Communication-Efficient Distributed Deep Metric Learning with Hybrid Synchronization

Yuxin Su
Department of Computer Science and Engineering
The Chinese University of Hong Kong
yxsu@cse.cuhk.edu.hk

Michael Lyu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
lyu@cse.cuhk.edu.hk

Irwin King
Department of Computer Science and Engineering
The Chinese University of Hong Kong
king@cse.cuhk.edu.hk

ABSTRACT

Deep metric learning is widely used in extreme classification and image retrieval because of its powerful ability to learn the semantic low-dimensional embedding of high-dimensional data. However, the heavy computational cost of mining valuable pair or triplet of training data and updating models frequently in existing deep metric learning approaches becomes a barrier to apply such methods to a large-scale real-world context in a distributed environment. Moreover, existing distributed deep learning framework is not designed for deep metric learning tasks, because it is difficult to implement a smart mining policy of valuable training data. In this paper, we introduce a novel distributed framework to speed up the training process of the deep metric learning using multiple machines. Specifically, we first design a distributed sampling method to find the hard-negative samples from a broader scope of candidate samples compared to the single-machine solution. Then, we design a hybrid communication pattern and implement a decentralized data-parallel framework to reduce the communication workload while the quality of the trained deep metric models is preserved. In experiments, we show excellent performance gain compared to a full spectrum of state-of-the-art deep metric learning models on multiple datasets in terms of image clustering and image retrieval tasks.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**;

KEYWORDS

Distributed computation, Distance metric learning, Deep metric learning

ACM Reference Format:
Yuxin Su, Michael Lyu, and Irwin King. 2018. Communication-Efficient Distributed Deep Metric Learning with Hybrid Synchronization. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271807

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271807

1 INTRODUCTION

Distance metric learning attempts to learn an advanced metric space in which the mapped representation of the raw data preserves the short distance between similar data points and the long distance between the dissimilar data points [3, 35]. This well-learned representation plays important role in information retrieval [28] and recommender systems [13]. For the past few years, with the success of deep learning, deep metric learning has received much attention. Because the distance metric learning shares similar assumption and objective with deep learning [17]. However, at odds with the conventional deep learning, which are successfully used to learn category related concepts on a large number of labeled data, deep metric learning aims to learn a nonlinear embedding of the data using deep neural network on a semantic metric space, which preserves the general concept of distance metric in data space.

Compared to deep metric learning, the drawback of the traditional deep neural networks such as GoogleNet [29] and ResNet [11] comes from the following two-fold: (a), the computational complexity of training and inference in deep learning is linearly proportional to the number of classes. It becomes impractical for extreme or fine-grained classification. (b), the most of deep learning models tend to overfit easily on a small amount of data per class. Therefore, deep metric learning yields promising results on many applications such as face recognition [23], zero-shot classification [5] and image retrieval [2].

The general procedure of deep metric learning contains two steps. First, a modified Siamese network [4] is optimized with the minimum value of one of the following loss: contrastive loss [10], random triplet loss [34], triplet loss with semi-hard negative mining strategy [23], lifted structured embedding loss [27], N-pair metric loss [25] and facility location loss [26]. Then, the semantic representing space generated from the Siamese network can be employed by many algorithms with efficient nearest-neighbor inference using the labeled data [12].

However, the existing deep metric learning approaches encounter a huge computational challenging for a large-scale dataset. Typically, deep metric learning considers a pair or a triplet of images as a training sample. Hence, n images can generate $O\left(n^2\right)$ or $O\left(n^3\right)$ training samples, which are intractable to iterate for large-scale dataset. A potential solution to reduce the size of search space is to find hard samples for training [7, 36]. Unfortunately, the hard sample mining brings enormous computation demands because it involves the inference for all data, which constantly varies during the training. To reduce the inference complexity, [7] involves human to label hard negative images from the evaluation of the model in each epoch. [32] samples triplets randomly during the

first ten training epochs, then trains the model with negative hard triplets in each minibatch after ten epochs. Although a compromise between the inference complexity and the convergence process during the training is reached, the inefficient problem still exists in deep metric learning framework.

Usually, distributed computation is a prominent tool to provide enough computational power. In the traditional deep learning community, distributed computation among multiple machines is also widely involved to handle the substantial computation demands [9]. Therefore, a deep metric learning oriented distributed algorithm is crucial to bring deep metric learning into big data applications. Since the smart sampling is a special issue in deep metric learning. It is difficult to utilize the conventional distributed deep learning platforms to find valuable training data globally. However, to our best knowledge, there is no distributed algorithm designed for or platform optimized for deep metric learning task.

In this paper, to conduct deep metric learning in distributed environment efficiently, we first construct a distributed sampling method to find the valuable triplet training data efficiently among multiple machines. We analyze the characteristics of several deep metric learning algorithms, then construct an empirical framework to distribute the computation task of deep metric learning into machines. Precisely, a typical Siamese network consists of two cascading components: Convolutional Neural Network (CNN) part and the last layer part. In many deep metric learning algorithms, the first CNN part is initialized with a pre-trained model from traditional classification task. Based on the observation that the CNN part with the massive number of parameters varies infrequently compared to the frequently updated last layer part with fewer parameters, we have designed different communication patterns: ring topology for CNN part, All-Reduce topology for the last layer part to synchronize these two components among machines.

In summary, the contributions of this paper are listed as follows:

- To the best of our knowledge, we are the first to propose a distance metric learning oriented distributed framework to speed up the training of deep metric learning among multiple machines.
- We have verified the decentralized distributed computation with mixed communication topology is reasonable in the context of deep metric learning.
- We also demonstrate the proposed framework is appropriately coupled with several state-of-the-art deep metric learning algorithms on CUB200-2011, CARS196, and Stanford Online Products and achieves a remarkable improvement with four machines regarding accuracy and runtime speedup.

