# Stochastic Class-based Hard Example Mining for Deep Metric Learning

Yumin Suh[1] Bohyung Han[1] Wonsik Kim[2] Kyoung Mu Lee[1]
[1]ECE & ASRI, Seoul National University, Korea [2]Samsung Research, Samsung Electronics
{n12345, bhhan, kyoungmu}@snu.ac.kr wonsik16.kim@samsung.com

## Abstract

*Performance of deep metric learning depends heavily on the capability of mining hard negative examples during training. However, many metric learning algorithms often require intractable computational cost due to frequent feature computations and nearest neighbor searches in a large-scale dataset. As a result, existing approaches often suffer from trade-off between training speed and prediction accuracy. To alleviate this limitation, we propose a stochastic hard negative mining method. Our key idea is to adopt class signatures that keep track of feature embedding online with minor additional cost during training, and identify hard negative example candidates using the signatures. Given an anchor instance, our algorithm first selects a few hard negative classes based on the class-to-sample distances and then performs a refined search in an instance-level only from the selected classes. As most of the classes are discarded at the first step, it is much more efficient than exhaustive search while effectively mining a large number of hard examples. Our experiment shows that the proposed technique improves image retrieval accuracy substantially; it achieves the state-of-the-art performance on the several standard benchmark datasets.*

## 1. Introduction

Deep metric learning is a fundamental problem applicable to various tasks in computer vision including image retrieval [14, 23, 24, 6, 35], person re-identification [38, 7], face recognition [20, 30], and many others. The goal of deep metric learning is to approximate a feature embedding function that maps data—images in our domain—onto a common feature space. After learning, visually similar images are supposed to be clustered while the ones with heterogeneous contents are expected to be located apart from each other. To meet this requirement, one can consider a triplet loss [20], which is defined on all the triplets of images in the training set. The triplet loss penalizes the cases that the distances between the images in the same classes are larger than the ones between images with different labels.

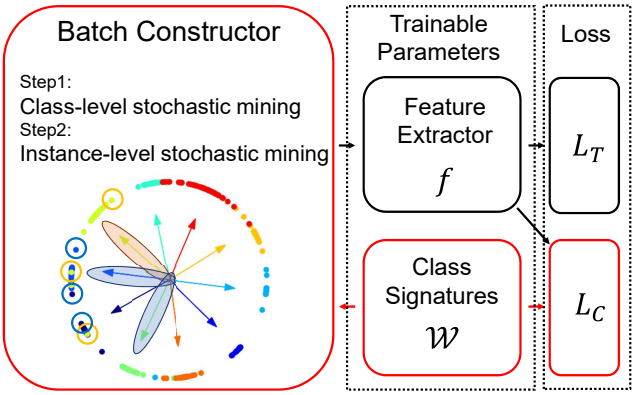


Figure 1. Overview of the training procedure with our stochastic hard example mining

A critical drawback of the triplet loss is the high computational cost in identifying hard negative examples for training, which is partly because embedding functions are changing throughout training procedure and one needs to search for the new triplets violating the desired constraints in each iteration [23, 6, 30, 21, 3, 39]. A naïve implementation of a metric learning algorithm based on triplet loss requires a forward propagation of the whole training dataset through feature extractor and distance computation between every pair of examples in each iteration, which is computationally infeasible in a large-scale datasets.

Existing approaches have explored two directions to reduce computational overhead while maintaining accuracy. One is to search for hard negative examples only within individual mini-batches [20, 7] constructed by random sampling; this strategy requires a large mini-batch size, *e.g.*, a few thousands in case of [20], to ensure to have a sufficient number of hard examples. The other is to exploit a fixed approximate global structure of dataset by using precomputed features [18, 6]. However, this approach is problematic because the representation of each example changes during training procedure and, consequently, the global structure is updated as well.

We propose a stochastic hard example mining technique to tackle the limitations in the existing approaches. Specif-

ically, we identify the nearest neighbor classes from a set of stochastically sampled instances in an anchor class, and draw hard examples from the classes only. Our key idea is to use class signatures during training, which track the change of the embedding function, and update the hard negative classes based on them during training. Assuming the instance-level feature embeddings have small distances to their class representatives, a sample-to-sample distance is approximated by a sample-to-class distance while circumventing the need to compute a distance between every pair of samples per iteration. The class signatures are updated at every iteration in an efficient way. Since the idea is much more efficient than exhaustive search, it allows us to change embedding functions adaptively and update image representations in every iteration.

According to our experiment, the proposed hard example mining technique improves accuracy in image retrieval tasks compared to several baseline methods on the standard benchmark datasets including CARS-196 [10], CUB-200-2011 [29], In-shop retrieval [11] and Stanford online products [25]. In addition, by adopting a cross-channel correlation technique [4] to further enhance the representation power, our method achieves the state-of-the-art performance in the standard datasets.

## 2. Related Works

Hard example mining is a popular technique to speed up convergence and enhance the discriminative power of the learned embeddings in deep metric learning [20, 7, 2, 23]. To reduce the computational overhead to identify hard examples, existing works have explored two directions: an exact search within each minibatch [20, 7, 19] and an approximate search from the whole dataset. In this paper, we focus on exploiting the class membership information to mine hard negatives from the whole dataset efficiently.

There are several approaches that approximate distances between instances by class-level distances to reduce the computational cost for hard example mining. Their common strategy is to identify the neighboring classes with a low computational cost and construct a minibatch based only on the classes [19, 30, 32]. The underlying assumption is that the neighboring classes are likely to contain hard negative examples.

To this end, early works used the precomputed embeddings to find the neighboring classes and fixed them during the training. However, this approach suffers from poor embedding quality and results in inefficient mining performance since the embedding space is gradually updated during the training. To figure out this issue, the neighboring class adaptation has been explored [23, 21, 18, 5, 22]. Among the techniques in this category, a naïve approach is to use an embedding of a random sample to represent its class label [23, 21]. The representation quality can be improved by using the average embedding of the examples in a class [18, 5]. However, it requires high computational cost because it repetitively feed forward the whole training samples through the network for adaptation. Also, since they commonly use coarse class-level approximation, they often fail to identify the hardest examples, especially when intra-class variation is large. Smirov *et al*. [22] attempt to mitigate this problem by constructing a minibatch with the hardest examples among the members in the mined hard positive and negative classes. However, they employ precomputed features to detect the hardest examples with each class, which is inaccurate due to embedding space updates.

