities by adopting a surrogate model $\mathcal{P}_{\boldsymbol{\pi}}$ parameterized by $\boldsymbol{\pi}$ to perform estimation:

$$\boldsymbol{\ell} \xrightarrow{\mathcal{P}_{\boldsymbol{\pi}}} \boldsymbol{o},\ where\ \boldsymbol{o} \in \boldsymbol{O} \subseteq \mathbb{R}^{2^h} \tag{4}$$

where $\boldsymbol{o}$ is probabilities of a Categorical distribution $p\left(\boldsymbol{O} \mid \boldsymbol{\mathcal{X}}\right) \mathrel{\widehat{=}} p\left(\boldsymbol{\mathcal{B}} \mid \boldsymbol{\mathcal{X}}\right)$, which contains 2^h entries. Each entry of $\boldsymbol{o}$ estimates one of a specific joint probability by feeding $\boldsymbol{\ell}$ into $\mathcal{P}_{\boldsymbol{\pi}}$:

$$\boldsymbol{o}_i \mathrel{\widehat{=}} p\left(\left(\boldsymbol{b}_k > 0\right)^{\mathbb{1}\{\boldsymbol{\iota}_k = 1\}}, \left(\boldsymbol{b}_k < 0\right)^{\mathbb{1}\{\boldsymbol{\iota}_k = 0\}}\right),\ \begin{smallmatrix} 1 \leq k \leq h, \\ 0 \leq i \leq 2^h - 1 \end{smallmatrix}$$

where $\boldsymbol{\iota}$ is the h bits binary representation of i. For example, $\boldsymbol{o}_3 = p\left(\boldsymbol{b} = (\text{-1-1+1+1})\right)$ when $h = 4$. To train $\mathcal{P}_{\boldsymbol{\pi}}$, we perform Maximum Likelihood Estimation (MLE) over $p\left(\boldsymbol{O} \mid \boldsymbol{\mathcal{X}}\right)$:

$$\begin{aligned} L_{\boldsymbol{\pi}}\left(\mathcal{P}_{\boldsymbol{\pi}}; \boldsymbol{o}\right) &= -\log \boldsymbol{o}_i, \\ where\ i &= \sum_{k=1}^{h} \mathbb{1}\left\{\boldsymbol{b}_k > 0\right\} \cdot 2^{k-1}. \end{aligned} \tag{5}$$

If $\mathcal{P}_{\boldsymbol{\pi}}$ is ready for estimation, we could directly adopt it as a nice non-linear function to maximize $p\left(d\left(\boldsymbol{b}, \boldsymbol{t}\right) = k\right)$. Note that to realize Eq. (3), we want $\boldsymbol{b}$ and its corresponding center $\boldsymbol{c}$ are closest *i.e.* maximize $p\left(d\left(\boldsymbol{b}, \boldsymbol{c}\right) = 0\right)$. So, we could reformulate this maximization as another MLE:

$$\begin{aligned} L_{\boldsymbol{\theta}}\left(\mathcal{F}_{\boldsymbol{\theta}}; \boldsymbol{b}\right) &= -\log p\left(d\left(\boldsymbol{b}, \boldsymbol{c}\right) = 0\right), \\ \textit{where } p\left(d\left(\boldsymbol{b}, \boldsymbol{c}\right) = 0\right) &= p\left(\boldsymbol{b}_i = \boldsymbol{c}_i, 1 \leq i \leq h\right), \end{aligned} \tag{6}$$

which could be optimized by calculating surrogate gradients of Eq. (6) through $\mathcal{P}_{\boldsymbol{\pi}}$:

$$\frac{\partial \hat{L}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} = \frac{\partial \left\{-\log \boldsymbol{o}_i;\ i = \sum_{k=1}^{h} \mathbb{1}\left\{\boldsymbol{c}_k > 0\right\} \cdot 2^{k-1}\right\}}{\partial \boldsymbol{\pi}} \frac{\partial \boldsymbol{\pi}}{\partial \boldsymbol{\ell}} \frac{\partial \boldsymbol{\ell}}{\partial \boldsymbol{\theta}} \mathrel{\widehat{=}} \frac{\partial L_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}. \tag{7}$$

Unfortunately, such joint probability estimation on MVB requires $\mathcal{O}\left(2^h\right)$ complexity. This is not flexible when h is large. By adopting the idea of block code [44], above estimation is able to perform on long hash bits by separating hash codes into a series of blocks. Specifically, any h bits hash codes can be split into u blocks while each block consumes h/u bits. Correspondingly, u independent surrogate networks $\mathcal{P}_{\boldsymbol{\pi}_i}, 1 \leq i \leq u$ are adopted to perform above estimation and back-propagation simultaneously. With such decomposition, the whole problem is transformed into u sub-problems with a reduced complexity $\mathcal{O}\left(2^h\right) \rightarrow \mathcal{O}\left(u \cdot 2^{h/u}\right)$.

