Long-Tail Hashing

ABSTRACT

Hashing, which represents data items as compact binary codes, has been becoming a more and more popular technique, e.g., for large-scale image retrieval, owing to its super fast search speed as well as its extremely economical memory consumption. However, existing hashing methods all try to learn binary codes from artificially balanced datasets which are not commonly available in real-world scenarios. In this paper, we propose *Long-Tail Hashing Network* (LTHNet), a novel two-stage deep hashing approach that addresses the problem of learning to hash for more realistic datasets where the data labels roughly exhibit a long-tail distribution. Specifically, the first stage is to learn relaxed embeddings of the given dataset with its long-tail characteristic taken into account via an end-to-end deep neural network; the second stage is to binarize those obtained embeddings. A critical part of LTHNet is its extended dynamic meta-embedding module which can adaptively realize visual knowledge transfer between head and tail classes, and thus enrich image representations for hashing. Our experiments have shown that LTHNet achieves dramatic performance improvements over all state-of-the-art competitors on long-tail datasets, with no or little sacrifice on balanced datasets. Further analyses reveal that while to our surprise directly manipulating class weights in the loss function has little effect, the extended dynamic meta-embedding module, the usage of cross-entropy loss instead of square loss, and the relatively small batch-size for training all contribute to LTHNet's success.

CCS CONCEPTS

• **Information systems → Image search**; • **Computing methodologies → Image representations**.

KEYWORDS

learning to hash, long-tail datasets, memory network, large-scale multimedia retrieval

ACM Reference Format:

. 2021. Long-Tail Hashing. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, July 11–15, 2021, Montréal, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Hashing, in the context of information retrieval, refers to a special embedding technique that aims to encode data samples into binary codes (i.e., hash codes) [68, 81]. Since binary codes could be economically stored and also quickly computed, hashing has witnessed wide applications in large-scale retrieval systems for images etc. Generally speaking, there are two kinds of hashing methods: data-independent and data-dependent. The most representative method in the former category is Locality Sensitive Hashing (LSH) [18] which simply utilizes random projections as the hash functions. The latter category is also known as *learning to hash*, because such methods learn the hash functions from a set of training samples first and then use the obtained hash functions to predict the binary codes for the test (query) samples [3, 16, 24–26, 47, 52, 53, 55, 62, 64, 66, 69, 70, 75, 77, 80, 81, 83, 85].

Specifically, learning to hash can be conducted in two different ways: unsupervised or supervised. Unsupervised learning to hash methods such as ITQ [19], DGH [49], DeepBit [42], SSDH [76], and KNNH [29] rely on the training samples themselves only. In contrast, *supervised* learning to hash methods such as FastHash [41], SDH [61], DPSH [39], HashNet [2], DSDH [37], FSSH [55], and SCDH [6] make use of not only the training samples but also their semantic labels, therefore they usually outperform unsupervised methods.

However, existing learning to hash methods, whether unsupervised or supervised, are mostly trained and tested on artificially balanced datasets which are not commonly available in real-world scenarios. Many recent studies [31, 51, 84, 86] indicate that real-life image datasets have skewed distributions with a *long tail*, i.e., a few dominant classes (a.k.a. *head classes*) account for most examples while the other classes (a.k.a. *tail classes*) each contain only a few examples. Current hashing methods may not work well on such long-tail datasets, especially for those data-poor classes. The absence of techniques for learning to hash from realistic long-tail datasets is the motivation of this work.

To address the above challenging problem, we propose a novel learning to hash method named *Long-Tail Hashing Network* (LTHNet) that marries up the *deep learning* approach to hashing and the mechanism of *visual memory*. In recent years, deep neural networks have proved to be superb at learning from large-scale datasets. Particularly, deep hashing methods like DPSH [39], HashNet [2], DSDH [37], and CSQ [78] have shown great advantages over classic, non-deep hashing methods (a.k.a. shallow hashing methods). In order to handle the long-tail distribution, we equip our deep hashing model with an extended *dynamic meta-embedding* module which adaptively combines direct features and memory features [51]. The adaptation and utilization of dynamic meta-embedding enrich the semantic representations of images and thus facilitate the knowledge transfer from data-rich head classes to data-poor tail classes.

Our proposed LTHNet method consists of two stages: the first is to learn relaxed embeddings of the given dataset with its long-tail characteristic taken into account via an end-to-end neural network; the second is to binarize those obtained embeddings. Extensive experiments have been conducted to demonstrate that LTHNet can achieve dramatic performance improvements over all state-of-the-art competitors on long-tail datasets, with no or little sacrifice on

SIGIR '21, July 11–15, 2021, Montréal, Canada

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

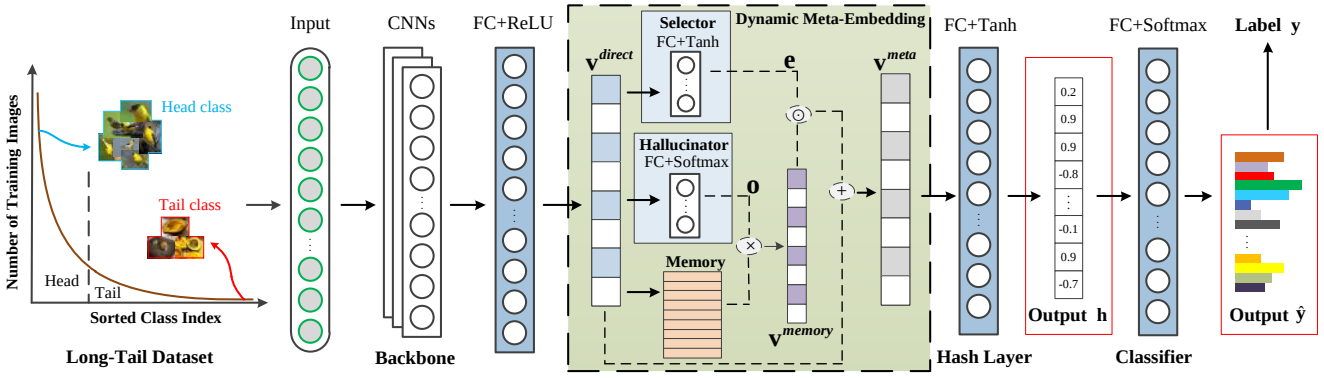


Figure 1: The architecture of Long-Tail Hashing Network (LTHNet).

balanced datasets. Furthermore, the parameter sensitivity analyses and ablation studies lead to new insights that the problem of long-tail hashing cannot be solved simply by reweighting different classes in the loss function, but the combination of extended dynamic meta-embedding, cross-entropy loss, and small batch-size will do the trick.

As far as we know, this is the first work to learn from long-tail datasets for hashing. Since most real-world datasets exhibit long-tail characteristics, LTHNet is likely to be effective for a wide range of applications, though we focus on image retrieval in this paper.

2 RELATED WORK

Obviously, LTHNet is related to the existing work in both learning to hash and learning from long-tail data.

2.1 Learning to Hash

The problem of learning to hash is to obtain binary codes for out-of-sample queries by learning the hash functions from training samples. Here, we focus on supervised learning to hash methods because they usually perform much better than unsupervised ones. Similar to the case in *learning to rank* [1, 48], such supervised learning to hash methods could be further divided into three groups: pointwise, pairwise, and listwise.

Pointwise methods formulate learning to hash as a classification problem, i.e., to train a classifier based on ground-truth labeled data and use it to predict each sample's class label. The representative methods include SDH [61], FSDH [22], R2SDH [21], HC-SDH [34], DSDH [37, 38], and SDMH [54]. All those methods could be viewed as different variants of SDH each of which has its particular emphasis. For example, FSDH and R2SDH try on enhancing the efficiency and the robustness of SDH respectively; DSDH aims to boost the performance by utilizing deep neural networks; and SDMH is tailored for multimedia search.

Pairwise methods formulate learning to hash as a regression problem, i.e., aligning the learned binary codes' pairwise similarities with those derived from the class labels. The typical examples in this group are KSH [50], LFH [83], COSDISH [33], SCDH [6], EDMH [7], NRDH [77], DPSH [39], HashNet [2] and CSQ [78]. Although those methods are based on essentially the same underlying idea, they have different focuses or strengths. For example, the first five methods listed above are shallow hashing methods that can be trained with higher efficiency, while the rest are deep hashing methods that are likely to achieve higher effectiveness; COSDISH and SCDH benefit from their specifically designed algorithms for fast discrete optimization; CSQ produces noticeable improvements by pulling the similar samples together and pushing the dissimilar ones apart; and EDMH generalizes the technique to cross-modal retrieval.