We circumvent the abovementioned problems by introducing class signatures, which track the changes of embedding space with minor additional cost. Specifically, our algorithm first selects a few hard negative classes based on the class-to-instance distances and then performs a refined search in an instance-level only from the selected classes. As most of the classes are discarded at the first stage, it is more efficient than exhaustive search while effectively mining a large number of hard examples. The methods proposed by Movshovitz-Attias *et al*. [14] and Wen *et al*. [34] are related to ours in a sense that class representatives are jointly trained with the feature extractor. However, their goal is to formulate new losses using the class representatives whereas we use them for hard negative mining.

Recently, a few generation-based approaches have been proposed to train hard example generators to avoid costly mining process [1, 3, 39]. For a given anchor instance, they generate a fake example that looks similar to the anchor class but belongs to a negative class selected randomly. Although they attempt to generate hard examples in the randomly selected classes, their impact may not be significant if the selected classes can be trivially differentiated.

## 3. Proposed Method

### 3.1. Overview

Our goal is to obtain an optimal feature extractor that maps an image $\mathbf{I}$ to a vector $\mathbf{x}$ by deep metric learning. The desired condition of the learned function is to make distances between the representations of similar images small while locating dissimilar ones apart from each other. The notion of similarity between two images is typically defined by their semantic information, which is often derived from whether their class labels are same or not. A pair of images with the same label are considered to be positive while a pair with different labels are called negative.

Given a triplet of samples, $\tau = [\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n] \in \mathcal{T}$, which consists of an anchor $\mathbf{x}_a$ and a positive sample $\mathbf{x}_p$ with label $y_a$, and a negative sample $\mathbf{x}_n$ with label $y_n$, the triplet loss penalizes the case that the distance from an anchor to a positive sample is not sufficiently smaller than the distance

to a negative one, which is formally given by

$$\ell_T(\tau) = \max(0, d(\mathbf{x}_a, \mathbf{x}_p) - d(\mathbf{x}_a, \mathbf{x}_n) + m), \quad (1)$$

$$L_T(\mathcal{X}) = \frac{1}{\sum_{\tau\in\mathcal{T}} \omega(\tau)} \sum_{\tau\in\mathcal{T}} \omega(\tau)\ell_T(\tau), \quad (2)$$

where $d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|^2$, $m$ is the margin for the difference between the distances from $\mathbf{x}_a$ to $\mathbf{x}_p$ and $\mathbf{x}_n$, and $\omega(\tau)$ denotes an importance of triplet $\tau$. When every triplet has a same weight, *i.e.*, $w(\tau) = 1$, Eq. (2) becomes identical to the conventional triplet loss, which is given by

$$L_T(\mathcal{X}) = \frac{1}{|\mathcal{T}|} \sum_{\tau\in\mathcal{T}} \ell_T(\tau). \quad (3)$$

Based on the observation that weighing more on the semi-hard triplets enhances the performance [7], we use following binary weight for $\omega(\tau)$ in Eq.(2) for all the experiments:

$$\omega(\tau) = \begin{cases} 1, & \text{if } \ell_T(\tau) > 0 \\ 0, & \text{otherwise} \end{cases}. \quad (4)$$

In our experiments, the weighting scheme in Eq. (4) consistently improves the accuracy with respect to the baseline triplet loss based on the uniform weights.

To facilitate deep metric learning based on triplet loss, each minibatch should contain many hard triplet examples while diversifying examples over iterations to avoid overfitting. Our main idea is to learn and use the class signature vectors, which represent individual classes in a discriminative manner, to reduce the computational overhead for hard triplet search. Intuitively, if two classes are located closely in an embedding space, the instances in a class are likely to be hard negatives with respect to the other. For the purpose, we first search for the nearest neighbor classes from an anchor class, which is based on the distances from samples in the anchor class to the rest of classes. Then, we seek for the nearest neighbors in an instance-level, only among the examples in the identified nearest neighbor classes. We perform both class- and instance-level search stochastically to increase sample diversity within a minibatch.

### 3.2. Neighbor Class Mining by Class Signatures

Given an anchor class, we aim to find the nearest neighbor classes based on their signatures, denoted by $\mathcal{W} = \{\mathbf{w}_1, \mathbf{w}_2, \cdots \mathbf{w}_{|\mathcal{Y}|}\}$, which are optimized by making the signature of each class coherent to the embeddings of its members while maximizing the discriminativeness between the class signatures.

For a given instance $\mathbf{x}$ with label $y_\mathbf{x}$, a sample-to-class similarity function $S(\mathbf{w}_c, \mathbf{x})$ should be large if the label of $\mathbf{x}$ is $c$, and small otherwise. Hence, to find the nearest neighbor classes based on the sample-to-class similarity, we define the following loss function:

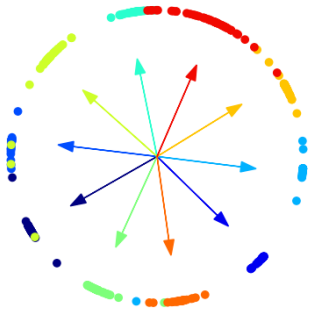

Figure 2. A 2D embedding example of class signatures and individual instances for the MNIST dataset. Circles and arrows denote instances and class signatures, respectively, which are given by our model. Class labels are color-coded.

$$\begin{aligned} L_C(\mathcal{W}, \mathcal{X}) &= -\frac{1}{N} \sum_{\mathbf{x}\in\mathcal{X}} \log\left(P(\mathbf{x}; \mathcal{W})\right) \quad (5) \\ &= -\frac{1}{N} \sum_{\mathbf{x}\in\mathcal{X}} \log\left(\frac{\exp(S(\mathbf{w}_{y_\mathbf{x}}, \mathbf{x}))}{\sum_c \exp(S(\mathbf{w}_c))}\right) \\ &= -\frac{1}{N} \sum_{\mathbf{x}\in\mathcal{X}} \log\left(\frac{\exp(\cos\theta_{y_\mathbf{x}})}{\sum_c \exp(\cos\theta_c)}\right), \quad (6) \end{aligned}$$

where $\theta_c = \angle(\mathbf{w}_c, \mathbf{x})$. It can also be interpreted as the log likelihood of $\mathbf{x}$ with respect to class $y_\mathbf{x}$. Note that we use a $\ell_2$-normalized feature vector $\mathbf{x}$, following the common trick to increase accuracy [38, 17, 31]. By constraining the class signatures to have a unit norm, *i.e.*, $\|\mathbf{w}_c\|_2 = 1$ for all $c$'s, we can use cosine similarity measure to compare the representations of instances and class signatures. Ideally, $\theta_c = 0$ if $c = y_\mathbf{x}$ and $\theta_c = \pi/2$ otherwise. Figure 2 illustrates a distribution of the instances and their class signatures trained on the MNIST dataset.