Summarizing all of things, overall optimization via gradient descent is placed in Alg. 1. We first perform Eq. (2) step by using pre-defined centers and then perform Eq. (3) step by back-propagation via Eq. (7). Two models $\mathcal{F}_{\boldsymbol{\theta}}, \mathcal{P}_{\boldsymbol{\pi}}$ are optimized with learning rate η_1, η_2 respectively.

Algorithm 1 One of implementations under supervised circumstance.

procedure TRAIN($\mathcal{F}_{\boldsymbol{\theta}},\ \mathcal{P}_{\boldsymbol{\pi}}$) ▷ Training procedure of two models $\mathcal{F}_{\boldsymbol{\theta}},\ \mathcal{P}_{\boldsymbol{\pi}}$.
  Generate class-specific centers $\boldsymbol{c} \in \boldsymbol{C}, |\boldsymbol{C}| =$ Class-num; ▷ (Eq. (2)).
  repeat ▷ Main training loop.
    Sample $\boldsymbol{x}$ from $\boldsymbol{\mathcal{X}}$ with label y;
    $\boldsymbol{\ell} = \mathcal{F}_{\boldsymbol{\theta}}\left(\boldsymbol{x}\right)$;
    $\boldsymbol{o} = \mathcal{P}_{\boldsymbol{\pi}}\left(\boldsymbol{\ell}\right)$;
    $\boldsymbol{\pi} \leftarrow \boldsymbol{\pi} - \eta_1 \frac{\partial L_{\boldsymbol{\pi}}}{\partial \boldsymbol{\pi}}$; ▷ (Eq. (5)).
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_2 \frac{\partial \hat{L}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}$ with corresponding center $\boldsymbol{c}$; ▷ Eq. (3), Eq. (7).
  until Total epoch exceeds;
  return $\mathcal{F}_{\boldsymbol{\theta}}$; ▷ Optimized hashing model $\mathcal{F}_{\boldsymbol{\theta}}$
end procedure

6 Experiments

We conduct extensive experiments on three benchmark datasets to confirm effectiveness of our proposed method. To make fair comparisons, we first provide experiments setup.

6.1 Setup

Our experiments focus on performance evaluation on two typical hash-based tasks, fast retrieval and recognition. The pipeline is to firstly train models on training split, then hash all samples in base split. Then, hashed queries in query split get rank lists from base split for retrieval, while queries' predictions from classifier are used for recognition.

6.1.1 Datasets

Our experiments are conducted on three datasets, varying from scale, variety and perplexity to validate methods' performance in different scenarios. Both single-label and multi-label datasets are included. We follow previous works to generate three splits, which is detailed below:

CIFAR-10 [20] is a single-label 10-class dataset. The whole dataset contains $6,000$ images for each class. Following [5], we form the dataset by $500 \mid 5,400 \mid 100$ for each class randomly as train, base, query splits, respectively.

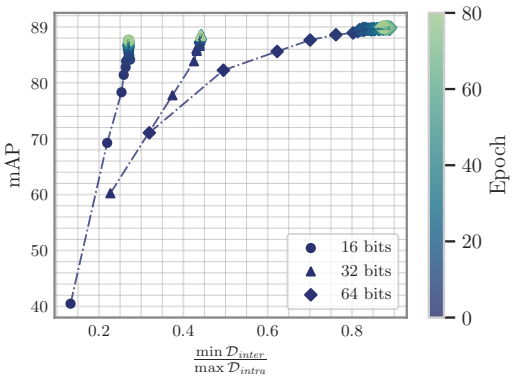


Figure 1: mAP *w.r.t.* $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ per epoch on ImageNet 64 bits during training. These two values have a positive relationship.

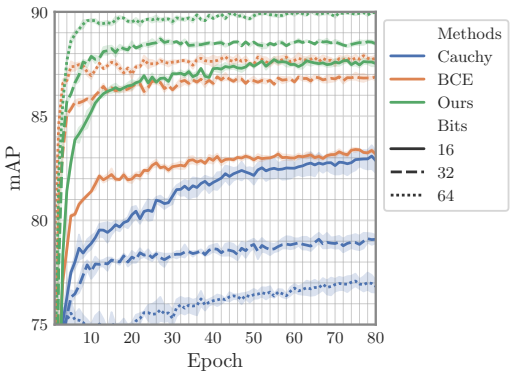


Figure 2: Convergence curve of three variants on ImageNet 64 bits. Our $\mathcal{P}_{\boldsymbol{\pi}}$ helps for achieving higher performance than other two.