Listwise methods are devised to maximize the consistency between the ground-truth relevance list and the calculated ranking positions for any given query. Among them, RPH [71] directly optimizes the nDCG measure to obtain effective hashing codes with high ranking quality; RSH [67], DTSH [73] and TDH [11] all convert the ranking list to a set of triplets and then learn the hash functions from those triplets.

In some sense, our proposed LTHNet approach is developed on top of the popular pointwise SDH [61] framework.

2.2 Learning from Long-Tail Data

The phenomenon of long-tail distributions is ubiquitous in IR [8, 13, 16, 17, 46, 79, 82]. Specifically, for learning from datasets with a skewed, long-tail distribution of class labels, several strategies have been proposed in previous studies.

Data resampling tries to reshape the original imbalanced dataset to enforce a uniform distribution of class labels. It could be done by either *over-sampling*, i.e., duplicating some samples in the tail classes [4, 23, 27], or *under-sampling*, i.e., discarding some samples in the head classes [32, 40]. Although resampling has been shown to be helpful when the dataset is imbalanced, it also brings some risks: duplicating too many samples could cause overfitting for the tail classes [4] while discarding too many samples might lead to underfitting for the head classes [32].

Class reweighting puts different importance weights on different classes in the loss function for learning. Specifically, we would give large weights to tail classes and small weights to head classes, in order to mitigate the undesirable influences of class size. Lin et al. [43, 44] generalized the cross-entropy loss function to accommodate weighted training samples. Cui et al. [9] replace the raw number of samples in a class with the *effective number* which can be regarded as a form of reweighting. In principle, such reweighting

methods are essentially equivalent to the aforementioned resampling methods, but usually they are more computationally efficient.

Knowledge transfer is based on the idea that the hidden knowledge could be shared across different classes and be leveraged to enrich data representations via meta learning or attention mechanisms. Wang et al. [74] and Cui et al. [10] deal with class imbalance by transferring the knowledge learned from major classes to minor classes. Liu et al. [51] devised a dynamic meta-embedding module which combines direct image features with corresponding memory features to enrich both head and tail samples' representations. In brief, this kind of methods are targeted at enriching the data representation rather than reshaping the data distribution for downstream tasks.

Other strategies beyond the above end-to-end learning paradigm have emerged recently. A couple of latest papers [31, 86] reveal that it could be advantageous to decouple representation learning and classification into two separate stages when dealing with imbalanced datasets. In addition, an ensemble approach, RIDE [72], trains diverse distribution-aware experts and routes an instance to additional experts when necessary for long-tail recognition.

In this paper, we mainly explore the potentials of class reweighting and knowledge transfer for learning to hash on long-tail datasets.

3 PROBLEM STATEMENT

Given a set of samples (e.g. images) $\mathfrak{X} = \{(\mathbf{x}_n, l_n)\}_{n=1}^N$, $\mathbf{x}_n \in \mathbb{R}^d$ denotes the d-dimensional feature vector for the n-th sample and $l_n \in \{1, 2, \cdots, C\}$ corresponds to its class label index, where N is the number of samples and C is the number of classes in $\mathfrak{X}$. Besides, let s_i represent the number of samples in the i-th class ($i = 1, 2, \cdots, C$). Without loss of generality, we assume that $s_1 \geq s_2 \geq \cdots \geq s_C$. Then the concept of long-tail datasets and long-tail hashing can be formally defined as follows.

Definition 1 (**Long-Tail Dataset**). *A dataset $\mathfrak{X}$ is called a long-tail dataset if the sizes of its sorted classes follow the Zipf's law [56], i.e.,*

$$s_i = s_1 \times i^{-\mu}, \tag{1}$$

*where μ is a parameter controlling the degree of data imbalance that is measured by the **imbalance factor** (**IF** for short) s_1/s_C.*

Definition 2 (**Long-Tail Hashing**). *Given a long-tail dataset $\mathfrak{X} = \{(\mathbf{x}_n, l_n)\}_{n=1}^N$, the problem of long-tail hashing is to learn a set of hash functions $\{\mathbf{h}_j(\cdot)\}_{j=1}^q$ based on it so that*

$$\mathbf{H}(\mathbf{x}_n) \stackrel{\Delta}{=} [\mathbf{h}_1(\mathbf{x}_n), \cdots, \mathbf{h}_q(\mathbf{x}_n)]^T = \mathbf{b}_n, \tag{2}$$

where $\mathbf{b}_n \in \{-1, +1\}^q$ denotes the hash code for the n-th sample and q is the code length.

For any data sample $\mathbf{x}$, its q-bit hash code $\mathbf{b}$ can be calculated as $\mathbf{H}(\mathbf{x})$ with the learned mapping $\mathbf{H}$ that consists of q hash functions each corresponding to a specific bit.

4 THE PROPOSED METHOD

In this section, we elaborate on our proposed LTHNet, a novel deep hashing method that is designed to learn a set of hash functions $\mathbf{H}$ effectively from long-tail datasets. Fig. 1 illustrates the architecture of LTHNet which contains four key components: (1) direct feature learning $\mathbf{v}^{direct}$, (2) extended dynamic meta-embedding $\mathbf{v}^{meta}$, (3) the hash layer $\mathbf{h}$, and (4) the classifier $\hat{\mathbf{y}}$. Although our objective here is not really to perform classification, a classifier is included to enable *supervised* learning of hash functions from labeled datasets. Table 1 describes the detailed configurations of LTHNet.

Table 1: LTHNet configurations.

Layer	Configuration	
0: Input	Image (e.g., $\mathbf{x}$)	
1: Backbone	ResNet34 (pre-trained)	
2: FC+ReLU	512×2000; **ReLU(·)**	
3: Extended DME	FC+Tanh	2000×2000; **Tanh(·)**
	FC+Softmax	2000×$(k+1)C$; **Softmax(·)**
	Memory	$(k+1)C$×2000
4: Hash Layer	2000×q; **Tanh(·)**	
5: Classifier	$q \times C$; **Softmax(·)**	
6: Output	$[\mathbf{v}^{direct}, \mathbf{h}, \hat{\mathbf{y}}]$	

4.1 Direct Feature Learning

Deep *convolutional neural networks* (CNNs) have achieved great success in feature learning (representation learning) for images and further facilitated a large number of downstream tasks [28, 36, 45, 63]. In this paper, we choose the popular ResNet34[1] pre-trained on ImageNet [12] as the backbone of our LTHNet (Layer#1 in Table 1), which is followed by "FC+ReLU", a fully connected layer of neurons with the ReLU activation function (Layer#2 in Table 1). From Layer#0 to Layer#2, the direct feature $\mathbf{v}^{direct}$ would be learned. The reason why one more FC layer is built on top of the direct output of ResNet34 is to allow for experimental comparisons between the "deep hashing" methods and those "deep features + shallow hashing" methods. The latter refers to supplying the 512-dimensional direct features output of the pre-trained ResNet34 to a traditional shallow hashing model such as SDH [61], FSSH [55], and SCDH [6]. Since those (kernel-based) shallow hashing models usually utilize 2000 anchors to achieve a good trade-off between competitive performance and fast speed, we make use of 512×2000 FC (Layer#2) for a fair comparison.

4.2 Extended Dynamic Meta-Embedding

For head classes, there are abundant samples for embedding via CNNs, but that is not the case for tail classes. To augment the direct feature $\mathbf{v}^{direct}$ especially for tail classes, we extend the idea of *dynamic meta-embedding* (DME) that was originally developed for pattern recognition [51] and apply it to hashing. Specifically, it merges direct features with memory features [60], which would enable the transfer of semantic knowledge between data-rich and data-poor classes. As for the visual memory, it could simply be represented as a set of class centroids $\mathcal{M} = \{\mathbf{m}_i\}_{i=1}^C$, which in fact summarizes the visual concept for each class of images in the training dataset [30]. Let $\mathbf{m}_i$ denote the centroid of the i-th class's

[1] https://pytorch.org/docs/stable/torchvision/models.html

samples:

$$\mathbf{m}_i = \frac{\sum_{n=1}^{N} \mathbf{v}_n^{direct} \mathbf{1}\{l_n = i\}}{\sum_{n=1}^{N} \mathbf{1}\{l_n = i\}}, \tag{3}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function, $\mathbf{v}_n^{direct}$ is the n-th sample's direct feature, and $i = 1, 2, \cdots, C$.

Moreover, deviating from the original DME, we argue that it is often insufficient for a single prototype to represent a category, especially for the tail classes [87]. Therefore, we use the *determinantal point process* (DPP)[2] [5, 15] to find k more diverse samples similar to the centroid of each class to further enrich the memory:

$$\mathcal{M}_i = \{\mathbf{m}_i\} \cup \mathbf{DPP}_k(i), \tag{4}$$

where $\mathbf{DPP}_k(i)$ is a function that returns a set of k samples for the i-th class as its summarizing prototypes. Since k should be smaller than the minimum size of classes, we set it to 3 by default, which would not incur much additional cost of storage or computation. Our experiments will show that it is indeed beneficial to employ multiple prototypes than a single one for each class in the long-tail setting (see Section 6.6).