Given $\mathcal{W}$, we can approximate the similarity between two samples, $\mathbf{x}$ and $\mathbf{x}'$ with labels $y$ and $y'$, respectively, by the corresponding class-to-class similarity and class-to-sample similarity as

$$S(\mathbf{x}, \mathbf{x}') \approx S(\mathbf{x}, \mathbf{w}_{y'}) \approx S(\mathbf{w}_y, \mathbf{w}_{y'}), \quad (7)$$

where $S(\cdot, \cdot)$ is a similarity between two vectors. Figure 3 shows the average sample-to-sample distance between classes with respect to the rank of their class-to-class distances in In-shop retrieval and SOP datasets. It illustrates that the expected sample-to-sample distance tends to increase as the class-to-class distance does. This result implies that the nearest classes obtained based on class-to-class distances are of reasonably good quality and also supports that our class-level approximation for hard example mining is effective.

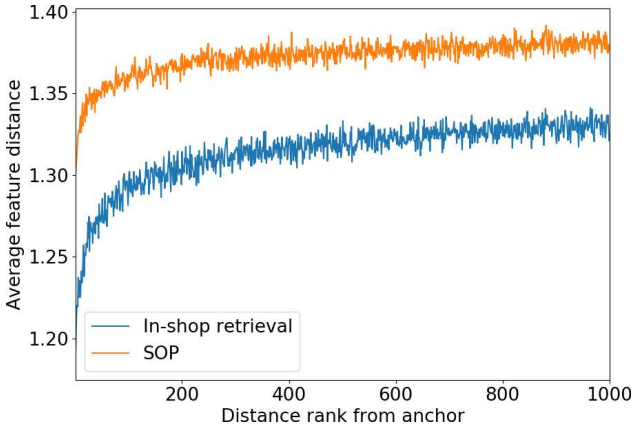


Figure 3. The average sample-to-sample distance between classes with respect to the rank of their class-to-class distances in In-shop retrieval and SOP dataset, averaged over 100 random anchor classes. This result shows that expected sample-to-sample distance is proportional to the corresponding class-to-class distance.

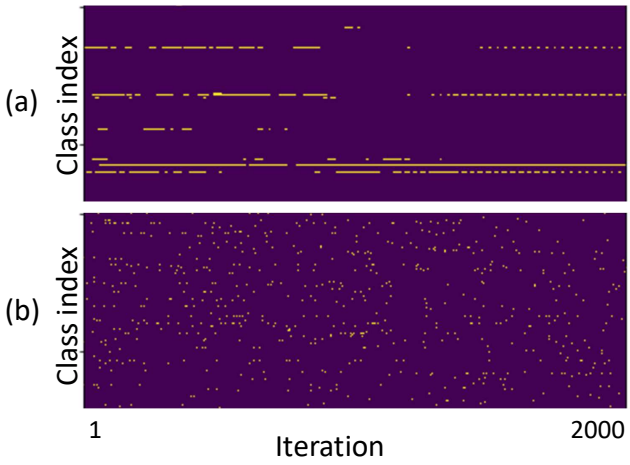


Figure 4. For a given anchor class $c_a$, the diversity of chosen negative classes in each minibatch are illustrated in the In-shop retrieval dataset. The $x$-axis and $y$-axis correspond to the training iteration and class index, respectively. The highlighted cells denote the selected classes. It shows that the selected classes in a minibatch are diverse over the iteration in the stochastic hard example mining (b) while mostly fixed in the deterministic class-level mining in Algorithm 1 (a).

## 3.3. Batch Construction

We now discuss how to construct minibatches and facilitate training using the identified hard examples.

### 3.3.1 Baseline protocol [38]

We adopt the approach in Zhao *et al*. [38] as the baseline batch construction protocol. At each iteration, it constructs a minibatch by first randomly sampling $K$ classes and then randomly sampling $\eta$ images from each of the classes, which makes the minibatch size $M = K\eta$. The loss is given by every possible triples composable from a minibatch. This approach is popularly used [38, 23, 7] due to its simplicity and performance. Note that [23] is a special case when $K = M/2$ and $\eta = 2$.

### 3.3.2 Improving baseline by hard class mining

We increase the expected number of hard triplets in each minibatch compared to the baseline protocol by composing each minibatch with the instances randomly sampled from an anchor class and its $(K-1)$-nearest classes. Formally, given an anchor class $c_a$ at each iteration, its $(K-1)$-nearest classes $\mathcal{N}$ are identified by the following optimization:

$$\underset{\mathcal{N}\subset\mathcal{Y}}{\text{argmax}} \sum_{c\in\mathcal{N}} S(\mathbf{w}_c, \mathbf{w}_{c_a}) \quad (8)$$
$$\text{subject to } c_a \notin \mathcal{N} \text{ and } |\mathcal{N}| = K-1,$$

where $\mathcal{Y}$ is a set of class labels. Once the nearest classes are selected, $\eta$ instances are randomly sampled from each of the classes, which constructs a minibatch $\mathcal{B}$. Algorithm 1 describes the detailed procedure.

A drawback of this approach is that the variation of sample-to-sample distances within a class becomes large when its intra-class variation is large. As a result, the class similarity measured by class signatures has too much error to approximate the similarity of instances belonging to the classes. A more serious issue is the limited diversity of the sampled hard negative classes in the naïve approach. As illustrated in Figure 4(a), given an anchor class $c_a$, only a few hard negative classes are sampled repeatedly over iterations while most of them fail to be selected. It implies that only a limited number of negative classes are used for training for each anchor class. To alleviate these limitations, we propose a stochastic hard example mining approach.