NUS-WIDE [7] consists of 81 labels and images may have one or more labels. We follow previous works [5] to pick the most frequent 21 labels and their associated images $(195,834)$ for experiments. Specifically, $193,734$ images are randomly picked to form base split while remaining $2,100$ images are adopted for queries. $10,500$ images are randomly sampled from base split for training models.

Table 1: Total training time with different variants, where we substitute our surrogate model with BCE loss from [53] or Cauchy loss from [4].

Method	Training time per epoch (s)		
	16 bits	32 bits	64 bits
BCE	38.13	38.20	38.52
Cauchy	38.21	38.30	38.42
Ours	38.95	39.78	40.84

ImageNet [11] is a large-scale dataset consists of $1,000$ classes. To conduct experiments, we follow [5] to pick a subset of 100 classes where all images of these classes in training set / validation set are as base split / query split respectively $(128,503 \mid 4,983$ images). We then randomly sample 100 images per class in base split for training.

6.1.2 Implementation Details

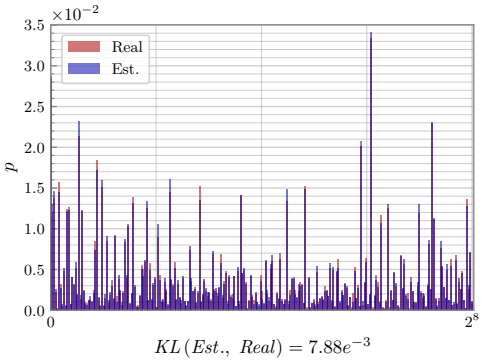
Our method is able to integrate into common hashing models. Due to limitation of computation resources, we choose a series of representative deep hashing methods for comparison, including **HashNet** [5], **DBDH** [55], **DSDH** [26], **DCH** [4], **GreedHash** [41] and **CSQ** [53] and denote methods with our integration as $(\cdot)_D$. All methods adopted for experiments are from a public benchmark implemented with PyTorch [35].[1] For fair comparisons, we conduct experiments with same backbone (ResNet-50 [16]) and hyper-parameters for all tested methods. We adopt Adam [37] optimizer with default configuration and learning rate of our method $\eta_1 = \eta_2 = 1e^{-3}$ for training. For multi-label datasets, we simply modify Alg. 1 Line 8 with sum of multiple losses. Block number u of $\mathcal{P}$ is set to ${}^{bits}/_{8}$. For example, if hash code length is 64, there will be 8 sub models $\mathcal{P}_{\boldsymbol{\pi}_1} \sim \mathcal{P}_{\boldsymbol{\pi}_8}$ trained in parallel. To mitigate randomness, we report average performance for 5 runs on all experiments. Evaluation metric adopted for retrieval is mean Average Precision (mAP@K) where $K = 54,000 \mid 5,000 \mid 1,000$ on CIFAR-10, NUS-WIDE, ImageNet, respectively. For recognition performance, if methods have auxiliary classification branch, we directly use predictions from it. Otherwise, we adopt a kNN classifier built on base split and vote with 100 nearest neighbors.

6.2 Ablation Study

In ablation study, we try to reveal correctness of our proposed objectives Eqs. (2) and (3), and effectiveness of our proposed posterior estimation model $\mathcal{P}_{\boldsymbol{\pi}}$. We answer following questions by conducting experiments on ImageNet to demystify above concerns. We focus on retrieval performance in ablation study, while we observe similar results on recognition.

[1] https://github.com/swuxyj/DeepHash-pytorch

Is hash model's performance lower-bounded by Eq. (1)? Validating correctness of Eq. (1) is important, but it is essentially hard to conduct experiments to confirm it. Nevertheless, we still reveal correlation between model performance and inter-class distinctiveness / intra-class compactness by tracking mAP *w.r.t.* $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ during training. To calculate $\mathcal{D}_{inter}$ and $\mathcal{D}_{intra}$, we first hash all samples from base split and calculate centers of them over all classes. The 99.9 percentile of $\min \mathcal{D}_{inter}$ and $\max \mathcal{D}_{intra}$ is picked to avoid outliers. We conduct tracking on CSQ_D with $16, 32, 64$ bits per epoch to draw Fig. 1. As figure shows, lines goes from lower left to upper right with small fluctuations, which indicates linear relationship between mAP and $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ on all bit-lengths. This may partially confirm our theoretical analysis in Sec. 4.



Method	KL_{real}
Naïve	0.538
Ours	0.008

Figure 3: Multivariate Bernoulli estimation by our proposed $\mathcal{P}_{\boldsymbol{\pi}}$, and KL-divergence comparisons with naïve independent estimation.