2 PRELIMINARIES AND RELATED WORK

Siamese neural networks [4, 14] employ pair or triplet neural network with shared parameters to learn a non-linear function embedding raw high-dimensional data into low-dimensional metric, in which a contrastive loss is trained to distinguish between similar and dissimilar pairs of data.

Let $x \in \mathcal{X}$ be an input data and $y \in \{1, \ldots, L\}$ is the corresponding output label. In the pairwise framework, Siamese network is trained with contrastive loss calculated with pairs of examples $\left(x_i, x_j\right)$ which are either similar or dissimilar. The objective of

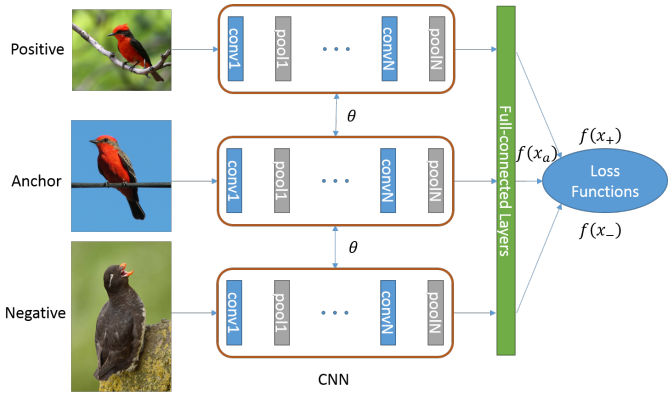


Figure 1: Siamese network. In triplet framework, the Siamese network generates the embedding data $f(x_a)$, $f(x_+)$, $f(x_-)$ for the anchor x_a, the positive x_+ and the negative x_- respectively.

learning is that the distances between embedding vectors of similar examples with some fixed margin should be smaller than the distances between that of dissimilar examples. More precisely, it minimizes the following loss:

$$\ell(\mathcal{X}, Y) = \frac{1}{|\mathcal{P}|} \sum_{(x_i, x_j) \in \mathcal{P}} y_{i,j} g\left(x_i, x_j\right) + \left(1 - y_{i,j}\right) \left[\alpha - g\left(x_i, x_j\right)\right]_+, \quad (1)$$

where $g(\cdot, \cdot)$ represents the output of Siamese network for a given pair of input. The label $y_{i,j} \in \{0, 1\}$ indicates whether a pair $\left(x_i, x_j\right)$ comes from the same class or not. The operation $[\cdot]_+$ denotes the hinge function which takes the positive component. α denotes a fixed margin. Usually, g is defined as a Euclidean distance between two embedding data points:

$$g\left(x_i, x_j\right) = \left\| f\left(x_i\right) - f\left(x_j\right) \right\|_2, \quad (2)$$

where $f(x)$ generates the representing features of the given input x in deep neural network.

The main idea is to construct a Euclidean space to make sure that positive pairs are close to each other while negative pairs are pushed away. Although the training process only requires a weaker form of supervision, the contrastive loss is focused on absolute distances, where the relative distance is more critical in many situations.

Similarly, in triplet-wise framework [34], the model is trained with triplets of examples (x_a, x_+, x_-). The positive and negative data of a given anchor point x_a is denoted as $x_+$and x_- respectively. It means that x_a and x_+ comes from the same class while x_- is from different class to x_a. The objective of learning is that the distance between embedding vectors of similar pair (x_a, x_+) is less than that of dissimilar pair (x_a, x_-). Within the triplet-wise framework, [23] constructs triplets by finding a semi-hard negative data, which has the smallest distance to the anchor point x_a among all negative data points $\{x_-\}$.

The triplet loss is defined as the following:

$$\ell(\mathcal{X}, Y) = \frac{1}{|\mathcal{T}|} \sum_{(x_a, x_+, x_-) \in \mathcal{T}} [g(x_a, x_+) + \alpha - g(x_a, x_-)]_+, \quad (3)$$

where $\mathcal{T}$ is the set of triplets.

[27] considers all the positive and negative pairs, they follow [23] to randomly sample several positive pairs, then select their semi-hard negative pairs to the training mini-batch. Finally, they optimize a structured prediction objective on the lifted problem by converting the vector of pair-wise distances in the mini-batch to the matrix of pairwise distance. [24] finds an aggressive mining method to improves the discrimination of model by combing both positive and negative pairs during the training. [26] finds that minimizing the metric loss for mini-batches does not necessarily lead to a more discriminative metric space and propose to consider the global structure of the metric space during the training.

The following two ideas closely relate to deep metric learning, but it is difficult to extend these methods into a distributed setting. [16] proposes an effective deep metric learning approach by carefully partitioning the training dataset. [30] estimates the distribution of positive and negative data pairs and take advantages of a distribution loss for deep metric learning.

Although there is no existing distributed framework designed for deep metric learning algorithm, we can easily employ conventional distributed deep learning frameworks [21, 37] to boost deep metric learning algorithms. Parameter server [18] is a widely adopted solution in distributed deep learning platforms [1, 6] The challenge is how to allocate resources between workers and parameter servers to fully utilize CPU and reduce the communication workload around the parameter servers. [19, 33] explore to reduce the communication demand in deep learning framework by decentralizing the process of the gradient aggregation. However, for many deep metric learning algorithms, a straightforward adoption from a single machine to distributed environment cannot achieve a remarkable performance gain. We will discuss the reasons with more details in the following sections.

3 PROPOSED FRAMEWORK

For the training process with large-scale datasets, we need larger mini-batch, partitioned into multiple machines. However, the deep metric learning with large mini-batch involves more communication demands because of the special selection method for mini-batch. In order to obtain a reasonable speedup with satisfying the accuracy, we need a smart architecture and policy for distributed training.