### 3.3.3 Stochastic hard example mining

Instead of relying only on class signatures to find nearest neighbor classes, we employ the distances computed between a set of instances in the anchor class and the class signatures of the rest of the classes. At every iteration, we first randomly select an anchor class and a subset of examples from it; given an anchor class $c_a$, we construct $\mathcal{B}_{c_a}$, which is a subset of minibatch $\mathcal{B}$, with $\eta$ examples randomly sampled from the class. Then, we search for a pool of nearest neighbor classes based on instance-to-class distances between the anchor instances in $\mathcal{B}_{c_a}$ and the set of class signatures $\mathcal{W}\backslash\{c_a\}$. The nearest neighbor samples are finally identified by collecting the sampled instances within this pool. The use of instance-to-class distances is effective to cope with the classes with a large intra-class variation

**Algorithm 1** Improved baseline with class-level mining

**Parameters** $K, \eta$

**for** $t = 1 : T$ **do**
  Random sample anchor class $c_a$ from $\mathcal{Y}$
  $\mathcal{B} \leftarrow$ Sample $\eta$ instances from $\{\mathbf{x}|y_{\mathbf{x}} = c_a\}$
  Get $\mathcal{N}$ of size $(K-1)$ by Eq. (8)
  **for** $c \in \mathcal{N}$ **do**
    $\mathcal{B}_c \leftarrow$ Sample $\eta$ instances from $\{\mathbf{x}|y_{\mathbf{x}} = c\}$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{B}_c$
  **end for**
  Perform one iteration of training to minimize the loss Eq. (12) using minibatch $\mathcal{B}$
**end for**

and helpful to diversify the identified nearest classes. In addition, the refined instance-level search only from the selected classes reduces the computational cost.

For a formal description, let us first define a similarity $S_g(\cdot,\cdot)$ between a set of vectors $\mathcal{U}$ and a vector $\mathbf{v} \in \mathcal{V}$, which is given by

$$S_g(\mathcal{U}, \mathbf{v}) \equiv \max_{\mathbf{u}\in\mathcal{U}} S(\mathbf{u}, \mathbf{v}), \tag{9}$$

where $\mathcal{U}$ and $\mathcal{V}$ are sets of vectors.

We first search for nearest neighbor classes from an anchor class based on distances from samples in the anchor class to the signatures of the rest of classes, which is given by

$$\begin{aligned} \mathcal{P}_c = \underset{\mathcal{V}'\subset\mathcal{W}\setminus\{\mathbf{w}_{c_a}\}}{\arg\max} \sum_{\mathbf{v}\in\mathcal{V}'} S_g(\mathcal{B}_{c_a}, \mathbf{v}) \\ \text{subject to } \ |\mathcal{V}'| = \alpha(K-1), \end{aligned} \tag{10}$$

where $\mathcal{W}$ denotes a set of class signatures and $\alpha(\geq 1)$ is chosen randomly at each iteration, for sample diversification. In a nutshell, $\mathcal{P}_c$ consists of top $\alpha(K-1)$ negative class signatures, which have largest similarities $S_g$ from the set of sampled anchor instances, $\mathcal{B}_{c_a}$. We use $\mathcal{P}_c$ as class candidates for the refined search, thereby reducing the number of candidate instances. For a given $\mathcal{P}_c$, to diversify the training examples, we randomly sample $(K-1)\eta$ instances among the instance pool, $\mathcal{P}_s$ by

$$\begin{aligned} \mathcal{P}_s = \underset{\mathcal{V}'\subset\{\mathbf{x}|y_{\mathbf{x}}=c, c\in\mathcal{P}_c\}}{\arg\max} \sum_{\mathbf{v}\in\mathcal{V}'} S_g(\mathcal{B}_{c_a}, \mathbf{v}) \\ \text{subject to } \ |\mathcal{V}'| = \beta(K-1)\eta, \end{aligned} \tag{11}$$

where $\beta(\geq 1)$ is to increase the number of candidates for the final selection of examples to be included in a minibatch. In a nutshell, $\mathcal{P}_s$ consists of top $\beta(K-1)\eta$ samples from the class set $\mathcal{P}_c$, which have largest similarities $S_g$ from the set of sampled anchor instances, $\mathcal{B}_{c_a}$. Note that each minibatch contains $K\eta$ instances, which include the elements in $\mathcal{B}_{c_a}$ and the subsampled instances from the nearest neighbor classes. Algorithm 2 summarizes the overall training process.

**Algorithm 2** Training with stochastic hard example mining

**Parameters** $K, \eta$

**for** $t = 1 : T$ **do**
  Random sample $\alpha(\geq 1)$
  Random sample an anchor class $c_a$
  $\mathcal{B}_{c_a} \leftarrow$ Sample $\eta$ instances from $\{\mathbf{x}|y_{\mathbf{x}} = c_a\}$
  $\mathcal{B} \leftarrow \mathcal{B}_{c_a}$
  Get a class pool $\mathcal{P}_c$ using Eq. (10).
  Get a instance pool $\mathcal{P}_s$ using $\mathcal{P}_c$ and Eq. (11).
  $\mathcal{B}_a \leftarrow$ Random sample $(K-1)\eta$ elements from $\mathcal{P}_s$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{B}_a$
  Perform one iteration of training to minimize the loss Eq. (12) using minibatch $\mathcal{B}$
**end for**

Figure 4(b) illustrates the diversity of the selected negative classes over iterations. Compared with Figure 4(a), the proposed stochastic hard example mining strategy allows to identify more diverse classes over iterations and learns the model more efficiently.

#### 3.3.4 Computation efficiency

Suppose that there are $n$ samples in each of $|\mathcal{Y}|$ classes. The naïve approach to find hard negative examples would require to scan the whole dataset, which requires $|\mathcal{Y}|n$ forward passes of the network in total for feature extraction. In contrast, our algorithm reduces the computational cost for feature extraction by first identifying a small set of nearest classes based on the learned class signatures and then searching for the nearest instances only within the set. Consequently, the proposed method extracts features only from $cn$ samples, where $c$ is a number of the candidate nearest classes. Note that one needs to repetitively recompute features during training due to gradual update of embedding space. Since the feature extraction dominates the computation time, our approach still has the advantage over the naïve one despite the potentially large value of $n$.