Convergence speed and efficiency of $\mathcal{P}_{\boldsymbol{\pi}}$. To verify convergence speed and efficiency of our posterior estimation model $\mathcal{P}_{\boldsymbol{\pi}}$, we test it by substituting with two variants: BCE from [53] and Cauchy loss from [4]. mAP is evaluated per epoch for three variants with three bit-lengths $16, 32, 64$. The convergence curve is plotted in Fig. 2. From the figure, we could see convergence speed of ours and BCE are similar, while Cauchy is slightly slow. Meanwhile, our method has continuously higher performance than other two after ~ 10 epochs for all bit-lengths, which is potentially due to our method could perform low-bias optimization. Note that when training with Cauchy loss, performance is instead lower when codes are longer. This may because inner-product based Cauchy loss could not handle long bits. We also measure averaged training time per epoch of all variants on a single NVIDIA RTX 3090, which is placed in Tab. 1. In this table, our model does not consume significant longer time than others to train. Above observations reveal efficiency of our proposed method. It could be a drop-in replacement without introducing significant overhead. Moreover, convergence speed and efficiency will not drop when bit-length becomes long, which reveals flexibility of the blocked design.

Could $\mathcal{P}_{\boldsymbol{\pi}}$ estimate multivariate Bernoulli distribution? We design an extra toy experiments to validate the ability of estimating multivariate Bernoulli distribution by our surrogate model $\mathcal{P}_{\boldsymbol{\pi}}$. Specifically, 8 bits MVB is generated with randomly 256 joint probabilities. We then take $10,000$ samples from it as inputs to train $\mathcal{P}_{\boldsymbol{\pi}}$. The model will further estimate distribution by feeding another 100 samples and taking mean of all $\boldsymbol{o}$s (Eq. (4)) as result. The estimated distribution is evaluated by Kullback–Leibler divergence [23] (KL) with real distribution, as well as bar plot, demonstrated in Fig. 3. As figure shows, our predicted joint probabilities almost cover real probabilities and KL is low. As a comparison, if we estimate real distribution with product of edge probabilities directly (w/o correlations between variables), KL will be significantly increased (row 1 in table). Such toy experiment reveals our method is better to estimate MVBs than the naïve one.

6.3 Performance Enhancements when Integrating into State-of-the-Art

To evaluate performance gain when integrating our proposed method into hashing models, we conduct experiments on three datasets. Specifically, we first report original performance of tested models and then re-run with our method incorporated to make comparison. Both $16, 32, 64$ bits results are reported to show performance from short codes to long codes.

Retrieval Performance. Retrieval performance comparisons under mAP is shown in Tab. 2. With our method integrated, all of tested hashing methods have performance enhancement. We observe an up to 26.5% increase and 5.02% in average, which is a significant improvement. Specifically, performance increase on HashNet, DBDH and DSDH is higher than other methods (first 3 rows), especially on ImageNet dataset. A potential reason is all of these three methods use pairwise metric learning objectives and inner-product to approximate Hamming distance, which may not handle optimization well when dataset size and class number is large. Meanwhile, our method works on

Table 2: mAP comparisons on three benchmark datasets for $16, 32, 64$ bits codes. $\uparrow(\cdot)$ indicates performance enhancement with our method integrated.

