Learning to Hash Naturally Sorts

Jiaguo Yu[1] *, **Yuming Shen**[2] *, **Menghan Wang**[3], **Haofeng Zhang**[1] †, **Philip H.S. Torr**[2]
[1]Nanjing University of Science and Technology
[2]University of Oxford
[3]eBay
{yujiaguo, zhanghf}@njust.edu.cn, {yuming.shen, philip.torr}@eng.ox.ac.uk, menghanwang@ebay.com

Abstract

Learning to hash pictures a list-wise sorting problem. Its testing metrics, *e.g.*, mean-average precision, count on a sorted candidate list ordered by pair-wise code similarity. However, scarcely does one train a deep hashing model with the sorted results end-to-end because of the non-differentiable nature of the sorting operation. This inconsistency in the objectives of training and test may lead to sub-optimal performance since the training loss often fails to reflect the actual retrieval metric. In this paper, we tackle this problem by introducing Naturally-Sorted Hashing (NSH). We sort the Hamming distances of samples' hash codes and accordingly gather their latent representations for self-supervised training. Thanks to the recent advances in differentiable sorting approximations, the hash head receives gradients from the sorter so that the hash encoder can be optimized along with the training procedure. Additionally, we describe a novel Sorted Noise-Contrastive Estimation (SortedNCE) loss that selectively picks positive and negative samples for contrastive learning, which allows NSH to mine data semantic relations during training in an unsupervised manner. Our extensive experiments show the proposed NSH model significantly outperforms the existing unsupervised hashing methods on three benchmarked datasets.

1 Introduction

Learning to hash [Gionis *et al.*, 1999], naturally treated as a representation learning task in deep learning, indeed subclasses Approximate Nearest Neighbour (ANN) search that *learns to sort* at scale. The codes' Hamming distances between a query and a bunch of candidates measure their degrees of relevance and further determine their order of presence in the retrieval results. Hence, the conventional evaluation metrics explicitly reflect the positional and order sensitivity of the retrieved candidates in accuracy, which involves an $\mathrm{argsort}$ process, including mean-Average Precision (mAP), top-k accuracy and even Normalized Discounted Cumulative Gain (NDCG) in recommending systems. Oddly enough, most of the existing deep hashing models usually do not implement the concept of sorting, but instead resort to some alternative learning objectives such as recognition and pair-wise/triplet losses [Luo *et al.*, 2021b]. This counter-intuitive convention raises the question that *can we and shall we train the deep hashing model after sorting to match up with its evaluation metrics?*

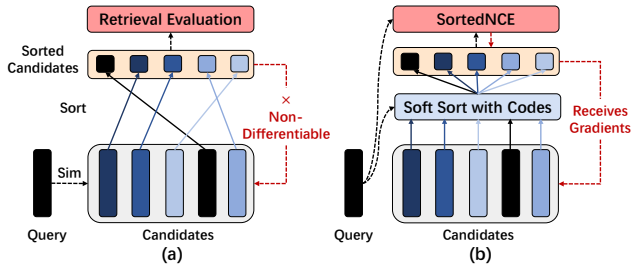

Figure 1: A brief motivation of NSH. **(a)** The actual testing metrics of learning to hash involves non-differentiable $\mathrm{argsort}$ operators. Hence, they can not be directly used for training. **(b)** The proposed NSH model best mimics the testing procedure that sorts the code similarity with soft approximations and is trained with a list-wise SortedNCE objective end-to-end.

Recalling one of the best practices in deep learning, the training objective of a model should explicitly represent its ultimate goal or testing measurement. For instance, a segmentation model is usually evaluated by the Intersection over Union (IoU) score and is trained with a similar IoU loss; a density estimation model expects a likelihood objective that describes the sample's density. The coherence between training and testing makes it non-trivial towards better performance. However, in the context of unsupervised hashing, there exist two main challenges to practice this vision.

The Non-Differentiable Kinks in Sorting and ANN As is discussed above, the main-stream measurements such as

*Equal contribution.

†Corresponding author. This work is supported by the National Natural Science Foundation of China (NSFC) under Grants No. 61872187, No. 62077023, No. 62072246, the UKRI grant: Turing AI Fellowship EP/W002981/1 and EPSRC/MURI grant: EP/N019474/1, and the University of Oxford's COVID-19 Research Response Fund: BRD00230. We would like to thank the Royal Academy of Engineering and FiveAI.

mAP and top-k accuracy are based on $\mathrm{argsort}$ operations, which does not derive a differentiable pathway from the actual scores to the hash layers. This makes it impossible to construct an end-to-end deep framework that is trained with the sorted output. We illustrate this problem in Fig. 1 (a). Though the order of relevance can be alternatively converted to positive/negative data pairs that facilitates ranking-based training, this solution, after all, just represents a part of the sorted retrieval list, and it consequently discard some information that potentially matters [Wang *et al.*, 2019]. Additionally, it also leads us to the second question, *i.e.*, how can be determine the order of similarity in an unsupervised manner?

The Lack of Clues of Relevance Unlike its supervised sibling where similarity labels are off the shelf, unsupervised hashing only observes data themselves. This also hinders existing model from mimicking the sorting-based evaluation metrics during training. Though it is possible to construct pair-wise pseudo labels in an alternating scheme [Shen *et al.*, 2018; Yang *et al.*, 2019], end-to-end training therefore becomes infeasible for deep models, and the errors of pseudo labels may propagate.