### 3.4. Loss

We jointly train the parameters of feature extractor and the class signatures $\mathcal{W}$ to minimize both triplet loss and the class signature loss. We hope that joint learning of triplet and classification loss improves the accuracy of both tasks as reported in [6, 40]. The loss function is formally given by

$$L(\mathcal{W}, \mathcal{X}) = L_T(\mathcal{X}) + L_C(\mathcal{W}, \mathcal{X}), \tag{12}$$

where $L_T(\cdot)$ and $L_C(\cdot,\cdot)$ denote the triplet and the class signature loss, respectively. Note that the gradient from the

class signature loss is back-propagated all the way down to the feature extractor.

### 3.5. Feature Extractor

Our baseline feature extractor is almost identical to the original version [6, 14] based on Inception_v1 [27] except that it has a batch normalization layer after the last average pooling layer.

Our feature extractor improves the baseline by introducing the second-order pooling [4, 26, 13]. The second order pooling actually exploits cross-channel correlation, which turns out to be useful to improve several computer vision tasks including classification [4, 13]. For an input feature map $\mathbf{G} \in \mathbb{R}^{w \times h \times c}$, the second-order pooling is defined by

$$\text{Pooling}(\mathbf{G}) = \frac{1}{hw} \sum_{xy} \text{vec}(\mathbf{g}_{xy} \otimes \mathbf{g}_{xy}), \qquad (13)$$

where $\mathbf{g}_{xy}$ is a feature vector in $\mathbf{G}$ at position $(x, y)$, $\otimes$ is the outer-product operator and $\text{vec}(\cdot)$ denotes the vectorization of an input. We adopt a technique called Tensor Sketch [16, 4] to reduce the computational overhead required for handling outer-product in Eq. (13). Refer to [16] for the details. We found that simply increasing the image resolution increases the accuracy by better exploiting the spatial information. To enlarge the resolution of the input feature map to the second-order pooling layer, we drop the layers from Inception_v1 *5a* block. More specifically, we use the network from the input to the Inception_v1 *4e* block, followed by a $1 \times 1$ convolution (512-dim) and a batch normalization layer to extract the feature map $\mathbf{G}$. Then, the second-order pooling is performed over the extracted feature map followed by $\ell_2$-normalization.

### 3.6. Image Retrieval

We perform image retrieval based on the similarity between the representations of a pair of images. The triplet loss is employed to learn the representation, which incorporates stochastic hard negative mining to facilitate training. The similarity between a pair of images is conceptually given by

$$\text{sim}(\mathbf{G}, \mathbf{G}') = \frac{1}{hw} \sum_{xyx'y'} \left\langle \mathbf{g}_{xy}, \mathbf{g}'_{x'y'} \right\rangle^2, \qquad (14)$$

where $\mathbf{G}$ and $\mathbf{G}'$ are the feature maps of two images. Note that each term is non-negative and thus any local descriptor $\mathbf{g}_{xy}$ with non-zero strength increases the image similarity. In our experiments, this implicitly enforced the local features located at the background regions to have small $\ell_2$-norm to avoid an adversarial effect.

Table 1 shows that the model using cross-channel correlation consistently improves accuracy from the baseline. In addition, accuracy increases as the input resolution grows from $224 \times 224$ to $336 \times 336$. Compared to the baseline, this change does not increase the number of parameters while taking about 2.25 times more computation in FLOPs.

Table 1. R@1 (%) in CARS-196 and CUB-200-2011 dataset for different feature extractors input resolutions

| | Method | $224 \times 224$ | $336 \times 336$ |
|---|---|---|---|
| CARS-196 | Inception_v1 | 83.6 | 89.7 |
| | Channel-correlation | 86.9 | 91.3 |
| CUB-200-2011 | Inception_v1 | 55.1 | 60.9 |
| | Channel-correlation | 58.1 | 65.2 |

## 4. Experiments

This section describes our setting for experiment and reports the performance of our algorithm compared to existing methods.

### 4.1. Datasets

The proposed approach is tested on the following standard benchmark datasets for image retrieval. We do not use ground-truth bounding box annotations in all experiments.

**CARS-196** [10] This dataset consists of $16,183$ images in 196 different classes of cars. We used the first 98 classes for training ($8,052$ images) and the rest of classes for testing ($8,131$ images), following the setting in [25].

**CUB-200-2011** [29] This dataset is based on the images of 200 different bird species. We used the first 100 classes for training ($5,864$ images) and the other 100 classes for testing ($5,924$ images), following the previous work [25].

**In-shop retrieval** [11] This dataset has $54,642$ images in $11,735$ classes of clothing items. We used $3,997$ classes for training ($25,882$ images) and another $3,985$ classes for testing ($28,760$ images), which follows the previous work [11]. In the test set, $14,218$ images are used as queries and the remaining $12,612$ images are used as the database for retrieval.

**Stanford onilne products (SOP)** [25] This is a large-scale dataset with $120,053$ product images of $22,634$ classes. Training split is composed of $59,551$ images in $11,318$ classes while $11,316$ classes with $60,499$ images are used for testing.

### 4.2. Implementation Details

We describe the details of our implementation including data augmentation, hyperparameter setting, and optimization method.

**Data augmentation** During the training, we resize an input image to $256 \times 256$ and perform standard random crops to $224 \times 224$ with random horizontal flipping for data augmentation. For the testing, we first resize input images to $256 \times 256$ and then crop the center to $224 \times 224$.

**Hyperparameters** We set the batch size to 60 ($M = 60$). Given the batch size, we choose $\eta = 10$ for the small datasets (CARS-196 and CUB-200-2011), which has 60 to 80 samples per class, and $\eta = 5$ for the larger datasets (In-shop retrieval and Stanford Online Products), which only has 5 to 7 examples per class. In Algorithm 2, $\alpha$ is randomly chosen from $\{3, 4, 5\}$ at each iteration with $\beta = 5$ for CARS-196 and CUB-200-2011, and from $\{15, 20, 25\}$ with $\beta = 1$ for In-shop retrieval and SOP.

**Optimization** We use Adam [9] for optimization. The initial learning rate and weight decay are set to $1 \times 10^{-4}$ and $5 \times 10^{-4}$, respectively. The learning rate is exponentially decayed to $1 \times 10^{-7}$ from epoch 200 to 400.

