This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2848925, IEEE Transactions on Pattern Analysis and Machine Intelligence

Deep Metric Learning with BIER: Boosting Independent Embeddings Robustly

Michael Opitz, Georg Waltner, Horst Possegger, and Horst Bischof

Abstract—Learning similarity functions between image pairs with deep neural networks yields highly correlated activations of embeddings. In this work, we show how to improve the robustness of such embeddings by exploiting the independence within ensembles. To this end, we divide the last embedding layer of a deep network into an embedding ensemble and formulate the task of training this ensemble as an online gradient boosting problem. Each learner receives a reweighted training sample from the previous learners. Further, we propose two loss functions which increase the diversity in our ensemble. These loss functions can be applied either for weight initialization or during training. Together, our contributions leverage large embedding sizes more effectively by significantly reducing correlation of the embedding and consequently increase retrieval accuracy of the embedding. Our method works with any differentiable loss function and does not introduce any additional parameters during test time. We evaluate our metric learning method on image retrieval tasks and show that it improves over state-of-the-art methods on the CUB-200-2011, Cars-196, Stanford Online Products, In-Shop Clothes Retrieval and VehicleID datasets. Therefore, our findings suggest that by dividing deep networks at the end into several smaller and diverse networks, we can significantly reduce overfitting.

Index Terms—Metric Learning, Deep Learning, Convolutional Neural Network.

1 Introduction

Deep Convolutional Neural Network (CNN) based metric learning methods map images to a high dimensional feature space. In this space semantically similar images should be close to each other, whereas semantically dissimilar images should be far apart from each other. To learn such metrics, several approaches based on image pairs (*e.g.* [1], [2]), triplets (*e.g.* [3], [4]) or quadruples (*e.g.* [5], [6]) have been proposed in the past. Metric learning has a variety of applications, such as image or object retrieval (*e.g.* [7], [8], [9]), single-shot object classification (*e.g.* [7], [8], [10]), keypoint descriptor learning (*e.g.* [11], [12]), face verification (*e.g.* [3], [13]), person re-identification (*e.g.* [8], [14]), object tracking (*e.g.* [15]).

In this work, we focus on learning simple similarity functions based on the dot product, since they can be computed rapidly and thus facilitate approximate search methods (*e.g.* [16]) for large-scale image retrieval. However, the accuracy of these methods tends to saturate or decline due to over-fitting, especially when large embeddings are used [7].

To address this issue, we present a learning approach, called Boosting Independent Embeddings Robustly (BIER), which leverages large embedding sizes more effectively. The main idea is to divide the last embedding layer of a CNN into multiple non-overlapping groups (see Fig. 1). Each group is a separate metric learning network on top of a shared feature representation. The accuracy of an ensemble depends on the accuracy of individual learners as well as the correlation between them [17]. Ideally, individual learners are highly accurate and have low correlation with each other, so that they complement each other during test time.

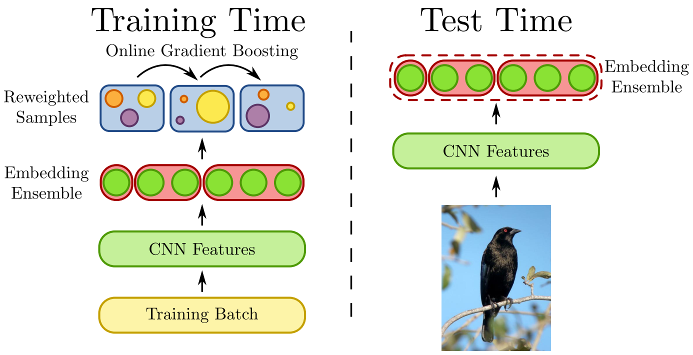

Fig. 1. BIER divides a large embedding into an ensemble of several smaller embeddings. During training we reweight the training set for successive learners in the ensemble with the negative gradient of the loss function. During test time we concatenate the individual embeddings of all learners into a single embedding vector.

Naïvely optimizing a global loss function for the whole ensemble shows no benefits since all learners have access to the same feature representation and the same training samples. All groups will end up learning highly correlated embeddings, which results in no performance improvements at all. To overcome this problem, we formulate the ensemble training as an online gradient boosting problem. In online gradient boosting, each learner reweights a training sample for successive learners according to the gradient of the loss function. This is done by multiplying the loss of successive learners for the training sample with a weight. Consequently, successive learners will focus on different samples than the previous learners, resulting in a more diverse feature representation (Section 4). To encourage the individual embeddings to have low correlation with each other already at the beginning of the training, we propose a novel initialization method for our embedding matrix (Section 4.1 and Section 4.2).

- *M. Opitz, G. Waltner, H. Possegger and H. Bischof are with the Institute of Computer Graphics and Vision, Graz University of Technology, Austria, 8010 Graz.*
E-mail: {michael.opitz, waltner, possegger, bischof}@icg.tugraz.at

Manuscript received January 15, 2018.

The matrix is initialized from a solution of an optimization problem which implicitly minimizes the correlation between groups. During test time we simply concatenate the individual embeddings of all learners into a single embedding feature vector. Therefore, BIER does not impose any significant additional runtime cost.

In comparison to our earlier version of this work [18], we extend BIER by integrating our weight initialization method as auxiliary loss function directly into the training objective (Section 4.2.2). As we show in our evaluation (Section 5.6), this allows us to train BIER at higher learning rates which significantly reduces training time. By jointly training our network with this loss function, we can further reduce the correlation between learners and improve the accuracy of our method (Section 5.6).

Additionally, we improve our the performance by introducing a novel Adversarial Loss, which learns adversarial regressors between pairs of embeddings (Section 4.1.2). These regressors learn a non-linear transformation between embeddings. Their objective is to maximize similarity between embeddings. Between our embeddings and the regressors, we insert a gradient reversal layer [19]. This layer changes the sign of the gradients during backpropagation and behaves like the identity function during forward propagation. As a consequence, our embeddings are trained to maximize this loss function w.r.t. our adversarial regressors and hence our ensemble becomes even more diverse.

We demonstrate the effectiveness of our metric on several image retrieval datasets [7], [20], [21], [22], [23]. In our evaluation we show that BIER significantly reduces the correlation of large embeddings (Section 5.1) and works with several loss functions (Section 5.2) while increasing retrieval accuracy by a large margin. BIER does not introduce any additional parameters into a CNN and has only negligible additional cost during training time and runtime. We show that BIER achieves state-of-the-art performance on the CUB-200-2011 [23], Cars-196 [20], Stanford Online Products [7], In-Shop Clothes Retrieval [22] and VehicleID [21] datasets (Section 5.8). Further, by employing our novel Adversarial Loss during training time as auxiliary loss, we can significantly outperform the state-of-the-art on these datasets.

2 Related Work

Our work is related to metric learning (Section 2.1) and boosting in combination with CNNs (Section 2.2). Additionally, since we propose a novel initialization method, we discuss related data dependent initialization methods for CNNs (Section 2.3). Next, we discuss techniques to increase the diversity of ensembles related to our auxiliary function (Section 2.4). Finally, we summarize adversarial loss functions for CNNs (Section 2.5), as we use an adversarial loss to encourage diversity of our learners.

2.1 Metric Learning

The main objective of metric learning in Computer Vision is to learn a distance function $d(\cdot,\cdot) : \mathbb{R}^k \times \mathbb{R}^k \mapsto \mathbb{R}^+$ mapping two k-dimensional input vectors, which are typically an input image or a feature representation of an image, to a distance between images. Typically, these distance functions have the form $d(\boldsymbol{x},\boldsymbol{y})^2 = (\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{M}(\boldsymbol{x}-\boldsymbol{y})$, where $\boldsymbol{M}$ is a positive semidefinite matrix. $\boldsymbol{M}$ can be factorized as $(\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{L}\boldsymbol{L}^\top(\boldsymbol{x}-\boldsymbol{y}) = \left\|\boldsymbol{x}^\top\boldsymbol{L} - \boldsymbol{y}^\top\boldsymbol{L}\right\|^2$, where $\boldsymbol{L} \in \mathbb{R}^{k\times d}$ projects an image, or a feature representation of an image into a d-dimensional vector space. In this vector space, semantically similar images should be close to each other, whereas semantically dissimilar images should be far apart from each other.

For a complete review of metric learning approaches we refer the interested reader to [24]. In this work we focus our discussion on boosting based metric learning approaches and deep CNN based approaches.

2.1.1 Boosting Based Metric Learning

In boosting based approaches, weak learners are typically rank one matrices. The ensemble then combines several of these matrices to form a positive semidefinite matrix $\boldsymbol{M}$, *e.g.* [25], [26], [27], [28]. Kedem *et al.* [29] propose gradient boosted trees for metric learning. They learn the non-linear mapping $f(\cdot)$ with an ensemble of regression trees, by minimizing a Large Margin Nearest Neighbor (LMNN) loss function [4] with the gradient boosting framework. Further, they initialize their first learner as the solution of the linear LMNN optimization problem. In contrast to these offline boosting based works, our method is an online boosting method, which directly integrates into deep CNN training. Our weak learners are fully connected layers on top of a shared CNN feature representation and, compared to these methods, typically have a higher rank. Further, we use auxiliary loss functions to explicitly encourage diversity in our metric ensemble.

2.1.2 CNN Based Metric Learning

CNN based methods learn a non-linear transformation of an input image of the form $\phi(\cdot) : \mathbb{R}^k \mapsto \mathbb{R}^h$. This CNN based feature extractor, *i.e.* $\phi(\cdot)$, can be pre-trained on other tasks, such as large scale image classification, *e.g.* [30], and is then fine-tuned on metric learning datasets. To map the feature representation into the d-dimensional vector space, an additional linear embedding layer is typically added at the end of a CNN feature extractor as $f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{W}$, $\boldsymbol{W} \in \mathbb{R}^{h\times d}$. Hence, metric learning CNNs learn the distance function $d(\boldsymbol{x},\boldsymbol{y})^2 = (\phi(\boldsymbol{x})-\phi(\boldsymbol{y}))^\top \boldsymbol{W}\boldsymbol{W}^\top(\phi(\boldsymbol{x})-\phi(\boldsymbol{y}))$, which is equivalent to $(\phi(\boldsymbol{x})-\phi(\boldsymbol{y}))^\top \boldsymbol{M}(\phi(\boldsymbol{x})-\phi(\boldsymbol{y}))$. To jointly learn all parameters of the CNN and the embedding, special loss functions operating on image pairs, triplets or quadruples are used. One of the most widely used pairwise loss functions for metric learning is the contrastive loss function, *e.g.* [1], [2], [7]. This loss function minimizes the squared Euclidean distance between positive feature vectors while encouraging a margin between positive and negative pairs. To train networks with this loss function, a Siamese architecture, *i.e.* two copies of a network with shared weights, is commonly used, *e.g.* [1], [2].

Other approaches adopt the LMNN formulation [4] and sample triplets consisting of a positive image pair and a negative image pair, *e.g.* [3], [7], [9], [13]. The loss function encourages a margin between distances of positive and negative pairs. Hence, positive image pairs are mapped closer to each other in the feature space compared to negative image pairs.

