Learning to Hash Naturally Sorts

Jiaguo Yu\textsuperscript{1} *, Yuming Shen\textsuperscript{2} *, Menghan Wang\textsuperscript{3}, Haofeng Zhang\textsuperscript{1} \textdagger, Philip H.S. Torr\textsuperscript{2}
\textsuperscript{1}Nanjing University of Science and Technology
\textsuperscript{2}University of Oxford
\textsuperscript{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, \textit{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 (SortNCE) 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 \textit{et al.}, 1999], naturally treated as a representation learning task in deep learning, indeed subclasses Approximate Nearest Neighbour (ANN) search that \textit{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 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 \textit{et al.}, 2021b]. This counter-intuitive convention raises the question that \textit{can we and shall we train the deep hashing model after sorting to match up with its evaluation metrics?}

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...
mAP and top-k accuracy are based on 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 discards some information that potentially matters [Wang et al., 2019]. Additionally, it also leads us to the second question, i.e., how can we 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 models 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 presents the properties of sorting during training towards better retrieval performance. As per the first challenge, we adopt the recent advances in differentiable 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 softmax 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) briefly 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; Ghasedari 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-labeling 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 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 $s \in \mathbb{R}^N$ as:

$$\text{softmax}(s) = \frac{\text{softmax}(\text{sort}(s) \mathbb{1}^T, \mathbb{1}s^T)}{\tau_s}$$

where $\text{softmax}(\cdot)$ is an arbitrary differentiable almost everywhere, semi–metric distance function, usually an L-1 norm, and $\tau_s$ 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 $\theta$ that encodes a datum $x$:

$$[h, z] = f_\theta(x), \quad b = \text{sign}(h).$$

$b$ is usually followed by a gradient estimator, e.g., $\partial b / \partial h := \mathbb{1}$, to enable end-to-end training. Since the hash code is usually short and less informative, $h$ and $b$ are only used to compute the pair-wise data similarity that act as the query/key in the attention mechanism, while $z$ carries the detailed information of $x$ and plays the role of value in attention. Hence, an arbitrary loss built on the top of $f_\theta(\cdot)$ automatically tunes $b$ 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.

Note that $d_\ell(\cdot)$ is not the Hamming distance function.
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 \((b, \tilde{b})\) and latents \((\tilde{z}, \tilde{Z})\). 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 \( S \) as:

\[
S = \tilde{B}\tilde{B}^T/2d_b + 0.5, \tag{3}
\]

so that its each entry represents the normalized pair-wise similarity, i.e., \( S[i, j] = 1 - \text{Hamming}(b_i, b_j)/d_b \). 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 \( S \) to prevent the model from falling into the pseudo-labeling 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 \( p_i \) for each \( x_i \), i.e.,

\[
p_i = \text{softsort}(\sim s_i) \in (0, 1)^n \times n, \tag{4}
\]

where \( s_i \in [0, 1]^n \) is the \( i \)-th row of \( S \) (Eq. (3)) that describes the relevance of the \( i \)-th item in the batch to all the others. Namely, an entry of \( p_i [j, k] \) at the position of \([j, k]\) stands for the probability of \( x_k \) becoming the most \( j \)-th related item to \( x_i \) [Prillo and Eisenschlos, 2020].

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

\[
e_i = p_i \tilde{Z} \in \mathbb{R}^{n \times d_z}, \tag{5}
\]

Here \( d_z \) is the size of the continuous latents \( \tilde{z} \). To clarify, \( e_i \) is just the sorted embedding matrix for \( x_i \), while a batch of data makes it a three-dimensional tensor, i.e., \( E = [e_1; \cdots; e_n] \in \mathbb{R}^{n \times n \times d_z} \). 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. \( E \) is literally \( n \) sorted lists of the latents \( \tilde{Z} \), of which each list \( e_i \) contains the representations of the in-batch retrieval results of \( x_i \) in a descending order of similarity. Notably, as \( b_i \) and \( \tilde{b}_i \) are expected to be identical, the first row of \( e_i \), i.e., \( e_i [1, :] \), will naturally be close to \( \tilde{z}_i \), representing \( x_i \) itself as the most relevant.