In this paper, we echo the title as the main motivation and tackle the challenges above by proposing Naturally-Sorted Hashing (NSH) that maximally present the properties of sorting during training towards better retrieval performance. As per the first challenge, we adopt the recent advances in differentiable $\mathrm{softsort}$ approximations [Prillo and Eisenschlos, 2020] as one building block that connects the hash encoder on bottom and the sorting-based loss on top. In particular, NSH follows [Shen *et al.*, 2020] that encodes two representations, *i.e.*, a binary hash code for similarity comparison and a continuous latent vector that carries detailed information. The code distances between samples feed the $\mathrm{softsort}$ operator and re-order the latent vectors for self-supervised training. To handle the second challenge, we propose a novel Sorted Noise-Contrastive Estimation (SortedNCE) loss that selectively picks the most related samples as the positive ones for contrastive learning. Hence, NSH mines data semantic relations during training and optimizes sorted retrieval candidate list of each datum on the fly. Fig. 1 (b) briefs our main idea, which better represents the testing scenario of deep hashing. Our main contributions include:

- For the first time, we describe an end-to-end NSH model that optimizes the sorted retrieval list that best represents the testing scenario of learning to hash and only requires gradient descent.
- To implement this vision, we additionally propose the SortedNCE loss that trains the sorted features in an unsupervised way.
- We show the superiority of NSH in retrieval performance on three benchmarked datasets against the most recent unsupervised deep hashing methods.

2 Related Work

Unsupervised Hashing Early work in unsupervised hashing method mainly focuses on learning compact representations [Gong *et al.*, 2013]. Several recent work with deep learning focus on the hash code quality [Li and van Gemert, 2021; Su *et al.*, 2018; Luo *et al.*, 2021b; Ghasedi Dizaji *et al.*, 2018]. Some others pay attention to the semantic awareness of the code [Shen *et al.*, 2018; Yang *et al.*, 2019; Shen *et al.*, 2020], while the majority resort to a pseudo-labelling scheme to mine data similarity as an individual module apart from neural network. Their performance is usually evaluated on the sorted candidates. However, they are not seek to implement the concept of sorting to mine similarity during training.

Hashing with Contrastive Learning Contrastive learning is a method to learn the general features of a dataset without labels by construct positive and negative pairs [Hadsell *et al.*, 2006]. Instance discrimination [Wu *et al.*, 2018] proposes a non-parametric cross-entropy loss to optimize the model at the instance level. Most recent works also use contrastive learning for hashing [Qiu *et al.*, 2021; Luo *et al.*, 2021b]. They adopt instance discrimination as the objective where positive and negative instances may still have overlapped semantics. Namely, each query image in a batch only treat its augmented view as a positive sample, which means even for extremely similar samples, they must be pushed apart. In addition, CIBHash [Qiu *et al.*, 2021] and CIMON [Luo *et al.*, 2021b] adjust the contrastive loss to suit the hashing learning criterion, yet employing the hash code directly in the contrastive loss does not allow for a good integration of the image's semantic content.

3 Preliminaries

Neural Sorting Operators Vanilla $\mathrm{argsort}$ is definitely non-differentiable, but recent research finds several approximations that are compatible with neural networks. Though it can be as well viewed as a linear programming problem [Blondel *et al.*, 2020], we opt to employ a simpler softmax approach [Prillo and Eisenschlos, 2020] that determines the permutation of a vector of similarity scores $\mathbf{s} \in \mathbb{R}^N$ as:

$$\mathrm{softsort}(\mathbf{s}) = \mathrm{softmax}\,\frac{-\,\mathrm{d_s}(\mathrm{sort}(\mathbf{s})\mathbb{1}^\intercal, \mathbb{1}\mathbf{s}^\intercal)}{\tau_\mathrm{s}}, \qquad (1)$$

where $\mathrm{d_s}(\cdot)$ is an arbitrary differentiable almost everywhere, semi–metric distance function[1], usually an L-1 norm, and τ_s is a temperature hyperparameter.

Twin-Bottleneck Hash Encoder Though factorized outputs are widely witnessed in deep learning, [Shen *et al.*, 2020] specifies different functionalities two outputs of a hash encoder $f_\theta(\cdot)$ parametrized by θ that encodes a datum $\mathbf{x}$

$$[\mathbf{h}, \mathbf{z}] = f_\theta(\mathbf{x}),\ \mathbf{b} = \mathrm{sign}(\mathbf{h}). \qquad (2)$$

$\mathbf{b}$ is usually followed by a gradient estimator, *e.g.*, $\partial\mathbf{b}/\partial\mathbf{h} \coloneqq \mathbb{I}$, to enable end-to-end training. Since the hash code is usually short and less informative, $\mathbf{h}$ and $\mathbf{b}$ are only used to compute the pair-wise data similarity that act as the *query/key* in the attention mechanism, while $\mathbf{z}$ carries the detailed information of $\mathbf{x}$ and plays the role of *value* in attention. Hence, an arbitrary loss built on the top of $f_\theta(\cdot)$ automatically tunes $\mathbf{b}$ to reflect the semantic locality. We follow the above idea to build the backbone of NSH, but our contributions lie in the operations on the top.

[1]Note that $\mathrm{d_s}(\cdot)$ is not the Hamming distance function.