Recently, several new loss functions for metric learning have been proposed. Song *et al.* [7] propose to lift a mini-batch to a matrix of pairwise distances between samples. They use a structural loss function on this distance matrix to train the neural network. Ustinova *et al.* [8] propose a novel histogram loss. They also lift a mini-batch to a distance matrix and compute a histogram of positive and negative distances. Their loss operates on this histogram and minimizes the overlap between the distribution of positive and negative distances. Huang *et al.* [31] introduce a position dependent deep metric unit which is capable of learning

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2848925, IEEE Transactions on Pattern Analysis and Machine Intelligence

a similarity metric adaptive to the local feature space. Sohn [32] generalizes the triplet loss to n-tuples and propose a more efficient batch construction scheme. Song *et al.* [33] propose a structured clustering loss to train embedding networks. Wang *et al.* [34] propose a novel angular loss, which improves the traditional triplet loss by imposing geometric constraints for triplets. Movshovitz-Attias *et al.* [35] propose a proxy-loss where they introduce a set of proxies which approximate the dataset. Their Proxy-Neighborhood Component Analysis (NCA) loss function optimizes distances to these proxies. Rippel *et al.* [36] propose a "magnet" loss function which models multimodal data distributions and minimizes the overlap between distributions of different classes.

Our work is complementary to these approaches. We show in our evaluation that combining existing loss functions with our method yields significant improvements (Section 5.2).

Another line of work aims at improving the sample mining strategy used for embedding learning. Schroff *et al.* [3] propose a semi-hard mining strategy for the triplet loss. Within a mini-batch, they only use samples for training where the negative image pair has a larger distance than the positive pair. This avoids getting stuck in a local minima early in training [3]. Harwood *et al.* [37] use offline sampling of training samples. To avoid the large computational cost, they use approximate nearest neighbor search methods to accelerate distance computation. Wu *et al.* [38] propose a distance weighted sampling method in combination with a margin based loss function to improve metric learning.

Although the main objective of our method is to reduce correlation in a large embedding, we apply a form of hard negative mining. We reweight samples for successive learners according to the gradient of the loss function. More difficult samples are typically assigned a higher gradient than easier samples. Hence, successive learners focus on harder examples than previous learners. However, we do not use any sample mining strategy for our first learner and hypothesize that our method can benefit from the above approaches, *e.g.* by selecting better samples from the training-set or mini-batch.

Most closely related to our method is the concurrent work of Yuan *et al.* [39]. They propose a hard-aware deeply cascaded embedding. This method leverages the benefits of deeply supervised networks [40], [41] by employing a contrastive loss function and train lower layers of the network to handle easier examples, and higher layers in a network to handle harder examples. In contrast to this multi-layer approach, we focus on reducing the correlation on just a single layer. Further, our method allows continuous weights for samples depending on the loss function. Finally, we show that employing auxiliary loss functions during initialization or training decreases correlation of learners and consequently improves the accuracy of the ensemble.

2.2 Boosting for CNNs

Boosting is a greedy ensemble learning method, which iteratively trains an ensemble from several weak learners [42]. The original boosting algorithm, AdaBoost [42], minimizes an exponential loss function. Friedman [43] extends the boosting framework to allow minimizing arbitrary differentiable loss functions. They show that one interpretation of boosting is that it performs gradient descent in function space and propose a novel method leveraging this insight called gradient boosting. Successive learners in gradient boosting are trained to have high correlation with the negative gradient of the loss function. There are several algorithms which extend gradient boosting for the online learning setting, *e.g.* [44], [45], [46], [47]. In contrast to offline boosting, which has access to the full dataset, online boosting relies on online weak learners and updates the boosting model and their weak learners one sample at a time.

In the context of CNNs these methods are rarely used. Several works, *e.g.* [48], [49] use CNN features in an offline boosting framework. These approaches, however, do not train the network and the weak learners end-to-end, *i.e.* the CNN is typically only used as a fixed feature extractor. In contrast to these approaches, we train our system end-to-end. We directly incorporate an online boosting algorithm into training a CNN.

Similarly, Walach *et al.* [50] leverage gradient boosting to train several CNNs within an offline gradient boosting framework for person counting. The ensemble is then fine-tuned with a global loss function. In contrast to their work, which trains several copies of full CNN models, our method trains a single CNN with an online boosting method. Similar to dropout [51], all our learners share a common feature representation. Hence, our method does not introduce any additional parameters.

Very recently, Han *et al.* [52] propose to use boosting to select discriminative neurons for facial action unit classification. They employ decision stumps on top of single neurons as weak learners, and learn weighting factors for each of these neurons by offline AdaBoost [42] applied to each mini-batch separately. Weights are then exponentially averaged over several mini-batches. They combine the weak learner loss functions with a global loss function over all learners to train their network. In contrast to this work, we use weak learners consisting of several neurons (*i.e.* linear classifiers). Further, our method is more tightly integrated in an online boosting framework. We reweight the training set according to the negative gradient of the loss function for successive weak learners. This encourages them to focus on different parts of the training set. Finally, our method does not rely on optimizing an explicit discriminative global loss function.

2.3 Initialization Methods

Most initialization methods for CNNs initialize weights randomly, either with carefully chosen variance parameters, *e.g.* [53], or depending on the fan-in and fan-out of a weight matrix, *e.g.* [54], [55], with the goal of having an initialization which provides a large gradient during learning. Rather than focusing on determining the variance of the weight matrix, Saxe *et al.* [56] propose to initialize the weight matrix as orthogonal matrix.

Recently, several approaches which initialize weights depending on the input data were proposed, *e.g.* [57], [58]. These methods typically scale a random weight matrix such that the activations on the training set have unit variance.

Another line of work, *e.g.* [59], [60], greedily initializes a network layer-by-layer, by applying unsupervised feature learning, such as Autoencoders or Restricted Bolzman Machines (RBMs). These methods seek for a weight matrix which minimizes the reconstruction error or a matrix which learns a generative model of the data.

Our initialization method is also a form of unsupervised pre-training of a single layer, as we use unsupervised loss functions for initializing the weights of our embedding layer. However, as opposed to minimizing a reconstruction loss or learning a generative model of the data, we initialize the weight matrix from a solution of an optimization problem which implicitly minimizes correlation between groups of features. With this initialization our weak learners already have low correlation at the beginning of the training process.

2.4 Diversity in Ensembles

Previous approaches which exploit diversity in ensembles are based on Negative Correlation Learning (NCL) [61], *e.g.* [61], [62]. These methods train neural networks in an ensemble to be negatively correlated to each other by penalizing the cross-correlation of their predictions. As a consequence, they complement each other better during test time. These approaches are typically focused on training regressor ensembles, as opposed to classification or metric ensembles and do not use boosting. Further, they train several full regressor networks from scratch as opposed to using a single shared feature extractor CNN.

More closely related is AdaBoost.NC [63], which extends NCL to AdaBoost for classification. AdaBoost.NC defines an ambiguity penalty term based on the deviation of the predictions of the weak learners to the ensemble prediction. Intuitively, if many learners deviate from the ensemble prediction for a sample, the ambiguity is high. This ambiguity measure is used to update the weights for the samples for successive learners in the ensemble. In contrast to this work, we encourage diversity in our ensemble by directly using a differentiable loss function for our learners.

Finally, in an earlier work we applied auxiliary loss functions for a deep CNN based classification ensemble with a shared feature representation [64]. Similar to this work, for computational efficiency, we share all low level CNN features and divide the network at the end into several non-overlapping groups. In contrast to our earlier work, we use online boosting to build our metric ensemble and different loss functions which are compatible with metric learning to encourage diversity.

2.5 Adversarial Loss Functions

Adversarial networks, such as Generative Adversarial Networks (GANs) [65], have several applications such as image generation (*e.g.* [65], [66], [67]), style transfer (*e.g.* [68]), domain adaptation (*e.g.* [69]), *etc*. These approaches typically consist of two neural networks, a discriminator and a generator. During training, discriminator and generator are playing a two-player minimax game. The discriminator minimizes a loss function to distinguish real-world images from fake images, which are generated by the generator. On the other hand, the generator tries to confuse the discriminator by generating plausible fake images. To achieve this, it maximizes the loss function the discriminator tries to minimize. During training, GANs use alternating Stochastic Gradient Descent (SGD) to optimize the two networks. In the first step the parameters of the generator are updated, keeping the parameters of the discriminator fixed. Then, in the second step the discriminator is updated, while keeping the generator fixed *e.g.* [65], [66], [67], [68].

Most closely related to our work are methods which apply GANs and adversarial loss functions for domain adaptation. Tzeng *et al.* [70] propose an adversarial loss at feature level for domain adaptation. They train a linear classifier on top of a hidden feature representation to categorize the domain of a sample. The feature generator (*i.e.* the hidden representation of the neural network) is trained to maximize the loss function of this classifier. Consequently, the hidden representation of samples from different domains will be aligned and hence undistinguishable for the classifier.

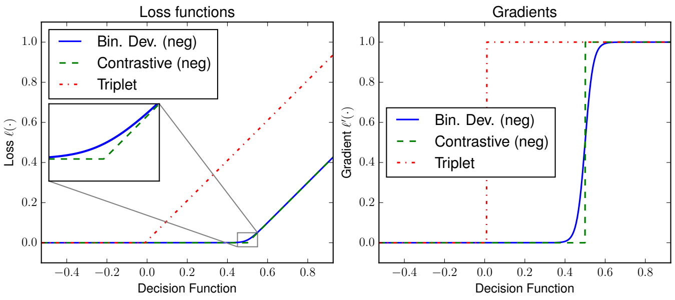

Fig. 2. Illustration of triplet loss, contrastive loss (for negative samples) and binomial deviance loss (for negative samples) and their gradients. Triplet and contrastive loss have a non-continuous gradient, whereas binomial deviance has a continuous gradient.

Similar to the GAN setup, Ganin *et al.* [19] propose Domain Adversarial Neural Networks (DANNs). This method uses a gradient reversal layer for domain adaptation. They insert a discriminator on top of a neural network feature generator. The discriminator minimizes a loss function to distinguish samples of two different domains. Between the discriminator and feature extractor they insert a gradient reversal layer which flips the sign of the gradients during backpropagation. As a consequence, the feature extractor maximizes the loss function of the discriminator, making the hidden layer representation of different domains undistinguishable for the discriminator. Compared to GAN based approaches, DANNs do not need alternating updates of the generator and discriminator. At each step, the method updates the parameters of both, the generator and the discriminator.

As opposed to aligning two domains with each other, our method makes embeddings more diverse. To this end, we adopt the gradient reversal layer of DANNs to make different learners more diverse from each other. We train a regressor, as opposed to a discriminator, which projects features from one learner to the other with a non-linear neural network. We optimize the regressor to maximize the similarity between embeddings. By inserting the gradient reversal layer between the regressor and our embeddings, we force our embeddings to be more diverse to each other. To the best of our knowledge, domain adaptation approaches have not been applied to increase diversity among classifiers.

