BIER - Boosting Independent Embeddings Robustly

Michael Opitz, Georg Waltner, Horst Possegger, Horst Bischof
Institute for Computer Graphics and Vision
Graz University of Technology, Austria
{michael.opitz, waltner, possegger, bischof}@icg.tugraz.at

Abstract

Learning similarity functions between image pairs with deep neural networks yields highly correlated activations of large embeddings. In this work, we show how to improve the robustness of embeddings by exploiting independence in ensembles. We divide the last embedding layer of a deep network into an embedding ensemble and formulate training this ensemble as an online gradient boosting problem. Each learner receives a reweighted training sample from the previous learners. This leverages large embedding sizes more effectively by significantly reducing correlation of the embedding and consequently increases retrieval accuracy of the embedding. Our method does not introduce any additional parameters and works with any differentiable loss function. 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 by a significant margin.

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. Metric learning has a large variety of applications, such as image or object retrieval (*e.g.* [35, 48, 53]), single-shot object classification (*e.g.* [35, 48, 51]), keypoint descriptor learning (*e.g.* [25, 43]), face verification (*e.g.* [36, 40]), person re-identification (*e.g.* [42, 48]), object tracking (*e.g.* [46]), *etc.* To learn such metrics, several approaches based on image pairs [9, 13], triplets [40, 52] or quadruples [26, 56] have been proposed in the past. In this work we focus on learning simple similarity functions based on the dot product, since they can be computed rapidly and can be used by approximate search methods (*e.g.* [33]) for large scale image retrieval. Typically, especially with large embedding sizes, the accuracy of these methods saturates or declines due to over-fitting [35].

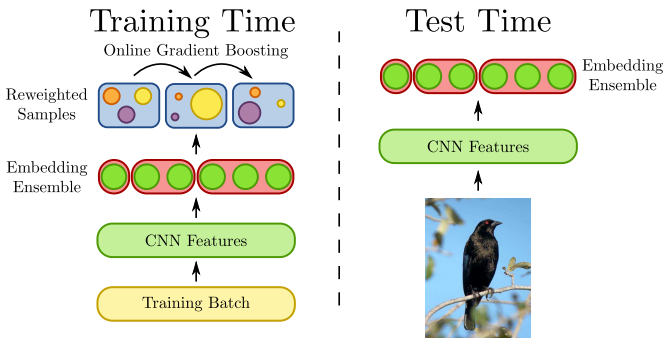

Figure 1. BIER divides a large embedding into an ensemble of several smaller embeddings. During training we reweight the training set for successive learners in the ensemble with the negative gradient of the loss function. During test time we concatenate the individual embeddings of all learners into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

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 Figure 1). Each group is a separate metric 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 [7]. Ideally, individual learners are highly accurate and have low correlation with each other, so that they complement each other during test time.

Naively optimizing a global loss function for this ensemble cannot show any benefits since all learners have access to the same feature representation and the same training samples. All groups will end up learning a highly correlated embedding, which results in no performance improvements. 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. Consequently, successive learners will focus on distinct samples from the previous learners, resulting

in a more diverse feature representation (Section 3.1). 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 3.2). The matrix is initialized from a solution of an optimization problem which minimizes the correlation between groups.

We demonstrate the effectiveness of our metric on several image retrieval datasets [23, 29, 31, 35, 49]. In our evaluation we show that BIER significantly reduces the correlation of large embeddings (see Section 4.1) and works with several loss functions (see Section 4.2) 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. The only design parameter of our method is the number of groups into which we divide our embedding. We show that BIER outperforms state-of-the-art methods on the CUB-200-2011 [49], Cars-196 [23], Stanford Online Products [35], In-Shop Clothes Retrieval [31] and VehicleID [29] datasets (Section 4.6).

2. Related Work

Our work is related to metric learning and boosting in combination with CNNs. Additionally, since we propose a novel initialization method, we discuss related data dependent initialization methods for CNNs.