To facilitate visual knowledge transfer from data-rich to data-poor classes, the memory feature is designed as:

$$\mathbf{v}^{memory} = \mathbf{o}^T \mathbf{M} = \sum\nolimits_{j=1}^{(k+1)C} o_j \mathbf{m}_j, \tag{5}$$

where $\mathbf{o} \in \mathbb{R}^{(k+1)C}$ could be viewed as the *attention* [65] over class prototypes and its extensions *hallucinated* from the direct feature. Concretely, we use "FC+Softmax" to obtain the coefficients from $\mathbf{v}^{direct}$, i.e., $\mathbf{o} = \mathbf{Softmax}(\mathbf{FC}(\mathbf{v}^{direct}))$.

Since the memory feature would have different impacts upon different classes (to wit, the memory feature is more important for the data-poor tail classes than for the data-rich head classes in terms of feature enrichment), we introduce an adaptive selector (see Fig. 1). Thus, the final output embedding $\mathbf{v}^{meta}$, which combines the direct feature and the memory-related feature, is written as:

$$\mathbf{v}^{meta} = \mathbf{v}^{direct} + \mathbf{e} \odot \mathbf{v}^{memory}, \tag{6}$$

where $\mathbf{e}$ acts as the adaptive selector of concepts and $\odot$ denotes the Hadamard product. Specifically, we utilize the "FC+Tanh" to derive the selector weights from $\mathbf{v}^{direct}$, i.e., $\mathbf{e} = \mathbf{Tanh}(\mathbf{FC}(\mathbf{v}^{direct}))$.

4.3 Hash Layer

After Layer#3, each sample's embedding would have been semantically enriched. Then, a hash layer (Layer#4) is further appended for the generation of binary codes:

$$\mathbf{h}^{true} = \mathbf{sgn}(\mathbf{FC}(\mathbf{v}^{meta})), \tag{7}$$

where $\mathbf{sgn}(\cdot)$ is the element-wise sign function, i.e., it outputs +1 when the input is non-negative and −1 otherwise. Hence, $\mathbf{h}^{true} \in \{-1, +1\}^q$ represents the hash code for the input sample $\mathbf{x}$.

It is worth mentioning that $\mathbf{sgn}(\cdot)$ is discontinuous and thus not differentiable at 0, and worst of all, for all other input values its gradient would be just zero. Thus $\mathbf{sgn}(\cdot)$ poses an obstacle to the *back-propagation* training of neural network [57]. To overcome this problem, we adopt a two-stage strategy: first, the direct "hard" hash mapping Eq. (7) is relaxed into:

$$\mathbf{h} = \mathbf{Tanh}(\mathbf{FC}(\mathbf{v}^{meta})), \tag{8}$$

whose output will consist of real values between −1 and +1, as illustrated in Fig. 1; second, after the end-to-end learning from the long-tail training dataset, the real-valued output vector $\mathbf{h}$ is binarized with:

$$\mathbf{b} = \mathbf{sgn}(\mathbf{h}), \tag{9}$$

and $\mathbf{b} \in \{-1, +1\}^q$ is the final hash code for the input image sample. Although it has been found in previous studies that such a real relaxation of binary constraints might lead to large quantization errors [6, 33, 49, 55, 61], it is the simplest way to train a deep neural network for binary outputs without introducing extra intermediate variables and complex optimization techniques [37]. More importantly, this simple two-stage strategy works well in practice delivering significant performance gains on both traditional balanced datasets and realistic long-tail datasets.

Algorithm 1: Long-Tail Hashing Network (LTHNet)

```
/* A deep neural network for learning to hash
   from long-tail datasets                          */
Input: the training dataset 𝔛 = {(x_n, l_n)}_{n=1}^N, the number
  of classes C, the maximum number of epochs MaxEpoch,
  and the hyperparameter β and k;
Initialize LTHNet parameters θ;
while not MaxEpoch do
    /* Memory: update M = {m_j}_{j=1}^{(k+1)C}      */
    for n = 1 to N do
        [v_n^direct,~,~] = LTHNet(x_n; M, θ);
    end
    Compute the centroid for each class via Eq. (3) and
      retrieve k-more diverse and similar samples for each
      centroid via Eq. (4), and the memory is updated as
      M = {m_j}_{j=1}^{(k+1)C};
    /* LTHNet training: update θ                    */
    for x in Dataloader(𝔛) do
        [~,~,ŷ] = LTHNet(x; M, θ);
        L_CB(ŷ, y);
        θ = RMSprop(L_CB, θ);
    end
end
  /* Out-of-samples (x_oos) Hashing                 */
[~,h_oos,~] = LTHNet(x_oos; M, θ);
b_oos = sgn(h_oos);
```

4.4 Classifier

Intuitively, better hash codes should lead to more accurate classifications. Therefore, a classifier (Layer#5) is introduced at the end of LTHNet so as to carry out supervised learning:

$$\hat{\mathbf{y}} = \mathbf{Softmax}(\mathbf{FC}(\mathbf{h})), \tag{10}$$

[2] https://github.com/laming-chen/fast-map-dpp

Table 2: Statistics of long-tail benchmarks with various IFs.

Cifar100	μ	$n_{\max}$	$n_{\min}$	n_{db}	n_{query}	n_{train}
IF=1	0.000	500	500	50k	10,000	50,000
IF=50	0.830	500	10	50k	10,000	3,732
IF=100	0.990	500	5	50k	10,000	2,598
ImageNet100	μ	$n_{\max}$	$n_{\min}$	n_{db}	n_{query}	n_{train}
IF=1	0.000	100	100	130k	5,000	10,000
IF=50	0.845	1300	26	130k	5,000	9,437
IF=100	0.990	1300	13	130k	5,000	6,834

where $\hat{\mathbf{y}}$ is the predicted probability distribution over class labels. Finally, the input sample is going to be categorized into the class of the highest probability.

Bringing all the above pieces together, our designed LTHNet can be summed up as:

$$[\mathbf{v}^{direct}, \mathbf{h}, \hat{\mathbf{y}}] = \mathbf{LTHNet}(\mathbf{x}; \boldsymbol{\mathcal{M}}, \boldsymbol{\theta}), \tag{11}$$

where $\mathbf{v}^{direct}$, $\mathbf{h}$ and $\hat{\mathbf{y}}$ represent the input sample $\mathbf{x}$'s direct feature, relaxed hash code and predicted class distribution respectively, and $\boldsymbol{\theta}$ denotes the model parameters. Among the LTHNet outputs, $\mathbf{v}^{direct}$ is used for updating the visual memory $\boldsymbol{\mathcal{M}}$, $\mathbf{h}$ is the generation of binary codes, and $\hat{\mathbf{y}}$ serves the purpose of supervise learning. Given the probabilistic prediction $\hat{\mathbf{y}}$ and the corresponding ground-truth one-hot vector $\mathbf{y}$, the commonly used *cross-entropy* loss function for classification is:

$$\mathbf{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i=1}^{C} y_i \mathbf{log}(\hat{y}_i). \tag{12}$$

In order to deal with the severe class imbalance in long-tail datasets, we generalize Eq. (12) to a class-weighted version [9]:

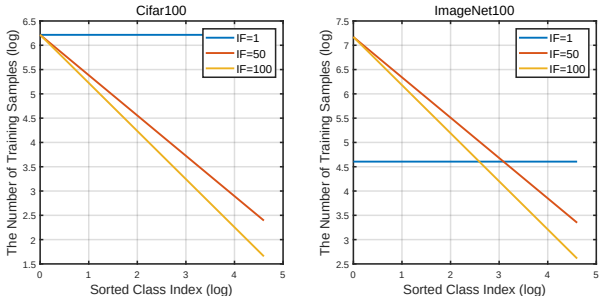
$$\mathbf{L}_{CB}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{E_{n_{\mathbf{y}}}} \mathbf{L}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1-\beta}{1-\beta^{n_{\mathbf{y}}}} \mathbf{L}(\hat{\mathbf{y}}, \mathbf{y}), \tag{13}$$

where $E_{n_{\mathbf{y}}} = \frac{1-\beta^{n_{\mathbf{y}}}}{1-\beta}$ is the *effective number* of samples in class $\mathbf{y}$, which is calculated using the actual number of samples($n_{\mathbf{y}}$) and a hyperparameter $\beta \in [0, 1)$. Note that $\beta = 0$ means no reweighting, i.e., backing off to Eq. (12), while $\beta \to 1$ indicates reweighting each class by the reciprocal of its actual size.