A common architecture for deep neural network systems takes advantage of data-parallelism [1]: a set of machines train model replicas on partitions of the input data in parallel. The model replicas are kept synchronized by all-reduced operation in Message Passing Interface (MPI) or parameter server, which maintains a global partition of the trained model and working machines fetch updated model from the parameter server periodically. These two methods guarantee the strong consistency of models among machines, but saturate the available network bandwidth, which becomes the bottleneck to accelerate the training process. In details, parameter server deteriorates the utility of the number of machines, because

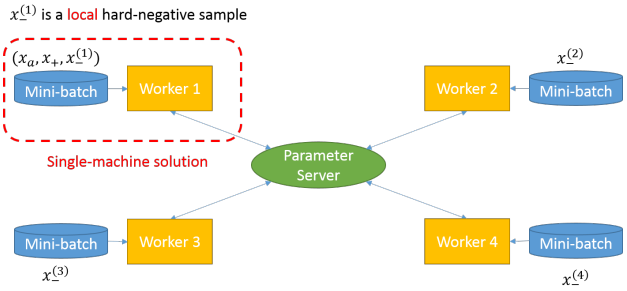


Figure 2: Illustration of the semi-hard negative mining in distributed environment.

several physical machines are equipped with server maintaining global parameters. The network around the server easily becomes a bottleneck because of heavy communication tasks around the servers. For all-reduced operation, there is no centralized parameter server, but the frequent broadcast operations consume lots of network resources. Meanwhile, the computation unit must wait for the synchronization of large models. Therefore, it is an open problem that how deep neural network systems balance the use of computation and network resources to achieve the fastest model convergence.

Compared to traditional deep neural network, deep metric learning contains two unique features: (1), the selection of the mini-batch with valuable training data containing the hard-positive and hard-negative samples is costly. (2), the major component of the whole model is initialized with a sophisticated pre-trained model. To take advantage of the two features, we propose a distributed framework to conduct distributed sampling described in subsection 3.1 to find the valuable training data with hard-negative data globally. Then we design a mixed communication topology to synchronize these components with the different level of consistency, which is explained in the subsection 3.2.

3.1 Distributed Sampling and Evaluation

In this subsection, we first discuss the challenge of the adoption from single-machine solution to distributed solution for deep metric learning algorithm. Then, we proposed a distributed sampling and evaluation policy to alleviate the disadvantage of the single-machine solution.

Existing distributed deep learning frameworks only focus on the gradient synchronization (aggregation and scattering) among machines. There is no further consideration about data sampling methods, which are independent to machines in conventional platforms. Therefore, as illustrated in fig. 2, a straightforward conversion from single machine algorithm to distributed algorithm leads to a simple aggregation from several single-machine solutions. For each machine, they can not take advantage of a large portion of samples from other machines. This constrained sampling scope becomes a barrier to enhance the overall performance of deep metric learning algorithms. Because a proper sampling strategy plays an equally important role as a suitable loss function [20].

In order to achieve a fast convergence rate, it is crucial to sample the hard positive and the hard negative samples from a large mini-batch. For a given anchor sample x_a, the hard positive is calculated from:

$$x_+ \leftarrow \underset{i:y[i]=y[a]}{\arg\max} \|f(x_a) - f(x_i)\|_2^2 \tag{4}$$

The hard negative sample is calculated from:

$$x_- \leftarrow \underset{i:y[i]\neq y[a]}{\arg\min} \|f(x_i) - f(x_a)\|_2^2 \tag{5}$$

To mine the hard negative sample, we formalize our proposed method with optimization to identify a negative example from a large scope of mini-batch co-located in all machines. For machine p, suppose the number of classes in a mini-batch is $|C[p]|$. For each class, we randomly select an anchor sample x_a and then find the associated hard positive samples from the same class in the mini-batch. We denote the indicator set of the anchor sample for each class as $C[p]$. To support the above sampling rules, we should construct the mini-batch with more classes and keep the number of samples per class relatively constant.

The search for hard negative sample involves the online evaluation of deep neural networks, which is computationally demanding for a large number of instances from distinct classes. In our proposed framework, we dispatch the evaluation and comparison tasks to multiple machines. At the first stage, for each machine p, it will broadcast the embedding vector of anchor samples $\left\{f_p\left(x_a^{(i)}\right)\right\}_{i\in C[p]}$ to other machines. Note that the deep neural network model could be slightly different among machines, it will be explained in the next subsection. We denote f_p as the deep neural network in the machine p.

At the second stage, each machine seeks for the hard negative sample corresponding to the incoming anchor samples. Specifically, for each machine q, it conducts the online evaluation of deep neural network for all samples in the mini-batch first, then calculates the distances between the embedding vectors and the incoming anchor samples to find the hard negative samples. For an anchor point x_{a_p} from the machine p, the hard negative sample is computed as:

$$x_-^{(a_p)} \leftarrow \underset{i:y[i]\neq y[a_p]}{\arg\min} \left\|f_q(x_i) - f_p\left(x_{a_p}\right)\right\|_2^2. \tag{6}$$

When all hard-negative samples are found, the machine q will scatter the embedding vectors of these samples to other machines. Finally, when the set of triplet samples is ready in machine p, we can employ the back-propagation method with the following loss definition:

$$\ell\left([X]_p, [Y]_p\right) = -\frac{1}{|C[p]|}\sum_{a\in C[p]} \log \frac{\exp\left\{f(x_a)^T f\left(x_{\mathcal{P}(a)}\right)\right\}}{\exp\left\{f(x_a)^T f\left(x_{\mathcal{P}(a)}\right)\right\} + \exp\left\{f(x_a)^T f\left(x_{\mathcal{N}(a)}\right)\right\}} + \frac{\lambda}{m}\sum_i^m \|f(x_i)\|_2, \tag{7}$$

where $\mathcal{P}(\cdot)$ and $\mathcal{N}(\cdot)$ are the indicators of the hard positive sample and negative sample respectively. m is the size of mini-batch.