2.1. Metric Learning

The main objective of metric learning in Computer Vision is to learn a function $f(\cdot) : \mathbb{R}^k \mapsto \mathbb{R}^d$ which maps a k-dimensional input vector, which is typically an input 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, and semantically dissimilar images should be far apart from each other.

For a complete review of metric learning approaches we refer the interested reader to [2]. In this work we focus our discussion on boosting based metric learning approaches and deep CNNs based approaches.

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 $\mathbf{M}$, *e.g*. [6, 30, 34, 41]. $\mathbf{M}$ can be factorized into $\mathbf{M} = \mathbf{L}\mathbf{L}^\top$, where $\mathbf{L} \in \mathbb{R}^{k \times d}$ is the projection into the d-dimensional vector space. Further, Kedem *et al*. [20] proposed gradient boosted trees for metric learning.

CNN based methods typically use special loss functions to train deep networks which operate on image pairs, triplets or quadruples. One of the most widely used pairwise loss functions for metric learning is the contrastive loss function, *e.g*. [9, 13, 35]. 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*. [9, 13].

Other approaches adopt the Large Margin Nearest Neighbor (LMNN) formulation [52] and sample triplets consisting of a positive image pair and a negative image pair, *e.g*. [35, 36, 40, 53]. 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.

Recent approaches propose new loss functions for metric learning. Song *et al*. [35] propose to lift a mini-batch to a matrix of pairwise distances between samples. Further, they use a structural loss function on this distance matrix to train the neural network. Ustinova *et al*. [48] 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*. [17] introduce a position dependent deep metric unit which is capable of learning a similarity metric adaptive to the local feature space. Sohn *et al*. [44] generalize the triplet loss to n-tuples and propose a more efficient batch construction scheme. Rippel *et al*. [37] propose a "magnet" loss function which models multimodal data distributions and minimizes the overlap between distributions of different classes. Our work is complementary to the above approaches. We show that combining existing loss functions with our method yields significant improvements (see Section 4.2).

Yuan *et al*. [55] propose a hard-aware deeply cascaded embedding. They leverage the benefits of deeply supervised networks [27, 45] 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. To achieve this, we split our high dimensional embedding into several learners which we train with online gradient boosting. Successive learners are trained on reweighted samples, which significantly reduces the correlation between the learners and consequently within the embedding. Further, in contrast to the hard-aware deeply cascaded embedding, our method allows continuous weights for samples depending on the loss function.

2.2. Boosting for CNNs

Boosting is a greedy ensemble learning method, which iteratively trains an ensemble from several weak learners [10]. The original boosting algorithm, AdaBoost [10], minimizes an exponential loss function. Friedman *et al*. [11] extend the boosting framework to allow minimiz-

ing 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.* [4, 5, 8, 28]. 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.* [19, 54] 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 used as 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 finetuned 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, all our learners share a common feature representation. Hence, our method does not introduce any additional parameters.

Very recently, Han *et al.* [14] proposed 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 [10] 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 global loss function.

2.3. Initialization Methods

Most initialization methods for CNNs initialize weights randomly either with carefully chosen variance parameters, *e.g.* [24], or depending on the fan-in and fan-out of a weight matrix, *e.g.* [12, 15]. Rather than focusing on determining the variance of the weight matrix, Saxe *et al.* [39] propose to initialize the weight matrix as orthogonal matrix.

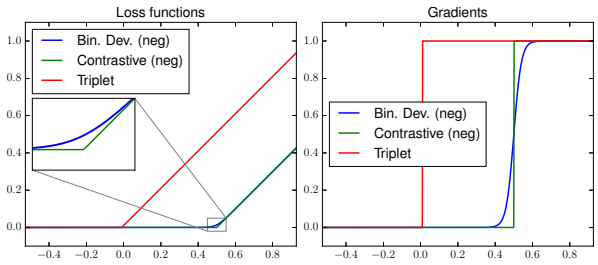

Figure 2. Illustration of triplet loss, contrastive loss (for negative samples) and binomial deviance loss (for negative samples) and their gradients. Triplet and contrastive loss have a non-continuous gradient, whereas binomial deviance has a continuous gradient.