5 LEARNING ALGORITHM

The parameters of the LTHNet model to be learned in the training stage include $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\theta}$. For the update of parameter $\boldsymbol{\mathcal{M}}$ in each iteration, we carry over all the training samples, compute the class centroids via Eq. (3), and then retrieve k-more diverse and similar samples via Eq. (4) which constitute the renewed memory. With respect to update of parameters $\boldsymbol{\theta}$ in each iteration, we sample a *mini-batch* of images from the training set, and then perform *back-propagation* using the gradients calculated on these sampled images. Algorithm 1 describes the complete learning procedure which iteratively updates $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\theta}$ until they have converged or the number of epochs has reached the preset maximum.

Given any new test (query) image $\mathbf{x}_{oos}$, we can use the learned LTHNet model to compute the real-valued hash vector $\mathbf{h}_{oos}$ first, and then binarize it into the final hash code $\mathbf{b}_{oos}$.



(a) Cifar100 (b) ImageNet100

Figure 2: The log-log plots of curated datasets with various IFs.

6 EXPERIMENTS

We have conducted extensive experiments to evaluate LTHNet against several state-of-the-art hashing methods on both balanced and long-tail benchmarks. All the datasets and source codes for our experiments will be made available to the public on Github.

6.1 Datasets

We have curated 2 balanced and 4 long-tail benchmarks based on two public datasets Cifar100[3] and ImageNet100[4].

Cifar100 [35] includes a 100-class database with 500 images per class as well as a 100-class query set with 100 images per class. To construct a variety of long-tail training sets, we randomly choose images from the database with $s_1 = 500$ and $\mu = 0, 0.83, 0.99$, thus the sizes of the generated classes would meet the Zipf's law (Eq. (1)). In the end, we have 3 benchmarks with different IFs (1, 50 and 100).

ImageNet100 [2] is a 100-class subset randomly selected from the original 1000-class single-labeled ImageNet benchmark [12, 58]. It contains a database with 1300 images per class and also a query set with 50 images per class. First, we randomly choose 100 images per class from the database and then construct a balanced training set (i.e., IF = 1). Note that here we do not use all the 1300 images available for each of the 100 classes because that would make our experiments take too long to finish while using just 100 images per class would actually be enough to get satisfactory results. Second, following the Zipf's law (Eq. (1)), we randomly choose images from the database with $s_1 = 1300$ and $\mu = 0.845, 0.99$ to finally obtain two long-tail training sets (IF = 50 and 100).

Overall, 6 benchmarks are used for our experiments, as shown in Table 2. Each benchmark comprises a database, a query set, and a training set. Note that $n_{\max} = s_1$, $n_{\min} = s(100)$, and n_{db}, n_{query}, n_{train} correspond to the sizes of the database, the query set, and the training set, respectively. Fig. 2 visualizes the class label distributions of the above benchmarks.

For each benchmark, we train different hashing models on the training set, and then employ them to compute the hash codes for the images in the database as well as the query set. Given a query

[3] https://www.cs.toronto.edu/~kriz/cifar.html
[4] http://www.image-net.org/

image, a result image returned from hash code based search of the database is deemed to be relevant if they share the same label.

6.2 Competitors and Metrics

To testify LTHNet's effectiveness, we compare it with the following competitive models: LSH[5] [18], PCAH[6] [19], ITQ[6] [20], KNNH[7] [29], SDH[8] [61], COSDISH[9] [33], FastHash[10] [41], FSSH[11] [55], SCDH[12] [6], DPSH[13] [39], HashNet[14] [2], DSDH[15] [37], and CSQ[16] [78]. The first 9 competitors are shallow hashing methods, while the rest are all deep hashing methods.

Regarding the performance measure, we adopt Mean Average Precision (MAP). Although some IR researchers are against the usage of MAP[14], there exist different opinions in the IR community[59]. More importantly, MAP has been used as the single or major retrieval performance measure in almost all the *learning to hash* literature [19, 29, 33, 37, 41, 47, 49, 55, 61, 66], so we follow the convention to make our experimental results comparable with the others'. Formally, we have $MAP = \frac{1}{Q}\sum_{i=1}^{Q} AP(q_i)$, where Q denotes the number of query samples, q_i represents the i-th query sample and $AP(q_i) = \frac{1}{l_{q_i}}\sum_{r=1}^{R} p_{q_i}(r)\delta_{q_i}(r)$, where l_{q_i} is the number of ground-truth neighbors of the query sample q_i, R is the total number of data items, $p_{q_i}(r)$ denotes the precision at cutoff r for the ranking list, $\delta_{q_i}(r)$ indicates whether the r-th data item is relevant to the query sample q_i.

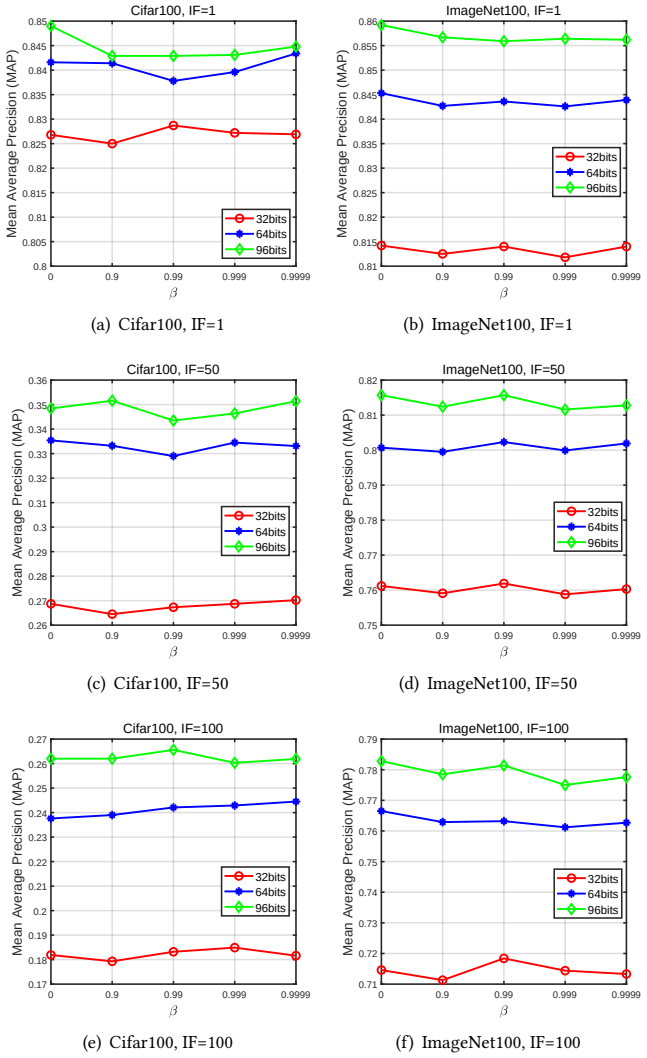
6.3 Settings

To ensure a fair comparison, we take the output of the ResNet34 model (pre-trained on ImageNet) — 512-dimensional feature vectors — as the input to shallow hashing methods, and use the original images directly as the input to deep hashing methods.

All the selected baseline hashing methods would have their respective hyperparameters properly tuned on the training set for the most competitive results (MAP scores), following the suggested protocols in the corresponding original papers. As explained earlier, the number of anchors is all set to 2000 for those kernel-based shallow competitors such as SDH [61], FSSH [55] and SCDH [6].

All the deep hashing methods in comparison, including LTHNet, would use exactly the same pre-trained ResNet34 as the backbone. We use the mini-batch RMSprop algorithm with the learning rate 1e−5 and weight decay 5e−4 to update the network parameters $\boldsymbol{\theta}$. Again, those hyperparameters are tuned for the most competitive results. Besides, the cosine annealing strategy implemented in PyTorch is adopted to adjust the learning rate within each epoch. The hyperparameters are set as follows: $\beta = 0$, $MaxEpoch = 100$, $batchsize = 8$ for Cifar100 and $batchsize = 16$ for ImageNet100.