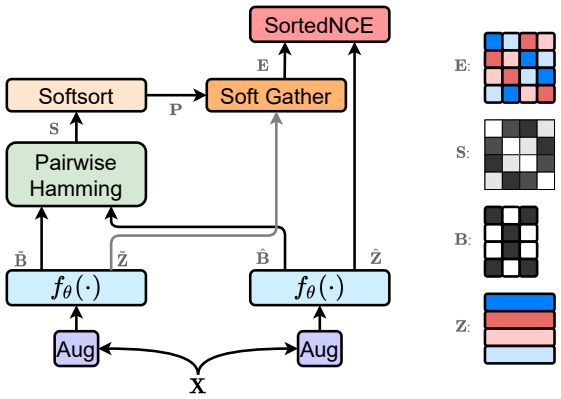

Figure 2: The training schematic of NSH. We illustrate the batch-based output shapes on the right where different colors refer to different instances. $\mathbf{Z} \in \mathbb{R}^{n\times d_z}$ are the latent representations. $\mathbf{B} \in \{-1,1\}^{n\times d_b}$ are the hash codes. $\mathbf{S} \in [0,1]^{n\times n}$ is the code-based similarity matrix for sorting. $\mathbf{E} \in \mathbb{R}^{n\times n\times d_z}$ are n soft-sorted lists of representations, of which each is ordered by the Hamming distance to the corresponding instance in the batch for SortedNCE.

4 Method

NSH considers an unsupervised hashing problem that maps a d_x-dimensional datum $\mathbf{x} \in \{\mathbf{x}\}_{i=1}^N$ to a binary vector $\mathbf{b} \in \{-1,1\}^{d_b}$, with N being the size of the entire dataset and d_b being the code length. In this paper, we by default consider an image hashing problem to match up with the conventional experimental settings, but NSH applies to arbitrary data modalities as long as random augmentations apply.

4.1 Model Structure

As is discussed in Sec. 3, we are inspired by [Shen *et al.*, 2020] to employ a twin-bottleneck hashing backbone to produce the actual hash code $\mathbf{b}$ and another set of latents $\mathbf{z}$ for each $\mathbf{x}$ during training, where the loss is imposed on the top of the transformation that both involves $\mathbf{z}$ and the Hamming distances between $\mathbf{b}$. NSH also adopts the recent advances in contrastive learning [Chen *et al.*, 2020a] as an unsupervised framework, which requires two sets of independent random data augmentations. In the following, we denote the outputs from the two augmented counterparts using $\tilde{\cdot}$ and $\hat{\cdot}$, *e.g.*, $\tilde{\mathbf{x}}$ and $\hat{\mathbf{x}}$, $\tilde{\mathbf{b}}$ and $\hat{\mathbf{b}}$, $\tilde{\mathbf{z}}$ and $\hat{\mathbf{z}}$, *etc*. In addition, as our training procedure involves multiple instances for sorting, we will use row-ordered batch-wise notations with capital letters when necessary, *e.g.*, $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1;\cdots;\tilde{\mathbf{x}}_n]$, $\hat{\mathbf{Z}} = [\hat{\mathbf{z}}_1;\cdots;\hat{\mathbf{z}}_n]$, *etc*, with n being the batch size to enable batch-based training.

Training Objective as a Function of Sorting Our ultimate goal is to build a fully-differentiable model that is trained on a bunch of semantically-sorted candidates:

$$\mathcal{L}_{\text{Sorted}} \coloneqq \underbrace{\text{NCE}}_{\text{Sec. 4.3}} \circ \underbrace{\text{Sort\&Gather} \circ \text{Hamming}}_{\text{Sec. 4.2}} \circ f_\theta(\mathbf{x}),$$

where each stacked component allows gradient to propagate back to $f_\theta(\cdot)$. Fig. 2 gives a glimpse of the structure of NSH. The instances under two different sets of augmentations are both rendered to the backbone encoder $f_\theta(\cdot)$, producing two groups of hash codes ($\tilde{\mathbf{B}}$ and $\hat{\mathbf{B}}$) and latents ($\tilde{\mathbf{Z}}$ and $\hat{\mathbf{Z}}$). The hash codes then determines the way to sort and permute the latents (Sec. 4.2). Then, the permuted latent tensor contributes to the proposed SortedNCE loss for training (Sec. 4.3), while the training procedure is given in Sec. 4.4.

4.2 Sorting and Gathering the Latents

Similar to [Shen *et al.*, 2020], we implant the computation of pair-wise hash code distances in the training model so that the code similarity can be optimized by any losses on the top. Formally, we define the code-based affinity matrix $\mathbf{S}$ as:

$$\mathbf{S} = \tilde{\mathbf{B}}\hat{\mathbf{B}}^\intercal/2d_b + 0.5, \tag{3}$$

so that its each entry represents the normalized pair-wise similarity, *i.e.*, $\mathbf{S}[i,j] = 1 - \text{Hamming}(\tilde{\mathbf{b}}_i, \hat{\mathbf{b}}_j)/d_b$. Eq. (3) only involves linear operations that are differentiable everywhere so that it can be used as a building block in neural network for back-propagation. Note that we are **not** imposing any loss term on $\mathbf{S}$ to prevent the model from falling into the pseudo-labelling training scheme, as discussed in Sec. 1

Sort out the Relevance with Hash Codes As the key operation of NSH, we treat each sample in the batch as the retrieval query, while letting the whole batch as the candidates. For each datum, one can pick the most relevant ones and place it at the head of the retrieval sequence and so on. To reflect this procedure in the model, we recall Eq. (1) to compute the sort-permutation matrix $\mathbf{p}_i$ for each $\mathbf{x}_i$, *i.e.*,

$$\mathbf{p}_i = \text{softsort}(-\mathbf{s}_i) \in (0,1)^{n\times n}, \tag{4}$$

where $\mathbf{s}_i \in [0,1]^n$ is the i-th row of $\mathbf{S}$ (Eq. (3)) that describes the relevance of the i-th item in the batch to all the others. Namely, an entry of $\mathbf{p}_i[j,k]$ at the position of $[j,k]$ stands for the probability of $\mathbf{x}_k$ becoming the most j-th related item to $\mathbf{x}_i$ [Prillo and Eisenschlos, 2020].