3 Review of Metric Learning for CNNs

Our method builds upon metric CNNs, *e.g.* [7], [8], [31], [32]. The main objective of these networks is to learn a high-dimensional non-linear embedding $f(\boldsymbol{x})$, which maps an image $\boldsymbol{x}$ to a feature space $\mathbb{R}^d$. In this space, similar image pairs should be close to each other and dissimilar image pairs should be far apart from each other. To achieve this, instead of relying on a softmax output layer, these methods use a final linear layer consisting of an embedding matrix $\boldsymbol{W} \in \mathbb{R}^{h \times d}$, which maps samples from the last hidden layer of size h into the feature space $\mathbb{R}^d$. To learn this embedding matrix $\boldsymbol{W}$ and the parameters of the underlying network, these networks are typically trained on pairs or triplets of images and use loss functions to encourage separation of positive and negative pairs, *e.g.* [7].

As opposed to learning a distance metric, in our work we learn a cosine similarity score $s(\cdot,\cdot)$, which we define as dot product between two embeddings

$$s(f(\boldsymbol{x}^{(1)}), f(\boldsymbol{x}^{(2)})) = \frac{f(\boldsymbol{x}^{(1)})^\top f(\boldsymbol{x}^{(2)})}{\|f(\boldsymbol{x}^{(1)})\| \cdot \|f(\boldsymbol{x}^{(2)})\|}. \quad (1)$$

This has the advantage that the similarity score is bounded between $[-1,+1]$.

In our framework, we do not use a Siamese architecture, *e.g.* as [1], [2]. Instead, we follow recent work, *e.g.* [3], [7], [8], and sample a mini-batch of several images, forward propagate them through the network and sample pairs or triplets in the last loss layer of the network. The loss is then backpropagated through all layers of the network. This has the advantage that we do not need to keep several separate copies of the network in memory and that we can improve the computational efficiency.

We consider three different loss functions (see Fig. 2), which are commonly used to train metric networks, *e.g.* [3], [11], [13], [14], *i.e.* the binomial deviance loss ℓ_{BD}, the contrastive loss ℓ_C and the triplet loss ℓ_T. These loss functions are defined as

$$\ell_{BD}(s,y) = \begin{cases} \log(1+e^{-\beta_1(s-\beta_2)}) & \text{if } y=1 \\ \log(1+e^{\beta_1(s-\beta_2)C}) & \text{otherwise,} \end{cases} \quad (2)$$

$$\ell_C(s,y) = \begin{cases} (s-1)^2 & \text{if } y=1 \\ \max(0, s-m) & \text{otherwise,} \end{cases} \quad (3)$$

$$\ell_T(s^+, s^-) = \max(0, s^- - s^+ + m), \quad (4)$$

where $s = s(f(\boldsymbol{x}^{(1)}), f(\boldsymbol{x}^{(2)}))$ is the similarity score between image $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$. $y \in \{1,0\}$ denotes the label of the image pair (*i.e.* 1 for similar pairs, and 0 for dissimilar pairs). s^- denotes the similarity score for a negative image pair and s^+ denote the similarity score for a positive image pair. Further, m denotes the margin for the contrastive and triplet loss, which is set to 0.5 and 0.01, respectively. β_1 and β_2 are scaling and translation parameters and are set to 2 and 0.5, similar to [8]. Finally, we follow [8] and set the cost C to balance positive and negative pairs for the binomial deviance loss to 25 for negative pairs.

The binomial deviance loss is similar to the contrastive loss, but has a smooth gradient (see Fig. 2). In contrast, the contrastive and triplet loss have a gradient of either 0 or 1. As we show in our evaluation (Section 5.2) the binomial deviance loss benefits more from our method compared to the triplet and contrastive loss. We hypothesize that the main reason for that is that the gradient of the binomial deviance loss is smooth compared to the triplet loss or the contrastive loss. As a consequence, our method assigns smooth weights to training samples which conveys more information for successive learners.

4 Online Gradient Boosting CNNs

To encourage diverse learners we borrow ideas from online gradient boosting. Online gradient boosting iteratively minimizes a loss function using a fixed number of M weak learners, *e.g.* [44], [45], [46], [47]. Learners are trained on reweighted samples according to the gradient of the loss function. Correctly classified samples typically receive a lower weight while misclassified samples are assigned a higher weight for successive learners. Hence, successive learners focus on different samples than previous learners, which consequently encourages higher diversity among weak learners.

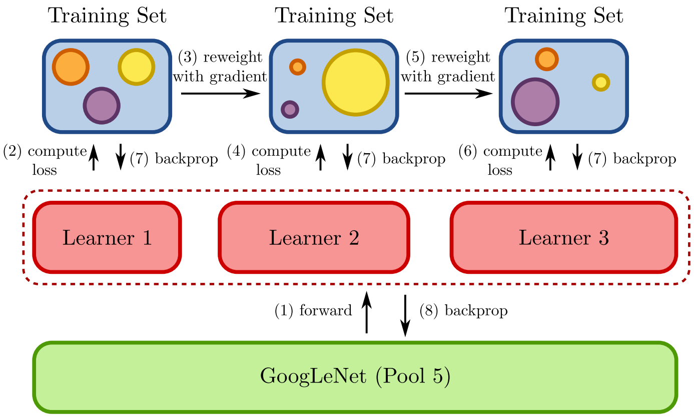

Fig. 3. We divide the embedding (shown as dashed layer) of a metric CNN into several weak learners and cast training them as online gradient boosting problem. Each learner iteratively reweights samples according to the gradient of the loss function. Training a metric CNN this way encourages successive learners to focus on different samples than previous learners and consequently reduces correlation between learners and their feature representation.

More formally, for a loss $\ell(\cdot)$, we want to find a set of weak learners $\{f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \dots, f_M(\boldsymbol{x})\}$ and their corresponding boosting model

$$F(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}) = \sum_{m=1}^{M} \alpha_m s(f_m(\boldsymbol{x}^{(1)}), f_m(\boldsymbol{x}^{(2)})), \quad (5)$$

where $F(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)})$ denotes the ensemble output and α_m is the weighting factor of the m-th learner. The m-th learner of the ensemble is trained on a reweighted training batch according to the negative gradient $-\ell'(\cdot)$ of the loss function at the ensemble prediction until stage $m-1$.

To train the weak learners $f_m(\cdot)$ in an online fashion, we adapt an online gradient boosting learning algorithm [44] with fixed weights α_m and integrate it within a CNN. Naïvely training multiple CNNs within the boosting framework is, however, computationally too expensive. To avoid this additional computational cost, we divide the embedding layer of our CNN into several non-overlapping groups, as illustrated in Fig. 3. A single group represents a weak learner. All our weak learners share the same underlying feature representation, which is a pre-trained ImageNet CNN in all our experiments.

Our network is trained end-to-end on mini-batches with SGD and momentum. We illustrate the training procedure for loss functions operating on pairs and a single example per batch in Algorithm 1. Our algorithm also works with triplets, but for the sake of clarity we omit a detailed explanation here and refer the interested reader to the supplementary material. The training procedure can be easily integrated into the standard backpropagation algorithm, introducing only negligible additional cost, since most time during training is spent on computing convolutions. First, in the forward pass we compute similarity scores s_n^m for each input sample n and each group m. In the backward pass we backpropagate the reweighted losses for each group iteratively. The weight w_n^m for the n-th sample and the m-th learner is computed from the negative gradient $-\ell'(\cdot)$ of the ensemble prediction until stage $m-1$. Hence, successive learners focus on examples which have large gradients (*i.e.* are misclassified) by previous learners.

```
Let $\eta_m = \frac{2}{m+1}$, for $m = 1, 2, \dots, M$,
$M$ = number of learners, $I$ = number of iterations
for $n = 1$ to $I$ do
    /* Forward pass */
    Sample pair ($\boldsymbol{x}_n^{(1)}$, $\boldsymbol{x}_n^{(2)}$) and corresponding label $y_n$
    $s_n^0 := 0$
    for m = 1 to M do
        $s_n^m := (1-\eta_m)s_n^{m-1} + \eta_m s(f_m(\boldsymbol{x}_n^{(1)}), f_m(\boldsymbol{x}_n^{(2)}))$
    end
    Predict $s_n$ = $s_n^M$

    /* Backward pass */
    $w_n^1 := 1$
    for $m = 1$ to $M$ do
        Backprop $w_n^m \ell(s(f_m(\boldsymbol{x}_n^{(1)}), f_m(\boldsymbol{x}_n^{(2)})), y_n)$
        $w_n^{m+1} := -\ell'(s_n^m, y_n)$
    end
end
```

Algorithm 1: Online gradient boosting algorithm for our CNN.

This online gradient boosting algorithm yields a convex combination of weak learners $f_m(\cdot)$, $1 \leq m \leq M$. Successive learners in the ensemble typically have to focus on more complex training samples compared to previous learners and therefore, should have a larger embedding size. We exploit this prior knowledge and set the group size of learner m to be proportional to its weight $\alpha_m = \eta_m \cdot \prod_{n=m+1}^{M}(1-\eta_n)$ in the boosting algorithm, where $\eta_m = \frac{2}{m+1}$. We experimentally verify this design choice in Section 5.1.

During test time our method predicts a single feature vector for an input image $\boldsymbol{x}$. We simply compute the embeddings from all weak learners $f_1(\cdot), f_2(\cdot), \dots f_M(\cdot)$, L_2-normalize each of them individually and weight each of them according to the boosting weights α_m. Finally, we concatenate all vectors to a single feature vector, which is the embedding $f(\boldsymbol{x})$ of the input image $\boldsymbol{x}$. As a consequence, distances between our vectors can be efficiently computed via dot products and hence, our vectors can be used by approximate search methods, *e.g.* [16].

4.1 Diversity Loss Functions

Rather than relying on boosting alone to increase the diversity in our ensemble, we propose additional loss functions which make learners more diverse from each other. We present two different loss functions to encourage the diversity of learners. These can either be used for weight initialization or as auxiliary loss function during training (see Section 4.2). Our first loss function, which we denote as *Activation Loss*, optimizes the embeddings such that for a given sample, only a single embedding is active and all other embeddings are close to zero (see Section 4.1.1). As second loss function, we propose an *Adversarial Loss*. We train a regressor on top of our embeddings which maps one embedding to a different embedding, maximizing their similarity. By inserting a gradient reversal layer [19] between the regressors and our embeddings, we update our embeddings so that they minimize the similarity between each other with respect to these regressors which results in more diverse embeddings (see Section 4.1.2).