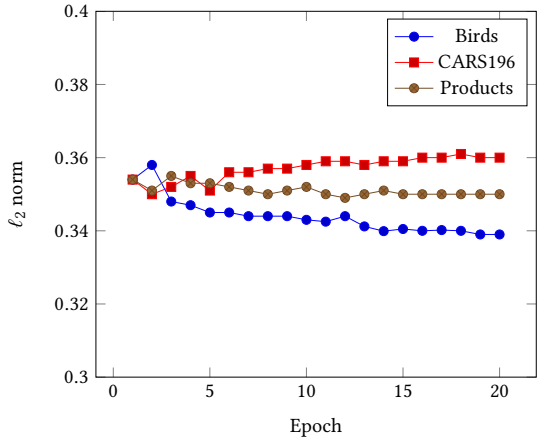


Figure 4: The average of the ℓ_2 norm of all parameters in CNN module during the training process.

3.2 Hybrid Synchronization

Suppose there are P machines, machine i contains a cascading model parameterized by θ_i with two distinct components C_i and F_i, which denote the parameters of the CNN part initialized with the pre-trained model and the last fully-connected layer respectively.

Typically, the parameters are calculated by the back-propagation algorithm with gradient descent, which updates θ iteratively:

$$\theta_i^{(t+1)} \leftarrow \theta_i^{(t)} + \eta^{(t)} \sum_{j\in P} \nabla f\left(\theta_j^{(t)}\right), \tag{8}$$

where θ_t are the parameters in iteration t, $\eta^{(t)}$ is the learning rate in iteration t.

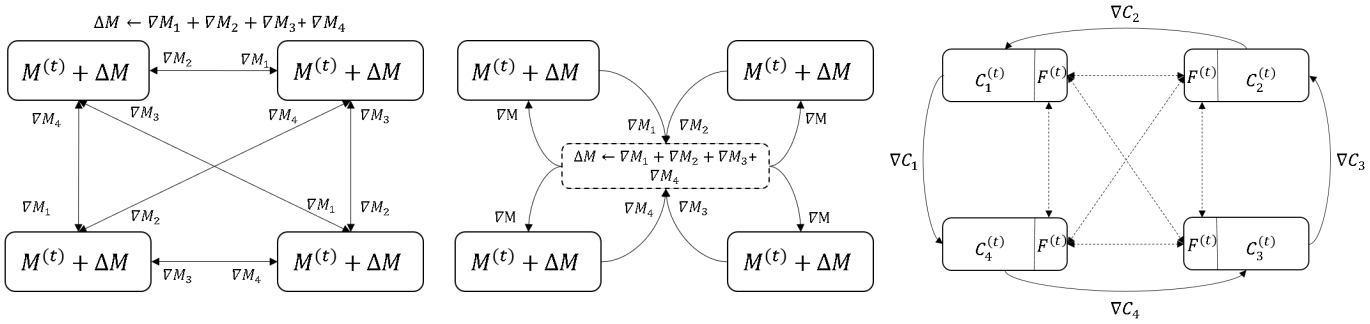
In the setting of All-Reduce or the synchronous parameter server, $\theta_j = \theta_k\ \forall j,k \in [P]$. It requires the parameter server to collects all gradients and to conduct the gradient aggregation in Eq. (8), then broadcast the updated parameters to all machines. This strong consistency requirement brings a heavy burden for network communication and degrades the training speed accordingly.

Empirically, we have the two following observations: (a), many deep metric learning algorithms are designed for two settings: end-to-end and last-layer. The latter means that the CNN part is initialized with a pre-trained model and then keeps fixed. (b), we observe that the average of ℓ_2 norm CNN part C_i varies insignificantly on many datasets, illustrated in fig. 6. Therefore, inspired by the decentralized framework [33], to increase the communication efficiency, we slacken the strong consistency constraints to allow the models in multiple machines are different. We propose the ring-based synchronization topology as follows:

$$C_i^{(t+1)} \leftarrow C_i^{(t)} + \eta^{(t)}\left(\beta\nabla f\left(C_i^{(t)}\right) + (1-\beta)\nabla f\left(C_{\text{Left}(i)}^{(t)}\right)\right), \tag{9}$$

where Left $(\cdot)$ denotes the index of last machine in a ring topology, $\eta^{(t)}$ represent the learning rate at iteration t. Currently, $\beta = 0.5$ is a predefined constant.

Ring-based synchronization is an optimal bandwidth solution to synchronize the parameters between machines, because the communication complexity of each machine reduces from $O(P)$ to $O(1)$.



(a) All-Reduce **(b) Parameter Server** **(c) Hybrid Synchronization**

Figure 3: Comparisons of several distributed communication topologies

In experiments, we have profiled the training process to monitor the change of all CNN components $C_{[P]}$. Figure 6 demonstrates the convergence between any two adjacent machines in the ring topology.

For the last layer, we follow the strong consistency requirements to update the last layer F_i, because the number of parameters in F_i is relatively small and the synchronous update does not barrier the training process.

In overall, the proposed hybrid synchronization consists of ring-based synchronization to update the large amount parameters in CNN models, and All-Reduce synchronization to update the frequently changed parameters in the last layer. Figure 3 illustrates the difference of the three communication topologies. Algorithm 1 summaries the proposed distributed deep metric learning algorithm.

3.3 Implementation Details

We have implemented our framework [1] with the popular PyTorch [22]. In order to reduce the waiting time of gradient synchronization, we modified the default PyTorch architecture to allow us to synchronize the gradients of the current layer in an asynchronous way while computing the gradients of the next one simultaneously.

For network setting, we basically follow the configuration in [26]. We employ existing network with pretrained parameters and fine-tune the network on new datasets. At the end of deep neural network, we conduct ℓ_2 normalization first to the embedding vector before computing the loss. The experimental ablation study in [27] reports that the dimensionality of the embedding vector does not play a crucial role during the training and testing process. We explored the dimensionality of embedding vector from 64 [26] to 512 [27] . The larger embedding size will perform slightly better than the smaller ones in accuracy but bring heavy communication requests in a distributed setting. Therefore, we select 128 as a trade-off. For each iteration, we only need to sample m examples and labels from different classes at random. In particular, we randomly select 5 images per class in a mini-batch. There is no need to prepare the data in any rigid paired or triplets format.