4 Method

NSH considers an unsupervised hashing problem that maps a \( d_x \)-dimensional datum \( x \in \{X\}_{i=1}^N \) to a binary vector \( b \in \{-1, 1\}^{d_x} \), 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 \( b \) and another set of latents \( z \) for each \( x \) during training, where the loss is imposed on the top of the transformation that both involves \( x \) and the Hamming distances between \( b \). 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 \( \cdot \) and \( \hat{\cdot} \), e.g., \( x \) and \( \hat{x} \), \( b \) and \( \hat{b} \), \( z \) and \( \hat{z} \), 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., \( X = [x_1; \cdots; x_n] \), \( \hat{Z} = [\hat{z}_1; \cdots; \hat{z}_n] \), 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:

\[
\mathcal{L}_{\text{Sorted}} = \mathbb{E} \left( \text{NCE} \circ \text{SoftGather} \circ \text{Hamming} \circ f_{\theta}(x) \right), \tag{Sec. 4.3}
\]

where each stacked component allows gradient to propagate back to \( f_{\theta}(\cdot) \). Fig. 2 gives a glimpse of the structure of

![Figure 2: The training schematic of NSH. We illustrate the batch-based output shapes on the right where different colors refer to different instances.](image)

**Image Caption:**

Figure 2: The training schematic of NSH. We illustrate the batch-based output shapes on the right where different colors refer to different instances. \( Z \in \mathbb{R}^{n \times d_z} \) are the latent representations. \( B \in \{-1, 1\}^{n \times d_b} \) are the hash codes. \( S \in [0, 1]^{n \times n} \) is the code-based similarity matrix for sorting. \( E \in \mathbb{R}^{n \times n \times d_z} \) 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.
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 $E$. Hence, one can easily determine the positive/negative samples of each $x_i$ 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 $E$ 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 $E$ is ordered, we define the SortedNCE loss based on the positions of the logits and let the first $m$ samples being positive, i.e.,

$$L_{\text{Sorted}} = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{m} \log \frac{\kappa(e_i[j],z_i)}{\sum_{k=1}^{m} \kappa(e_i[k],z_i)}$$

where $\kappa(a,b) = \exp(\cos(a,b)/\tau_c)$. $m$ and $\tau_c$ are treated as hyperparameters. Intuitively, Eq. (6) constructs $m$ contrastive for $m$ positive candidates in $e_i$, of which each shares a single-label cross-entropy term. Importantly, since the features of positive samples have fixed positions of $1 \cdots m$ in $e_i$, NSH does not requires an additional 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 $L_{\text{Sorted}}$, it only requires the very conventional quantization loss $L_R = (\|s\text{g}(B) - H\|_2 + \|s\text{g}(B) - H\|_2)/2n$ [Erin Liong et al., 2015] to enhance the concreteness of the hashing layer, i.e.,

$$L_{\text{NSH}} = L_{\text{Sorted}} + L_R.$$

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

### Algorithm 1: The Training Procedure of NSH

**Input:** Dataset $D = \{x_i\}_{i=1}^n$ and batch size $n$.

**Output:** Network parameters $\theta$.

```python
for batch in $D$ do
    batch1, batch2 = aug(batch), aug(batch) # [n, d]
    [b1, z1], [b2, z2] = f0(batch1), f0(batch2)
    s = matmul(b1, b2) / 2 / d1 + 0.5 # 1 - Hamming/d1
    p = softmax(-s) # [n, n]
    e = einsum(‘nn, nd -> nnd’, p, z1)
    # The code below describes SortedNCE
    labels = onehot(zeros([n]), n-m+1)
    loss = 0
    for i in range(n)
        pos, neg = cos[i], cos[i, :]
        logit = softmax(concat([pos, neg]), /tau)
        loss += cross_entropy(logits, labels)/m
    end
    loss += quantization_loss
    optimizer.apply_gradients(loss, $\theta$)
end
```

Without interacting with the other ones, while the semantic awareness is memorized by the parameters during training.