4.1.1 Activation Loss

Our Activation Loss directly operates on the activations of our embeddings, making our learners more diverse by suppressing all activations except those of a single embedding (see Fig. 4). As a consequence, for a given sample, only a single embedding is active and all other embeddings are close to zero. More formally, let M denote the number of groups (*i.e.* weak learners) and G_i denote the index set of neurons of group i, $1 \leq i \leq M$. We want to increase the diversity of the embedding matrix $\boldsymbol{W} \in \mathbb{R}^{h \times d}$, where d denotes the embedding size and h the input feature dimensionality, *i.e.* the size of the last hidden layer in a CNN. Finally, let $X = \left\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(N)}\right\}$ denote the training set. For our initialization experiments, which we will discuss in Section 4.2.1, we use feature vectors extracted from the last hidden layer of a pre-trained CNN, which we denote as $\phi(\boldsymbol{x}) : \mathbb{R}^k \mapsto \mathbb{R}^h$, where k denotes the input image dimensionality and h the dimensionality of the last hidden layer of our feature extractor. When we apply our loss function as auxiliary loss during end-to-end training, we jointly optimize this loss function with the metric loss, as will be shown in Section 4.2.2. Intuitively, we want to ensure that activations are not correlated between groups. For a sample $\boldsymbol{x}^{(n)}$, we encourage this with the following suppression loss function

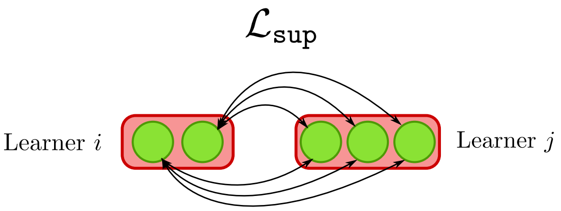

Fig. 4. Illustration of our Activation Loss. Neurons (green) of different embeddings (red) suppress each other. We apply this loss during training time between all pairs of our learners.

$$\mathcal{L}_{\texttt{sup}_{(\texttt{i},\texttt{j})}}(\boldsymbol{x}^{(n)}) = \sum_{\substack{k \in G_i, \\ l \in G_j}} (f_i(\boldsymbol{x}^{(n)})_k \cdot f_j(\boldsymbol{x}^{(n)})_l)^2, \tag{6}$$

where $f_i(\boldsymbol{x}^{(n)}) = \phi(\boldsymbol{x}^{(n)})^\top \boldsymbol{W}_i$ denotes the i-th embedding $(1 \leq i \leq M)$ of input image $\boldsymbol{x}^{(n)}$, $\boldsymbol{W}_i$ denotes the sub-matrix of $\boldsymbol{W}$ corresponding to the i-th embedding and $f_i(\boldsymbol{x}^{(n)})_k$ the k-th dimension of $f_i(\boldsymbol{x}^{(n)})$. Naïvely solving this problem, however, leads to the trivial solution $\boldsymbol{W} = \boldsymbol{0}$. To prevent this trivial solution, we add the regularization term

$$\mathcal{L}_{\texttt{weight}} = \sum_{i=1}^{d} (\boldsymbol{w}_i^\top \boldsymbol{w}_i - 1)^2, \tag{7}$$

where $\boldsymbol{w}_i$ (with $1 \leq i \leq d$) are the row vectors of $\boldsymbol{W}$. This term forces the squared row vector norms of $\boldsymbol{W}$ to be close to 1 and hence avoids a trivial solution. Our final Activation Loss combines both $\mathcal{L}_{\texttt{sup}}$ and $\mathcal{L}_{\texttt{weight}}$

$$\mathcal{L}_{\texttt{act}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{\substack{i=1, \\ j=i+1}}^{M} \mathcal{L}_{\texttt{sup}_{(\texttt{i},\texttt{j})}}(\boldsymbol{x}^{(n)}) + \lambda_{\text{w}} \cdot \mathcal{L}_{\texttt{weight}}, \tag{8}$$

where λ_{w} is a regularization parameter, which we set high enough such that all row-vectors have a squared norm close to $1 \pm 1e^{-3}$.

4.1.2 Adversarial Loss

The previous Activation Loss imposes a rather strong constraint on the embeddings, *i.e.* for a given sample only a single embedding should be active and all other embeddings should be close to zero.

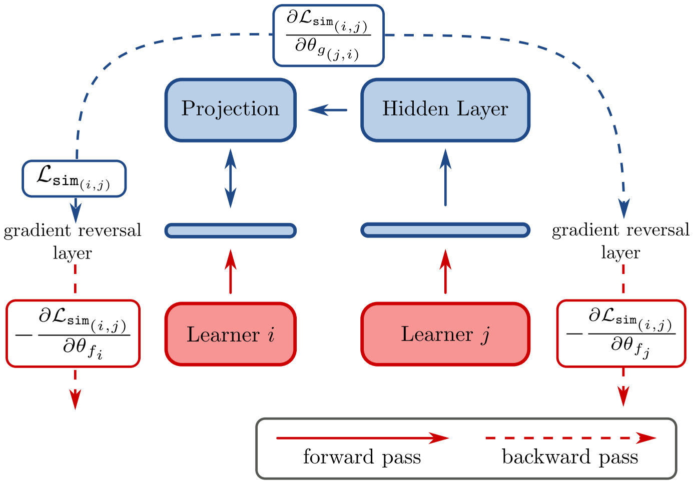
$\frac{\partial \mathcal{L}_{\texttt{sim}(i,j)}}{\partial \theta_{g(j,i)}}$
Projection
Hidden Layer
$\mathcal{L}_{\texttt{sim}(i,j)}$
gradient reversal layer
gradient reversal layer
$-\frac{\partial \mathcal{L}_{\texttt{sim}(i,j)}}{\partial \theta_{f_i}}$
Learner i
Learner j
$-\frac{\partial \mathcal{L}_{\texttt{sim}(i,j)}}{\partial \theta_{f_j}}$
forward pass
backward pass

Fig. 5. Illustration of our adversarial regressors (blue) between learner i and learner j of our embedding (red). We learn a regressor which maps the vector of learner j to learner i, maximizing the similarity of feature vectors. The gradient reversal layer flips the sign of the gradients which are backpropagated to our embeddings, therefore minimizing the similarity of feature vectors. We apply these regressors during training between all pairs of our learners.

While this improves the results, our objective is to maximize diversity between two feature vectors $f_i(\boldsymbol{x}) \in \mathbb{R}^{d_i}$ and $f_j(\boldsymbol{x}) \in \mathbb{R}^{d_j}$ extracted from embedding i and j, where in general $d_i \neq d_j$. Rather than imposing the constraint that only a single embedding is active for a given training sample, we could also aim for a weaker constraint. We want the two vectors $f_i(\boldsymbol{x})$ and $f_j(\boldsymbol{x})$ to be different, nonetheless discriminative. Therefore, the distance between the two vectors should be large. Unfortunately, there is no straightforward way to measure distances between two different vector spaces, since they can *e.g.* be of different dimensionality or be permuted.

To overcome this problem, we introduce an adversarial loss function, which we illustrate in Fig. 5. We learn regressors between pairs of embeddings which project $f_j(\boldsymbol{x})$ into the feature space $f_i(\boldsymbol{x})$, maximizing the similarity between embeddings $f_i(\boldsymbol{x})$ and $f_j(\boldsymbol{x})$, by minimizing a loss function. On the other hand, our learners try to maximize this loss function w.r.t. these adversarial regressors and therefore minimize their similarity. To achieve this, we use a reverse gradient layer [19] between regressors and embeddings. During the forward pass this layer behaves like the identity function. However, during the backward pass, this layer flips the sign of the gradients. As a consequence, our embedding learners minimize this loss function with respect to the regressors, *i.e.* increasing their diversity.

More formally, let $f_m(\boldsymbol{x}) \in \mathbb{R}^{d_m}$ denote the d_m dimensional embedding of the m-th learner. The objective of our adversarial regressor is to learn a function $g_{(j,i)}(\cdot) : \mathbb{R}^{d_j} \mapsto \mathbb{R}^{d_i}$ from the d_j-dimensional embedding j to the d_i-dimensional embedding i, maximizing similarity between vectors from embedding j and i via the loss

$$\mathcal{L}_{\texttt{sim}_{(i,j)}}(\boldsymbol{x}^{(n)}) = -\frac{1}{d_i}\sum (f_i(\boldsymbol{x}^{(n)}) \odot g_{(j,i)}(f_j(\boldsymbol{x}^{(n)})))^2, \quad (9)$$

where $\odot$ denotes the Hadamard (*i.e.* elementwise) product. Our embeddings on the other hand, try to minimize the similarity w.r.t. of these regressors:

$$\max_{f_i, f_j} \min_{g_{(i,j)}} \mathcal{L}_{\texttt{sim}_{(i,j)}}(\boldsymbol{x}^{(n)}). \quad (10)$$

To optimize this adversarial loss function, we use a reverse gradient layer between regressors and embeddings (see Fig. 5) and minimize w.r.t. all parameters. Consequently, the loss is maximized w.r.t. f_i and f_j, minimizing the embedding similarity.

$\mathcal{L}_{\texttt{sim}_{(i,j)}}$ can be made arbitrary small by scaling the weights $\widehat{\boldsymbol{W}}$ of the regressors $g_{(j,i)}$ as well as the weights $\boldsymbol{W}$ of the embedding. Hence, we penalize large weights $\widehat{\boldsymbol{W}}$ and biases $\widehat{\boldsymbol{b}}$ of $g_{(j,i)}$, and the weights $\boldsymbol{W}$ of our embedding as

$$\mathcal{L}_{\texttt{weight}} = \max(0, \widehat{\boldsymbol{b}}^{\top}\widehat{\boldsymbol{b}} - 1) + \sum_i (\widehat{\boldsymbol{w}}_i^{\top}\widehat{\boldsymbol{w}}_i - 1)^2 + \sum_i (\boldsymbol{w}_i^{\top}\boldsymbol{w}_i - 1)^2, \quad (11)$$

where $\widehat{\boldsymbol{w}}_i$ denotes the i-th row of the weight matrix $\widehat{\boldsymbol{W}}$ and $\boldsymbol{w}_i$ denotes the i-th row of the weight matrix $\boldsymbol{W}$. We combine both terms to train the regressor with our adversarial loss

$$\mathcal{L}_{\texttt{adv}} = \frac{1}{N}\sum_{n=1}^{N}\sum_{\substack{i=1\\ j=i+1}}^{M} \mathcal{L}_{\texttt{sim}_{(i,j)}}(\boldsymbol{x}^{(n)}) + \lambda_{\texttt{w}} \cdot \mathcal{L}_{\texttt{weight}}, \quad (12)$$

where M is the number of learners in our ensemble. $\lambda_{\texttt{w}}$ is a regularization parameter, which we set high enough so that our weight vectors have a squared norm close to $1 \pm 1e^{-3}$.

Backpropagating the errors of this loss function to our learners increases their correlation and reduces their diversity. However, since we use a gradient reversal layer between our learners and the regressors, we actually force our learners to minimize $\mathcal{L}_{\texttt{sim}_{(i,j)}}$, consequently increasing their diversity. In our experiments, we use two-layer neural networks with a Rectified Linear Unit (ReLU) as non-linearity in the hidden layer for the regressor $g_{(j,i)}$. Further, we choose a hidden layer size of 512.

We use the loss function in Eq. (12) as auxiliary loss function during training as shown in the following section. At test time, we simply discard the regressors. Hence, we do not introduce any additional parameters during test time with this adversarial loss function. During training time, computational cost is dominated by calculating the forward and backward pass of the convolution layers. Further, since we are only using a gradient reversal layer as opposed to alternating updates of our adversarial network and our base network, we can update the parameters of both networks in a single forward and backward pass. Hence, we do not introduce significant additional computational cost during training time.

