Cross-Batch Memory for Embedding Learning

Xun Wang∗, Haozhi Zhang∗, Weilin Huang†, Matthew R. Scott
Malong Technologies
{xunwang, haozhang, whuang, mscott}@malong.com

Abstract

Mining informative negative instances are of central importance to deep metric learning (DML), however this task is intrinsically limited by mini-batch training, where only a mini-batch of instances is accessible at each iteration. In this paper, we identify a “slow drift” phenomena by observing that the embedding features drift exceptionally slow even as the model parameters are updating throughout the training process. This suggests that the features of instances computed at preceding iterations can be used to considerably approximate their features extracted by the current model. We propose a cross-batch memory (XBM) mechanism that memorizes the embeddings of past iterations, allowing the model to collect sufficient hard negative pairs across multiple mini-batches - even over the whole dataset. Our XBM can be directly integrated into a general pair-based DML framework, where the XBM augmented DML can boost performance considerably. In particular, without bells and whistles, a simple contrastive loss with our XBM can have large R@1 improvements of 12%-22.5% on three large-scale image retrieval datasets, surpassing the most sophisticated state-of-the-art methods [37, 26, 2]. Moreover, existing pair-based DML methods can be unified as weighting schemes under a General Pair Weighting (GPW) framework [37]. The performance of pair-based methods heavily rely on their capability of mining informative negative pairs. To collect sufficient informative negative pairs from each mini-batch, many efforts have been devoted to improving the sampling schemes, which can be categorized into two main directions: (1) sampling informative mini-batches based on global data distribution [32, 6, 28, 32, 9]; (2) weighting informative pairs within each individual mini-batch [22, 30, 37, 35, 40].

Various sophisticated sampling schemes have been developed, but the hard mining ability is inherently limited by the size of a mini-batch, which the number of possible training pairs depends on. Therefore, to improve the sampling scheme, it is straightforward to enlarge the mini-batch, which can boost the performance of pair-based DML methods immediately. We demonstrate by experiments that the performance of pair-based approaches, such as contrastive loss [3] and recent MS loss [37], can be improved strikingly when the mini-batch grows larger on large-scale datasets (Figure 1, left and middle). It is not surprising because the number of negative pairs grows quadratically w.r.t. the mini-batch size. However, simply enlarging a mini-batch is not an ideal solution to solve the hard mining problem due to two limitations: (1) the mini-batch size is limited by the GPU memory and computational cost; (2) a large mini-batch (e.g. 1800 used in [29]) often requires cross-device synchronization, which is a challenging engineering task. A naive solution to collect abundant informative pairs is to compute the features of instances in the whole training set at each training iteration, and then search for hard negative pairs from the whole dataset. Obviously, this solution is extremely time consuming, especially for a large-scale dataset, but it inspired us to break the limit of mining hard negatives within a single mini-batch.

1. Introduction

Deep metric learning (DML) aims to learn an embedding space where instances from the same class are encouraged to be closer than those from different classes. As a fundamental problem in computer vision, DML has been applied to various tasks, including image retrieval [39, 12, 7], face recognition [38], zero-shot learning [47, 1, 16], visual tracking [17, 34] and person re-identification [44, 13].

∗Equal contribution †Corresponding author
In this paper, we identify an interesting “slow drift” phenomena that the embedding of an instance actually drifts at a relatively slow rate throughout the training process. It suggests that the deep features of a mini-batch computed at past iterations can considerably approximate to those extracted by current model. Based on the “slow drift” phenomena, we propose a cross-batch memory (XBM) module to record and update the deep features of recent mini-batches, allowing for mining informative examples across multiple mini-batches. Our XBM can provide plentiful hard negative pairs by directly connecting each anchor in the current mini-batch with the embeddings from recent mini-batches.