[5] http://www.cad.zju.edu.cn/home/dengcai/Data/DSH.html
[6] https://github.com/willard-yuan/hashing-baseline-for-image-retrieval
[7] https://github.com/HolmesShuan/K-Nearest-Neighbors-Hashing
[8] https://github.com/bd622/DiscretHashing
[9] http://cs.nju.edu.cn/lwj/learningtohash.html
[10] https://bitbucket.org/chhshen/fasthash
[11] https://lcbwlx.wixsite.com/fssh
[12] https://github.com/keneeth/scdh
[13] https://github.com/jiangqy/DPSH-pytorch
[14] https://github.com/zhjy2016/HashNet
[15] https://github.com/TreezzZ/DSDH_PyTorch
[16] https://github.com/yuanli2333/Hadamard-Matrix-for-hashing



(a) Cifar100, IF=1 (b) ImageNet100, IF=1

(c) Cifar100, IF=50 (d) ImageNet100, IF=50

(e) Cifar100, IF=100 (f) ImageNet100, IF=100

Figure 3: The retrieval performance (w.r.t. MAP) of our LTHNet (k=3) with different β values, IFs and hash code lengths. Note that LTHNet (k=0) exhibits similar results.

More parameter setting details can be found in the parameter sensitivity analysis (Section 6.5) and also our released source code for experiments.

6.4 Results

Tables 3 and 4 show all the hashing methods' MAP scores on Cifar100 and ImageNet100 respectively, with different IFs and hash code lengths, where the best scores are in **boldface** and the second best underlined. Note that in addition to the standard LTHNet model, these tables also contain the results of LTHNet$_{sq}$, a modified version of LTHNet (with a different loss function), which will be explained later in the ablation study (Section 6.6).

Table 3: MAP scores of all methods on the Cifar100 dataset with various IFs (imbalance factors) and hash code lengths.

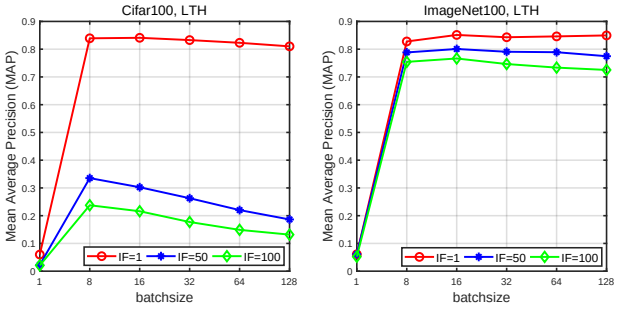
Settings /MAP/ Methods	IF=1			IF=50			IF=100		
	32 bits	64 bits	96 bits	32 bits	64 bits	96 bits	32 bits	64 bits	96 bits
LSH	0.0333	0.0458	0.0608	0.0333	0.0467	0.0615	0.0307	0.0480	0.0585
PCAH	0.0569	0.0612	0.0605	0.0532	0.0617	0.0627	0.0519	0.0608	0.0625
ITQ	0.0806	0.0976	0.1026	0.0709	0.0858	0.0903	0.0677	0.0824	0.0896
KNNH	0.0844	0.1012	0.1088	0.0703	0.0840	0.0906	0.0689	0.0810	0.0871
SDH	0.1950	0.2472	0.2739	0.1115	0.1363	0.1460	0.1006	0.1182	0.1258
COSDISH	0.1262	0.1921	0.2143	0.0695	0.0875	0.1000	0.0583	0.0724	0.0809
FastHash	0.2239	0.3161	0.3636	0.0787	0.1061	0.1211	0.0714	0.0903	0.1001
FSSH	0.1849	0.2207	0.2671	0.1101	0.1384	0.1512	0.0957	0.1146	0.1274
SCDH	0.2415	0.3003	0.3316	0.1282	0.1536	0.1661	0.1138	0.1335	0.1415
DPSH	0.3113	0.4506	0.4957	0.1069	0.1407	0.1634	0.0978	0.1216	0.1383
HashNet	0.4380	0.5719	0.6311	0.1726	0.1950	0.2079	0.1444	0.1559	0.1631
DSDH	0.5398	0.6100	0.6407	0.1119	0.1000	0.0999	0.0940	0.0872	0.0807
CSQ	0.7711	0.7984	0.7821	0.2221	0.2745	0.2669	0.1716	0.1992	0.1658
LTHNet$_{sq}$ (k=0)	0.8191	0.8321	0.8362	0.2220	0.2144	0.2330	0.1624	0.1546	0.1508
LTHNet$_{sq}$ (k=3)	0.8232	0.8360	0.8390	0.2432	0.2794	0.3116	0.1750	0.2079	0.2264
LTHNet (k=0)	0.8195	0.8336	0.8400	0.2427	0.3028	0.3309	0.1752	0.2240	0.2415
LTHNet (k=3)	**0.8268**	**0.8416**	**0.8490**	**0.2687**	**0.3354**	**0.3484**	**0.1819**	**0.2376**	**0.2620**

Table 4: MAP scores of all methods on the ImageNet100 dataset with various IFs (imbalance factors) and hash code lengths.

Settings /MAP/ Methods	IF=1			IF=50			IF=100		
	32 bits	64 bits	96 bits	32 bits	64 bits	96 bits	32 bits	64 bits	96 bits
LSH	0.0613	0.1066	0.1557	0.0606	0.1121	0.1475	0.0556	0.1097	0.1510
PCAH	0.1478	0.2004	0.2042	0.1306	0.1817	0.1919	0.1280	0.1788	0.1947
ITQ	0.1965	0.2687	0.2907	0.1803	0.2458	0.2731	0.1719	0.2371	0.2667
KNNH	0.1996	0.2778	0.3007	0.1830	0.2537	0.2798	0.1766	0.2411	0.2666
SDH	0.4416	0.5108	0.5385	0.3553	0.4188	0.4414	0.3126	0.3733	0.3975
COSDISH	0.2875	0.4040	0.4559	0.2072	0.2900	0.3320	0.1763	0.2395	0.2731
FastHash	0.3178	0.4295	0.4744	0.2462	0.3274	0.3741	0.1932	0.2703	0.3100
FSSH	0.4746	0.5184	0.5528	0.3681	0.4533	0.4702	0.3312	0.4017	0.4314
SCDH	0.4894	0.5598	0.5854	0.3937	0.4726	0.4954	0.3601	0.4194	0.4422
DPSH	0.4887	0.6055	0.6514	0.2186	0.3125	0.3791	0.1788	0.2832	0.3468
HashNet	0.4410	0.6006	0.6421	0.3465	0.4034	0.4240	0.3101	0.3770	0.3800
DSDH	0.6554	0.7015	0.7231	0.2568	0.2617	0.2744	0.1841	0.2134	0.2429
CSQ	**0.8507**	**0.8733**	**0.8657**	0.6629	0.7022	0.6823	0.5989	0.5620	0.5495
LTHNet$_{sq}$ (k=0)	0.7338	0.7713	0.8130	0.5523	0.5154	0.5530	0.3924	0.3490	0.4132
LTHNet$_{sq}$ (k=3)	0.7896	0.8259	0.8465	0.7133	0.7491	0.7753	0.6333	0.6587	0.6958
LTHNet (k=0)	0.7924	0.8267	0.8382	0.7369	0.7804	0.7920	0.6771	0.7350	0.7528
LTHNet (k=3)	0.8142	0.8453	0.8592	**0.7612**	**0.8007**	**0.8157**	**0.7146**	**0.7665**	**0.7828**

Looking at the experimental results on the traditional balanced datasets (i.e., IF=1), we can see that the deep hashing methods always outperform the shallow hashing competitors, especially for longer code lengths. This is not surprising, because modern deep neural networks are known to have stronger fitting and generalization abilities than classic shallow learning methods, when there are a massive amount of data for training. Most notably, our proposed LTHNet method performs better than all the competitors on Cifar100 while comes second only to the newly emerged CSQ method on ImageNet100, which validates its effectiveness in the traditional balanced setting.

Let us check the experimental results of the same hashing methods on the more realistic long-tail datasets (i.e., IF=50 and 100). It is clear that the more skewed the label distribution, the more challenging the learning to hash task is due to the scarcity of training samples for tail classes. Interestingly, on long-tail datasets, existing deep hashing methods such as DSDH and DPSH can hardly work: sometimes their performances are even inferior to that of the shallow method SCDH. By contrast, our proposed LTHNet method



(a) Cifar100 (b) ImageNet100

Figure 4: The retrieval performance (w.r.t. MAP) of our LTHNet (k=3) with different batchsizes (64 bits); and the results with other bit-settings hold the similar trends as above. Note that LTHNet (k=0) exhibits similar results.

can outperform all the competing hashing methods including the strongest contender CSQ. According to the paired t-test, the performance improvements made by LTHNet (k=3) over the best baseline CSQ are all *statistically significant* (p-value < 0.05). This confirms LTHNet's superior performance in the realistic long-tail setting.

Overall, LTHNet is on a par with state-of-the-art hashing techniques like CSQ on traditional balanced datasets, but it works significantly better than all of them on realistic long-tail datasets.