Binomial Dev.	$\log(1 + e^{-(2y-1)\alpha(s-\beta)C_y})$
Contrastive	$(1-y)\max(0, s-m) + y(s-1)^2$
Triplet	$\max(0, s^- - s^+ + m)$

Table 1. Overview of loss functions used in our work.

Recently, several approaches which initialize weights depending on the input data were proposed, *e.g.* [22, 32]. These methods typically scale a random weight matrix such that the activations on the training set have unit variance.

Several works, *e.g.* [3, 16], greedily initialize 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.

In contrast to these works, we initialize the weight matrix from a solution of an optimization problem which explicitly minimizes correlation between groups of features. With this initialization our weak learners already have low correlation at the beginning of the training process.

3. Boosting a Metric Network

Our method builds upon metric CNNs, *e.g.* [17, 35, 44, 48]. 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 that similar pairs should be close to each other and dissimilar pairs should be far apart from each other, *e.g.* [35].

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:

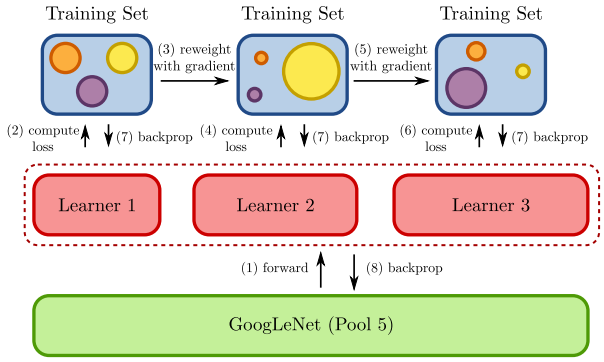

Figure 3. We divide the embedding of a metric CNN into several weak learners and cast training them as online gradient boosting problem. Each learner iteratively reweights samples according to the gradient of the loss function. Training a metric CNN this way encourages successive learners to focus on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

$s(f(\boldsymbol{x}^{(1)}), f(\boldsymbol{x}^{(2)})) = \frac{f(\boldsymbol{x}^{(1)})^\top f(\boldsymbol{x}^{(2)})}{\|f(\boldsymbol{x}^{(1)})\| \cdot \|f(\boldsymbol{x}^{(2)})\|}$. This has the advantage that the similarity is bounded between $[-1, +1]$.

In our framework, we do not use a Siamese architecture, *e.g.* as [9, 13]. Instead, we follow recent work, *e.g.* [35, 40, 48], 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.

We consider three different loss functions (defined in Table 1 and illustrated in Figure 2), which are commonly used to train metric networks, *e.g.* [25, 36, 40, 42]. To avoid cluttering the notation, let $s = s(f(\boldsymbol{x}^{(1)}), f(\boldsymbol{x}^{(2)}))$ be the similarity score between image $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$. Let $y \in \{1, 0\}$ denote the label of the image pair (*i.e.* 1 for similar pairs, and 0 for dissimilar pairs). Let s^- denote the similarity score for a negative image pair and s^+ denote the similarity score for a positive image pair. m denotes the margin for the contrastive and triplet loss, which is set to 0.5 and 0.01, respectively, in all our experiments. α and β are scaling and translation parameters and are set to 2 and 0.5, similar to [48]. Finally, we follow [48] and set the cost C_y to balance positive and negative pairs as

$$C_y = \begin{cases} 1 & \text{if } y = 1 \\ 25 & \text{if } y = 0. \end{cases} \quad (1)$$

3.1. Online Gradient Boosting CNNs for Metric Learning

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.* [4, 5, 8, 28]. 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.

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

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

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 set 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 [4] with fixed weights α_m and integrate it within a CNN. Naively 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 Figure 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 Stochastic Gradient Descent (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 supplemental 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 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 high gradients (*i.e.* are misclassified) by previous learners.

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. We experimentally verify this design choice in Section 4.1.

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

Algorithm 1: Online gradient boosting algorithm for our CNN.