Re-order Features by Soft Gathering/Permutation For each $\mathbf{x}_i$, NSH produces a matrix of candidates' embeddings in the batch $\mathbf{e}_i$ where the features of the most similar items are placed on the top. As now we already have the soft-permutation matrix from Eq. (4), $\mathbf{e}_i$ can be defined by the following soft-gathering process:

$$\mathbf{e}_i = \mathbf{p}_i\tilde{\mathbf{Z}} \in \mathbb{R}^{n\times d_z}. \tag{5}$$

Here d_z is the size of the continuous latents $\tilde{\mathbf{z}}$. To clarify, $\mathbf{e}_i$ is just the sorted embedding matrix for $\mathbf{x}_i$, while a batch of data makes it a three-dimensional tensor, *i.e.*, $\mathbf{E} = [\mathbf{e}_1;\cdots;\mathbf{e}_n] \in \mathbb{R}^{n\times n\times d_z}$. The batch-based implementation of Eq. (5) can be easily achieved by the Einstein summation convention in recent deep learning toolboxes (see Alg. 1).

Remark: The Order of E Matters. $\mathbf{E}$ is literally n sorted lists of the latents $\tilde{\mathbf{Z}}$, of which each list $\mathbf{e}_i$ contains the representations of the in-batch retrieval results of $\mathbf{x}_i$ in a descending order of similarity. Notably, as $\tilde{\mathbf{b}}_i$ and $\hat{\mathbf{b}}_i$ are expected to be identical, the first row of $\mathbf{e}_i$, *i.e.*, $\mathbf{e}_i[1,:]$, will naturally be close to $\tilde{\mathbf{z}}_i$, representing $\mathbf{x}_i$ itself as the most relevant

one in the batch. From the second row on, the relevance decreases. The steps defined by Eqs. (3) to (5) automatically selects semantically-related instance and place them at the beginning of $\mathbf{E}$. Hence, one can easily determine the positive/negative samples of each $\mathbf{x}_i$ in a fully-differentiable way. This procedure mimics the real testing scenario of retrieval and enables list-wise training for better performance.

4.3 SortedNCE

Many existing unsupervised hashing models struggle in determining the similarity between data points, and they usually needs a held-out pseudo-labelling step that introduce additional noise [Su *et al.*, 2018; Luo *et al.*, 2021b]. However, NSH already has a bunch of sorted features $\mathbf{E}$ off the shelf, which makes it extremely easy to determine positive samples.

On the other hand, there would be multiple samples that share the same semantic information, but the vanilla single-label InfoNCE [Oord *et al.*, 2018] loss does not consider this overlap of semantics. We propose a learning objective that works with multiple positive pairs. In particular, since $\mathbf{E}$ is ordered, we define the SortedNCE loss based on the positions of the logits and let the first m samples being positive, *i.e.*,

$$\mathcal{L}_{\text{Sorted}} = \frac{-1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m} \log \underbrace{\frac{\kappa(\mathbf{e}_i[\,j\,,:],\hat{\mathbf{z}}_i)}{\kappa(\mathbf{e}_i[\,j\,,:],\hat{\mathbf{z}}_i)+\sum_{k=m+1}^{n}\kappa(\mathbf{e}_i[\,k\,,:],\hat{\mathbf{z}}_i)}}_{\text{Positive and negative samples only depend on the sorting positions}}, \tag{6}$$

where $\kappa(\mathbf{a},\mathbf{b}) = \exp(\cos(\mathbf{a},\mathbf{b})/\tau_{\text{c}})$. m and τ_{c} are treated as hyperparameters. Intuitively, Eq. (6) constructs m contrastive for m positive candidates in $\mathbf{e}_i$, of which each shapes a single-label cross-entropy term. Importantly, since the features of positive samples have fixed positions of $1\cdots m$ in $\mathbf{e}_i$, NSH does not requires an additional $\operatorname{argsort}$ operator to find the most relevant items that involves non-differentiable computations. We discuss the benefits of this design in Sec. 4.5. A single m-label cross-entropy term may work here too, by treating the first m logits as positive. This solution is similar to SupCon [Khosla *et al.*, 2020], but we experimentally show in Sec. 5.3 that it underperforms SortedNCE for our case.

4.4 Training and Inference

Training NSH is extremely simple. In addition to $\mathcal{L}_{\text{Sorted}}$, it only requires the very conventional quantization loss $\mathcal{L}_{\text{R}} = (\|\operatorname{sg}(\tilde{\mathbf{B}}) - \tilde{\mathbf{H}}\|_2 + \|\operatorname{sg}(\hat{\mathbf{B}}) - \hat{\mathbf{H}}\|_2)/2n$ [Erin Liong *et al.*, 2015] to enhance the concreteness of the hashing layer, *i.e.*,

$$\mathcal{L}_{\text{NSH}} = \mathcal{L}_{\text{Sorted}} + \mathcal{L}_{\text{R}}. \tag{7}$$

Here, $\operatorname{sg}(\cdot)$ is the stop-gradient operation. We place $\operatorname{sg}(\cdot)$ here to avoid duplicated gradients from $\mathbf{B}$ since we have previously manually defined the gradient estimator of $\partial\mathbf{b}/\partial\mathbf{h}$. Alg. 1 describes the training process of NSH in Python-style pseudo codes.