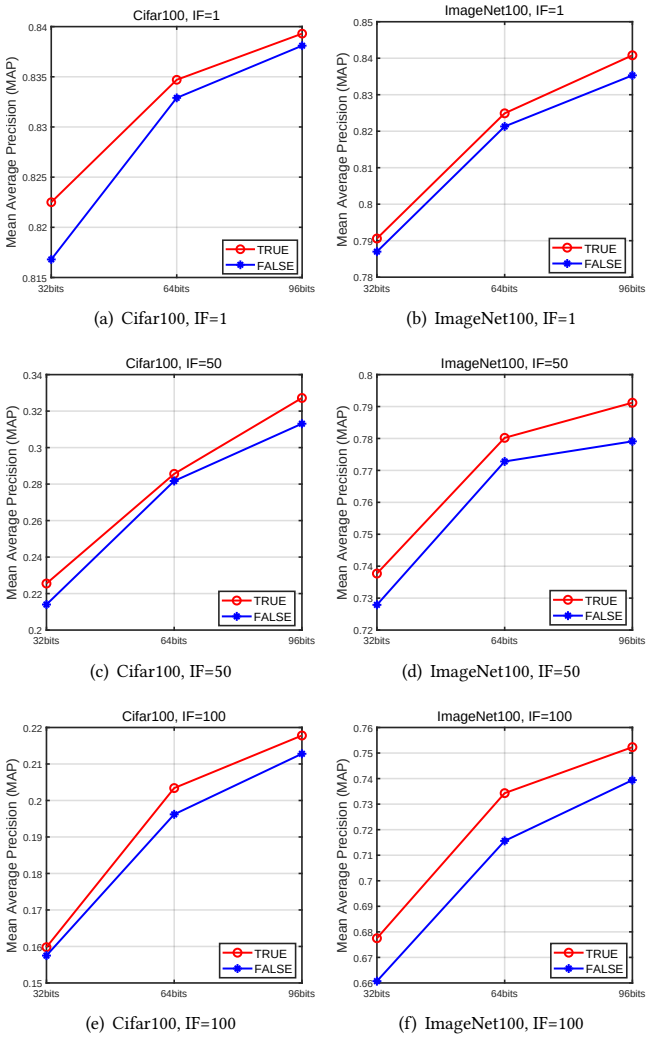
6.5 Parameter Sensitivity

Fig. 3 shows the retrieval performance (w.r.t. MAP scores) for different β values under various IFs and hash code lengths. To our surprise, from Fig. 3(a) to Fig. 3(f), the performance curves are mostly stable with only slight fluctuations: the changes of MAP scores across various β values are less than 0.006. This reveals that the intuitive idea of directly balancing different classes in the loss function, i.e., assigning more weights to samples from small classes and less weights to samples from big classes, does not really work for long-tail hashing. So we just set $\beta = 0$ in our LTHNet experiments.

Fig. 4 draws LTHNet's performances when it is trained using different batchsizes. On Cifar100, when *batchsize* equals 8, LTHNet performs the best; while on ImageNet100, the optimal *batchsize* is 16. Therefore, we set *batchsize* to 8 and 16 for Cifar100 and ImageNet100 benchmarks, respectively, in our experiments. Generally speaking, on balanced datasets, LTHNet works well with a wide range of batchsizes, but on long-tail datasets, LTHNet seems to require the batchsizes to be sufficiently small (e.g., 8 or 16) in order to yield good results. In addition, we have also tuned the *batchsize* of other deep hashing methods in comparison, and found the similar phenomenon that a relatively small *batchsize* such as 16/32/64 is likely to perform better than a large *batchsize* such as 128 on long-tail datasets. For more details, please refer to Fig. 6.

6.6 Ablation Study

Is the dynamic meta-embedding module (Fig. 1) indeed useful for long-tail hashing? To answer this question, we make a comparison



(a) Cifar100, IF=1 (b) ImageNet100, IF=1

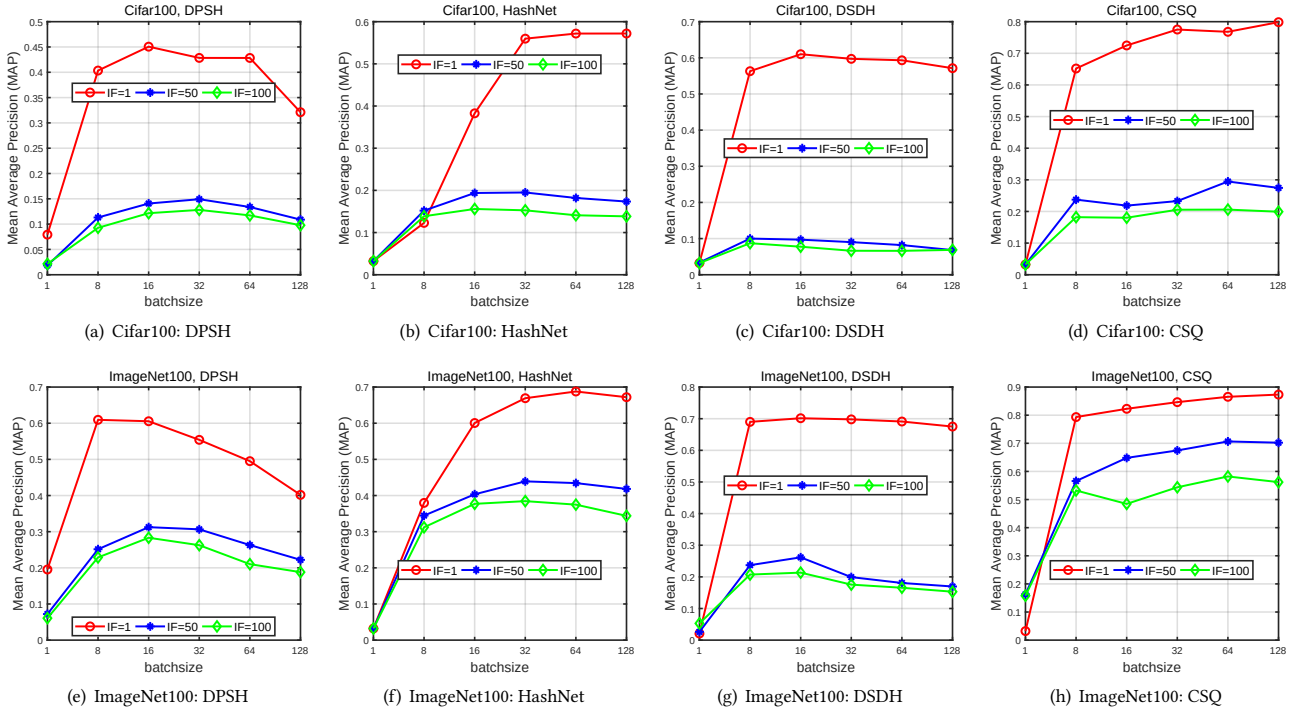
(c) Cifar100, IF=50 (d) ImageNet100, IF=50

(e) Cifar100, IF=100 (f) ImageNet100, IF=100

Figure 5: The retrieval performance (w.r.t. MAP) of our LTHNet (k=0) with and without the extended DME module (denoted as "TRUE" and "FALSE", respectively), under various settings.

between LTHNet (k=0) with and without the extended DME module. Fig. 5 plots their MAP scores under different settings. Obviously, the red performance curves (LTHNet with extended DME) are always above the blue performance curves (LTHNet without extended DME). Such consistent performance improvements under various settings verify the benefits of DME for learning to hash, especially on long-tail datasets. Besides, by comparing LTHNet (k=0) with LTHNet (k=3) (or LTHNet$_{sq}$ (k=0) v.s. LTHNet$_{sq}$ (k=3)) in Tables 3 and 4, we found that an enriched memory could further boost LTHNet's performance.

Furthermore, we wonder how the performance would change if the *cross-entropy loss* (a.k.a. log loss) in the LTHNet model is replaced with the *square loss* which is used by SDH [61] and DSDH [37].



(a) Cifar100: DPSH (b) Cifar100: HashNet (c) Cifar100: DSDH (d) Cifar100: CSQ

(e) ImageNet100: DPSH (f) ImageNet100: HashNet (g) ImageNet100: DSDH (h) ImageNet100: CSQ

Figure 6: The retrieval performance (w.r.t. MAP) of deep hashing models on Cifar100 and ImageNet100 with various batchsizes (64 bits). The results of other bit-settings hold the similar trends as above.