[1] https://github.com/yxsu/ddml-hs

ALGORITHM 1: Proposed Distributed Deep Metric Learning Algorithm

Input: Image datasets with C labeled categories, P : number of machines, T : number of iteration, η : step size, λ: regularization parameter
Output: $f(\theta)$: optimized deep neural network parameterized by θ
for $t = 1, \ldots, T$ **do**
  for all *machine* $p \in [1, P]$ **do in parallel**
    Randomly sample to from the whole dataset;
    Randomly select anchor sample x_a for each class. $C[p]$ represents the indicator set of anchor sample for each class;
    Find the hard positive sample by:
    $x_+ \leftarrow \underset{i:y[i]=y[a]}{\arg\max} \|f(x_a) - f(x_i)\|_2^2$
    Broadcast $\left\{f_p\left(x_a^{(i)}\right)\right\}_{i \in C[p]}$ to other machines;
    Receive $\left\{f_q\left(x_a^{(i)}\right) : i \in C[q], q \in [1, P]\right\}$ from other machines;
    Find all hard negative samples by: $x_-^{(a_q)} \leftarrow$
    $\underset{i:y[i] \neq y[a_q]}{\arg\min} \left\|f_p(x_i) - f_q\left(x_{a_q}\right)\right\|_2^2 \quad \forall a_q \in C[q], q \in [1, P]$;
    Scatter the above hard negative samples;
    After receiving the hard negative samples, conduct back-propagation on the loss defined in Eq. (7) to update θ;
    All-reduce the FC parameters F_p among all machines;
    Send CNN parameters C_p to its right neighbor Right(p);
    Update C_p with Eq. (9);
  end
end

For the network architecture, the CNN components are initialized with the ResNet-34 [11] pre-trained on the popular ImageNet / ILSVRC 2012-CLS dataset. The final fully-connected layers FC are randomly initialized.

We fine-tuned the network on the training datasets with two configurations:

- End-to-End: we fine-tune the overall model and update the parameters of all the layers during the back-propagation with the proposed hybrid synchronization. In this case, we perform 20 epochs of gradient descent.

- Last Layer: we keep all CNN components pretrained on ImageNet unchanged and update the last layer with our hybrid synchronization. In this situation, the communication in our framework is equivalent to the default All-reduce pattern in PyTorch.

All the input images are resized to 256×256 and cropped at 227×227. We use a random crop with random horizontal mirroring for training and a single center crop for testing. [25] takes multiple random crops and compute the average from the cropped images as the embedding vector.

4 EXPERIMENTS

In experiments, we attempt to answer the following two questions: (a), is our proposed distribution framework correct in the context of deep metric learning? We can obtain the answer from the comparisons between proposed framework and the several existing deep metric learning algorithms regarding accuracy in clustering and image retrieval. (b), is our framework communication-efficient and does it enjoy good scalability regarding the number of the machines? The observation on the execution time of the proposed framework with different hardware setting will illustrate the efficiency issue.

In the experiments, we evaluate our distributed framework on the following widely-used fine-grained datasets and employ the same train/test splits.

- The Caltech-UCSD Birds (CUB200-2011) [31] consists of 11,788 images of birds from 200 different species/categories. We use the standard configuration in which the first 100 categories (5,864 images) is used for training and the rest for test (5,924).
- CARS196 [15] consists of 16,185 images of cars from 196 model categories. The first 98 categories (8,054 images) are used for training and the rest for test (8,131 images).
- Stanford Online Products [27] consists of 120,253 images from 22,634 online product categories. The default configuration contains 59,551 images from 11,318 categories for training and 60,502 images from 11,316 categories for test.

In these experiments, we choose to disjoint the categories for training and testing separately, although they belong to the same context (they all represent birds, cars or products). This makes the problem more challenging because traditional deep neural network models may easily overfit on the training datasets with a vast number of categories.

All experiments are conducted on 4 servers with 2 Pascal GPUs each connected by 10 Gbit/s network. To make fair comparisons, we closely follow [26, 27] for deep metric learning configuration in experiments.

For clustering evaluation, we calculate the embedding vector on all the test images first, then conduct affinity propagation clustering algorithm [8] with bisection method for the predefined number of clusters, which is equal to the number of classes in the test set. We use the standard Normalized Mutual Information (NMI) metrics to quantitatively measure the clustering quality [26]. NMI is defined by the ratio of mutual information entropy of clusters and labels.

For the image retrieval evaluation, we employ the standard Recall@K metric in the standard K nearest neighbor retrieval task.

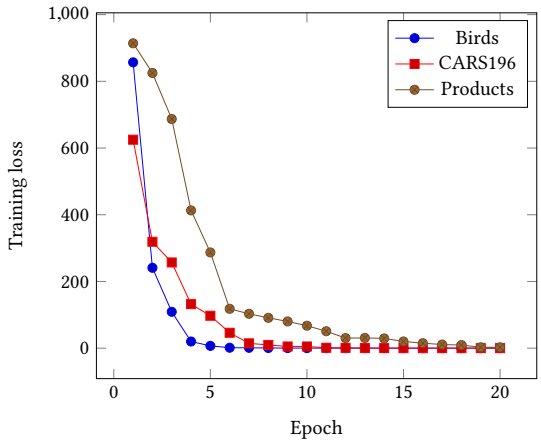


Figure 5: The training loss of the averaged deep neural network model in our proposed framework with 4 machines.

Recall@K is defined as the fraction of queries for which there exists a data sample of the same class as that of the query instance with the first K positions of the retrieved list.

We compare our proposed methods against several state-of-the-art methods including triplet loss with semi-hard-negative mining [23], deep metric learning via lifted structure [27], deep metric learning with histogram loss [30], deep metric learning with N-pair loss [25], deep metric learning via facility location (also called NMI-based) [26] and deep spectral clustering [16]. In experiments, to make completed comparisons, we have also adopted the state-of-the-art methods from single-machine solution to distributed computation environment with All-reduce communication topology, which is the standard distributed implementation in TensorFlow [1] and PyTorch [22].