### 4.3. Evaluation Metrics

We employ the Recall@$K$ (R@$K$) metric for evaluation. For each sample, $K$ nearest neighbors are retrieved from the remaining test set. If retrieved images include at least one sample from the same class, it is considered to be correct. The Recall@$K$ metric measures the number of correct sample over entire sample. For the distance measure, Euclidean distance is used, which is equivalent to the cosine distance in our case, because the feature are $\ell_2$-normalized.

### 4.4. Effect of the Stochastic Hard Example Mining

We evaluate the proposed stochastic hard example mining method in aspect of the hardness.Figure 5 illustrates the number of triplets with non-zero losses within the minibatch constructed in each iteration of training. Compared to the random sampling [38] and class-level mining (Algorithm 1), the proposed stochastic hard negative mining strategy (Algorithm 2) is obviously more effective to find desirable triplets that have non-zero losses.

To evaluate the effect of the proposed stochastic hard class mining, we compare our method (Algorithm 2) with baseline protocol [38], hard class mining (Algorithm 1), and two variants of ours. Table 2 present the accuracies of all the compared methods. It is known that training the feature extractor with joint loss of triplet and classification enhances the accuracy from the baseline which uses only triplet loss. To discriminate the effect of the hard sample mining and the addition of loss (Eq. 5), we show both results, only with mining (*var2*) and the full model. In *var2*, the class signature loss does not back-propagate to the feature extractor. Another variant (*var1*) replaces the proposed class signature with the class-wise average of features, extracted from the Inception_v1 pre-trained on the ImageNet dataset. Table 2 shows that the proposed method consistently improves the accuracy compared to the random sampling baseline in all the datasets. We reimplemented three existing mining methods [6, 21, 30] for comparison with the proposed algorithm. For fair comparison, we used the same setting in the algorithms except for the mining strategies, where the feature extractor is trained by the triplet loss without the signature loss. As shown in Table 2, the proposed approach (*var2*) achieves the best accuracy in general among the compared methods.

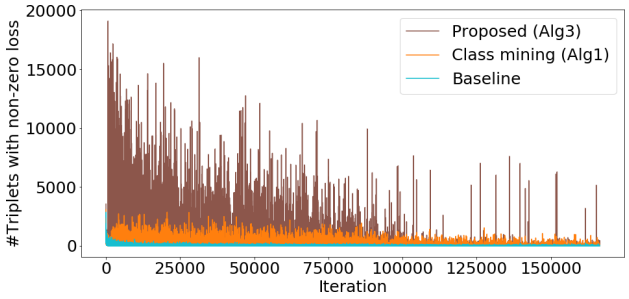


Figure 5. Comparison of the number of triplets with non-zero loss in a minbatch during training.

### 4.5. Comparison with the Existing Methods

We compare our method to the state-of-the-art methods in Table.3-6. Since the backbone network architecture affects the retrieval accuracy, we show the architecture in the parenthesis. When compared to the existing hard sample mining method, SmartMining [6], ours achieve higher accuracy. Note that they provided the result only on small datasets. We also report the accuracy of proposed mining method applied to the channel-correlation model with higher input image resolution of $336 \times 336$ (Channel-correlation). In every dataset, our method outperforms the previous state-of-the-art with comparable compuational cost.

## 5. Conclusion

We proposed a stochastic hard negative example mining method for triplet loss. Unlike existing works, our method tracks the change of neighbor relations between classes in during training with minor additional cost. Based on the relations, it identifies hard examples through a coarse-to-fine search and stochastically samples hard examples from a pool of candidates to diversify the examples used for training. Experimental results show that our method consistently improves the baseline.

**Acknowledgement** This work was partially supported by Samsung Resaearch and the Visual Turing Test project (IITP-2017-0-01780) from the Ministry of Science and ICT of Korea.

## References

[1] Shuo Chen, Chen Gong, Jian Yang, Xiang Li, Yang Wei, and Jun Li. Adversarial metric learning. In *IJCAI*, 2018. 2

Table 2. Recall@K (%) comparison with the baselines

| | | CARS-196 | | | | CUB-200-2011 | | | | In-shop retrieval | | | | Stanford online products | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $K$ | | 1 | 2 | 4 | 8 | 1 | 2 | 4 | 8 | 1 | 10 | 20 | 30 | 1 | 10 | $10^2$ | $10^3$ |
| SmartMining [6] (reproduced) | | 72.7 | 82.2 | 88.4 | 92.7 | 50.2 | 62.5 | 73.3 | 82.7 | – | – | – | – | – | – | – | – |
| Doppelganger [21] (reproduced) | | 80.7 | 87.7 | 92.5 | 95.2 | 55.0 | 67.0 | 77.3 | 85.8 | 87.7 | 96.9 | 97.9 | 98.3 | 68.3 | 83.8 | 92.5 | 97.5 |
| HowToTrain100k [30] (reproduced) | | 78.9 | 86.5 | 91.6 | 95.0 | 53.4 | 65.2 | 75.7 | 83.8 | 87.0 | 96.5 | 97.5 | 98.0 | 69.2 | 83.7 | 82.0 | 97.0 |
| Inception_v1 | Baseline [38] | 78.2 | 86.0 | 90.9 | 94.2 | 52.4 | 64.4 | 74.9 | 84.2 | 86.4 | 96.5 | 97.9 | 98.4 | 67.8 | 84.0 | 93.2 | 97.9 |
| | Class Mining (Alg. 1) | 81.3 | 87.8 | 92.6 | 95.6 | 52.9 | 64.8 | 75.6 | 84.1 | 88.0 | 96.7 | 97.8 | 98.3 | 70.6 | 84.9 | 93.1 | 97.7 |
| | Stochastic Mining (Alg. 2, *var1*) | 81.3 | 88.3 | 92.3 | 95.5 | 54.1 | 66.3 | 76.7 | 84.8 | 87.3 | 96.3 | 97.4 | 97.9 | 68.7 | 82.4 | 90.8 | 96.2 |
| | Stochastic Mining (Alg. 2, *var2*) | 82.5 | 89.2 | 93.4 | 96.2 | 55.1 | 66.4 | 76.2 | 84.8 | 88.9 | 97.2 | 98.2 | 98.6 | 72.1 | 85.9 | 93.3 | 97.6 |
| | Stochastic Mining (Alg. 2) | 83.4 | 89.9 | 93.9 | 96.5 | 56.0 | 68.3 | 78.2 | 86.3 | 90.7 | 97.8 | 98.5 | 98.8 | 75.2 | 87.5 | 93.7 | 97.4 |