4.2 Optimizing Diversity Loss Functions

We present two ways to apply the previously defined loss functions to improve our boosting based method. In our first approach, we use one of our diversity loss functions, *i.e.* either our Activation Loss or our Adversarial Loss, for initializing the embedding matrix $\boldsymbol{W}$. We fix all lower level CNN parameters and solve an optimization problem for the embedding matrix $\boldsymbol{W}$. Then, we perform end-to-end training of the CNN with this initialization and our boosting based method (Section 4.2.1). Our second method applies the diversity loss during training time as auxiliary loss together with our boosting based method (Section 4.2.2).

4.2.1 Initialization Method

During initialization we want to find an initial estimate of the embedding matrix $\boldsymbol{W}$, so that our learners already have low correlation with each other at the beginning of the training.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2848925, IEEE Transactions on Pattern Analysis and Machine Intelligence

Therefore, we omit end-to-end training and instead fix all the CNN parameters except the embedding matrix $\boldsymbol{W}$. We minimize a loss function which encourages diversity of learners by solving the following optimization problem with SGD and momentum

$$\arg\min_{\boldsymbol{W}} \mathcal{L}_{\texttt{div}}, \tag{13}$$

where $\mathcal{L}_{\texttt{div}}$ is either $\mathcal{L}_{\texttt{act}}$ (*c.f.* Eq. (8)) if we use our Activation Loss or $\mathcal{L}_{\texttt{adv}}$ (*c.f.* Eq. (12)) if we use our Adversarial Loss.

Compared to training a full CNN, solving this problem takes only seconds to a few minutes depending on the size of the dataset. The main reason for this is that we can pre-compute all lower level CNN features and just optimize with respect to the last layer (*i.e.* the embedding matrix $\boldsymbol{W}$). As a consequence, the number of parameters for which we are optimizing is smaller and the computational load is lower, hence convergence is quicker.

We show the benefits of both, our Adversarial Loss and Activation Loss as initialization method in Section 5.5. Both loss functions significantly improve the accuracy of our boosting based method, as they reduce the correlation between embeddings already from the beginning of the training.

4.2.2 Auxiliary Loss Function

When we apply the loss functions as auxiliary loss during training, we sample the matrix $\boldsymbol{W}$ uniformly random [54] and introduce an additional weighting parameter $\lambda_{\texttt{div}}$, which controls the strength of our diversity regularizer. More formally, during training time we optimize the following loss

$$\mathcal{L} = \mathcal{L}_{\texttt{metric}} + \lambda_{\texttt{div}} \cdot \mathcal{L}_{\texttt{div}}, \tag{14}$$

where $\mathcal{L}_{\texttt{metric}}$ is the discriminative metric loss (*e.g.* binomial deviance, contrastive, triplet), which is minimized by our boosting based algorithm and $\mathcal{L}_{\texttt{div}}$ is our loss function which encourages diversity in our ensemble. We either use $\mathcal{L}_{\texttt{act}}$ (Eq. (8)) or $\mathcal{L}_{\texttt{adv}}$ (Eq. (12)) for $\mathcal{L}_{\texttt{div}}$, depending on whether we use our Activation Loss or our Adversarial Loss, respectively. The weighting parameter $\lambda_{\texttt{div}}$ controls the strength of the diversity and can be set via cross-validation.

We also need to include the constraints on the weights during training (see Eq. (7) and Eq. (11)). The main reason for that is that our embeddings are L_2 normalized. Consequently, the discriminative loss function does not significantly change the magnitude of the weight vectors. On the other hand, our auxiliary loss functions typically try to make the weights as small as possible. Therefore, during training our weights are significantly decreasing, which eventually causes numeric problems during optimization. To avoid this problem, we constrain the weights of our embedding layer to have a squared L_2 norm of 1 for all row vectors. For our hidden layers, however, we have found that adding such constraints typically destroys the learned ImageNet features. Therefore, we only backpropagate the auxiliary loss to the embedding layer, which we add on top of the last hidden layer of our feature extractor. During training time this has only a small computational overhead compared to standard backpropagation, as only the last layer is affected.

We show the benefits of using our Activation Loss and our Adversarial Loss as auxiliary loss function in Section 5.6. When applied as auxiliary loss, our Adversarial Loss is more effective than our Activation Loss, *i.e.* it reduces the correlation between embeddings more without impairing their accuracy and as a result achieves higher ensemble accuracy.

5 Evaluation

We first conduct a detailed ablation study on the CUB-200-2011 [23] dataset. We follow the evaluation protocol proposed in [7] and use the first 100 classes $(5,864$ images) for training and the remaining 100 classes $(5,924$ images) for testing.

For evaluation we use the Recall@K metric [7]. For each image in the test set, we compute the feature vectors from our CNN and then retrieve the K most similar images from the remaining test set. If one of the K retrieved images has the same label as the query image, it is a match and increases the recall score by 1. The final Recall@K score is the average over all test images.

We implement our method with Tensorflow [71]. As network architecture, we follow previous works (*e.g.* [7], [8]) and use a GoogLeNet* [41] which is pre-trained on the ImageNet dataset [30]. As optimization method we use ADAM [73] with a learning rate of $1e^{-6}$. When we use auxiliary loss functions, we can increase the learning rate by an order of magnitude to $1e^{-5}$ (see Section 5.6). We construct a mini-batch by first sampling a fixed number of categories from the dataset and then sampling several images for each of these categories. Each mini-batch consists of approximately 5-10 images per category.

For preprocessing, we follow previous work, *e.g.* [7], [8] and resize the longest axis of our images to 256 pixels and pad the shorter axis with white pixels such that images have a size of 256×256 pixels. We subtract the mean from the ImageNet dataset channel-wise from the image. During training time, we crop random 224×224 pixel patches from the images and randomly mirror them. During test time, we use the 224×224 pixel center crop from an image to predict the final feature vector used for retrieval.

In the following section, we show the impact of an ensemble trained with BIER on the strength (*i.e.* accuracy) and correlation of an embedding (Section 5.1). Next, we show that BIER works with several widely used loss functions (Section 5.2), we analyse the impact of the number of groups in an embedding (Section 5.3) and the embedding size (Section 5.4). Then, we demonstrate the effectiveness of our diversity loss functions during initialization (Section 5.5) and as auxiliary loss function during training (Section 5.6). We show the influence of our weighting parameter λ_{div} (Section 5.7). Finally, we show that our method outperforms state-of-the-art methods on several datasets [7], [20], [21], [22], [23] (Section 5.8).

5.1 Strength and Correlation

The performance of an ensemble depends on two elements: the strength (*i.e.* accuracy) of individual learners and the correlation between the learners [17]. Ideally, learners of an ensemble are highly accurate and lowly correlated, so that they can complement each other well.

To evaluate the impact of our contributions on strength and correlation, we compare several models. First, we train a model with a regular loss function with an embedding size of 512 (*Baseline*). Next, we use a simple model averaging approach, where we split the last embedding layer into three non-overlapping groups of size 170, 171 and 171 respectively, initialize them with our Activation Loss initialization method and optimize a discriminative metric loss function on each of these groups separately (*Init-*

*We dump the weights of the network from the Caffe [72] model.

TABLE 1
Evaluation of classifier (Clf.) and feature correlation on CUB-200-2011 [23]. **Best** results are highlighted.

Method	Clf. Corr. ↓	Feature Corr. ↓	R@1 ↑
Baseline-512	-	0.1530	51.76
Init-170-171-171	0.8362	0.1005	53.73
Learner-1-170			51.94
Learner-2-171			51.99
Learner-3-171			52.26
Init-96-160-256	0.9008	0.1197	53.93
Learner-1-96			50.35
Learner-2-160			52.60
Learner-3-256			53.36
BIER-170-171-171	0.7882	0.0988	54.76
Learner-1-170			51.47
Learner-2-171			52.28
Learner-3-171			52.38
BIER-96-160-256	**0.7768**	**0.0934**	**55.33**
Learner-1-96			49.95
Learner-2-160			52.82
Learner-3-256			54.09

170-171-171). Finally, we apply our boosting based reweighting scheme on the three groups (*BIER-170-171-171*).

As discussed in Section 4, we propose to use groups of different sizes proportional to the weighting of the online boosting algorithm, as subsequent learners have to deal with harder samples. To this end, we divide the embedding into differently sized groups. We assign the first learner a size of 96 neurons, the second learner 160 neurons and the last learner 256 neurons. Finally, we train a model with our Activation Loss initialization method (*Init-96-160-256*) and add our boosting method (*BIER-96-160-256*) on top of these learners.

As shown in Table 1, initializing the weight matrix such that activations are independent already achieves a notable improvement over our baseline model. Additionally, our boosting method significantly increases the accuracy of the ensemble. Without boosting, the individual classifiers are highly correlated. By training successive classifiers on reweighted samples, the classifiers focus on different training examples leading to less correlated classifiers. Interestingly, the individual weak learners trained with just our Activation Loss initialization method achieve similar accuracy compared to our boosted learners (*e.g.* 51.94 vs 51.47 of *Learner-1-170*), but the combination achieves a significant improvement since each group focuses on a different part of the dataset.

5.2 Loss Functions

To show that BIER works with several loss functions such as triplet loss or contrastive loss, we train a baseline CNN with embedding size of 512 and then with our boosting based method. For our method, we set the group size to 96, 160 and 256 respectively. In Table 2 we see that binomial deviance, triplet loss and contrastive loss can benefit from our method.

Further, we see that our method performs best for loss functions with smooth (*i.e.* continuous) gradient. We hypothesize that this is due to the fact that non-smooth loss functions convey less information in their gradient. The gradient of the triplet and contrastive loss (for negative samples) is either 0 or 1, whereas the gradient of binomial deviance has continuous values between 0 and 1.

TABLE 2
Evaluation of loss functions on CUB-200-2011 [23].

Method	Feature Corr. ↓	R@1 ↑
Triplet-512	0.2122	50.12
Triplet-96-160-256	**0.1158**	**53.31**
Contrastive-512	0.1639	50.62
Contrastive-96-160-256	**0.1246**	**53.8**
Binomial-Deviance-512	0.1530	51.76
Binomial-Deviance-96-160-256	**0.0934**	**55.33**

5.3 Number of Groups

We demonstrate the influence of the number of groups on our method. To this end, we fix the embedding size to 512 and run our method with $M = \{2, 3, 4, 5\}$ groups. The group size is proportional to the final weights of our boosting algorithm (see Section 4). In Table 3 we report the correlation of the feature embedding, the R@1 score of the ensemble and the average of the R@1 score of each individual learner. We see that with a fixed embedding size of 512, the optimal number of learners for our method is 3-4. For a larger number of groups the strength of individual learners declines and hence performance decreases. For a smaller number of groups the individual embeddings are larger. They achieve higher individual accuracy, but are more correlated with each other, since they benefit less from the gradient boosting algorithm.

TABLE 3
Evaluation of group sizes on CUB-200-2011 [23].

Group Sizes	Clf. Corr. ↓	Avg R@1 ↑	R@1 ↑
Baseline	-	-	51.76
170-342	0.8252	**53.06**	54.66
96-160-256	0.7768	52.29	55.33
52-102-152-204	0.7091	50.67	**55.62**
34-68-102-138-170	**0.6250**	48.5	54.9

5.4 Embedding Sizes

Next, we show the effect of different embedding sizes. We train a CNN with embedding sizes of 384, 512, 1024 with BIER and compare it to a regular CNN. For our method, we split the embeddings into several groups according to the weights of the learners (see Section 4). We divide the 384 sized embedding into groups of size 64, 128 and 192, respectively. For the embedding of size 512 we use groups of size 96, 160 and 256. Finally, for the largest embedding we use groups of size 50, 96, 148, 196, 242 and 292.