Encoding Testing Samples On testing, NSH does not require any sorting or gathering layers. The hash code of a testing sample can be directly obtained from the backbone $f_\theta(\cdot)$ without interacting with the other ones, while the semantic awareness is memorized by the parameters during training.

Algorithm 1: The Training Procedure of NSH

Input: Dataset $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{N}$ and batch size n.
Output: Network parameters θ.

```
for batch in D.repeat() do
    batch1, batch2 = aug(batch), aug(batch) # [n d_x]
    [b1, z1], [b2, z2] = f_θ(batch1), f_θ(batch2)
    s = matmul(b1, b2.T)/2/d_l + 0.5 # 1 − Hamming/d_l
    p = softsort(−s) # [n n n]
    e = einsum('nnn, nd → nnd', p, z1)
    # The code below describes SortedNCE
    labels = onehot(zeros([n]), n−m+1)
    cos = einsum('nnd, nd → nn', e, z2)
    loss = 0
    for i in range(m) do
        pos, neg = cos[:, i], cos[:, m:]
        logits = softmax(concat([pos, neg])/τ_c)
        loss += cross_entropy(logits, labels)/m/n
    end
    loss += quantization_loss
    optimizer.apply_gradients(loss, θ)
end
```

4.5 Discussion: What Makes It Different?

NSH defines a specific gradient pathway from the contrastive loss to the hash codes that reflect the rewards to both the code semantic similarity and the sorted candidates' quality end-to-end. We slightly abuse the differential notations to represent the chain rules of these concepts as follows:

$$\frac{\partial\mathcal{L}_{\text{Sorted}}}{\partial\mathbf{B}} = \underbrace{\frac{\partial\mathcal{L}_{\text{Sorted}}}{\partial\mathbf{P}}}_{\substack{\text{Sorting/Retrieval}\\ \text{Rewards}}} \underbrace{\frac{\partial\mathbf{P}}{\partial\mathbf{S}}}_{\substack{\text{Similarity}\\ \text{Rewards}}} \underbrace{\frac{\partial\mathbf{S}}{\partial\mathbf{B}}}_{\substack{\text{Code}\\ \text{Quality}}}. \tag{8}$$

In this way, the model automatically optimizes the key components of hashing task to best fit and favour the presence of $\mathcal{L}_{\text{Sorted}}$, even if the similarity ground truth is not given.

As to a Hard-Sort Baseline One can easily build a baseline by replacing Eqs. (4) and (5) with $\operatorname{argsort}$ and computing the contrastive loss according to the highest $\operatorname{argsort}$ entries. However, we argue this design is just enhancing the decisions of $\operatorname{argsort}$, because the absence of the gradients $\partial\operatorname{argsort}/\partial\mathbf{S}$ does not allow the model to mine the data similarity and fail to update $\mathbf{S}$ during gradient descent.

As to Many Existing Models Some methods does not consider optimizing $\mathbf{S}$ during training [Su *et al.*, 2018; Luo *et al.*, 2021b], which may lead to biased results. The majority of recent works treat $\mathbf{S}$ as pseudo labels that are updated out of the training loop, since their design do not compute the gradients of $\partial\mathcal{L}/\partial\mathbf{S}$ of Eq. (8) [Shen *et al.*, 2018; Yang *et al.*, 2019]. As discussed previously, this design allows the errors caused by false assignments to propagate during network training. Last but not least, NSH is the first model to optimize the mocked sorting results through $\partial\mathcal{L}/\partial\mathbf{P}$ of Eq. (8), which literally performs unsupervised list-wise training end-to-end. Echoing our motivation, this strategy better fits the retrieval task that undergoes list-wise evaluation measurements.

Method	Reference	CIFAR-10			NUS-WIDE			MS COCO		
		16 bits	32 bits	64 bits	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
AGH [Liu *et al.*, 2011]	ICML11	0.333	0.357	0.358	0.592	0.615	0.616	0.596	0.625	0.631
ITQ [Gong *et al.*, 2013]	PAMI13	0.305	0.325	0.349	0.627	0.645	0.664	0.598	0.624	0.648
DGH [Liu *et al.*, 2014]	NeurIPS14	0.335	0.353	0.361	0.572	0.607	0.627	0.613	0.631	0.638
DeepBit [Lin *et al.*, 2016]	CVPR16	0.194	0.249	0.277	0.392	0.403	0.429	0.407	0.419	0.430
SGH [Dai *et al.*, 2017]	ICML17	0.435	0.437	0.433	0.593	0.590	0.607	0.594	0.610	0.618
BGAN [Song *et al.*, 2018]	AAAI18	0.525	0.531	0.562	0.684	0.714	0.730	0.645	0.682	0.707
BinGAN [Zieba *et al.*, 2018]	NeurIPS18	0.476	0.512	0.520	0.654	0.709	0.713	0.651	0.673	0.696
GreedyHash [Su *et al.*, 2018]	NeurIPS18	0.448	0.473	0.501	0.633	0.691	0.731	0.582	0.668	0.710
HashGAN [Ghasedi Dizaji *et al.*, 2018]	CVPR18	0.447	0.463	0.481	-	-	-	-	-	-
DVB [Shen *et al.*, 2019]	IJCV19	0.403	0.422	0.446	0.604	0.632	0.665	0.570	0.629	0.623
DistillHash [Yang *et al.*, 2019]	CVPR19	0.284	0.285	0.288	0.667	0.675	0.677	-	-	-
TBH [Shen *et al.*, 2020]	CVPR20	0.532	0.573	0.578	0.717	0.725	0.735	0.706	0.735	0.722
$\mathrm{MLS}^3\mathrm{RDUH}$ [Tu *et al.*, 2020]	IJCAI20	0.369	0.394	0.412	0.713	0.727	0.750	0.607	0.622	0.641
DATE [Luo *et al.*, 2021a]	MM21	0.577	0.629	0.647	**0.793**	0.809	0.815	-	-	-
MBE [Li and van Gemert, 2021]	AAAI21	0.561	0.576	0.595	0.651	0.663	0.673	-	-	-
CIMON [Luo *et al.*, 2021b]*	IJCAI21	0.451	0.472	0.494	-	-	-	-	-	-
CIBHash [Qiu *et al.*, 2021]	IJCAI21	0.590	0.622	0.641	0.790	0.807	0.815	0.737	0.760	0.775
NSH	**Proposed**	**0.706**	**0.733**	**0.756**	0.758	**0.811**	**0.824**	**0.746**	**0.774**	**0.783**