Our XBM is conceptually simple, easy to implement and memory efficient. The memory module can be updated using a simple enqueue-dequeue mechanism by leveraging the computation-free features computed at the past iterations, with only about a negligible 0.2 GB of extra GPU memory utilized. More importantly, our XBM can be directly integrated into most existing pair-based methods with just several lines of codes, and can boost performance considerably. We evaluate our XBM with various conventional pair-based DML techniques on three widely used large-scale image retrieval datasets: Stanford Online Products (SOP) [22], In-shop Clothes Retrieval (In-shop) [20], and PKU VehicleID (VehicleID) [19]. In Figure 1 (middle and right), our approach demonstrates excellent robustness and brings consistent performance improvements across all settings; under the same configurations, our XBM obtains extraordinary R@1 improvements on all three datasets compared with the corresponding pair-based methods (e.g., over 20% for contrastive loss). Furthermore, with our XBM, a simple contrastive loss can easily outperform the most state-of-the-art sophisticated methods, such as [37, 26, 2], by a large margin.

In parallel to our work, He et al. [10] built a dynamic dictionary as a queue of preceding mini-batches to provide a rich set of negative samples for unsupervised learning (with a contrastive loss). However, unlike [10] which uses a specific encoding network to compute the features of current mini-batch, our features are computed more efficiently by taking them directly from the forward of the current model with no additional computational cost. More importantly, to solve the problem of feature drift, He et al. designed a momentum update that slowly progresses the key encoder to ensure the consistency between different iterations, while we identify the “slow drift” phenomena which suggests that the features can become stable by themselves when the early phase of training finishes.

2. Related Work

Pair-based DML. Pair-based DML methods can be optimized by computing the pair-wise similarities between instances in the embedding space [8, 22, 29, 35, 30, 37]. Contrastive loss [8] is one of the classic pair-based DML methods, which learns a discriminative metric via Siamese networks. It encourages the deep features of positive pairs to be closer to each other, and those of negative pairs to be farther than a fixed threshold. Triplet loss [29] requires the similarity of a positive pair to be higher than that of a negative pair (with the same anchor) by a given margin.

Inspired by contrastive loss and triplet loss, a number of pair-based DML algorithms have been developed, which attempted to weight all pairs in a mini-batch, such as up-weighting informative pairs (e.g., N-pair loss [30], Multi-Similarity (MS) loss [37]) through a log-exp formulation, or sampling negative pairs uniformly w.r.t. pair-wise distance [40]. Generally, pair-based methods can be cast into a unified weighting formulation through General Pair Weighting (GPW) framework [37].

However, most deep models are trained with SGD where only a mini-batch of samples are accessible at each iteration, and the size of a mini-batch can be relatively small compared to the whole dataset, especially for a large-scale dataset. Moreover, a large fraction of the pairs is less informative as the model learns to embed most trivial pairs correctly. Thus the conventional pair-based DML techniques suffer from lacks of hard negative pairs, which are critical to promote model training.

To alleviate the aforementioned problems, a number of approaches have been developed to collect more potential information contained in a mini-batch, such as building...
a class-level hierarchical tree [6], updating class-level signatures to select hard negative instances [32], or obtaining samples from an individual cluster [28]. Unlike these approaches which aim to enrich the information in a mini-batch, our XBM are designed to directly mine hard negative examples across multiple mini-batches.

**Proxy-based DML.** There is another branch of DML methods aiming to learn the embeddings by comparing each sample with proxies, including proxy NCA [21], NormSoftmax [46] and SoftTriple [25]. In fact, our XBM module can be regarded as the proxies to some extent. However, there are two main differences between the proxy-based methods and our XBM module: (1) proxies are often optimized along with the model weights, while the embeddings of our memory are directly taken from the past mini-batches; (2) proxies are used to represent the class-level information, whereas the embedding of our memory computes the information for each individual instance. Both proxy-based methods and our XBM augmented pair-based methods are able to capture the global distribution of data over the whole dataset during training.

**Feature Memory Module.** Non-parametric memory module for embedding learning has shown power in various computer visual tasks [36, 43, 41, 42, 48, 18]. For examples, the external memory can be used to address the unaffordable computational demand of conventional NCA [41] in large-scale recognition, and encourage instance-invariance in domain adaptation [48, 42]. But only the positive pairs are optimized, while the negatives are ignored in [41]. Our XBM is able to provide a rich set of negative examples for the pair-based DML methods, which is more generalized and can make full use of the past embeddings. The key distinction is that existing memory modules only store the embeddings of current mini-batch [36], or maintain the whole dataset [41, 48] with a moving average update, while our XBM is maintained as a dynamic queue of mini-batches, which is more flexible and applicable in extremely large-scale datasets.