4.1 Convergence

In this subsection, we attempt to show the correctness of our proposed framework on all three datasets.

In our proposed framework, the deep neural network model could be slightly different with its neighbors. Since we loose the consistency requirement in our hybrid synchronization policy to the decentralized setting. The theoretical analysis on the convergence issue of this decentralized deep neural network models is still an open problem [19]. In this paper, we only conduct an empirical study on this convergence issue. Figure 6 illustrates the difference of models among neighbors on three datasets. From the figure, we can see that the ℓ_2 norm difference of the parameters in CNN model convergence to zero after 10 epochs. It means that all parameters among neighborhood convergences to the same values.

Figure 5 further demonstrates the convergence process of our proposed framework. During the training process, we take the average of all training loss in different machines as the overall training loss depicted in fig. 5. After training, we consider the averaged model as the final optimized model to conduct test tasks.

Table 1: Evaluations on the Caltech-UCSD Birds (CUB-200-2011) dataset. Recall@k is the recall with the top k results.

Method	Setting	Time (s) / epoch	NMI	Recall@1	Recall@4	Recall@8
Triplet semi-negative [23]	Original	524	56.12	40.46	58.15	69.28
	2 machines	458	56.10	40.98	57.45	69.56
	4 machines	362	56.98	41.05	58.34	69.17
Lifted struct [27]	Original	536	56.30	43.24	66.73	79.61
	2 machines	441	56.20	44.19	66.50	79.13
	4 machines	391	56.98	44.34	66.56	79.78
Histogram [30]	Original	520	-	49.34	68.61	80.58
	2 machines	437	-	49.10	66.97	78.24
	4 machines	358	-	48.34	65.10	77.10
N-pairs [25]	Original	513	58.87	44.29	67.26	79.18
	2 machines	420	58.97	45.14	67.26	79.78
	4 machines	340	59.20	45.19	67.98	80.10
NMI-based [26]	Original	702	60.19	48.38	72.47	82.25
	2 machines	599	60.15	48.19	72.38	81.98
	4 machines	493	60.17	48.56	72.50	82.01
Spectral [16]	Original	617	58.13	50.28	76.80	**85.79**
	2 machines	574	58.10	50.78	76.75	85.58
	4 machines	432	58.98	51.03	76.85	85.78
Ours	2 machines	378	61.19	52.45	77.08	84.24
	4 machines	**234**	**61.96**	**52.87**	**78.19**	85.07
	4 machines (last layer)	**86**	55.34	43.19	60.73	75.21

Table 2: Evaluations on the CARS196 dataset. R@k is the recall with the top k results.

Method	Setting	Time (s) / epoch	NMI	Recall@1	Recall@4	Recall@8
Triplet semi-negative [23]	Original	507	54.09	41.30	75.21	80.32
	2 machines	446	54.13	44.25	75.59	79.19
	4 machines	378	54.20	44.56	75.50	79.31
Lifted struct [27]	Original	530	56.90	53.70	75.98	83.30
	2 machines	439	57.02	53.98	76.04	83.34
	4 machines	395	57.13	53.88	76.05	83.50
Histogram [30]	Original	512	-	53.67	75.56	81.20
	2 machines	425	-	53.10	74.97	81.09
	4 machines	355	-	53.00	74.56	80.78
N-pairs [25]	Original	489	58.04	54.36	79.03	84.23
	2 machines	398	58.34	54.56	79.01	84.34
	4 machines	336	58.78	54.88	79.12	84.56
NMI-based [26]	Original	832	57.27	57.29	79.90	88.24
	2 machines	703	57.17	57.09	79.78	88.05
	4 machines	578	57.20	57.19	79.81	88.20
Spectral [16]	Original	798	61.08	69.35	81.08	90.35
	2 machines	637	61.10	69.56	81.10	90.29
	4 machines	501	61.21	69.70	81.23	**90.40**
Ours	2 machines	423	61.24	70.07	82.13	89.25
	4 machines	**298**	**61.80**	**70.73**	**82.87**	90.10
	4 machines (last layer)	**93**	53.10	44.37	77.19	82.28

Table 3: Evaluations on the Stanford Online Products dataset. R@k is the recall with the top k results.

Method	Setting	Time (s) / epoch	NMI	Recall@1	Recall@4	Recall@8
Triplet semi-negative [23]	Original	4,097	88.38	67.12	77.97	82.28
	2 machines	3,658	88.40	67.23	78.01	82.38
	4 machines	3,167	88.42	67.34	78.01	82.38
Lifted struct [27]	Original	3,814	88.19	65.50	78.23	81.08
	2 machines	3,512	88.22	65.68	78.19	81.00
	4 machines	3,170	88.25	65.71	78.41	81.56
Histogram [30]	Original	3,856	-	65.55	78.73	81.56
	2 machines	3,550	-	65.01	78.30	81.12
	4 machines	3,210	-	64.95	78.15	81.03
N-pairs [25]	Original	3,701	89.01	67.12	79.15	84.09
	2 machines	3,506	89.12	67.31	79.21	84.15
	4 machines	3,010	89.18	67.35	79.51	84.34
NMI-based [26]	Original	6,238	90.27	66.98	77.06	82.15
	2 machines	5,078	90.34	67.10	77.10	82.20
	4 machines	3,998	**90.35**	67.28	77.21	82.24
Spectral [16]	Original	5,713	87.38	66.09	78.98	83.12
	2 machines	4,892	87.50	66.12	79.01	83.20
	4 machines	3,820	87.55	66.15	79.23	83.20
Ours	2 machines	3,028	89.17	67.79	80.34	**84.73**
	4 machines	**2,356**	89.56	**68.02**	**80.48**	84.70
	4 machines (last layer)	**898**	86.57	64.14	76.23	80.18