During test time our method predicts a single feature vector for an input image $\boldsymbol{x}$, which can be used by approximate search methods, *e.g.* [33]. We simply compute the embeddings from all weak learners $f_1(\cdot), f_2(\cdot), \ldots f_M(\cdot)$, 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 $\boldsymbol{x}$.

3.2. Weight Initialization

We want to encourage that learners in our ensemble have low correlation with each other already at the beginning of the training. To this end we initialize the weight vector $\boldsymbol{W}$ of the last layer of our network such that the activations of the training set are independent across different groups.

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 learn the initialization of an 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 $X = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}\}$ denote the training set. In all our experiments we use feature vectors extracted from the last hidden layer of a pre-trained CNN. Intuitively, we want to ensure that activations are not correlated between groups. We formulate this with the following optimization problem:

$$\arg\min_{\boldsymbol{W}} \sum_{n=1}^{N} \sum_{\substack{i=1,\\ j=i+1}}^{M} \sum_{\substack{k \in G_i,\\ l \in G_j}} (a_k^{(n)} \cdot a_l^{(n)})^2 + \lambda \sum_{i=1}^{d} (\boldsymbol{w}_i^\top \boldsymbol{w}_i - 1)^2$$
$$\text{s.t. } \boldsymbol{a}^{(n)} = \boldsymbol{x}^{(n)\top} \boldsymbol{W}, \tag{3}$$

where $\boldsymbol{a}^{(n)}$ denotes the activation of feature vector $\boldsymbol{x}^{(n)}$, $a_k^{(n)}$ the k-th dimension of $\boldsymbol{a}^{(n)}$, $\boldsymbol{w}_i$ (with $1 \leq i \leq d$) are the column vectors of $\boldsymbol{W}$ and λ is a regularization parameter. The regularization term is necessary to avoid the trivial solution $\boldsymbol{W} = \boldsymbol{0}$. It forces the squared norms of the column vectors of $\boldsymbol{W}$ to be close to 1 and hence avoids a trivial solution. We set λ to 100 in all our experiments, which is large enough such that all column-vectors have a squared norm close to $1 \pm 1e^{-3}$.

We optimize this problem with SGD and momentum. We initialize $\boldsymbol{W}$ uniformly random according to the method proposed by Glorot *et al.* [12] and normalize the column-vectors to have a norm of 1. For this particular optimization problem to initialize $\boldsymbol{W}$ we do not train our network end-to-end. Instead, we fix the feature representation and just optimize for $\boldsymbol{W}$. Thus, the optimization converges typically within seconds to a few minutes, depending on the embedding size d and the dataset size N. This is negligible compared to neural network training, which typically takes several hours to converge. We show the effectiveness of our initialization method in Section 4.5.

4. Evaluation

In our evaluation we first run several experiments on the CUB-200-2011 [49] dataset. We follow the evaluation protocol proposed in [35] 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 [35]. 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 Caffe [18] and Theano [47]. As network architecture, we follow previous works (*e.g.* [35, 48]) and use a GoogLeNet [45] which is pre-trained on the ImageNet dataset [38]. As optimization method we use ADAM [21] with a learning rate of $1e^{-6}$. We sample 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. We follow previous work [35] and use a batch size of 128.

In Section 4.1 we show the impact of an ensemble trained with BIER on the strength (*i.e.* accuracy) and correlation of an embedding. Next, we show that BIER works with several widely used loss functions (Section 4.2). Further, we analyse the impact of the number of groups in an embedding (Section 4.3) and the embedding size (Section 4.4). Then, we demonstrate the effectiveness of our initialization method (Section 4.5). Finally, we show that our method outperforms state-of-the-art methods on several datasets [23, 29, 31, 35, 49] (Section 4.6).

4.1. Strength and Correlation

Performance of an ensemble depends on two elements: the strength (*i.e*. accuracy) of individual learners and the correlation between the learners [7]. 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 different 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 initialization method and optimize a loss function on each of these groups separately (*Init-170-171-171*). Finally, we apply our boosting based reweighting scheme on the three groups (*Ours-170-171-171*).