We use the binomial deviance loss function, as it consistently achieves best results compared to triplet loss or contrastive loss (recall Table 2). In Table 4 we see that our method yields a consistent gain for a variety of different embedding sizes. For larger embedding sizes a larger number of groups is more beneficial. We found that the main reason for this is that larger embeddings are

TABLE 4
Evaluation of embedding size on CUB-200-2011 [23].

Method	Feature Corr. ↓	R@1 ↑
Baseline-384	0.1453	51.57
BIER-64-128-192	**0.0939**	**54.66**
Baseline-512	0.1530	51.76
BIER-96-160-256	**0.0934**	**55.33**
Baseline-1024	0.1480	52.89
BIER-50-96-148-196-242-292	**0.0951**	**55.99**

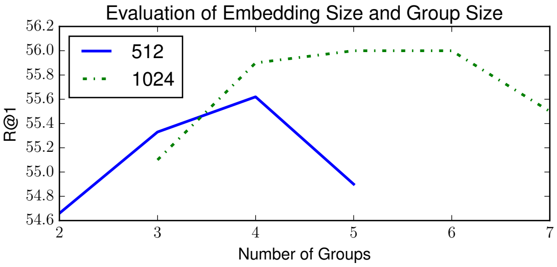

Fig. 6. Evaluation of different embedding sizes and group sizes on the CUB-200-2011 [23] dataset.

more likely to over-fit. Hence, it is more beneficial to train several smaller learners which complement each other better.

Further, we illustrate the effect of the number of learners and the number of groups in Fig. 6. We observe that with larger embedding sizes our method can use a larger number of groups. The main reason for that is that larger embedding sizes have typically more redundancy (hence higher correlation) compared to smaller embedding sizes. Therefore, it is more beneficial to split a larger embedding into a larger number of groups. We set the group sizes proportional to the weight of our boosting algorithm (Section 4). For the interested reader, we also list the corresponding group sizes in our supplementary.

5.5 Impact of Initialization

To show the effectiveness of both, our Activation Loss and Adversarial Loss for weight initialization, we compare it with random initialization, as proposed by Glorot *et al.* [54] and an orthogonal initialization method [56]. All networks are trained with binomial deviance as loss function with our proposed boosting based reweighting scheme. We report mean R@1 of the three methods.

In Table 5 we see that BIER with both our initialization methods achieves better accuracy compared to orthogonal or random initialization. This is due to the fact that with our initialization method learners are already less correlated at the beginning of the training. This makes it easier for the boosting algorithm to maintain diversity of our learners during training.

TABLE 5
Evaluation of Glorot, orthogonal and our Activation Loss and Adversarial Loss initialization method on CUB-200-2011 [23].

Method	R@1
Glorot	54.41
Orthogonal	54.58
Activation Loss	**55.33**
Adversarial Loss	55.04

5.6 Impact of Auxiliary Loss Functions

To show the benefits of adding our diversity loss functions during training as auxiliary loss function, we run several experiments on the CUB-200-2011 [23] dataset. We compare our Adversarial Loss function to our Activation Loss function and a network which does not use an auxiliary loss function during training. All networks are trained with the boosting based reweighting scheme and use an embedding size of 512 with 3 groups (*i.e.* 96, 160 and 256 learners). Further, we observe that we can train our networks with auxiliary loss function with an order of magnitude higher learning rate (*i.e.* $1e^{-5}$ instead of $1e^{-6}$), which results in significantly faster convergence times. We report the R@1 accuracy of all our methods.

As we can see in Table 6, by including an auxiliary loss during training we significantly improve over our previous baseline BIER [18], which used our boosting based training but the Activation Loss only during initialization. By including the auxiliary loss function during training, we can improve the stability of training, allowing our models to be trained with larger learning rates and therefore faster convergence. Training BIER [18] without auxiliary loss functions and with such high learning rates yields a significant drop in performance, since training becomes too unstable.

Finally, the Adversarial Loss function outperforms the Activation Loss function by a significant margin. We hypothesize this is due to the fact that the Activation Loss function constrains the individual learners too much. The Activation Loss encourages the ensemble that for a given training sample, only a single learner should be active and all other learners should be close to zero. In contrast to that, our Adversarial Loss minimizes similarity between embeddings w.r.t. an adversarial regressor, which tries to make two vector spaces as similar as possible under a non-linear mapping. According to our results, minimizing similarity is more effective for reducing correlation than suppressing entire vector spaces.

We also analyze the impact on strength and correlation of our auxiliary loss functions on our ensemble. We show these results in Table 7. Notably, by including an auxiliary loss function we can significantly reduce correlation of the feature vectors as well as the correlation between classifiers. This suggests that our auxiliary loss functions further reduce redundancies in our embedding and therefore improve results. Compared to the Activation Loss, our Adversarial Loss can reduce the correlation between classifiers more effectively and achieves a higher accuracy in terms of R@1.

The individual learners of the Adversarial Loss achieve comparable accuracy to the learners of the Activation Loss (*i.e.* 51.1% vs 51.3%, 53.8% vs 53.5% and 55.3% vs 55.2%). The Adversarial Loss, however, can significantly reduce the correlation between classifiers (*i.e.* 0.6031 vs 0.7310) and features (*i.e.* 0.0731 vs 0.0882). As a consequence, the individual learners are more diverse from each other and complement each other better. Therefore, our Adversarial Loss achieves a significantly better ensemble accuracy of 57.5% vs 56.5%.

When we use our Adversarial Loss as auxiliary loss during training, in contrast to the work of Ganin *et al.* [19], which uses the gradient reversal layer for domain adaptation, we do not require a dynamic schedule for the regularization parameter $\lambda_{\texttt{div}}$ (see Section 4.2.2). Instead, we keep this weighting parameter fixed. Rather than scaling back the gradients inside the gradient reversal layer, we weight the loss function of our adversarial network with $\lambda_{\texttt{div}}$. As a consequence, our adversarial auxiliary network

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2848925, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 6
Comparison of auxiliary loss functions on CUB [23]. Our adversarial loss function significantly improves accuracy over our baseline (BIER [18]) and enables higher learning rates and faster convergence.

Method	R@1	Learning Rate	Iterations
No Auxiliary Loss	55.3	$1e^{-6}$	50K
No Auxiliary Loss	52.3	$1e^{-5}$	15K
Activation Loss	56.5	$1e^{-5}$	15K
Adversarial Loss	**57.5**	$1e^{-5}$	15K

TABLE 7
Impact of auxiliary loss functions on strength and correlation.

Method	Clf. Corr. ↓	Feature Corr. ↓	R@1 ↑
BIER-96-160-256	0.7768	0.0934	55.3
Learner-1-96			50.0
Learner-2-160			52.8
Learner-3-256			54.1
Activation BIER-96-160-256	0.7130	0.0882	56.5
Learner-1-96			51.3
Learner-2-160			53.5
Learner-3-256			55.2
Adversarial BIER-96-160-256	**0.6031**	**0.0731**	**57.5**
Learner-1-96			51.1
Learner-2-160			53.8
Learner-3-256			55.3

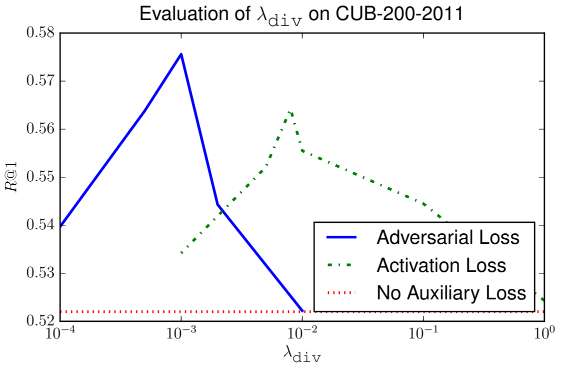

Fig. 7. Evaluation of $\lambda_{\texttt{div}}$ on CUB-200-2011 [23].

trains slower compared to our base network, which turns out to be beneficial for the training process. We hypothesize that the main reason for this is that the adversarial network gets too strong if we update it too fast, which in turn degrades the performance of the base network.

5.7 Evaluation of the Regularization Parameter

When we add our diversity loss functions during training time we introduce an additional parameter $\lambda_{\texttt{div}}$ (recall Section 4.2.2). To demonstrate its effect, we train several models on the CUB-200-2011 dataset [23] with a learning rate of $1e^{-5}$ and vary $\lambda_{\texttt{div}}$.

In Fig. 7 we see that for our Adversarial Loss $\lambda_{\texttt{div}}$ peaks around $1e^{-3}$, whereas for our Activation Loss $\lambda_{\texttt{div}}$ peaks around $1e^{-2}$. Further, our Adversarial Loss significantly outperforms our Activation Loss by about 1% R@1. Finally, applying any of our loss functions as auxiliary loss function with a learning rate of $1e^{-5}$ significantly improves R@1 compared to networks without an auxiliary loss function trained with the same learning rate. Therefore, by integrating any of our auxiliary loss functions, we can improve the training stability of BIER at higher learning rates.

5.8 Comparison with the State-of-the-Art

We show the robustness of our method by comparing it with the state-of-the-art on the CUB-200-2011 [23], Cars-196 [20], Stanford Online Product [7], In-Shop Clothes Retrieval [22] and VehicleID [21] datasets.

CUB-200-2011 consists of $11,788$ images of 200 bird categories. The Cars-196 dataset contains $16,185$ images of 196 cars classes. The Stanford Online Product dataset consists of $120,053$ images with $22,634$ classes crawled from Ebay. Classes are hierarchically grouped into 12 coarse categories (*e.g.* cup, bicycle, *etc.*). The In-Shop Clothes Retrieval dataset consists of $54,642$ images with $11,735$ clothing classes. VehicleID consists of $221,763$ images with $26,267$ vehicles.

For training on CUB-200-2011, Cars-196 and Stanford Online Products, we follow the evaluation protocol proposed in [7]. For the CUB-200-2011 dataset, we use the first 100 classes ($5,864$ images) for training and the remaining 100 classes ($5,924$ images) for testing. We further use the first 98 classes of the Cars-196 dataset for training ($8,054$ images) and the remaining 98 classes for testing ($8,131$ images). For the Stanford Online Products dataset we use the same train/test split as [7], *i.e.* we use $59,551$ images of $11,318$ classes for training and $60,502$ images of $11,316$ classes for testing. For the In-Shop Clothes Retrieval dataset, we use the predefined $25,882$ training images of $3,997$ classes for training. The test set is partitioned into a query set ($14,218$ images of $3,985$ classes) and a gallery set ($12,612$ images of $3,985$ classes). When evaluating on VehicleID, we use the predefined $110,178$ images of $13,134$ vehicles for training and the predefined test sets (Small, Medium, Large) for testing [21].