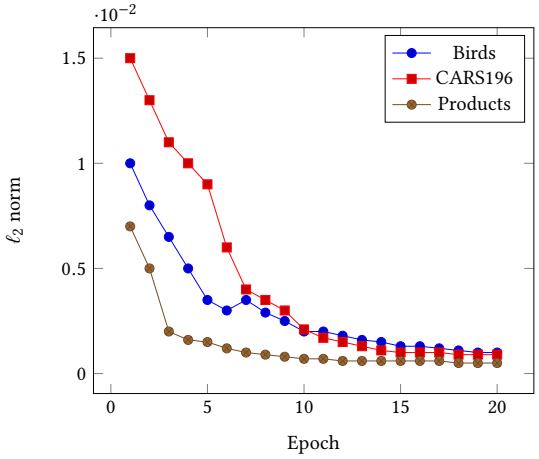


Figure 6: The average of $\left\|C_i - C_{\mathbf{Right}(i)}\right\|_2 \forall i \in [P]$. This figure displays the convergence of CNN models with 4 machines

4.2 Quantitative Results

In this subsection of quantitative evaluation, we try to empirically prove the effectiveness of our proposed distributed deep metric learning method in image retrieval and clustering tasks.

Tables 1, 2 and 3 display the comparison results between several state-of-the-art methods and our proposed method in terms of the NMI and k nearest neighbor task with the Recall@K metric. In this subsection, we only consider the "Original" setting for the state-of-the-art methods. "Original" means that all these algorithms are executed in the original single-machine solution with the same configuration in their original paper. However, we do not report the results of our proposed framework in the single machine setting. Because in a single machine, our sampling policy cannot take advantage of the large portion of data samples in a mini-batch. Therefore, the comparisons in the single-machine setting are unfair. The overall performance of our method in this environment is worse than the N-pairs method [25] with the similar loss definition.

From the tables, we can easily see a considerable improvement of our proposed method compared to the state-of-the-art methods, especially the N-pairs method. The improvement against N-pairs method demonstrates the effectiveness of our distributed sampling policy. Because our method has a similar loss definition compared to the N-pairs method and the significant difference comes from the extension of the sampling policy to the distributed configuration.

For the execution time, table 1, 2 and 3 also demonstrate distinguishable results from all these state-of-the-art methods. The execution time of the triplet loss with semi-hard-negative mining , the N-pairs method and the lifted structure method are comparable. Because these methods spend the majority of time on the online evaluation of the deep neural network. Compared to other methods, NMI-based method is slowest for each epoch. This is due to the cost of finding the facility location in this method.

For different datasets, the evaluation time of these methods for each mini-batch on the larger Stanford Online Products dataset is comparable to that on the smaller CUB-200-2011 and CARS196

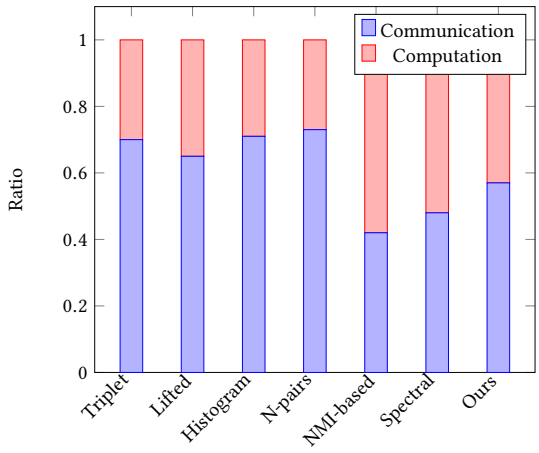


Figure 7: Comparison of computation time and communication time on the Caltech-UCSD Birds dataset with 4 machines. The results from other two datasets are similar.

datasets. Since all input images are scaled into the same size before evaluating the deep neural network. However, all methods are slow on the Stanford Online Products dataset. Because the Stanford Online Products dataset is larger than the others and has more rounds of mini-batching for each epoch. For the last-layer setting, as demonstrated in table 1, 2 and 3, the execution time is very fast. Since there is no computationally expensive back-propagation and heavy communication demands for the CNN components. However, in most cases, the performances of our proposed method with the last-layer setting on three datasets are not competitive concerning image retrieval and clustering tasks.

4.3 Scalability

In this subsection, we attempt to demonstrate the scalability of several state-of-the-art methods with the distributed setting. Then, we compare our proposed framework and the other methods with different distributed settings, in which the different behaviors motivate our work.

In triplet framework, large batch size brings a large scope of candidate samples, and it enjoys high probability to generate useful triplets with real hard-positive and hard-negative samples. Furthermore, there is generalization gap between large batch size and small batch size in learning theory. However, since existing deep metric learning algorithm are all designed for single machine environment, which contains a limited number of GPUs. The limited GPU memory constraints the maximum batch size as well as the model complexity of deep neural network. For example, we believe the overall performance of the state-of-the-art methods will increase if we only replace ResNet-34 with ResNet-151 and keep sampling policy unchanged in these methods. Unfortunately, the higher complexity of deep neural network will cost more GPU memory and lead to a smaller mini-batch size owing to the limited total GPU memory. Therefore, a suitable distributed framework is necessary for deep metric learning algorithm.

From table 1, 2 and 3, we can see that the overall execution time of all methods decreases when the number of machines increases. Specifically, the computation time for each epoch decreases. Because the rounds of mini-batching in each machine is less than that in single machine. However, the communication illustrated in fig 7 is costly. The strong consistency requirement in All-reduce distributed framework brings heavy communication workload in this case. For the performance in image retrieval and clustering tasks, we can not observe a stable improvement for the state-of-the-art methods when the number of machine increases. Therefore, a simple gradient aggregation policy implemented in All-reduce framework is not benefiting to distributed deep metric learning. This phenomenon is different from the conventional distributed deep learning approaches. The potential improvement should come from a smart sampling method which utilizes all machine resources.

Our proposed framework performs well in image retrieval and clustering tasks. In many cases, it achieves the state-of-the-art results. For the scalability issue, our proposed method performs better in general when the number of machine increases. These experiments prove that our distributed sampling method has a positive impact on the overall performance of deep metric learning algorithms.