Method	CIFAR-10			NUS-WIDE			ImageNet		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
HashNet	63.0	81.5	84.7	77.8	83.3	85.0	50.6	63.1	68.4
HashNet$_D$	**79.1**$_{\uparrow 16.1}$	**82.3**$_{\uparrow 0.8}$	**85.3**$_{\uparrow 0.6}$	**78.6**$_{\uparrow 0.8}$	**84.0**$_{\uparrow 0.7}$	**86.2**$_{\uparrow 1.2}$	**72.6**$_{\uparrow 22.0}$	**84.3**$_{\uparrow 21.2}$	**87.6**$_{\uparrow 19.2}$
DBDH	83.1	85.0	85.6	82.8	84.6	85.7	55.7	63.8	76.9
DBDH$_D$	**84.6**$_{\uparrow 1.5}$	**86.0**$_{\uparrow 1.0}$	**86.5**$_{\uparrow 0.9}$	**83.8**$_{\uparrow 1.0}$	**85.6**$_{\uparrow 1.0}$	**86.5**$_{\uparrow 0.8}$	**82.2**$_{\uparrow 26.5}$	**85.8**$_{\uparrow 22.0}$	**86.1**$_{\uparrow 9.2}$
DSDH	75.6	83.1	84.5	83.3	84.5	85.6	57.2	72.1	75.3
DSDH$_D$	**84.3**$_{\uparrow 8.7}$	**84.6**$_{\uparrow 1.5}$	**87.3**$_{\uparrow 2.8}$	**83.6**$_{\uparrow 0.3}$	**85.3**$_{\uparrow 0.8}$	**86.4**$_{\uparrow 0.8}$	**81.7**$_{\uparrow 24.5}$	**87.3**$_{\uparrow 15.2}$	**87.9**$_{\uparrow 12.6}$
DCH	83.4	84.4	85.3	80.7	81.7	80.9	85.5	86.2	86.4
DCH$_D$	**83.6**$_{\uparrow 0.2}$	**84.6**$_{\uparrow 0.2}$	**87.1**$_{\uparrow 1.8}$	**82.9**$_{\uparrow 2.2}$	**84.2**$_{\uparrow 2.5}$	**84.9**$_{\uparrow 4.0}$	**86.1**$_{\uparrow 0.6}$	**87.5**$_{\uparrow 1.3}$	**88.1**$_{\uparrow 1.7}$
GreedHash	83.3	84.3	86.9	78.6	80.3	82.0	83.1	85.9	86.4
GreedHash$_D$	**85.2**$_{\uparrow 1.9}$	**85.5**$_{\uparrow 1.2}$	**87.6**$_{\uparrow 0.7}$	**79.4**$_{\uparrow 0.8}$	**83.1**$_{\uparrow 2.8}$	**85.7**$_{\uparrow 3.7}$	**83.8**$_{\uparrow 0.7}$	**86.6**$_{\uparrow 0.7}$	**86.8**$_{\uparrow 0.4}$
CSQ	83.2	83.4	84.7	82.0	83.5	84.6	83.4	86.9	87.9
CSQ$_D$	**88.7**$_{\uparrow 5.5}$	**89.2**$_{\uparrow 5.8}$	**90.3**$_{\uparrow 5.6}$	**83.3**$_{\uparrow 1.3}$	**85.3**$_{\uparrow 1.8}$	**85.8**$_{\uparrow 1.2}$	**88.5**$_{\uparrow 5.1}$	**89.5**$_{\uparrow 2.6}$	**90.2**$_{\uparrow 2.3}$

multi-label datasets where all methods also obtain performance gain, indicating multi-label data may also benefit from our objective. Furthermore, we also observe 3.46% average mAP increase on CSQ. It adopts similar objective but different optimization approach with ours. This increase shows effectiveness of our proposed posterior estimation approach, which could perform better optimization than theirs.

Recognition Performance. Similarly, we report recognition performance of these methods by measuring classification accuracy on ImageNet with $16, 32, 64$ bits codes. Results are reported in Tab. 3. Generally, we observe similar performance gain. When methods have our integration, their accuracy is significantly increased, with an up to 20.5% and 5.3% in average. Specifically, some of them utilize an auxiliary classification branch to produce predictions, *i.e.*, GreedyHash and DSDH. While for others, we adopt a kNN classifier to classify samples by voting with base split. We obtain increase for both approaches. This indicates our approach also helps for hash-based recognition in different scenarios, by leveraging inter-class distinctiveness and intra-class compactness.

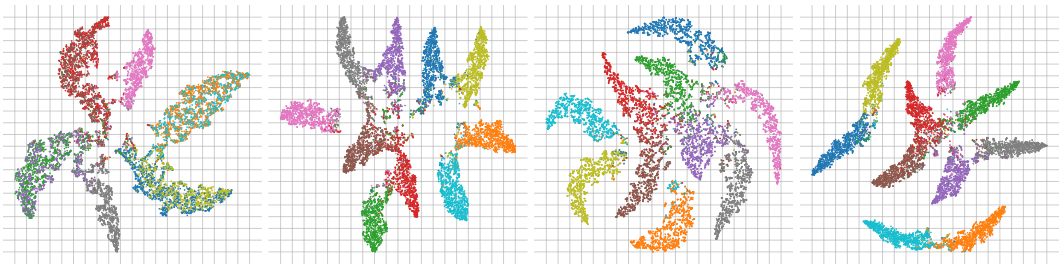
Table 3: Recognition performance on ImageNet.