We fix all our parameters and train BIER with the binomial deviance loss function and an embedding size of 512 and group size of 3 (*i.e.* we use groups of size 96, 160, 256). For the CUB-200-2011 and Cars-196 dataset we follow previous work, *e.g.* [7], and report our results in terms of Recall@K, $K \in \{1, 2, 4, 8, 16, 32\}$. For Stanford Online Products we also stick to previous evaluation protocols [7] and report Recall@K, $K \in \{1, 10, 100, 1000\}$, for the In-Shop Clothes Retrieval dataset we compare with $K \in \{1, 10, 20, 30, 40, 50\}$ and for VehicleID we evaluate with $K \in \{1, 5\}$. We also report the results for the last learner in our ensemble (*BIER Learner-3*), as it was trained on the most difficult examples. Further, we also show the benefits of using our adversarial loss function during training time in combination with BIER (*A-BIER*) on all datasets and also report the last learner in this ensemble (*A-BIER Learner-3*).

Results and baselines are shown in Tables 8, 9, 10, 11 and 12. Our method in combination with a simple loss function operating on pairs is able to outperform or achieve comparable performance to state-of-the-art methods relying on higher order tuples [7], [32], histograms [8], novel loss functions [34], [35] or hard sample mining strategies [37], [38], [39]. We consistently improve our strong baseline method by a large margin at R@1 on all datasets, which demonstrates the robustness of our approach. Further, by using our adversarial loss function during training (*A-BIER*), we significantly improve over BIER [18] and outperform state-of-the-art methods. On CUB-200-2011 and Cars-196 we can improve

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2848925, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 8
Comparison with the state-of-the-art on the CUB-200-2011 [23] and Cars-196 [20] dataset. **Best** results are highlighted.

	CUB-200-2011						Cars-196					
R@K	1	2	4	8	16	32	1	2	4	8	16	32
Contrastive [7]	26.4	37.7	49.8	62.3	76.4	85.3	21.7	32.3	46.1	58.9	72.2	83.4
Triplet [7]	36.1	48.6	59.3	70.0	80.2	88.4	39.1	50.4	63.3	74.5	84.1	89.8
LiftedStruct [7]	47.2	58.9	70.2	80.2	89.3	93.2	49.0	60.3	72.1	81.5	89.2	92.8
Binomial Deviance [8]	52.8	64.4	74.7	83.9	90.4	94.3	-	-	-	-	-	-
Histogram Loss [8]	50.3	61.9	72.6	82.4	88.8	93.7	-	-	-	-	-	-
N-Pair-Loss [32]	51.0	63.3	74.3	83.2	-	-	71.1	79.7	86.5	91.6	-	-
Clustering [33]	48.2	61.4	71.8	81.9	-	-	58.1	70.6	80.3	87.8	-	-
Proxy NCA [35]	49.2	61.9	67.9	72.4	-	-	73.2	82.4	86.4	87.8	-	-
Smart Mining [37]	49.8	62.3	74.1	83.3	-	-	64.7	76.2	84.2	90.2	-	-
HDC [39]	53.6	65.7	77.0	85.6	91.5	**95.5**	73.7	83.2	89.5	93.8	96.7	98.4
Angular Loss [34]	54.7	66.3	76.0	83.9	-	-	71.4	81.4	87.5	92.1	-	-
Ours Baseline	51.8	63.8	74.1	83.1	90.0	94.8	73.6	82.6	89.0	93.5	96.4	98.2
BIER Learner-3 [18]	54.1	66.1	76.5	84.7	91.2	95.3	76.5	84.9	90.9	94.9	97.6	98.7
BIER [18]	55.3	67.2	76.9	85.1	91.7	**95.5**	78.0	85.8	91.1	95.1	97.3	**98.7**
A-BIER Learner-3	55.3	67.0	76.8	86.0	91.1	95.3	80.6	88.2	92.3	95.8	97.6	98.6
A-BIER	**57.5**	**68.7**	**78.3**	**86.2**	**91.9**	**95.5**	**82.0**	**89.0**	**93.2**	**96.1**	**97.8**	**98.7**

TABLE 9
Comparison with the state-of-the-art on the cropped versions of the CUB-200-2011 [23] and Cars-196 [20] dataset.

	CUB-200-2011						Cars-196					
R@K	1	2	4	8	16	32	1	2	4	8	16	32
PDDM + Triplet [31]	50.9	62.1	73.2	82.5	91.1	94.4	46.4	58.2	70.3	80.1	88.6	92.6
PDDM + Quadruplet [31]	58.3	69.2	79.0	88.4	93.1	95.7	57.4	68.6	80.1	89.4	92.3	94.9
HDC [39]	60.7	72.4	81.9	89.2	93.7	96.8	83.8	89.8	93.6	96.2	97.8	98.9
Margin [38]	63.9	75.3	**84.4**	**90.6**	**94.8**	-	86.9	92.7	95.6	97.6	98.7	-
Ours Baseline	58.9	70.1	79.8	87.6	92.6	96.0	82.6	88.8	93.1	96.1	97.5	98.7
BIER Learner-3 [18]	62.8	73.5	81.9	89.0	93.7	96.7	85.8	91.7	94.8	97.2	98.4	99.2
BIER [18]	63.7	74.0	82.5	89.3	93.8	96.8	87.2	92.2	95.3	97.4	98.5	99.3
A-BIER Learner-3	64.0	74.3	83.1	89.2	94.1	96.9	88.5	93.2	98.9	97.7	98.5	99.2
A-BIER	**65.5**	**75.8**	83.9	90.2	94.2	**97.1**	**90.3**	**94.1**	**96.8**	**97.9**	**98.9**	**99.4**

TABLE 10
Comparison with the state-of-the-art on the Stanford Online Products [7] dataset.

R@K	1	10	100	1000
Contrastive [7]	42.0	58.2	73.8	89.1
Triplet [7]	42.1	63.5	82.5	94.8
LiftedStruct [7]	62.1	79.8	91.3	97.4
Binomial Deviance [8]	65.5	82.3	92.3	97.6
Histogram Loss [8]	63.9	81.7	92.2	97.7
N-Pair-Loss [32]	67.7	83.8	93.0	97.8
Clustering [33]	67.0	83.7	93.2	-
HDC [39]	69.5	84.4	92.8	97.7
Angular Loss [34]	70.9	85.0	93.5	**98.0**
Margin [38]	72.7	86.2	93.8	**98.0**
Proxy NCA [35]	73.7	-	-	-
Ours Baseline	66.2	82.3	91.9	97.4
BIER Learner-3 [18]	72.5	86.3	93.9	97.9
BIER [18]	72.7	86.5	**94.0**	**98.0**
A-BIER Learner-3	74.0	86.8	93.9	97.8
A-BIER	**74.2**	**86.9**	**94.0**	97.8

TABLE 11
Comparison with the state-of-the-art on the In-Shop Clothes Retrieval [22] dataset.

R@K	1	10	20	30	40	50
FasionNet + Joints [22]	41.0	64.0	68.0	71.0	73.0	73.5
FasionNet + Poselets [22]	42.0	65.0	70.0	72.0	72.0	75.0
FasionNet [22]	53.0	73.0	76.0	77.0	79.0	80.0
HDC [39]	62.1	84.9	89.0	91.2	92.3	93.1
Ours Baseline	70.6	90.5	93.4	94.7	95.5	96.1
BIER Learner-3 [18]	76.4	92.7	95.0	96.1	96.6	97.0
BIER [18]	76.9	92.8	95.2	96.2	96.7	97.1
A-BIER Learner-3	82.8	95.0	96.8	97.4	97.7	98.0
A-BIER	**83.1**	**95.1**	**96.9**	**97.5**	**97.8**	**98.0**

TABLE 12
Comparison with the state-of-the-art on VehicleID [21].

	Small		Medium		Large	
R@K	1	5	1	5	1	5
Mixed Diff+CCL [21]	49.0	73.5	42.8	66.8	38.2	61.6
GS-TRS loss [74]	75.0	83.0	74.1	82.6	73.2	81.9
Ours Baseline	78.0	87.5	73.0	84.7	67.9	82.4
BIER Learner-3 [18]	82.6	90.5	79.3	88.0	75.5	86.0
BIER [18]	82.6	90.6	79.3	88.3	76.0	86.4
A-BIER Learner-3	86.0	92.7	83.2	88.6	81.5	88.6
A-BIER	**86.3**	**92.7**	**83.3**	**88.7**	**81.9**	**88.7**

over the state-of-the-art significantly by about 2-4% at R@1. The Stanford Online Products, the In-Shop Clothes Retrieval and VehicleID datasets are more challenging since there are only few ($\approx$ 5) images per class. On these datasets our auxiliary adversarial loss achieves a notable improvement over BIER of 1.5%, 6.1% and 3-6%, respectively. *A-BIER* outperforms state-of-the-art methods on all datasets. Notably, even the last learner in our adversarial ensemble (*A-BIER Learner-3*), evaluated on its own, already outperforms the state-of-the-art on most of the datasets.

6 Conclusion

In this work we cast training an ensemble of metric CNNs with a shared feature representation as online gradient boosting problem. We further introduced two loss functions which encourage diversity in our ensemble. We applied these loss functions either during initialization or as auxiliary loss function during training. In our experiments we showed that our loss functions increase diversity among our learners and, as a consequence, significantly increase accuracy of our ensemble. Further, we showed that our novel Adversarial Loss function outperforms our previous Activation Loss function. This is because our Adversarial Loss increased the diversity in our ensemble more effectively. Consequently, the ensemble accuracy was higher for networks trained with our Adversarial Loss. Our proposed method does not introduce any additional parameters during test time and has only negligible additional computational cost, both, during training and test time. Our extensive experiments showed that BIER significantly reduces correlation on the last hidden layer of a CNN and works with several different loss functions. By training with our auxiliary loss function Adversarial BIER outperformed state-of-the-art methods on several datasets, *i.e.* [7], [20], [21], [22], [23].

Our results show that our main idea, splitting a CNN at the end into multiple learners and increasing the diversity of these learners, reduces overfitting in CNNs. We hope this technique can be beneficial in other areas in Computer Vision where overfitting is a problem, such as object detection, semantic segmentation, *etc*.

Acknowledgments

This project was supported by the Austrian Research Promotion Agency (FFG) projects MANGO (836488) and Darknet (858591). We gratefully acknowledge the support of NVIDIA Corporation with the donation of GPUs used for this research.

References

[1] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a Similarity Metric Discriminatively, with Application to Face Verification," in *Proc. CVPR*, 2005.

[2] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality Reduction by Learning an Invariant Mapping," in *Proc. CVPR*, 2006.

[3] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in *Proc. CVPR*, 2015.

[4] K. Q. Weinberger and L. K. Saul, "Distance Metric Learning for Large Margin Nearest Neighbor Classification," *JMLR*, vol. 10, no. 2, pp. 207–244, 2009.

[5] M. T. Law, N. Thome, and M. Cord, "Quadruplet-wise Image Similarity Learning," in *Proc. ICCV*, 2013.

[6] W. S. Zheng, S. Gong, and T. Xiang, "Reidentification by Relative Distance Comparison," *TPAMI*, vol. 35, no. 3, pp. 653–668, 2013.