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:

$$\frac{\partial L_{\text{Sorted}}}{\partial B} = \frac{\partial L_{\text{Sorted}}}{\partial P} \frac{\partial P}{\partial S} \frac{\partial S}{\partial B}.$$  

In this way, the model automatically optimizes the key components of hashing task to best fit and favour the presence of $L_{\text{Sorted}}$, 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 argsort and computing the contrastive loss according to the highest argsort entries. However, we argue this design is just enhancing the decisions of argsort, because the absence of the gradients $\partial \text{argsort}/\partial S$ does not allow the model to mine the data similarity and fail to update $S$ during gradient descent.

**As to Many Existing Models** Some methods does not consider optimizing $S$ during training [Su et al., 2018; Luo et al., 2021b], which may lead to biased results. The majority of recent works treats $S$ as pseudo labels that are updated out of the training loop, since their design do not compute the gradients of $\partial L/\partial S$ 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 $\partial L/\partial P$ 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.
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@H=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.

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 12,221 images. We randomly select 5,000 images as queries with the remaining being the gallery.

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@H=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 $1 \times 10^{-5}$, 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 b, while the latent feature head is followed by a L-2 normalization layer to produce z, with a dimensionality of $d_z = 1024$. The contrastive temperature $\tau_c$ 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 softsort temperature is set to the code length $\tau_c = d_z$.

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>16 bits</th>
<th>32 bits</th>
<th>64 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.323</td>
<td>0.405</td>
<td>0.477</td>
</tr>
<tr>
<td>(i) Hard-Sort Baseline</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ii) Without $L_R$</td>
<td>0.686</td>
<td>0.718</td>
<td>0.735</td>
</tr>
<tr>
<td>(iii) $L_{\text{Sorted}} \rightarrow \text{Decoding Loss}$</td>
<td>0.489</td>
<td>0.501</td>
<td>0.544</td>
</tr>
<tr>
<td>(iv) Single Bottleneck</td>
<td>0.606</td>
<td>0.631</td>
<td>0.650</td>
</tr>
<tr>
<td>(v) Without sortsort</td>
<td>0.641</td>
<td>0.695</td>
<td>0.707</td>
</tr>
<tr>
<td>(vi) $L_{\text{Sorted}} \rightarrow \text{Multi-Label NCE}$</td>
<td>0.689</td>
<td>0.710</td>
<td>0.730</td>
</tr>
</tbody>
</table>

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 $L_R$. We also evaluate how important the conventional quantization loss can be in NSH. From Tab. 3, we see that this traditional regularizer does not influence the final results much. Hence we can give full credit to our design for the good performance.

(iii) $L_{\text{Sorted}} \rightarrow \text{Decoding Loss}$. To demonstrate the effectiveness of our SortedNCE, we construct this baseline by removing $L_{\text{Sorted}}$ and add a decoder with a decoding loss after sorting. In this case, reconstruction only requires $e_i[1,:]$ 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 $z$ in the network, so that Eq. (5) gathers the hash code only, i.e., $e_i = p_i B$. 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 sortsort. This baseline removes the operations defined by Eqs. (4) and (5), and then compute $E = SB, \hat{E} = SB$. Hence, a conventional SimCLR-like contrastive learning loss [Chen et al., 2020a] can be built upon $E$ and $\hat{E}$. 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.

(vi) $L_{\text{Sorted}} \rightarrow \text{Multi-Label NCE}$. 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 $\text{softmax}$ operator fits single-label objective better.

5.4 Hyperparameters and Visualization

We study the influence of the temperature $\tau_c$ 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 $\tau_c$ as its value is recommended by [Prillo and Eisenschlos, 2020]. In addition, the proportion of $L_B$ 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 sortsort and proposed SortedNCE to implement our vision. Our experimental results endorsed our motivation and design, showing the superiority of NSH in performance.
References