Method	ImageNet		
	16 bits	32 bits	64 bits
HashNet	69.1	74.1	82.4
HashNet$_D$	**83.2**$_{\uparrow 14.1}$	**88.1**$_{\uparrow 14.0}$	**89.5**$_{\uparrow 7.1}$
DBDH	80.9	82.9	88.2
DBDH$_D$	**86.8**$_{\uparrow 5.9}$	**89.1**$_{\uparrow 6.2}$	**88.4**$_{\uparrow 0.2}$
DSDH	40.6	49.9	54.1
DSDH$_D$	**61.1**$_{\uparrow 20.5}$	**62.1**$_{\uparrow 12.2}$	**63.2**$_{\uparrow 9.1}$
DCH	88.5	89.2	88.1
DCH$_D$	**88.9**$_{\uparrow 0.4}$	**89.5**$_{\uparrow 0.3}$	**88.7**$_{\uparrow 0.6}$
GreedHash	79.1	86.8	87.0
GreedHash$_D$	**80.2**$_{\uparrow 1.1}$	**87.1**$_{\uparrow 0.3}$	**88.4**$_{\uparrow 1.4}$
CSQ	87.7	89.0	89.4
CSQ$_D$	**87.9**$_{\uparrow 0.2}$	**89.1**$_{\uparrow 0.1}$	**91.0**$_{\uparrow 1.6}$

From above comparisons, we confirm effectiveness of our proposed method. Overall, our method is validated to be flexible and promising to deploy in above scenarios with various bits.

6.4 Qualitative Comparisons

t-SNE visualization. To further analyze how our objective affects hash codes by adjusting $\mathcal{D}_{inter}$ and $\mathcal{D}_{intra}$, we conduct codes visualization by t-SNE [14] on HashNet and DBDH. Specifically, we randomly sample $5,000$ 64 bits codes from CIFAR-10 base split. t-SNE is then performed on these codes as shown in Fig. 4, where points' color indicates class. From these figures, we give our humble explanation. Firstly, please look at Fig. 4(a), codes extracted from original HashNet are mixed among different classes. While in Fig. 4(c), clusters are loose. These representations result in low $\min \mathcal{D}_{inter}$ and high $\max \mathcal{D}_{intra}$. When they are trained along with our method (Figs. 4(b) and 4(d)), we get more separable and distinct clusters compared to original ones. Meanwhile, codes



(a) HashNet 64 bits, $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}} = 0.8562$ (b) HashNet$_D$ 64 bits, $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}} = 0.9334$ (c) DBDH 64 bits, $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}} = 0.9447$ (d) DBDH$_D$ 64 bits, $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}} = 0.9761$

Figure 4: t-SNE visualization for HashNet and DBDH $w/$ and w/o our integration, where the $(\cdot)_D$ variants have more compact structures and larger margin among different classes than original ones.

in a cluster are more compact than original methods. Quantitative results of $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ placed under figures also reveal this phenomenon.

mAP *w.r.t.* $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ visualization. To further confirm relationship between hashing performance and $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$, as in Sec. 6.2, we plot mAP *w.r.t.* $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ for all tested methods on 64 bits ImageNet base split (Fig. 5). As figure shows, most of methods obtain higher mAP with larger $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ and move from lower-left region to upper-right region (solid dots to translucent dots indicate original methods to integrated methods). Meanwhile, according to the positions of these dots, mAP and $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ are supposed to be under linear relationship for different methods, *i.e.* higher $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ leads to higher mAP. Therefore, we give regression line with 95% confidence interval on plot to confirm our observations. Extended from this, our proposed lower bound could also be a criterion of hash codes' performance.

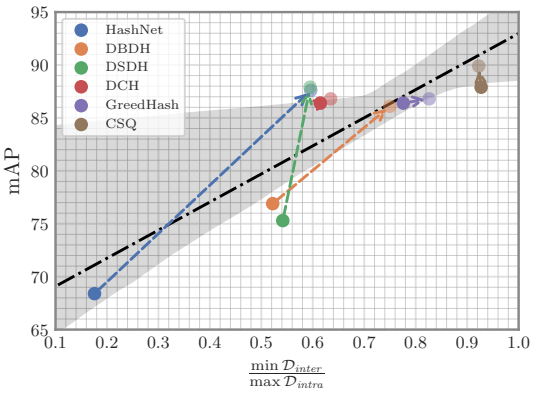


Figure 5: mAP *w.r.t.* $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ for different methods. By integrating our auxiliary objective, we could observe significant performance and $\frac{\min \mathcal{D}_{inter}}{\max \mathcal{D}_{intra}}$ increase in most cases (from solid dots to translucent dots). Regression line is also plotted with 95% confidence interval to reveal linear relationship between two metrics.

7 Conclusion

In this paper, we conduct comprehensive study on characteristics of hash codes. As a result, we prove hash codes' performance is lower-bounded by inter-class distinctiveness and intra-class compactness. Formulating such lower bound as an objective, we could further lift it in hash model training. Meanwhile, our proposed surrogate model for posterior estimation over hash codes' fully exploits above guidance to perform low-bias model optimization and finally produce good codes. Extensive experiments conducted on three benchmark datasets confirm effectiveness of our proposed method. We are able to boost current hashing models' performance with a flexible integration.