Table 1: Performance comparison (*w.r.t.* mAP) of NSH and the state-of-the-art **unsupervised** hashing methods. *Note that we use a more common setting on NUS-WIDE with the 21 most frequent classes, while some papers report results on 10 classes.

Method	CIFAR-10			MS COCO		
	16 bits	32 bits	64 bits	16 bits	32 bits	64 bits
ITQ	0.276	0.292	0.309	0.607	0.637	0.662
AGH	0.306	0.321	0.317	0.602	0.635	0.644
DGH	0.315	0.323	0.324	0.623	0.642	0.650
HashGAN	0.418	0.436	0.455	-	-	-
SGH	0.387	0.380	0.367	0.604	0.615	0.637
GreedyHash	0.322	0.403	0.444	0.603	0.624	0.675
TBH	0.497	0.524	0.529	0.646	0.698	0.701
CIBHash	0.526	0.570	0.583	0.734	0.767	0.785
NSH	**0.691**	**0.716**	**0.744**	**0.733**	**0.770**	**0.805**

Table 2: P@1000 results of NSH and compared methods on CIFAR-10 and MS COCO.

5 Experiments

5.1 Experimental Setup

CIFAR-10 [Krizhevsky and Hinton, 2009] comes with 60,000 images. We follow [Ghasedi Dizaji *et al.*, 2018] to have a 50,000-10,000 train-test split.

NUS-WIDE [Chua *et al.*, 2009] has of 81 categories of images. We adopt the 21-class subset following [Qiu *et al.*, 2021]. 100 images of each class are utilized as a query set, with the remaining being the gallery.

MS COCO [Lin *et al.*, 2014] is a benchmark for multiple tasks. We use the conventional set with 12,2218 images. We randomly select 5,000 images as queries with the remaining ones the database.

Evaluation Metric We adopt several widely-used evaluation metrics, including mean Average Precision (mAP), top-k precision (P@k), Precision-Recall (P-R) curves and precision of Hamming radius within 2 (P@r=2). Following the recent convention [Shen *et al.*, 2020; Qiu *et al.*, 2021], we adopt mAP@1000 for CIFAR-10, mAP@5000 for NUS-WIDE and MSCOCO. For all datasets, two data points will be considered as relevant if they share at least one common label.

Implementation Details The proposed method is implemented with Tensorflow. We use the Adam optimizer [Kingma and Ba, 2015] to train the networks with a learning rate of 1×10^{-5}, and the batch size is 50. We train the model for 200 epochs at most. All the images are resized to $224 \times 224 \times 3$ and we adopt the image augmentation strategies of MoCo-v2 [Chen *et al.*, 2020b]. We use the ResNet-50 [He *et al.*, 2016] until the last pooling layer and top two fully-connected layers as the hash head and the latent feature head. The hash head is followed by $\tanh$ and sign operations to produce $\mathbf{b}$, while the latent feature head is followed by a L-2 normalization layer to produce $\mathbf{z}$, with a dimensionality of $d_{\mathrm{z}} = 1024$. The contrastive temperature τ_{c} and the number of positive samples m we picked was set to $\{0.1, 0.5, 0.5\}$ and $\{2, 3, 3\}$ for CIFAR-10, NUS-WIDE and MS COCO. Following [Prillo and Eisenschlos, 2020], the $\mathrm{softsort}$ temperature is set to the code length $\tau_{\mathrm{s}} = d_{\mathrm{b}}$.

5.2 Comparison with the SotA

Baselines We compare NSH against 17 state-of-the-art baselines, including 3 traditional unsupervised hashing methods and 14 recent unsupervised hashing methods. For a fair comparison, we report the results with VGG features [Simonyan and Zisserman, 2015] which is pretrained on ImageNet if the baseline is not trained from scratch.

Results Tab. 1 shows the retrieval performance in mAP. It can be clearly observed that NSH obtain the best results on the three datasets. Another interesting observation is that NSH is significantly better than CIBHash across different hash bits and datasets. Note that both these two methods employ contrastive learning. In addition, the P-R curves and the precision within Hamming radius of 2 (P@H=2) of NSH and several baselines on CIFAR-10 are reported in Fig. 3.

5.3 Ablation Study

We validated the effectiveness of our motivation and design via the following baselines, with the results shown in Tab. 3.