### 3. Cross-Batch Memory Embedding Networks

In this section, we first analyze the limitation of existing pair-based DML methods. Then we introduce the “slow drift” phenomena, which provides the underlying evidence that supports our cross-batch mining approach. Finally, we describe our XBM module and integrate it into existing pair-based DML methods.

#### 3.1. Delving into Pair-based DML

Let \( \mathcal{X} = \{x_1, x_2, \ldots, x_N\} \) denotes a set of training instances, and \( y_i \) is the corresponding label of \( x_i \). An embedding function, \( f(\cdot; \theta) \), projects a data point \( x_i \) onto a \( D \)-dimensional unit hyper-sphere, \( v_i = f(x_i; \theta) \). We measure the similarity between two instances of a pair in the embedding space. During training, we denote an affinity matrix of all pairs within the current mini-batch as \( S \), whose \((i, j)\) element is the cosine similarity between the embeddings of the \(i\)-th sample and the \(j\)-th sample: \( v_i^T v_j \).

To facilitate further analysis, we delve into the pair-based DML methods by using the GPW framework described in [37]. With GPW, a pair-based function can be formulated in a unified pair-weighting form:

\[
L = \frac{1}{m} \sum_{i=1}^{m} \left( \sum_{y_j \neq y_i}^{m} w_{ij} S_{ij} - \sum_{y_j = y_i}^{m} w_{ij} S_{ij} \right),
\]

where \( m \) is the mini-batch size, and \( w_{ij} \) is the weight assigned to \( S_{ij} \). Eq. 1 shows that any pair-based method can be considered as a weighting scheme focusing on informative pairs. Here we list the weighting schemes of contrastive loss, triplet loss and MS loss.

- **Contrastive loss.** For each negative pair, \( w_{ij} = 1 \) if \( S_{ij} > \lambda \), otherwise \( w_{ij} = 0 \). The weights of all positive pairs are set to 1.

- **Triplet loss.** For each negative pair, \( w_{ij} = |P_{ij}| \), where \( P_{ij} \) is the valid positive set sharing the anchor. Formally, \( P_{ij} = \{x_{ik} | y_k = y_i, S_{ik} < S_{ij} + \eta \} \) and \( \eta \) is a predefined margin in triplet loss. Similarly, we can obtain the triplet weight for a positive pair.

- **MS loss.** Unlike contrastive loss and triplet loss which only assigns a weight with integer value, MS loss [37] is able to weight the pairs more properly by jointly considering multiple similarities. The MS weight for a negative pair is computed as:

\[
w_{ij} = \frac{e^{\beta (S_{ij} - \lambda)}}{1 + \sum_{k \in \mathcal{N}_i} e^{\beta (S_{ik} - \lambda)}},
\]

where \( \beta \) and \( \lambda \) are hyper-parameters, and \( \mathcal{N}_i \) is the valid negative set for the anchor \( x_i \). The MS weight for a positive pair can be computed similarly.

In fact, the main path of developing pair-based DML is to design a better weighting mechanism for pairs within a mini-batch. Generally, with a small mini-batch (e.g. 16 or 32), the sophisticated weighting schemes can perform much better (Figure 1, left). However, beyond the weighting scheme, the mini-batch size is also of great importance to DML. Figure 1 (left and middle) shows the R@1s of various pair-based methods are increased considerably by using a larger mini-batch on large-scale datasets. Intuitively, the number of negative pairs increase quadratically when the mini-batch size grows, which naturally provides more informative pairs. Instead of developing another sophisticated
but highly complicated algorithm to weight the informative pairs, our intuition is to simply collect sufficient informative negative pairs, where a simple weighting scheme, such as contrastive loss, can easily outperform the stage-of-the-art weighting approaches. This provides a new path that is straightforward yet more efficient to solve the hard mining problem in DML.

Naively, a straightforward solution to collect more informative negative pairs is to increase the mini-batch size. However, training deep networks with a large mini-batch is limited by GPU memory, and often requires massive data flow communication between multiple GPUs. To this end, we attempt to achieve the same goal by introducing an alternative approach using very low GPU memory and minimum computation burden. We propose a XBM module that allows the model to collect informative pairs over multiple past mini-batches, based on the “slow drift” phenomena as described below.