For communication time, we expect that the ratio of communication time is as low as possible. Figure 7 illustrates the ratio of communication time and computation time of several methods. From the figure, we can easily observe that the ratio of communication time in our proposed method is lower than many state-the-of-art methods in a conventional distributed framework. The advantage of our proposed framework is more obvious if the algorithm is conducted on a larger-scale cluster with more machines. For NMI-based and Spectral methods, the lower ratio of communication time does not reveal an excellent performance on a distributed framework. In fact, the computation of NMI-based and Spectral methods are more costly.

5 CONCLUSION

In this paper, we present a scalable solution from the perspective of distributed computation to tackle the big data challenge for deep metric learning algorithm. In particular, we first propose a distributed sampling method to find the hard-negative samples from all machines to improve the efficiency of triplet mining method. Then, with the help of the unique characteristics of deep metric learning, we further propose a hybrid synchronization policy to speed up the training process of deep metric learning algorithm by reducing the communication workload significantly in a distributed environment. Finally, extensive experimental results demonstrate the effectiveness of our proposed distributed deep metric learning framework on image retrieval and clustering tasks.

6 ACKNOWLEDGEMENT

The work described in this paper was partially supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14208815 and No. CUHK 14234416 of the General Research Fund), and Microsoft Research Asia (2018 Microsoft Research Asia Collaborative Research Award).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.* 265–283.

[2] Sean Bell and Kavita Bala. 2015. Learning visual similarity for product design with convolutional neural networks. *ACM Trans. Graph.* 34, 4 (2015), 98:1–98:10. https://doi.org/10.1145/2766959

[3] Aurélien Bellet, Amaury Habrard, and Marc Sebban. 2013. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709* (2013).

[4] Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification Using A "Siamese" Time Delay Neural Network. *IJPRAI* 7, 4 (1993), 669–688. https://doi.org/10.1142/S0218001493000339

[5] Maxime Bucher, Stéphane Herbin, and Frédéric Jurie. 2016. Improving Semantic Embedding Consistency by Metric Learning for Zero-Shot Classiffication. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part V.* 730–746.

[6] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *CoRR* abs/1512.01274 (2015).

[7] Yin Cui, Feng Zhou, Yuanqing Lin, and Serge J. Belongie. 2016. Fine-Grained Categorization and Dataset Bootstrapping Using Deep Metric Learning with Humans in the Loop. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016.* 1153–1162. https://doi.org/10.1109/CVPR.2016.130

[8] Brendan J Frey and Delbert Dueck. 2007. Clustering by passing messages between data points. *science* 315, 5814 (2007), 972–976.

[9] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *CoRR* abs/1706.02677 (2017).

[10] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality Reduction by Learning an Invariant Mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA.* 1735–1742. https://doi.org/10.1109/CVPR.2006.100

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016.* 770–778. https://doi.org/10.1109/CVPR.2016.90

[12] Elad Hoffer and Nir Ailon. 2016. Semi-supervised deep learning by metric embedding. *CoRR* abs/1611.01449 (2016).

[13] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge J. Belongie, and Deborah Estrin. 2017. Collaborative Metric Learning. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017.* 193–201.

[14] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, Vol. 2.

[15] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 2013. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops.* 554–561.

[16] Marc Teva Law, Raquel Urtasun, and Richard S. Zemel. 2017. Deep Spectral Clustering Learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017.* 1985–1994.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[18] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.* 583–598.

[19] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA.* 5336–5346.

[20] R. Manmatha, Chao-Yuan Wu, Alexander J. Smola, and Philipp Krähenbühl. 2017. Sampling Matters in Deep Embedding Learning. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017.* 2859–2867.

[21] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. 2017. Ray: A Distributed Framework for Emerging AI Applications. *CoRR* abs/1712.05889 (2017).

[22] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[23] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015.* 815–823.

[24] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. 2015. Discriminative Learning of Deep Convolutional Feature Point Descriptors. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015.* 118–126.

[25] Kihyuk Sohn. 2016. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain.* 1849–1857.

[26] Hyun Oh Song, Stefanie Jegelka, Vivek Rathod, and Kevin Murphy. 2017. Deep Metric Learning via Facility Location. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017.* 2206–2214.

[27] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. 2016. Deep Metric Learning via Lifted Structured Feature Embedding. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016.* 4004–4012.

[28] Yuxin Su, Irwin King, and Michael R. Lyu. 2017. Learning to Rank Using Localized Geometric Mean Metrics. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017.* 45–54.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015.* 1–9.

[30] Evgeniya Ustinova and Victor S. Lempitsky. 2016. Learning Deep Embeddings with Histogram Loss. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain.* 4170–4178.

[31] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. 2011. The caltech-ucsd birds-200-2011 dataset. (2011).

[32] Xiaolong Wang and Abhinav Gupta. 2015. Unsupervised Learning of Visual Representations Using Videos. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015.* 2794–2802.

[33] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter R. Pietzuch. 2016. Ako: Decentralised Deep Learning with Partial Gradient Exchange. In *Proceedings of the Seventh ACM Symposium on Cloud Computing, Santa Clara, CA, USA, October 5-7, 2016.* 84–97.

[34] Kilian Q Weinberger and Lawrence K Saul. 2009. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *Journal of Machine Learning Research* 10 (2009), 207–244.

[35] Eric P. Xing, Michael I. Jordan, Stuart J. Russell, and Andrew Y. Ng. 2002. Distance Metric Learning with Application to Clustering with Side-Information. In *Advances in Neural Information Processing Systems.* 521–528.

[36] Yuhui Yuan, Kuiyuan Yang, and Chao Zhang. 2017. Hard-Aware Deeply Cascaded Embedding. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017.* 814–823.

[37] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P. Xing. 2017. Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters. In *2017 USENIX Annual Technical Conference, USENIX ATC 2017, Santa Clara, CA, USA, July 12-14, 2017.* 181–193.