Limitation and Broader Impacts

Hashing and related compact representation learning are able to significantly reduce computational requirements while improving memory efficiency when deploying to real scenarios. However, a main challenge for these techniques is information loss which results in performance drop. To tackle this, our work studies on hash codes' performance gives a lower-bound, which is validated in two specific tasks. However, generalizing such lower-bounds to various scenarios, *e.g.* semi-supervised, unsupervised hash learning, are leaved for future study. Meanwhile, our posterior estimation approach is not verified for extremely long bits. Nevertheless, our work may still provide potentials of inspiring researchers to improve not only hash learning, but also broader areas that adopt metric learning. Our study on posterior estimation may also help for precise discrete optimizations.

References

[1] A. Andoni, P. Indyk, T. Laarhoven, I. P. Razenshteyn, and L. Schmidt. Practical and optimal LSH for angular distance. In *NeurIPS*, pages 1225–1233, 2015. 1, 2

[2] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *CVPR*, pages 5297–5307, 2016. 2, 3

[3] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 2

[4] Y. Cao, M. Long, B. Liu, and J. Wang. Deep cauchy hashing for hamming space retrieval. In *CVPR*, pages 1229–1237, 2018. 2, 6, 7

[5] Z. Cao, M. Long, J. Wang, and P. S. Yu. Hashnet: Deep learning to hash by continuation. In *ICCV*, pages 5609–5618, 2017. 2, 5, 6

[6] Y. Chen, Z. Lai, Y. Ding, K. Lin, and W. K. Wong. Deep supervised hashing with anchor graph. In *ICCV*, pages 9795–9803, 2019. 2

[7] T. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. NUS-WIDE: a real-world web image database from national university of singapore. In *CIVR*, 2009. 6

[8] S. Dahlgaard, M. B. T. Knudsen, and M. Thorup. Practical hash functions for similarity estimation and dimensionality reduction. In *NeurIPS*, pages 6615–6625, 2017. 1, 2

[9] B. Dai, R. Guo, S. Kumar, N. He, and L. Song. Stochastic generative hashing. In *ICML*, pages 913–922, 2017. 2

[10] C. Deng, Z. Chen, X. Liu, X. Gao, and D. Tao. Triplet-based deep hashing network for cross-modal retrieval. *IEEE Trans. Image Process.*, 27(8):3893–3903, 2018. 2

[11] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 1, 6

[12] K. D. Doan, P. Yang, and P. Li. One loss for quantization: Deep hashing with discrete wasserstein distributional matching. In *CVPR*, pages 9447–9457, 2022. 2

[13] Z. Dong, A. Mnih, and G. Tucker. Disarm: An antithetic gradient estimator for binary latent variables. In *NeurIPS*, 2020. 2

[14] C. Flexa, W. C. Gomes, I. Moreira, R. Alves, and C. Sales. Polygonal coordinate system: Visualizing high-dimensional data using geometric dr, and a deterministic version of t-sne. *Expert Syst. Appl.*, 175: 114741, 2021. 8

[15] R. Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984. 1

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 6

[17] J. T. Hoe, K. W. Ng, T. Zhang, C. S. Chan, Y. Song, and T. Xiang. One loss for all: Deep hashing with a single cosine similarity based learning objective. In *NeurIPS*, pages 24286–24298, 2021. 2

[18] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2010. 1, 2

[19] W. Kong and W. Li. Isotropic hashing. In *NeurIPS*, pages 1655–1663, 2012. 3

[20] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5

[21] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NeurIPS*, pages 1042–1050, 2009. 1, 2

[22] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(12):2143–2157, 2009. 1

[23] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22 (1):79 – 86, 1951. 7

[24] H. Lai, Y. Pan, Y. Liu, and S. Yan. Simultaneous feature learning and hash coding with deep neural networks. In *CVPR*, pages 3270–3278, 2015. 1

[25] C. Li, C. Deng, N. Li, W. Liu, X. Gao, and D. Tao. Self-supervised adversarial hashing networks for cross-modal retrieval. In *CVPR*, pages 4242–4251, 2018. 1, 2

[26] Q. Li, Z. Sun, R. He, and T. Tan. Deep supervised discrete hashing. In *NeurIPS*, pages 2482–2491, 2017. 1, 2, 3, 6

[27] Y. Li, W. Liu, and J. Huang. Sub-selective quantization for learning binary codes in large-scale image search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(6):1526–1532, 2018. 1, 3

[28] Y. Liang, Y. Pan, H. Lai, W. Liu, and J. Yin. Deep listwise triplet hashing for fine-grained image retrieval. *IEEE Trans. Image Process.*, 31:949–961, 2022. 2

[29] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. In *ECCV*, volume 8693, pages 740–755, 2014. 1