More concretely, we substitute

$$\hat{\mathbf{y}} = \mathbf{FC}(\mathbf{h}) \tag{14}$$

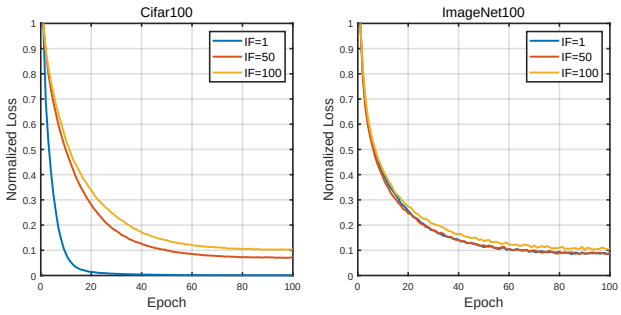
and

$$\mathbf{L}(\hat{\mathbf{y}}, \mathbf{y}) = ||\hat{\mathbf{y}} - \mathbf{y}||_2^2 \tag{15}$$

for Eq. (10) and Eq. (12) respectively. Let us use LTHNet$_{sq}$ to denote such a modified version of LTHNet, and show its experimental results along with those of the other hashing methods in Tables 3 and 4. Evidently, LTHNet$_{sq}$ is slightly inferior to the standard LTHNet when IF=1, but far behind LTHNet when IF=50 or 100, which indicates that cross-entropy loss is more suitable for LTHNet than square loss.

6.7 Convergence Analysis

Fig. 7 plots the normalized loss for each epoch on Cifar100 and ImageNet100 with various IFs. It is worth noting that to facilitate a fair comparison, the y-axis shows each LTHNet's loss normalized by the maximum loss among all its iterations. Clearly, all the curves go lower and lower from a large loss to a small loss until they become flat, which empirically corroborates the nice convergence properties of our LTHNet algorithm.



(a) Cifar100 (b) ImageNet100

Figure 7: Convergence curves of LTHNet (k=3) on Cifar100 and ImageNet100 with various IFs (64 bits); and the loss curves with other bit-settings hold the similar trends as above. Note that LTHNet (k=0) exhibits similar results.

7 CONCLUSION

In this paper, we put forward a novel two-stage deep hashing method named Long-Tail Hashing Network (LTHNet) for large-scale image retrieval. To our knowledge, this is the first work of its kind that addresses the problem of learning to hash on realistic long-tail datasets. A surprising finding is that the intuitive idea of directly reweighting different classes in the loss function actually does not work in this context. Nevertheless, the extended dynamic meta-embedding module, the usage of cross-entropy loss (instead of square loss), and the relatively small batch-size for training all help our proposed LTHNet achieve outstanding performances not only on traditional balanced datasets but, more importantly, also on realistic long-tail datasets.

REFERENCES

[1] Reda Alhajj and Jon G. Rokne. 2018. *Learning to Rank*. Encyclopedia of Social Network Analysis and Mining, 2nd Edition, Springer.

[2] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S. Yu. 2017. HashNet: Deep Learning to Hash by Continuation. In *ICCV*. 5609–5618.

[3] Suthee Chaidaroon, Travis Ebesu, and Yi Fang. 2018. Deep Semantic Text Hashing with Weak Supervision. In *SIGIR*. ACM, 1109–1112.

[4] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *JAIR* 16 (2002), 321–357.

[5] Laming Chen, Guoxin Zhang, and Eric Zhou. 2018. Fast Greedy MAP Inference for Determinantal Point Process to Improve Recommendation Diversity. In *NeurIPS*. 5627–5638.

[6] Yong Chen, Zhibao Tian, Hui Zhang, Jun Wang, and Dell Zhang. 2020. Strongly Constrained Discrete Hashing. *TIP* 29 (2020), 3596–3611.

[7] Yong Chen, Hui Zhang, Zhibao Tian, Jun Wang, Dell Zhang, and Xuelong Li. 2020. Enhanced Discrete Multi-modal Hashing: More Constraints yet Less Time to Learn. *TKDE* (2020), 1–13.

[8] Zhihong Chen, Rong Xiao, Chenliang Li, Gangfeng Ye, Haochuan Sun, and Hongbo Deng. 2020. ESAM: Discriminative Domain Adaptation with Non-Displayed Items to Improve Long-Tail Performance. In *SIGIR*. 579–588.

[9] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge J. Belongie. 2019. Class-Balanced Loss Based on Effective Number of Samples. In *CVPR*. 9268–9277.

[10] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge J. Belongie. 2018. Large Scale Fine-Grained Categorization and Domain-Specific Transfer Learning. In *CVPR*. 4109–4118.

[11] Cheng Deng, Zhaojia Chen, Xianglong Liu, Xinbo Gao, and Dacheng Tao. 2018. Triplet-Based Deep Hashing Network for Cross-Modal Retrieval. *TIP* 27, 8 (2018), 3893–3903.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*. 248–255.

[13] Doug Downey, Susan T. Dumais, and Eric Horvitz. 2007. Heads and Tails: Studies of Web Search with Common and Rare Queries. In *SIGIR*. 847–848.

[14] Norbert Fuhr. 2018. Some Common Mistakes in IR Evaluation, and How They Can Be Avoided. 51, 3 (2018), 32–41.

[15] Lu Gan, Diana Nurbakova, Léa Laporte, and Sylvie Calabretto. 2020. Enhancing Recommendation Diversity using Determinantal Point Processes on Knowledge Graphs. In *SIGIR*. 2001–2004.

[16] Ming Gao, Leihui Chen, Xiangnan He, and Aoying Zhou. 2018. BiNE: Bipartite Network Embedding. In *SIGIR*. 715–724.

[17] Darío Garigliotti, Dyaa Albakour, Miguel Martinez, and Krisztian Balog. 2019. Unsupervised Context Retrieval for Long-tail Entities. In *SIGIR*. 225–228.

[18] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *VLDB*. 518–529.

[19] Yunchao Gong and Svetlana Lazebnik. 2011. Iterative Quantization: A Procrustean Approach to Learning Binary Codes. In *CVPR*. 817–824.

[20] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval. *TPAMI* 35, 12 (2013), 2916–2929.

[21] Jie Gui and Ping Li. 2018. R2SDH: Robust Rotated Supervised Discrete Hashing. In *KDD*. 1485–1493.

[22] Jie Gui, Tongliang Liu, Zhenan Sun, Dacheng Tao, and Tieniu Tan. 2018. Fast Supervised Discrete Hashing. *TPAMI* 40, 2 (2018), 490–496.

[23] Hui Han, Wenyuan Wang, and Binghuan Mao. 2005. Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In *ICIC*. 878–887.

[24] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2019. Unsupervised Neural Generative Semantic Hashing. In *SIGIR*. 735–744.

[25] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2020. Content-aware Neural Hashing for Cold-start Recommendation. In *SIGIR*. 971–980.

[26] Casper Hansen, Christian Hansen, Jakob Grue Simonsen, Stephen Alstrup, and Christina Lioma. 2020. Unsupervised Semantic Hashing with Pairwise Reconstruction. In *SIGIR*. 2009–2012.

[27] Haibo He and Edwardo A. Garcia. 2009. Learning from Imbalanced Data. *TKDE* 21 (2009), 1263–1284.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[29] Xiangyu He, Peisong Wang, and Jian Cheng. 2019. K-Nearest Neighbors Hashing. In *CVPR*. 2839–2848.

[30] Yen-Chang Hsu, Zhaoyang Lv, and Zsolt Kira. 2018. Learning to Cluster in order to Transfer Across Domains and Tasks. In *ICLR*. 1–20.

[31] Bingyi Kang, Saining Xie, Marcus Rohrbach, Zhicheng Yan, Albert Gordo, Jiashi Feng, and Yannis Kalantidis. 2020. Decoupling Representation and Classifier for Long-Tailed Recognition. In *ICLR*. 1–16.

[32] Qi Kang, Xiaoshuang Chen, Sisi Li, and MengChu Zhou. 2017. A Noise-Filtered Under-Sampling Scheme for Imbalanced Classification. *IEEE Transactions on Cybernetics* 47, 12 (2017), 4263–4274.

[33] Wang-Cheng Kang, Wu-Jun Li, and Zhi-Hua Zhou. 2016. Column Sampling based Discrete Supervised Hashing. In *AAAI*. 1230–1236.

[34] Gou Koutaki, Keiichiro Shirai, and Mitsuru Ambai. 2018. Hadamard Coding for Supervised Discrete Hashing. *TIP* 27, 11 (2018), 5378–5392.

[35] Alex Krizhevsky. 2009. *Learning Multiple Layers of Features from Tiny Images*. Technical Report. University of Toronto. 1––60 pages.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*. 1106–1114.

[37] Qi Li, Zhenan Sun, Ran He, and Tieniu Tan. 2017. Deep Supervised Discrete Hashing. In *NeurIPS*. 2482–2491.