Table 3. Accuracy comparison on CARS-196

| Method | R@1 | R@2 | R@4 | R@8 |
|---|---|---|---|---|
| Lifted [25] (Inception_v1) | 53.0 | 66.7 | 76.0 | 84.3 |
| Facility [24] (Inception_v1) | 58.1 | 70.6 | 80.3 | 87.8 |
| SmartMining [6] (Inception_v1) | 64.7 | 76.2 | 84.2 | 90.2 |
| N-pair [23] (Inception_v1) | 71.1 | 79.7 | 86.5 | 91.6 |
| Angular [33] (Inception_v1) | 71.4 | 81.4 | 87.5 | 92.1 |
| Proxy NCA [14] (Inception_v1) | 73.2 | 82.4 | 86.4 | 88.7 |
| HDC [37] (Inception_v1 + ensemble) | 73.7 | 83.2 | 89.5 | 93.8 |
| DAML [3] (Inception_v1) | 75.1 | 83.8 | 89.7 | 93.5 |
| HTG [39] (Inception_v1 + att) | 76.5 | 84.7 | 90.4 | 94.0 |
| Margin [35] (ResNet-50) | 79.6 | 86.5 | 91.9 | 95.1 |
| HTL [5] (BN-Inception) | 81.4 | 88.0 | 92.7 | 95.7 |
| DVML [12] (Inception_v1) | 82.0 | 88.4 | 93.3 | 96.3 |
| A-Bier [15] (Inception_v1 + ensemble) | 82.0 | 89.0 | 93.2 | 96.1 |
| ABE [8] (Inception_v1 + ensemble) | 85.2 | 90.5 | 93.9 | 96.1 |
| DREML [36] (Inception_v3 + ensemble) | 84.2 | 89.4 | 93.2 | 95.5 |
| DREML [36] (ResNet-18 + ensemble) | 86.0 | 91.7 | 95.0 | 97.2 |
| Vo *et al.* [28] (VGG16-BN) | 87.8 | 92.7 | 95.6 | 97.5 |
| Proposed (Inception_v1) | 83.4 | 89.9 | 93.9 | 96.5 |
| Proposed (Channel-correlation) | **91.7** | **95.3** | **97.3** | **98.4** |

Table 4. Accuracy comparison on CUB-200-2011

| Method | R@1 | R@2 | R@4 | R@8 |
|---|---|---|---|---|
| SmartMining [6] (Inception_v1) | 49.8 | 62.3 | 74.1 | 83.3 |
| Proxy NCA [14] (Inception_v1) | 49.2 | 61.9 | 67.9 | 72.4 |
| N-pair [23] (Inception_v1) | 51.9 | 64.3 | 74.9 | 83.2 |
| DVML [12] (Inception_v1) | 52.7 | 65.1 | 75.5 | 84.3 |
| DAML [3] (Inception_v1) | 52.7 | 65.4 | 75.5 | 84.3 |
| HDC [37] (Inception_v1 + ensemble) | 53.6 | 65.7 | 77.0 | 85.6 |
| Angular [33] (Inception_v1) | 54.7 | 66.3 | 76.0 | 83.9 |
| HTL [5] (BN-Inception) | 57.1 | 68.8 | 78.7 | 86.5 |
| A-Bier [15] (Inception_v1 + ensemble) | 57.5 | 68.7 | 78.3 | 86.2 |
| HTG [39] (Inception_v1 + att) | 59.5 | 71.8 | 81.3 | 88.2 |
| ABE [8] (Inception_v1 + ensemble) | 60.6 | 71.5 | 79.8 | 87.4 |
| Margin [35] (ResNet-50) | 63.6 | 74.4 | 83.1 | 90.0 |
| DREML [36] (inception_v3 + ensemble) | 58.9 | 69.6 | 78.4 | 85.6 |
| DREML [36] (ResNet-18 + ensemble) | 63.9 | 75.0 | 83.1 | 89.7 |
| Vo [28] (VGG16-BN) | 66.4 | 77.5 | 85.4 | 91.3 |
| Proposed (Inception_v1) | 56.0 | 68.3 | 78.2 | 86.3 |
| Proposed (Channel-correlation) | **66.2** | **76.3** | **84.1** | **90.1** |

Table 5. Accuracy comparison on In-shop retrieval

| Method | R@1 | R@10 | R@20 | R@30 |
|---|---|---|---|---|
| HDC [37] (Inception_v1 + ensemble) | 62.1 | 84.9 | 89.0 | 91.2 |
| DREML [36] (ResNet-18 + ensemble) | 78.4 | 93.7 | 95.8 | 96.7 |
| HTG [39] (Inception_v1 + att) | 80.3 | 93.9 | 95.8 | 96.6 |
| HTL [5] (BN-Inception) | 80.9 | 94.3 | 95.8 | 97.2 |
| A-Bier [15] (Inception_v1 + ensemble) | 83.1 | 95.1 | 96.9 | 97.5 |
| ABE [8] (Inception_v1 + ensemble) | 87.3 | 96.7 | 97.9 | 98.2 |
| Proposed (Inception_v1) | 90.7 | 97.8 | 98.5 | 98.8 |
| Proposed (Channel-correlation) | **91.9** | **98.0** | **98.7** | **99.0** |

Table 6. Accuracy comparison on Stanford online products

| Method | R@1 | R@10 | R@$10^2$ | R@$10^3$ |
|---|---|---|---|---|
| N-pair [23] (Inception_v1) | 66.4 | 83.2 | 93.0 | – |
| DAML [3] (Inception_v1) | 68.4 | 83.5 | 92.3 | – |
| HDC [37] (Inception_v1 + ensemble) | 69.5 | 84.4 | 92.8 | 97.7 |
| DVML [12] (Inception_v1) | 70.2 | 85.2 | 93.8 | – |
| Margin [35] (ResNet-50) | 72.7 | 86.2 | 93.8 | 98.0 |
| Proxy NCA [14] (Inception_v1) | 73.7 | – | – | – |
| A-Bier [15] (Inception_v1 + ensemble) | 74.2 | 86.9 | 94.0 | 97.8 |
| HTL [5] (BN-Inception) | 74.8 | 88.3 | **94.8** | **98.4** |
| ABE [8] (Inception_v1 + ensemble) | 76.3 | 88.4 | **94.8** | 98.2 |
| Vo *et al.* [28] (VGG16-BN) | 74.8 | 88.3 | 95.2 | 98.5 |
| Proposed (Inception_v1) | 75.2 | 87.5 | 93.7 | 97.4 |
| Proposed (Channel-correlation) | **77.6** | **89.1** | 94.7 | – |