As discussed in Section 3.1, we propose to use groups of different sizes proportional to the weighting of the online boosting algorithm. To that 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 initialization method (*Init-96-160-256*) and add our boosting method (*Ours-96-160-256*) on top of these learners.

As shown in Table 2, just 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 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 data set.

4.2. 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 3 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 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. Therefore, the binomial deviance loss conveys more information for successive learners compared to triplet or contrastive loss.

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
Ours-170-171-171	0.7882	0.0988	54.76
Learner-1-170			51.47
Learner-2-171			52.28
Learner-3-171			52.38
Ours-96-160-256	**0.7768**	**0.0934**	**55.33**
Learner-1-96			49.95
Learner-2-160			52.82
Learner-3-256			54.09

Table 2. Evaluation of classifier (Clf.) and feature correlation on CUB-200-2011 [49]. **Best** results are highlighted.

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**

Table 3. Evaluation of loss functions on CUB-200-2011 [49].

4.3. 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 Algorithm 1). In Table 4 we report the correlation of the feature embedding, the R@1 score 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 larger group sizes the strength of individual learners declines and hence performance decreases. For smaller group sizes the individual embeddings are larger. They achieve higher individual accuracy, but are highly correlated with each other, since they benefit less from the gradient boosting algorithm.

4.4. Embedding Sizes

Next, we show the effect of different embedding sizes. We train a CNN with embedding sizes 384, 512, 1024 with our method and with a regular CNN. For our method, we

Group Sizes	Clf. Corr.	Avg R@1	R@1
Baseline	-	-	51.75
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

Table 4. Evaluation of group sizes on CUB-200-2011 [49].

split the embeddings into several groups. 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 (see Section 3.1).

We use the binomial deviance loss function, as it consistently achieves best results compared to triplet loss or contrastive loss (recall Table 3). In Table 5 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 more likely to over-fit. Hence, it is more beneficial to train several smaller embeddings which complement each other better.

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

Table 5. Evaluation of embedding size on CUB-200-2011 [49].

4.5. Impact of Initialization

To show the effectiveness of our initialization method, we compare it with random initialization, as proposed by Glorot *et al*. [12] and an orthogonal initialization method [39]. All networks are trained with binomial deviance as loss function with our proposed boosting based reweighting scheme. We report mean R@1 and standard deviation of the three methods.

In Table 6 we see that BIER with our initialization method achieves better accuracy compared to orthogonal or random initialization. Note that our method significantly reduces the standard deviation and hence achieves more consistent results. We hypothesize that this is due to the fact that with our initialization method learners are already low correlated at the beginning of the training.

4.6. 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 [49], Cars-196 [23], Stanford Online Product [35], In-Shop Clothes Retrieval [31] and VehicleID [29] datasets.

Method	R@1
Glorot	54.41 ± 0.43
Orthogonal	54.58 ± 0.12
Ours	$\mathbf{55.33 \pm 0.05}$

Table 6. Evaluation of Glorot, orthogonal and our initialization method on the CUB-200-2011 [49] dataset.

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, ...). 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 [35]. 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 [35], *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 [29].

We fix all our parameters and train our method 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.* [35] 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 [35] 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.

Results and baselines are shown in Tables 7, 8, 9, 10 and 11. Our method in combination with a simple loss function operating on pairs is able to outperform state-of-the-art methods relying on higher order tuples [44, 35] or

	CUB-200-2011						Cars-196					
R@	1	2	4	8	16	32	1	2	4	8	16	32
Contrastive [35]	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 [35]	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 [35]	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 [48]	52.8	64.4	74.7	83.9	90.4	94.3	-	-	-	-	-	-
Histogram Loss [48]	50.3	61.9	72.6	82.4	88.8	93.7	-	-	-	-	-	-
N-Pair-Loss [44]	51.0	63.3	74.3	83.2	-	-	71.1	79.7	86.5	91.6	-	-
HDC [55]	53.6	65.7	**77.0**	**85.6**	91.5	**95.5**	73.7	83.2	89.5	93.8	96.7	98.4
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	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	**55.3**	**67.2**	76.9	85.1	**91.7**	**95.5**	**78.0**	**85.8**	**91.1**	**95.1**	**97.3**	**98.7**

Table 7. Comparison with the state-of-the-art on the CUB-200-2011 [49] and Cars-196 [23] dataset. **Best** results are highlighted.