3.2. Slow Drift Phenomena

The embeddings of past mini-batches are usually considered out-of-date because the model parameters are changing throughout the training process [10, 32, 25]. Such out-of-date features are always discarded, but we learn that they can be an important resource, while being computation-free, by identifying the “slow drift” phenomena. We study the drifting speed of embeddings by measuring the difference of features for the same instance computed at different training iterations. Formally, the feature drift of an input $x$ at $t$-th iteration with step $\Delta t$ is defined as:

$$D(x, t; \Delta t) := ||f(x; \theta^t) - f(x; \theta^{t-\Delta t})||_2^2 \quad (2)$$

We train GoogleNet [33] from scratch with a contrastive loss, and compute the average feature drift for a set of randomly sampled instances at different steps: $\{10, 100, 1000\}$ (in Figure 3). The feature drift is consistently small, within only e.g. 10 iterations. For the large steps, e.g. 100 and 1000, the features change drastically at the early phase, but become relatively stable within about 3K iterations. Furthermore, when the learning rate decreases, the drift gets extremely slow. We define such phenomena as “slow drift”, which suggests that with a certain number of training iterations, the embeddings of instances can drift very slowly, resulting in marginal differences between the features computed at different training iterations.

Furthermore, we demonstrate that such “slow drift” phenomena can provide a strict upper bound for the error of gradients of a pair-based loss. For simplicity, we consider the contrastive loss of one single negative pair $L = v_i^T v_j$, where $v_i, v_j$ are the embeddings of current model and $v_j$ is an approximation of $v_j$.

**Lemma 1.** Assume $||v_j - \bar{v}_j||_2^2 < \epsilon$, $\bar{L} = v_i^T \bar{v}_j$, and $f$ satisfies Lipschitz continuous condition, then the error of gradients related to $v_i$ is,

$$||\frac{\partial L}{\partial \theta} - \frac{\partial \bar{L}}{\partial \theta}||_2^2 < C \epsilon, \quad (3)$$

where $C$ is the Lipschitz constant.

Proof and discussion of Lemma 1 are provided in Supplementary Materials. Empirically, $C$ is often less than 1.
Algorithm 1 Pseudocode of XBM.

```python
train network f conventionally with K epochs
initialize XBM as queue M
for x, y in loader: # x: data, y: labels
    anchors = f.forward(x)
    # memory update
    enqueue(M, (anchors.detach(), y))
    dequeue(M)
    # compare anchors with M
    sim = torch.matmul(anchors.transpose(), M.feats)
    loss = pair_based_loss(sim, y, M.labels)
    loss.backward()
    optimizer.step()
```

with the backbones used in our experiments. Lemma 1 suggests that the error of gradients is controlled by the error of embeddings under Lipschitz assumption. Thus, the “slow drift” phenomenon ensures that mining across mini-batches can provide negative pairs with valid information for pair-based methods.

In addition, we discover that the “slow drift” of embeddings is not a special phenomena in DML, and also exists in other conventional tasks, as shown in Supplementary Materials.

3.3. Cross-Batch Memory Module

We first describe our cross-batch memory (XBM) module, with model initialization and updating mechanism. Then we show that our memory module is easy to implement, can be directly integrated into existing pair-based DML framework as a plug-and-play module, by simply using several lines of codes (in Algorithm 1).

**XBM.** As the feature drift is relatively large at the early epochs, we warm up the neural networks with 1k iterations, allowing the model to reach a certain local optimal field where the embeddings become more stable. Then we initialize the memory module $M$ by computing the features of a set of randomly sampled training images with the warm-up model. Formally, $M = \{(\bar{v}_1, \bar{y}_1), (\bar{v}_2, \bar{y}_2), \ldots, (\bar{v}_M, \bar{y}_M)\}$, where $\bar{v}_i$ is initialized as the embedding of the $i$-th sample $x_i$, and $M$ is the memory size. We define a memory ratio as $R_M := M/N$, the ratio of memory size to the training size.