[30] W. Liu, C. Mu, S. Kumar, and S. Chang. Discrete graph hashing. In *NeurIPS*, pages 3419–3427, 2014. 2

[31] X. Luo, H. Wang, D. Wu, C. Chen, M. Deng, J. Huang, and X.-S. Hua. A survey on deep hashing methods. *ACM Trans. Knowl. Discov. Data*, 2022. 1, 2

[32] A. Miech, D. Zhukov, J.-B. Alayrac, M. Tapaswi, I. Laptev, and J. Sivic. HowTo100M: Learning a Text-Video Embedding by Watching Hundred Million Narrated Video Clips. In *ICCV*, 2019. 1

[33] M. Norouzi, D. J. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *NeurIPS*, pages 1070–1078, 2012. 1, 3

[34] M. Norouzi, A. Punjani, and D. J. Fleet. Fast exact search in hamming space with multi-index hashing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(6):1107–1119, 2014. 1

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, volume 32, pages 8026–8037, 2019. 6

[36] F. Radenovic, A. Iscen, G. Tolias, Y. Avrithis, and O. Chum. Revisiting oxford and paris: Large-scale image retrieval benchmarking. In *CVPR*, pages 5706–5715, 2018. 1

[37] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. In *ICLR*, 2018. 6

[38] F. Shen, C. Shen, W. Liu, and H. T. Shen. Supervised discrete hashing. In *CVPR*, pages 37–45, 2015. 2, 3

[39] J. Song, T. He, L. Gao, X. Xu, A. Hanjalic, and H. T. Shen. Unified binary generative adversarial network for image retrieval and compression. *Int. J. Comput. Vis.*, 128(8):2243–2264, 2020. 2

[40] J. Song, R. Lang, X. Zhu, X. Xu, L. Gao, and H. T. Shen. 3d self-attention for unsupervised video quantization. In *ACM SIGIR*, pages 1061–1070, 2020. 3

[41] S. Su, C. Zhang, K. Han, and Y. Tian. Greedy hash: Towards fast optimization for accurate hash coding in CNN. In *NeurIPS*, pages 806–815, 2018. 1, 2, 3, 6

[42] D. Svenstrup, J. M. Hansen, and O. Winther. Hash embeddings for efficient word representations. In *NeurIPS*, pages 4928–4936, 2017. 1, 2

[43] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*, pages 6306–6315, 2017. 1

[44] J. van Lint. *Introduction to Coding Theory*. Graduate Texts in Mathematics. Springer Berlin Heidelberg, 2013. ISBN 9783662079980. 5

[45] J. Wang, W. Liu, A. X. Sun, and Y. Jiang. Learning hash codes with listwise supervision. In *ICCV*, pages 3032–3039, 2013. 2

[46] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014. 1, 2

[47] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen. A survey on learning to hash. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4):769–790, 2018. 1, 2

[48] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NeurIPS*, pages 1753–1760, 2008. 1, 2

[49] T. Weyand, A. Araujo, B. Cao, and J. Sim. Google landmarks dataset v2 - A large-scale benchmark for instance-level recognition and retrieval. In *CVPR*, pages 2572–2581, 2020. 1

[50] W. Wu, B. Li, C. Luo, and W. Nejdl. Hashing-accelerated graph neural networks for link prediction. In *WWW*, pages 2910–2920, 2021. 1, 2

[51] H. Yang, K. Lin, and C. Chen. Supervised learning of semantics-preserving hash via deep convolutional neural networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(2):437–451, 2018. 2

[52] A. Yoshitaka and T. Ichikawa. A survey on content-based retrieval for multimedia databases. *IEEE Trans. Knowl. Data Eng.*, 11(1):81–93, 1999. 1

[53] L. Yuan, T. Wang, X. Zhang, F. E. H. Tay, Z. Jie, W. Liu, and J. Feng. Central similarity quantization for efficient image and video retrieval. In *CVPR*, pages 3080–3089, 2020. 2, 4, 6, 7

[54] F. Zhao, Y. Huang, L. Wang, and T. Tan. Deep semantic ranking based hashing for multi-label image retrieval. In *CVPR*, pages 1556–1564, 2015. 2

[55] X. Zheng, Y. Zhang, and X. Lu. Deep balanced discrete hashing for image retrieval. *Neurocomputing*, 403: 224–236, 2020. 2, 6

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Sec. 7.
   (c) Did you discuss any potential negative societal impacts of your work? [No] No explicit negative impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Sec. 4.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Sec. 4.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Sec. 6.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Sec. 6.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Sec. 6.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Sec. 6.
   (b) Did you mention the license of the assets? [Yes] See Sec. 6.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Codes appear in supplementary materials.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]