(i) **Hard-Sort Baseline.** We first explore the effect of our

	Baseline	16 bits	32 bits	64 bits
(i)	Hard-Sort Baseline	0.323	0.405	0.477
(ii)	Without $\mathcal{L}_R$	0.686	0.718	0.735
(iii)	$\mathcal{L}_{\text{Sorted}} \rightarrow$ Decoding Loss	0.489	0.501	0.544
(iv)	Single Bottleneck	0.606	0.631	0.650
(v)	Without softsort	0.641	0.695	0.707
(vi)	$\mathcal{L}_{\text{Sorted}} \rightarrow$Multi-Label NCE	0.689	0.710	0.730
	NSH	**0.706**	**0.733**	**0.756**

Table 3: Ablation study results of mAP@1000 on CIFAR-10.

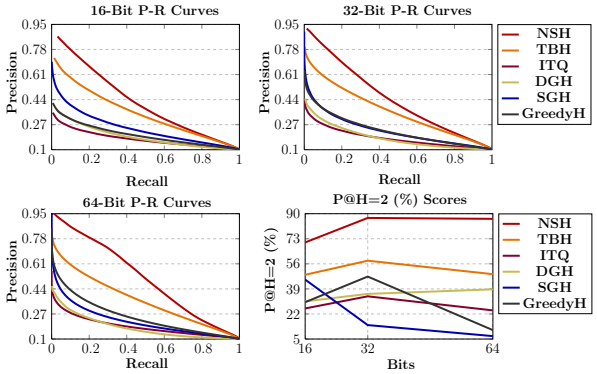

Figure 3: P-R curves and P@H≤2 results of NSH and compared methods on CIFAR-10.

core motivation with differentiable sort on hashing. This baseline is also described in Sec. 4.5 that replaces Eqs. (4) and (5) with argsort and computes the contrastive loss according to the highest argsort entries. As is previously discussed, this baseline fail to optimize the code during training, and thus the results are not promising.

(ii) **Without $\mathcal{L}_R$.** We also evaluate how important the convention quantization loss can be in NSH. From Tab. 3, we see that this traditional regularizer dose not influence the final results much. Hence we can give full credit to our design for the good performance.

(iii) **$\mathcal{L}_{\text{Sorted}} \rightarrow$Decoding Loss.** To demonstrate the effectiveness of our SortedNCE, we construct this baseline by removing $\mathcal{L}_{\text{Sorted}}$ and add a decoder with a decoding loss after sorting. In this case, reconstruction only requires $\mathbf{e}_i[1,:]$ as the most relevant latents, but the network is still fully trainable. It can be observed that this baseline obtains similar performance to TBH [Shen *et al.*, 2020] as both of them involve an auto-encoding structure.

(iv) **Single Bottleneck.** This baseline removes $\mathbf{z}$ in the network, so that Eq. (5) gathers the hash code only, *i.e.*, $\mathbf{e}_i = \mathbf{p}_i\tilde{\mathbf{B}}$. This baseline performs close to CIBHash [Qiu *et al.*, 2021] as both of their contrastive losses are imposed to the code-based features. The drop in performance accords our intuition to employ the twin-bottleneck encoder.

(v) **Without** softsort**.** This baseline removes the operations defined by Eqs. (4) and (5), and then compute $\tilde{\mathbf{E}} = \mathbf{S}\tilde{\mathbf{B}}, \hat{\mathbf{E}} = \mathbf{S}\hat{\mathbf{B}}$. Hence, a conventional SimCLR-like contrastive learning loss [Chen *et al.*, 2020a] can be built upon $\tilde{\mathbf{E}}$ and $\hat{\mathbf{E}}$. Though it produces good results as well, the performance margin between this baseline and NSH is still significant, showing that our core motivation to train a sorted list is valid.

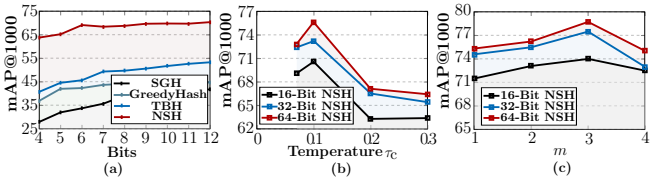

Figure 4: **(a)** mAP@1000 results with extremely short code lengths on CIFAR-10. **(b)** Effects of different temperatures τ_c. **(c)** Effects of different sizes of m in SortedNCE.

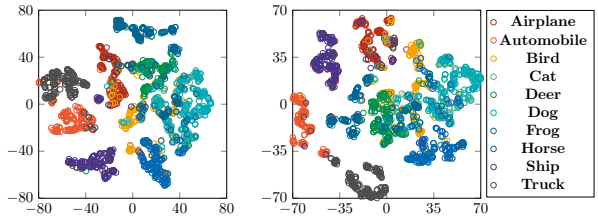

Figure 5: 32-bit and 64-bit visualization results with t-SNE on CIFAR-10.

(vi) **$\mathcal{L}_{\text{Sorted}} \rightarrow$Multi-Label NCE.** Our SortedNCE actually constructs m cross-entropy terms. Each one only has one positive label. It is also possible to replace it with a multi-label NCE loss such as SupCon [Khosla *et al.*, 2020], by marking the first m entries as positive. However, this baseline still underperforms NSH. We suspect this is because the softmax operator fits single-label objective better.