We maintain and update our XBM module as a queue: at each iteration, we enqueue the embeddings and labels of the current mini-batch, and dequeue the entities of the earliest mini-batch. Thus our memory module is updated with embeddings of the current mini-batch directly, without any additional computation. Furthermore, the whole training set can be cached in the memory module, because very limited memory is required for storing the embedding features, e.g. $512 \times d$ float vectors. See the other update strategy in Supplementary Materials.

**XBM augmented Pair-based DML.** We perform hard negative mining with our XBM on the pair-based DML. For a pair-based loss, based on GPW in [37], it can be cast into a unified weighting formulation of pair-wise similarities within a mini-batch in Eqn.(1), where a similarity matrix is computed within a mini-batch, $S$. To perform our XBM mechanism, we simply compute a cross-batch similarity matrix $\bar{S}$ between the instances of current mini-batch and the memory bank.

Formally, the memory augmented pair-based DML can be formulated as below:

$$L = \frac{1}{m} \sum_{i=1}^{m} L_i = \sum_{i=1}^{m} \left[ \sum_{\hat{y}_j \neq y_i}^{M} w_{ij} \bar{S}_{ij} - \sum_{\hat{y}_j = y_i}^{M} w_{ij} \bar{S}_{ij} \right],$$

(4)

where $\bar{S}_{ij} = v_i^T \bar{v}_j$. The memory augmented pair-based loss in Eqn.(4) is in the same form as the normal pair-based loss in Eqn.(1), by computing a new similarity matrix $\bar{S}$. Each instance in current mini-batch is compared with all the instances stored in the memory, enabling us to collect sufficient informative pairs for training. The gradient of the loss $L_i$ w.r.t. $v_i$ is,

$$\frac{\partial L_i}{\partial v_i} = \sum_{\hat{y}_j \neq y_i}^{M} w_{ij} \bar{v}_j - \sum_{\hat{y}_j = y_i}^{M} w_{ij} \bar{v}_j$$

(5)

and the gradients w.r.t. $v_i$, model parameters ($\theta$) can be computed through a chain rule:

$$\frac{\partial L_i}{\partial \theta} = \frac{\partial L_i}{\partial v_i} \frac{\partial v_i}{\partial \theta}$$

(6)

Finally, the model parameters $\theta$ are optimized through stochastic gradient descent. Lemma 1 ensures that the gradient error raised by embedding drift can be strictly constrained with a bound, which minimizes the side effect to the model training.

**Hard Mining Ability.** We investigate the hard mining ability of our XBM mechanism. We study the amount of valid negative pairs produced by our memory module at each iteration. A negative pair with non-zero gradient is considered as valid. The statistical result is illustrated in Figure 4. Throughout the training procedure, our memory module steadily contributes about 1,000 hard negative pairs per iteration, whereas less than 10 valid pairs are generated by the original mini-batch mechanism.

Qualitative hard mining results are shown in Figure 5. Given a bicycle image as an anchor, the mini-batch provides limited and different images, e.g. roof and sofa, as negatives. On the contrary, our XBM offers both semantically bicycle-related images and other samples, e.g. wheel and clothes. These results clearly demonstrate that the proposed
Figure 4. The number of valid negative examples from mini-batch and that from memory per iteration. Model is trained on SOP with \( R_M = 1 \), mini-batch size 64 and GoogleNet as the backbone.

Figure 5. Given an anchor image (yellow), examples of positive (green) and negative from mini-batch (gray) and that from memory (purple). Current mini-batch can only bring few valid negatives with less information, while our XBM module can provide a wide variety of informative negative examples.

XBM can provide diverse, related, and even fine-grained samples to construct negative pairs.

Our results confirm that (1) existing pair-based approaches suffer from the problem of lacking informative negative pairs to learn a discriminative model, and (2) our XBM module can significantly strengthen the hard mining ability of pair-based DML in a very simple yet efficient manner. See more examples in Supplementary Materials.