[2] Yin Cui, Feng Zhou, Yuanqing Lin, and Serge Belongie. Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop. In *CVPR*, 2016. 2

[3] Yueqi Duan, Wenzhao Zheng, Xudong Lin, Jiwen Lu, and Jie Zhou. Deep adversarial metric learning. In *CVPR*, 2018. 1, 2, 8

[4] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling. In *CVPR*, 2016. 2, 6

[5] Weifeng Ge, Weilin Huang, Dengke Dong, and Matthew R. Scott. Deep metric learning with hierarchical triplet loss. In *ECCV*, 2018. 2, 8

[6] Ben Harwood, Vijay Kumar B G, Gustavo Carneiro, Ian Reid, and Tom Drummond. Smart mining for deep metric learning. In *ICCV*, 2017. 1, 5, 6, 7, 8

[7] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv:1703.07737*, 2017. 1, 2, 3, 4

[8] Wonsik Kim, Bhavya Goyal, Kunal Chawla, Jungmin Lee, and Keunjoo Kwon. Attention-based ensemble for deep metric learning. In *ECCV*, 2018. 8

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*,

2014. 7

[10] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *ICCV Workshops*, 2013. 2, 6

[11] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. Deepfashion: powering robust clothes recognition and retrieval with rich annotations. In *ICCV Workshops*, 2013. 2, 6

[12] Xudong Lin, Yueqi Duan, Qiyuan Dong, Jiwen Lu, and Jie Zhou. Deep variational metric learning. In *ECCV*, 2018. 8

[13] Mingsheng Long, Zhangjie Cao, Jianmin Wang, and Michael I. Jordan. Conditional adversarial domain adaptation. In *NeurIPS*, 2018. 6

[14] Yair Movshovitz-Attias, Alexander Toshev, Thomas K. Leung, Sergey Ioffe, and Saurabh Singh. No fuss distance metric learning using proxies. In *ICCV*, 2017. 1, 2, 6, 8

[15] Michael Opitz, Georg Waltner, Horst Possegger, and Horst Bischof. Bier - boosting independent embeddings robustly. In *ICCV*, 2017. 8

[16] Ninh Pham and Rasmus Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *SIGKDD*, 2013. 6

[17] Rajeev Ranjan, Carlos D Castillo, and Rama Chellappa. L2-constrained softmax loss for discriminative face verification. *arXiv:1703.09507*, 2017. 3

[18] Oren Rippel, Manohar Paluri, Piotr Dollar, and Lubomir Bourdev. Metric learning with adaptive density discrimination. *ICLR*, 2016. 1, 2

[19] Ergys Ristani and Carlo Tomasi. Features for multi-target multi-camera tracking and re-identification. In *CVPR*, 2018. 2

[20] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015. 1, 2

[21] Evgeny Smirnov, Aleksandr Melnikov, Sergey Novoselov, Eugene Luckyanets, and Galina Lavrentyeva. Doppelganger mining for face representation learning. In *ICCVW*, 2017. 1, 2, 7, 8

[22] Evgeny Smirnov, Aleksandr Melnikov, Andrei Oleinik, Elizaveta Ivanova, Ilya Kalinovskiy, and E Lukyanets. Hard example mining with auxiliary embeddings. In *CVPR Workshop on Disguised Faces in the Wild*, 2018. 2

[23] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *NIPS*, 2016. 1, 2, 4, 8

[24] Hyun Oh Song, Stefanie Jegelka, Vivek Rathod, and Kevin Murphy. Deep metric learning via facility location. *CVPR*, 2017. 1, 8

[25] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *CVPR*, 2016. 2, 6, 8

[26] Yumin Suh, Jingdong Wang, Siyu Tang, Tao Mei, and Kyoung Mu Lee. Part-aligned bilinear representations for person re-identification. In *ECCV*, 2018. 6

[27] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 6

[28] Nam Vo and James Hays. Generalization in metric learning: Should the embedding layer be embedding layer? In *WACV*, 2018. 8

[29] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 2, 6

[30] Chong Wang, Xue Zhang, and Xipeng Lan. How to train triplet networks with 100k identities? *arXiv:1709.02940*, 2017. 1, 2, 7, 8

[31] Feng Wang, Xiang Xiang, Jian Cheng, and Alan L Yuille. Normface: $l\_2$ hypersphere embedding for face verification. *arXiv:1704.06369*, 2017. 3

[32] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *CVPR*, 2014. 2

[33] Jian Wang, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin. Deep metric learning with angular loss. In *ICCV*, 2017. 8

[34] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *ECCV*, 2016. 2

[35] Chao-Yuan Wu, R. Manmatha, Alexander J. Smola, and Philipp Krahenbuhl. Sampling matters in deep embedding learning. In *ICCV*, 2017. 1, 8

[36] Hong Xuan, Richard Souvenir, and Robert Pless. Deep randomized ensembles for metric learning. *arXiv preprint arXiv:1808.04469*, 2018. 8

[37] Yuhui Yuan, Kuiyuan Yang, and Chao Zhang. Hard-aware deeply cascaded embedding. In *ICCV*, 2017. 8

[38] Liming Zhao, Xi Li, Yueting Zhuang, and Jingdong Wang. Deeply-learned part-aligned representations for person re-identification. In *ICCV*, 2017. 1, 3, 4, 7, 8

[39] Yiru Zhao, Zhongming Jin, Guo jun Qi, Hongtao Lu, and Xian sheng Hua. A principled approach to hard triplet generation via adversarial nets. In *ECCV*, 2018. 1, 2, 8

[40] Zhedong Zheng, Liang Zheng, and Yi Yang. A discriminatively learned cnn embedding for person re-identification. *arXiv:1611.05666*, 2016. 5