5.4 Hyperparameters and Visualization

We study the influence of the temperature τ_c and the number of positive samples m we picked, which are reported in Fig. 4 (b) and (c). We also provide the results with extremely short code length in Fig. 4 (a). Though the performance under different settings of hyperparameters varies, it is overall stable and is yet representing the state-of-the-art. NSH is not very sensitive to the hyperparameters. We do not assess different values of τ_s as its value is recommended by [Prillo and Eisenschlos, 2020]. In addition, the proportion of $\mathcal{L}_R$ does not influence the performance much so we skip its weighing hyperparameter here. We plot the t-SNE [Maaten and Hinton, 2008] results in Fig. 5 to illustrate our semantic awareness.

6 Conclusion

In this paper, we discussed the disagreement in the presence of training and testing objectives of unsupervised deep hashing and then proposed NSH to solve this problem. NSH overcame the main difficulties to mine the data semantics by sorting and is trained in a list-wise fully-differentiable manner that better reflects the testing scenario of retrieval. We adopted the recent advances in softsort and proposed SortedNCE to implement our vision. Our experimental results endorsed our motivation and design, showing the superiority of NSH in performance.

References

[Blondel *et al.*, 2020] Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *ICML*, 2020.

[Chen *et al.*, 2020a] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[Chen *et al.*, 2020b] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[Chua *et al.*, 2009] 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 *CIVR*, 2009.

[Dai *et al.*, 2017] Bo Dai, Ruiqi Guo, Sanjiv Kumar, Niao He, and Le Song. Stochastic generative hashing. In *ICML*, 2017.

[Erin Liong *et al.*, 2015] Venice Erin Liong, Jiwen Lu, Gang Wang, Pierre Moulin, and Jie Zhou. Deep hashing for compact binary codes learning. In *CVPR*, 2015.

[Ghasedi Dizaji *et al.*, 2018] Kamran Ghasedi Dizaji, Feng Zheng, Najmeh Sadoughi, Yanhua Yang, Cheng Deng, and Heng Huang. Unsupervised deep generative adversarial hashing network. In *CVPR*, 2018.

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[Gong *et al.*, 2013] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2013.

[Hadsell *et al.*, 2006] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, 2006.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[Khosla *et al.*, 2020] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *NeurIPS*, 2020.

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

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

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

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

[Lin *et al.*, 2016] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. Learning compact binary descriptors with unsupervised deep neural networks. In *CVPR*, 2016.

[Liu *et al.*, 2011] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *ICML*, 2011.

[Liu *et al.*, 2014] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. Discrete graph hashing. In *NeurIPS*, 2014.

[Luo *et al.*, 2021a] Xiao Luo, Daqing Wu, Zeyu Ma, Chong Chen, Minghua Deng, Jianqiang Huang, and Xian-Sheng Hua. A statistical approach to mining semantic similarity for deep unsupervised hashing. In *MM*, 2021.

[Luo *et al.*, 2021b] Xiao Luo, Daqing Wu, Zeyu Ma, Chong Chen, Huasong Zhong, Minghua Deng, Jianqiang Huang, and Xiansheng Hua. Cimon: Towards high-quality hash codes. In *IJCAI*, 2021.

[Maaten and Hinton, 2008] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[Oord *et al.*, 2018] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[Prillo and Eisenschlos, 2020] Sebastian Prillo and Julian Eisenschlos. Softsort: A continuous relaxation for the argsort operator. In *ICML*, 2020.

[Qiu *et al.*, 2021] Zexuan Qiu, Qinliang Su, Zijing Ou, Jianxing Yu, and Changyou Chen. Unsupervised hashing with contrastive information bottleneck. In *IJCAI*, 2021.

[Shen *et al.*, 2018] Fumin Shen, Yan Xu, Li Liu, Yang Yang, Zi Huang, and Heng Tao Shen. Unsupervised deep hashing with similarity-adaptive and discrete optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):3034–3044, 2018.

[Shen *et al.*, 2019] Yuming Shen, Li Liu, and Ling Shao. Unsupervised binary representation learning with deep variational networks. *International Journal of Computer Vision*, 127(11-12):1614–1628, 2019.

[Shen *et al.*, 2020] Yuming Shen, Jie Qin, Jiaxin Chen, Mengyang Yu, Li Liu, Fan Zhu, Fumin Shen, and Ling Shao. Auto-encoding twin-bottleneck hashing. In *CVPR*, 2020.

[Simonyan and Zisserman, 2015] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

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

[Su *et al.*, 2018] Shupeng Su, Chao Zhang, Kai Han, and Yonghong Tian. Greedy hash: Towards fast optimization for accurate hash coding in cnn. In *NeurIPS*, 2018.

[Tu *et al.*, 2020] Rong-Cheng Tu, Xianling Mao, and Wei Wei. Mls3rduh: Deep unsupervised hashing via manifold based local semantic similarity structure reconstructing. In *IJCAI*, pages 3466–3472, 2020.

[Wang *et al.*, 2019] Xinshao Wang, Yang Hua, Elyor Kodirov, Guosheng Hu, Romain Garnier, and Neil M Robertson. Ranked list loss for deep metric learning. In *CVPR*, 2019.

[Wu *et al.*, 2018] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *CVPR*, 2018.

[Yang *et al.*, 2019] Erkun Yang, Tongliang Liu, Cheng Deng, Wei Liu, and Dacheng Tao. Distillhash: Unsupervised deep hashing by distilling data pairs. In *CVPR*, 2019.

[Zieba *et al.*, 2018] Maciej Zieba, Piotr Semberecki, Tarek El-Gaaly, and Tomasz Trzcinski. Bingan: Learning compact binary descriptors with a regularized gan. In *NeurIPS*, 2018.