4. Experiments and Results

4.1. Implementation Details

We follow the standard settings in [22, 30, 23, 14] for fair comparison. Specifically, we adopt GoogleNet [33] as the default backbone network if not mentioned. The weights of the backbone were pre-trained on ILSVRC 2012-CLS dataset [27]. A 512-d fully-connected layer with \( l_2 \) normalization is added after the global pooling layer. The default embedding dimension is set as 512. For all datasets, the input images are first resized to \( 256 \times 256 \), and then cropped to \( 224 \times 224 \). Random crops and random flips are utilized as data augmentation during training. For testing, we only use the single center crop to compute the embedding for each instance as [22]. In all experiments, we use the Adam optimizer [15] with \( 5 \times 10^{-4} \) weight decay and the PK sampler (P categories, K samples/category) to construct mini-batches.

4.2. Datasets

Our methods are evaluated on three large-scale datasets for few-shot image retrieval. Recall@k is reported. The training and testing protocol follow the standard setups:

- **Stanford Online Products (SOP)** [22] contains 120,053 online product images in 22,634 categories. There are only 2 to 10 images for each category. Following [22], we use 59,551 images (11,318 classes) for training, and 60,502 images (11,316 classes) for testing.
- **In-shop Clothes Retrieval (In-shop)** contains 72,712 clothing images of 7,986 classes. Following [20], we use 3,997 classes with 25,882 images as the training set. The test set is partitioned to a query set with 14,218 images of 3,985 classes, and a gallery set having 3,985 classes with 12,612 images.
- **PKU VehicleID (VehicleID)** [19] contains 221,736 surveillance images of 26,267 vehicle categories, where 13,134 classes (110,178 images) are used for training. Following the test protocol described in [19], evaluation is conducted on a predefined small, medium and large test sets which contain 800 classes (7,332 images), 1600 classes (12,995 images) and 2400 classes (20,038 images) respectively.

4.3. Ablation Study

We provide ablation study on SOP dataset with GoogleNet to verify the effectiveness of our XBM module.

**Memory Ratio.** The search space of our cross-batch hard mining can be dynamically controlled by memory ratio \( R_M \). We illustrate the impact of memory ratio to XBM augmented contrastive loss on three benchmarks (in Figure 1, right). Firstly, our method significantly outperforms the baseline (with \( R_M = 0 \)), with over 20% improvements on all three datasets using various configurations of \( R_M \). Secondly, our method with mini-batch of 16 can achieve better performance than the non-memory counterpart using 256 mini-batch, e.g. with an improvement of 71.7%→78.2% on recall@1, while saving GPU memory considerably.

More importantly, our XBM can boost the contrastive loss largely with small \( R_M \) (e.g. on In-shop, **52.0% → 79.4% on recall@1 with \( R_M = 0.01 \)) and its performance is going to be saturated when the memory expands to a moderate size. It makes sense, since the memory with a small \( R_M \) (e.g. 1%) already contains thousands of embeddings to generate sufficient valid negative instances on large-scale datasets, especially fine-grained ones, such as In-shop or VehicleID. Therefore, our memory scheme can have consistent and stable performance improvements with a wide range of memory ratios.
Mini-batch Size. Mini-batch size is critical to the performance of many pair-based approaches (Figure 1, left). We further investigate its impact to our memory augmented pair-based methods (shown in Figure 6). Our method has a 3.2% performance gain by increasing its mini-batch size from 16 to 256, while the original contrastive method has a significantly larger improvement of 25.1%. Obviously, with the proposed memory module, the impact of mini-batch size is reduced significantly. This indicates that the effect of mini-batch size can be strongly compensated by our memory module, which provides a more principled solution to address the hard mining problem in DML.

With General Pair-based DML. Our memory module can be directly applied to the GPW framework. We evaluate it with contrastive loss, triplet loss and MS loss. As shown in Table 1, our memory module can improve the original DML approaches significantly and consistently on all benchmarks. Specifically, the memory module remarkably boosts the performance of contrastive loss by 64.0% → 77.8% and MS loss by 69.7% → 76.2%. Furthermore, with sophisticated sampling and weighting approach, MS loss has 16.7% recall@1 performance improvement over contrastive loss on VehicleID Large test set. Such a large gap can be simply filled by our memory module, with a further 5.8% improvement. MS loss has a smaller improvement because it weights extremely hard negatives heavily which might be outliers, while such a harmful influence is weakened by the equally weighting scheme of contrastive loss. For a detailed analysis see Supplementary Materials (SM).