[38] Qi Li, Zhenan Sun, Ran He, and Tieniu Tan. 2020. A General Framework for Deep Supervised Discrete Hashing. *IJCV* 128, 8 (2020), 2204–2222.

[39] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. 2016. Feature Learning Based Deep Supervised Hashing with Pairwise Labels. In *IJCAIs*. 1711–1717.

[40] Guohua Liang and Chengqi Zhang. 2012. An efficient and simple under-sampling technique for imbalanced time series classification. In *CIKM*. 2339–2342.

[41] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and David Suter. 2014. Fast Supervised Hashing with Decision Trees for High-Dimensional Data. In *CVPR*. 1971–1978.

[42] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou. 2016. Learning Compact Binary Descriptors with Unsupervised Deep Neural Networks. In *CVPR*. 1183–1192.

[43] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2017. Focal Loss for Dense Object Detection. In *ICCV*. 2999–3007.

[44] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. 2020. Focal Loss for Dense Object Detection. *TPAMI* 42, 2 (2020), 318–327.

[45] Jack Lindsey, Samuel A. Ocko, Surya Ganguli, and Stéphane Deny. 2019. A Unified Theory of Early Visual Representations from Retina to Cortex through Anatomically Constrained Deep CNNs. In *ICLR*. 1–17.

[46] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep Learning for Extreme Multi-Label Text Classification. In *SIGIR*. 115–124.

[47] Song Liu, Shengsheng Qian, Yang Guan, Jiawei Zhan, and Long Ying. 2020. Joint-modal Distribution-based Similarity Hashing for Large-scale Unsupervised Deep Cross-modal Retrieval. In *SIGIR*. 1379–1388.

[48] Tie-Yan Liu. 2009. *Learning to Rank for Information Retrieval*. Foundations and Trends in Information Retrieval.

[49] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. 2014. Discrete Graph Hashing. In *NeurIPS*. 3419–3427.

[50] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. 2012. Supervised hashing with kernels. In *CVPR*. 2074–2081.

[51] Ziwei Liu, Zhongqi Miao, Xiaohang Zhan, Jiayun Wang, Boqing Gong, and Stella X. Yu. 2019. Large-Scale Long-Tailed Recognition in an Open World. In *CVPR*. 2537–2546.

[52] Fuchen Long, Ting Yao, Qi Dai, Xinmei Tian, Jiebo Luo, and Tao Mei. 2018. Deep Domain Adaptation Hashing with Adversarial Learning. In *SIGIR*. 725–734.

[53] Xu Lu, Lei Zhu, Zhiyong Cheng, Liqiang Nie, and Huaxiang Zhang. 2019. Online Multi-modal Hashing with Dynamic Query-adaption. In *SIGIR*. 715–724.

[54] Xu Lu, Lei Zhu, Jingjing Li, Huaxiang Zhang, and Heng Tao Shen. 2020. Efficient Supervised Discrete Multi-View Hashing for Large-Scale Multimedia Search. *TMM* 22, 8 (2020), 2048–2060.

[55] Xin Luo, Liqiang Nie, Xiangnan He, Ye Wu, Zhen-Duo Chen, and Xin-Shun Xu. 2018. Fast Scalable Supervised Hashing. In *SIGIR*. 735–744.

[56] William J. Reed. 2001. The Pareto, Zipf and other Power Laws. *Economics Letters* 74, 1 (2001), 15–19.

[57] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning Representations by Back-Propagating Errors. *Nature* 323 (1986), 533—-536.

[58] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115, 3 (2015), 211–252.

[59] Tetsuya Sakai. 2020. On Fuhr's Guideline for IR Evaluation. 54, 1 (2020), p14.

[60] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. 2016. Meta-Learning with Memory-Augmented Neural Networks. In *ICML*. 1842–1850.

[61] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. 2015. Supervised Discrete Hashing. In *CVPR*. 37–45.

[62] Shaoyun Shi, Weizhi Ma, Min Zhang, Yongfeng Zhang, Xinxing Yu, Houzhi Shan, Yiqun Liu, and Shaoping Ma. 2020. Beyond User Embedding Matrix: Learning to Hash for Modeling Large-Scale Users in Recommendation. In *SIGIR*. 319–328.

[63] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*. 1–14.

[64] Changchang Sun, Xuemeng Song, Fuli Feng, Wayne Xin Zhao, Hao Zhang, and Liqiang Nie. 2019. Supervised Hierarchical Cross-Modal Hashing. In *SIGIR*. 725–734.

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*. 5998–6008.

[66] Di Wang, Quan Wang, Yaqiang An, Xinbo Gao, and Yumin Tian. 2020. Online Collective Matrix Factorization Hashing for Large-Scale Cross-Media Retrieval. In *SIGIR*. 1409–1418.
[67] Jun Wang, Wei Liu, Andy X. Sun, and Yu-Gang Jiang. 2013. Learning Hash Codes with Listwise Supervision. In *ICCV*. 3032–3039.
[68] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. 2018. A Survey on Learning to Hash. *TPAMI* 40, 4 (2018), 769–790.
[69] Qifan Wang, Luo Si, Zhiwei Zhang, and Ning Zhang. 2014. Active Hashing with Joint Data Example and Tag Selection. In *SIGIR*. 405–414.
[70] Qifan Wang, Dan Zhang, and Luo Si. 2013. Semantic Hashing Using Tags and Topic Modeling. In *SIGIR*. 213–222.
[71] Qifan Wang, Zhiwei Zhang, and Luo Si. 2015. Ranking Preserving Hashing for Fast Similarity Search. In *IJCAI*. 3911–3917.
[72] Xudong Wang, Long Lian, Zhongqi Miao, Ziwei Liu, and Stella X. Yu. 2020. Long-ailed Recognition by Routing Diverse Distribution-Aware Experts. *arXiv* arXiv:2010.01809 (2020), 1–14.
[73] Xiaofang Wang, Yi Shi, and Kris M. Kitani. 2016. Deep Supervised Hashing with Triplet Labels. In *ACCV*. 70–84.
[74] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. 2017. Learning to Model the Tail. In *NeurIPS*. 7029–7039.
[75] Zijian Wang, Zheng Zhang, Yadan Luo, and Zi Huang. 2019. Deep Collaborative Discrete Hashing with Semantic-Invariant Structure. In *SIGIR*. 905–908.
[76] Erkun Yang, Cheng Deng, Tongliang Liu, Wei Liu, and Dacheng Tao. 2018. Semantic Structure-based Unsupervised Deep Hashing. In *IJCAI*. 1064–1070.
[77] Zhan Yang, Jun Long, Lei Zhu, and Wenti Huang. 2020. Nonlinear Robust Discrete Hashing for Cross-Modal Retrieval. In *SIGIR*. 1349–1358.
[78] Li Yuan, Tao Wang, Xiaopeng Zhang, Francis E. H. Tay, Zequn Jie, Wei Liu, and Jiashi Feng. 2020. Central Similarity Quantization for Efficient Image and Video Retrieval. In *CVPR*. 3080–3089.
[79] Weixin Zeng, Xiang Zhao, Wei Wang, Jiuyang Tang, and Zhen Tan. 2020. Degree-Aware Alignment for Entities in Tail. In *SIGIR*. 811–820.
[80] Dan Zhang, Fei Wang, and Luo Si. 2011. Composite Hashing with Multiple Information Sources. In *SIGIR*. 225–234.
[81] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu. 2010. Self-Taught Hashing for Fast Similarity Search. In *SIGIR*. 18–25.
[82] Hongfei Zhang, Xia Song, Chenyan Xiong, Corby Rosset, Paul N Bennett, Nick Craswell, and Saurabh Tiwary. 2019. Generic Intent Representation in Web Search. In *SIGIR*. 65–74.
[83] Peichao Zhang, Wei Zhang, Wu-Jun Li, and Minyi Guo. 2014. Supervised Hashing with Latent Factor Models. In *SIGIR*. 173–182.
[84] Xiao Zhang, Zhiyuan Fang, Yandong Wen, Zhifeng Li, and Yu Qiao. 2017. Range Loss for Deep Face Recognition with Long-Tailed Training Data. In *ICCV*. 5419–5428.
[85] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference Preserving Hashing for Efficient Recommendation. In *SIGIR*. 183–192.
[86] Boyan Zhou, Quan Cui, Xiu-Shen Wei, and Zhao-Min Chen. 2020. BBN: Bilateral-Branch Network With Cumulative Learning for Long-Tailed Visual Recognition. In *CVPR*. 9716–9725.
[87] Linchao Zhu and Yi Yang. 2020. Inflated Episodic Memory With Region Self-Attention for Long-Tailed Visual Recognition. In *CVPR*. 4343–4352.