	CUB-200-2011						Cars-196					
R@	1	2	4	8	16	32	1	2	4	8	16	32
PDDM + Triplet [17]	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 [17]	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 [55]	60.7	72.4	81.9	89.2	93.7	**96.8**	83.8	89.8	93.6	96.2	97.8	98.9
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	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	**63.7**	**74.0**	**82.5**	**89.3**	**93.8**	**96.8**	**87.2**	**92.2**	**95.3**	**97.4**	**98.5**	**99.3**

Table 8. Comparison with the state-of-the-art on the cropped versions of the CUB-200-2011 [49] and Cars-196 [23] dataset.

R@	1	10	100	1000
Contrastive [35]	42.0	58.2	73.8	89.1
Triplet [35]	42.1	63.5	82.5	94.8
LiftedStruct [35]	62.1	79.8	91.3	97.4
Binomial Deviance [48]	65.5	82.3	92.3	97.6
Histogram Loss [48]	63.9	81.7	92.2	97.7
N-Pair-Loss [44]	67.7	83.8	93.0	97.8
HDC [55]	69.5	84.4	92.8	97.7
Ours Baseline	66.2	82.3	91.9	97.4
BIER Learner-3	72.5	86.3	93.9	97.9
BIER	**72.7**	**86.5**	**94.0**	**98.0**

Table 9. Comparison with the state-of-the-art on the Stanford Online Products [35] dataset.

R@	1	10	20	30	40	50
FasionNet + Joints [31]	41.0	64.0	68.0	71.0	73.0	73.5
FasionNet + Poselets [31]	42.0	65.0	70.0	72.0	72.0	75.0
FasionNet [31]	53.0	73.0	76.0	77.0	79.0	80.0
HDC [55]	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	76.4	92.7	95.0	96.1	96.6	97.0
BIER	**76.9**	**92.8**	**95.2**	**96.2**	**96.7**	**97.1**

Table 10. Comparison with the state-of-the-art on the In-Shop Clothes Retrieval [31] dataset.

	Small		Medium		Large	
R@	1	5	1	5	1	5
Mixed Diff+CCL [29]	49.0	73.5	42.8	66.8	38.2	61.6
GS-TRS loss [1]	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	**82.6**	90.5	**79.3**	88.0	75.5	86.0
BIER	**82.6**	**90.6**	**79.3**	**88.3**	**76.0**	**86.4**

Table 11. Comparison with the state-of-the-art on VehicleID [29].

histograms [48]. We consistently improve our strong baseline method by a large margin at R@1 on all datasets, which demonstrates the robustness of our approach. On CUB-200-2011 and Cars-196 we can improve 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 (≈ 5) images per class. On these datasets we achieve a notable improvement of 3%, 14.8% and 3-8%, respectively. Notably, the last learner in our ensemble (*BIER Learner-3*) already outperforms the state-of-the-art on most of the datasets.

5. Conclusion

In this work we cast training an ensemble of metric CNNs with a shared feature representation as online gradient boosting problem. Our method does not introduce any additional parameters and has only negligible additional cost during training and test time. Our extensive experiments show that BIER significantly reduces correlation on the last hidden layer of a CNN and works with several different loss functions. Finally, BIER outperforms state-of-the-art methods on the Stanford Online Products [35], CUB-200-2011 [49], Cars-196 [23], In-Shop Clothes Retrieval [31] and VehicleID [29] datasets.

Acknowledgements. This project was supported by the Austrian Research Promotion Agency (FFG) projects MANGO (836488) and Darknet (858591). We also thank NVIDIA for their GPU donation.