[7] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese, "Deep Metric Learning via Lifted Structured Feature Embedding," in *Proc. CVPR*, 2016.

[8] E. Ustinova and V. Lempitsky, "Learning Deep Embeddings with Histogram Loss," in *Proc. NIPS*, 2016.

[9] P. Wohlhart and V. Lepetit, "Learning Descriptors for Object Recognition and 3D Pose Estimation," in *Proc. CVPR*, 2015.

[10] G. Waltner, M. Opitz, and H. Bischof, "BaCoN: Building a Classifier from only N Samples," in *Proc. CVWW*, 2016.

[11] B. Kumar, G. Carneiro, and I. Reid, "Learning Local Image Descriptors with Deep Siamese and Triplet Convolutional Networks by Minimising Global Loss Functions," in *Proc. CVPR*, 2016.

[12] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer, "Discriminative Learning of Deep Convolutional Feature Point Descriptors," in *Proc. CVPR*, 2015.

[13] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep Face Recognition." in *Proc. BMVC*, 2015.

[14] H. Shi, Y. Yang, X. Zhu, S. Liao, Z. Lei, W. Zheng, and S. Z. Li, "Embedding Deep Metric for Person Re-identification: A Study Against Large Variations," in *Proc. ECCV*, 2016.

[15] R. Tao, E. Gavves, and A. W. Smeulders, "Siamese Instance Search for Tracking," in *Proc. CVPR*, 2016.

[16] M. Muja and D. G. Lowe, "Scalable Nearest Neighbor Algorithms for High Dimensional Data," *TPAMI*, vol. 36, no. 11, pp. 2227–2240, 2014.

[17] L. Breiman, "Random Forests," *ML*, vol. 45, no. 1, pp. 5–32, 2001.

[18] M. Opitz, G. Waltner, H. Possegger, and H. Bischof, "BIER: Boosting Independent Embeddings Robustly," in *Proc. ICCV*, 2017.

[19] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-Adversarial Training of Neural Networks," *JMLR*, vol. 17, no. 59, pp. 1–35, 2016.

[20] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3D Object Representations for Fine-Grained Categorization," in *Proc. ICCV Workshops*, 2013.

[21] H. Liu, Y. Tian, Y. Wang, L. Pang, and T. Huang, "Deep Relative Distance Learning: Tell the Difference Between Similar Vehicles," in *Proc. CVPR*, 2016.

[22] Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang, "DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations," in *Proc. CVPR*, 2016.

[23] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," California Institute of Technology, Tech. Rep. CNS-TR-2011-001, 2011.

[24] A. Bellet, A. Habrard, and M. Sebban, "A survey on metric learning for feature vectors and structured data," *arXiv*, vol. abs/1306.6709, 2013.

[25] J. Bi, D. Wu, L. Lu, M. Liu, Y. Tao, and M. Wolf, "Adaboost on Low-Rank PSD Matrices for Metric Learning," in *Proc. CVPR*, 2011.

[26] M. Liu and B. C. Vemuri, "A Robust and Efficient Doubly Regularized Metric Learning Approach," in *Proc. ECCV*, 2012.

[27] R. Negrel, A. Lechervy, and F. Jurie, "MLBoost Revisited: A Faster Metric Learning Algorithm for Identity-Based Face Retrieval," in *Proc. BMVC*, 2016.

[28] C. Shen, J. Kim, L. Wang, and A. v. d. Hengel, "Positive Semidefinite Metric Learning using Boosting-Like Algorithms," *JMLR*, vol. 13, pp. 1007–1036, 2012.

[29] D. Kedem, S. Tyree, F. Sha, G. R. Lanckriet, and K. Q. Weinberger, "Non-linear Metric Learning," in *Proc. NIPS*, 2012.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *IJCV*, vol. 115, no. 3, pp. 1–42, 2015.

[31] C. Huang, C. C. Loy, and X. Tang, "Local Similarity-Aware Deep Feature Embedding," in *Proc. NIPS*, 2016.

[32] K. Sohn, "Improved Deep Metric Learning with Multi-Class n-Pair Loss Objective," in *Proc. NIPS*, 2016.

[33] H. O. Song, S. Jegelka, V. Rathod, and K. Murphy, "Deep Metric Learning via Facility Location," in *Proc. CVPR*, 2017.

[34] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin, "Deep Metric Learning With Angular Loss," in *Proc. ICCV*, 2017.

[35] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh, “No Fuss Distance Metric Learning Using Proxies,” in *Proc. ICCV*, 2017.

[36] O. Rippel, M. Paluri, P. Dollar, and L. Bourdev, “Metric Learning with Adaptive Density Discrimination,” in *Proc. ICLR*, 2016.

[37] B. Harwood, V. Kumar B G, G. Carneiro, I. Reid, and T. Drummond, “Smart Mining for Deep Metric Learning,” in *Proc. ICCV*, 2017.

[38] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krähenbühl, “Sampling Matters in Deep Embedding Learning,” in *Proc. ICCV*, 2017.

[39] Y. Yuan, K. Yang, and C. Zhang, “Hard-Aware Deeply Cascaded Embedding,” in *Proc. ICCV*, 2017.

[40] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu, “Deeply-Supervised Nets.” in *Proc. AISTATS*, 2015.

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” in *Proc. CVPR*, 2015.

[42] Y. Freund and R. E. Schapire, “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting,” *JCSS*, vol. 55, no. 1, pp. 119 – 139, 1997.

[43] J. H. Friedman, “Greedy Function Approximation: a Gradient Boosting Machine.” *AoS*, vol. 29, no. 5, pp. 1189–1232, 2001.

[44] A. Beygelzimer, E. Hazan, S. Kale, and H. Luo, “Online Gradient Boosting,” in *Proc. NIPS*, 2015.

[45] A. Beygelzimer, S. Kale, and H. Luo, “Optimal and Adaptive Algorithms for Online Boosting.” in *Proc. ICML*, 2015.

[46] S.-T. Chen, H.-T. Lin, and C.-J. Lu, “An Online Boosting Algorithm with Theoretical Justifications,” *Proc. ICML*, 2012.

[47] C. Leistner, A. Saffari, P. M. Roth, and H. Bischof, “On Robustness of On-line Boosting - a Competitive Study,” in *Proc. ICCV Workshops*, 2009.

[48] N. Karianakis, T. J. Fuchs, and S. Soatto, “Boosting Convolutional Features for Robust Object Proposals,” *arXiv*, vol. abs/1503.06350, 2015.

[49] B. Yang, J. Yan, Z. Lei, and S. Z. Li, “Convolutional Channel Features,” in *Proc. CVPR*, 2015.

[50] E. Walach and L. Wolf, “Learning to Count with CNN Boosting,” in *Proc. ECCV*, 2016.

[51] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *JMLR*, vol. 15, pp. 1929–1958, 2014.

[52] S. Han, Z. Meng, A. S. Khan, and Y. Tong, “Incremental Boosting Convolutional Neural Network for Facial Action Unit Recognition,” in *Proc. NIPS*, 2016.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Proc. NIPS*, 2012.

[54] X. Glorot and Y. Bengio, “Understanding the Difficulty of Training Deep FeedForward Neural Networks,” in *Proc. AISTATS*, 2010.

[55] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” in *Proc. ICCV*, 2015.

[56] A. M. Saxe, J. L. McClelland, and S. Ganguli, “Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks,” in *Proc. ICLR*, 2014.

[57] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell, “Data-dependent Initializations of Convolutional Neural Networks,” in *Proc. ICLR*, 2016.

[58] D. Mishkin and J. Matas, “All you need is a good init,” in *Proc. ICLR*, 2016.

[59] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle *et al.*, “Greedy Layer-Wise Training of Deep Networks,” in *Proc. NIPS*, 2006.

[60] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” *NECO*, vol. 18, no. 7, pp. 1527–1554, 2006.

[61] Y. Liu and X. Yao, “Ensemble Learning via Negative Correlation,” *Neural Netw.*, vol. 12, no. 10, pp. 1399–1404, 1999.

[62] H. Chen and X. Yao, “Multiobjective Neural Network Ensembles Based on Regularized Negative Correlation Learning,” *TKDE*, vol. 22, no. 12, pp. 1738–1751, 2010.

[63] S. Wang, H. Chen, and X. Yao, “Negative Correlation Learning for Classification Ensembles,” in *Proc. IJCNN*, 2010.

[64] M. Opitz, H. Possegger, and H. Bischof, “Efficient Model Averaging for Deep Neural Networks,” in *Proc. ACCV*, 2016.

[65] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Nets,” in *Proc. NIPS*, 2014.

[66] A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” in *Proc. ICLR*, 2016.

[67] X. Mao, Q. Li, H. Xie, R. Y. Lau, and Z. Wang, “Least Squares Generative Adversarial Networks,” in *Proc. ICCV*, 2017.

[68] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks,” in *Proc. ICCV*, 2017.

[69] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, “Adversarial Discriminative Domain Adaptation,” in *Proc. CVPR*, 2017.

[70] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, “Simultaneous Deep Transfer Across Domains and Tasks,” in *Proc. ICCV*, 2015.

[71] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org.

[72] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional Architecture for Fast Feature Embedding,” *arXiv*, vol. abs/1408.5093, 2014.

[73] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *Proc. ICLR*, 2015.

[74] Y. Bai, F. Gao, Y. Lou, S. Wang, T. Huang, and L.-Y. Duan, “Incorporating Intra-Class Variance to Fine-Grained Visual Recognition,” *arXiv*, vol. abs/1703.00196, 2017.

Michael Opitz received a MSc in Visual Computing from Vienna University of Technology, Austria, in 2013. He is currently working towards his PhD degree in the Institute of Computer Graphics and Vision, Graz University of Technology, Austria. His research interests include computer vision and machine learning with a focus on metric learning and object detection.

Georg Waltner received a MSc in Telematics in 2014 from Graz University of Technology, Austria. He is currently working towards his PhD degree in the Institute of Computer Graphics and Vision, Graz University of Technology, Austria. His research interest lies in classification of food items with respect to finding optimal embedding spaces and inserting new possibly unseen categories.

Horst Possegger is currently a Computer Science PhD student at the Institute for Computer Graphics and Vision Graz University of Technology, Austria. He received the BSc and MSc degrees in Software Development and Business Management from Graz University of Technology in 2011 and 2013, respectively. His research interests include visual object tracking and detection, human behaviour analysis, and video analysis in general.

Horst Bischof Horst Bischof received his MSc and Ph.D. degree in computer science from the Vienna University of Technology in 1990 and 1993. In 1998, he got his Habilitation (venia docendi) for applied computer science. Currently, he is Vice Rector for Research at Graz University of Technology and Professor at the Institute for Computer Graphics and Vision. He has published more than 650 peer reviewed scientific papers.

Horst Bischof is member of the European academy of sciences and has received several awards (20), among them the 29th Pattern Recognition award in 2002, the main price of the German Association for Pattern Recognition (DAGM) in 2007 and 2012, the best scientific paper award at the BMCV 2007, the BMVC best demo award 2012 and the best scientific paper awards at the ICPR 2008, ICPR 2010, PCV 2010, AAPR 2010 and ACCV 2012.