The results suggest that (1) both straightforward (e.g. contrastive loss) and carefully designed weighting scheme (e.g. MS loss) can be improved largely by our memory module, and (2) with our memory module, a simple pair-weighting method (e.g. contrastive loss) can easily outperform the most sophisticated, state-of-the-art methods such as MS loss [37] by a large margin.

Memory and Computational Cost. We analyze the complexity of our XBM module on memory and computational cost. On memory cost, the XBM module \( \mathcal{M} (O(DM)) \) and affinity matrix \( \mathcal{S} (O(mM)) \) requires a negligible 0.2 GB GPU memory for caching the whole training set (Table 2). On computational complexity, the cost of \( \mathcal{S} (O(DmDM)) \) increases linearly with memory size \( M \). With a GPU implementation, it takes a reasonable 34% amount of extra training time w.r.t. the forward and backward procedure.

It is also worth noting that XBM does not act in the inference phase. It only requires 1 hour extra training time and 0.2GB memory, to achieve a surprising 13.5% performance gain by using a single GPU. Moreover, our method can be scalable to an extremely large-scale dataset, e.g. with 1 billion samples, since our XBM module can generate a rich set of valid negatives with a small-memory-ratio XBM, which requires acceptable cost.

4.4. Quantitative and Qualitative Results

In this section, we compare our XBM augmented contrastive loss with the state-of-the-art DML methods on three benchmarks on image retrieval. Even though our method can reach better performance with a larger mini-batch size (Figure 6), we only use 64 mini-batch which can be implemented on a single GPU with ResNet50 [11]. Since the backbone architecture and embedding dimension can effect the recall metric, we list the results of our method with var-
### Table 3. Recall@K (%) performance on SOP. 'G', 'B' and 'R' denotes applying GoogleNet, InceptionBN and ResNet50 as backbone respectively, and the superscript is embedding size.

<table>
<thead>
<tr>
<th>Method</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>GS-TRS [5]</td>
<td>75.0</td>
<td>83.0</td>
<td>74.1</td>
</tr>
<tr>
<td>BIER [23]</td>
<td>82.6</td>
<td>90.6</td>
<td>79.3</td>
</tr>
<tr>
<td>A-BIER [24]</td>
<td>86.3</td>
<td>92.7</td>
<td>83.3</td>
</tr>
<tr>
<td>VANet [4]</td>
<td>83.3</td>
<td>95.9</td>
<td>81.1</td>
</tr>
<tr>
<td>MS [37]</td>
<td>91.0</td>
<td>96.1</td>
<td>89.4</td>
</tr>
<tr>
<td>Divide [28]</td>
<td>R\textsuperscript{128}</td>
<td>87.7</td>
<td>92.9</td>
</tr>
<tr>
<td>MIC [26]</td>
<td>R\textsuperscript{128}</td>
<td>86.9</td>
<td>93.4</td>
</tr>
<tr>
<td>FastAP [2]</td>
<td>R\textsuperscript{128}</td>
<td>91.9</td>
<td>96.8</td>
</tr>
<tr>
<td></td>
<td>Cont. w/ M G\textsuperscript{512}</td>
<td>94.0</td>
<td>96.3</td>
</tr>
<tr>
<td></td>
<td>Cont. w/ M B\textsuperscript{512}</td>
<td>94.6</td>
<td>96.9</td>
</tr>
<tr>
<td></td>
<td>Cont. w/ M R\textsuperscript{128}</td>
<td>94.7</td>
<td>96.8</td>
</tr>
</tbody>
</table>

### Table 4. Recall@K (%) performance on In-Shop.

<table>
<thead>
<tr>
<th>Method</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>GS-TRS [5]</td>
<td>75.0</td>
<td>83.0</td>
<td>74.1</td>
</tr>
<tr>
<td>BIER [23]</td>
<td>82.6</td>
<td>90.6</td>
<td>79.3</td>
</tr>
<tr>
<td>A-BIER [24]</td>
<td>86.3</td>
<td>92.7</td>
<td>83.3</td>
</tr>
<tr>
<td>VANet [4]</td>
<td>83.3</td>
<td>95.9</td>
<td>81.1</td>
</tr>
<tr>
<td>MS [37]</td>
<td>91.0</td>
<td>96.1</td>
<td>89.4</td>
</tr>
<tr>
<td>Divide [28]</td>
<td>R\textsuperscript{128}</td>
<td>87.7</td>
<td>92.9</td>
</tr>
<tr>
<td>MIC [26]</td>
<td>R\textsuperscript{128}</td>
<td>86.9</td>
<td>93.4</td>
</tr>
<tr>
<td>FastAP [2]</td>
<td>R\textsuperscript{128}</td>
<td>91.9</td>
<td>96.8</td>
</tr>
<tr>
<td></td>
<td>Cont. w/ M G\textsuperscript{512}</td>
<td>94.0</td>
<td>96.3</td>
</tr>
<tr>
<td></td>
<td>Cont. w/ M B\textsuperscript{512}</td>
<td>94.6</td>
<td>96.9</td>
</tr>
<tr>
<td></td>
<td>Cont. w/ M R\textsuperscript{128}</td>
<td>94.7</td>
<td>96.8</td>
</tr>
</tbody>
</table>

### Table 5. Recall@K (%) performance on VehicleID.

<table>
<thead>
<tr>
<th>Method</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>GS-TRS [5]</td>
<td>75.0</td>
<td>83.0</td>
<td>74.1</td>
</tr>
<tr>
<td>BIER [23]</td>
<td>82.6</td>
<td>90.6</td>
<td>79.3</td>
</tr>
<tr>
<td>A-BIER [24]</td>
<td>86.3</td>
<td>92.7</td>
<td>83.3</td>
</tr>
<tr>
<td>VANet [4]</td>
<td>83.3</td>
<td>95.9</td>
<td>81.1</td>
</tr>
<tr>
<td>MS [37]</td>
<td>91.0</td>
<td>96.1</td>
<td>89.4</td>
</tr>
<tr>
<td>Divide [28]</td>
<td>R\textsuperscript{128}</td>
<td>87.7</td>
<td>92.9</td>
</tr>
<tr>
<td>MIC [26]</td>
<td>R\textsuperscript{128}</td>
<td>86.9</td>
<td>93.4</td>
</tr>
<tr>
<td>FastAP [2]</td>
<td>R\textsuperscript{128}</td>
<td>91.9</td>
<td>96.8</td>
</tr>
<tr>
<td></td>
<td>Cont. w/ M G\textsuperscript{512}</td>
<td>94.0</td>
<td>96.3</td>
</tr>
<tr>
<td></td>
<td>Cont. w/ M B\textsuperscript{512}</td>
<td>94.6</td>
<td>96.9</td>
</tr>
<tr>
<td></td>
<td>Cont. w/ M R\textsuperscript{128}</td>
<td>94.7</td>
<td>96.8</td>
</tr>
</tbody>
</table>

Figure 7 shows that our memory module promotes the learning of a more discriminative encoder. For example, at the first row, our model is aware of the deer under the lamp which is a specific character of the query product, and retrieves the correct images. In addition, we also present some bad cases in the bottom rows, where our retrieved results are visually closer to the query than that of baseline model. See more visualizations in SM.

### 5. Conclusions

We have presented a conceptually simple, easy to implement, and memory efficient cross-batch mining mechanism for pair-based DML. In this work, we identify the “slow drift” phenomena that the embeddings drift exceptionally slow during the training process. Then we propose a cross-batch memory (XBM) module to dynamically update the embeddings of instances of recent mini-batches, which allows us to collect sufficient hard negative pairs across multiple mini-batches, or even from the whole dataset. Without bells and whistles, the proposed XBM can be directly integrated into a general pair-based DML framework, and improve the performance of existing pair-based methods significantly on image retrieval. In particular, with our XBM, a contrastive loss can easily surpass state-of-the-art methods [37, 26, 2] by a large margin on three large-scale datasets. This paves a new path in solving for hard negative mining which is a fundamental problem for various computer vision tasks. Furthermore, we hope that the dynamic memory mechanism can be extended to improve a wide variety of machine learning tasks because “slow drift” is a general phenomenon not only occurring in DML.
References