References

[1] Y. Bai, F. Gao, Y. Lou, S. Wang, T. Huang, and L.-Y. Duan. Incorporating Intra-Class Variance to Fine-Grained Visual Recognition. *arXiv*, abs/1703.00196, 2017. 8

[2] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *arXiv*, abs/1306.6709, 2013. 2

[3] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al. Greedy Layer-Wise Training of Deep Networks. In *Proc. NIPS*, 2006. 3

[4] A. Beygelzimer, E. Hazan, S. Kale, and H. Luo. Online Gradient Boosting. In *Proc. NIPS*, 2015. 3, 4

[5] A. Beygelzimer, S. Kale, and H. Luo. Optimal and Adaptive Algorithms for Online Boosting. In *Proc. ICML*, 2015. 3, 4

[6] 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*, pages 2617–2624, 2011. 2

[7] L. Breiman. Random Forests. *ML*, 45(1):5–32, 2001. 1, 6

[8] S.-T. Chen, H.-T. Lin, and C.-J. Lu. An Online Boosting Algorithm with Theoretical Justifications. *Proc. ICML*, 2012. 3, 4

[9] S. Chopra, R. Hadsell, and Y. LeCun. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *Proc. CVPR*, 2005. 1, 2, 4

[10] Y. Freund and R. E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *JCSS*, 55(1):119 – 139, 1997. 2, 3

[11] J. H. Friedman. Greedy Function Approximation: a Gradient Boosting Machine. *AoS*, 29(5):1189–1232, 2001. 2

[12] X. Glorot and Y. Bengio. Understanding the Difficulty of Training Deep FeedForward Neural Networks. In *Proc. AISTATS*, 2010. 3, 5, 7

[13] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *Proc. CVPR*, 2006. 1, 2, 4

[14] S. Han, Z. Meng, A.-S. KHAN, and Y. Tong. Incremental Boosting Convolutional Neural Network for Facial Action Unit Recognition. In *Proc. NIPS*, 2016. 3

[15] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proc. ICCV*, 2015. 3

[16] G. E. Hinton, S. Osindero, and Y.-W. Teh. A Fast Learning Algorithm for Deep Belief Nets. *NECO*, 18(7):1527–1554, 2006. 3

[17] C. Huang, C. C. Loy, and X. Tang. Local Similarity-Aware Deep Feature Embedding. In *Proc. NIPS*, 2016. 2, 3, 8

[18] 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*, abs/1408.5093, 2014. 5

[19] N. Karianakis, T. J. Fuchs, and S. Soatto. Boosting Convolutional Features for Robust Object Proposals. *arXiv*, abs/1503.06350, 2015. 3

[20] D. Kedem, S. Tyree, F. Sha, G. R. Lanckriet, and K. Q. Weinberger. Non-linear Metric Learning. In *Proc. NIPS*, pages 2573–2581, 2012. 2

[21] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv*, abs/1412.6980, 2014. 5

[22] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell. Data-dependent Initializations of Convolutional Neural Networks. In *Proc. ICLR*, 2016. 3

[23] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *Proc. ICCV Workshops*, 2013. 2, 5, 7, 8

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proc. NIPS*, 2012. 3

[25] B. Kumar, G. Carneiro, I. Reid, et al. Learning Local Image Descriptors with Deep Siamese and Triplet Convolutional Networks by Minimising Global Loss Functions. In *Proc. CVPR*, 2016. 1, 4

[26] M. T. Law, N. Thome, and M. Cord. Quadruplet-wise Image Similarity Learning. In *Proc. ICCV*, 2013. 1

[27] C.-Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu. Deeply-Supervised Nets. In *Proc. AISTATS*, 2015. 2

[28] C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. On Robustness of On-line Boosting - a Competitive Study. In *Proc. ICCV Workshops*, 2009. 3, 4

[29] 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. 2, 5, 7, 8

[30] M. Liu and B. C. Vemuri. A Robust and Efficient Doubly Regularized Metric Learning Approach. In *Proc. ECCV*, pages 646–659. Springer, 2012. 2

[31] 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. 2, 5, 7, 8

[32] D. Mishkin and J. Matas. All you need is a good init. In *Proc. ICLR*, 2016. 3

[33] M. Muja and D. G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *PAMI*, 36(11):2227–2240, 2014. 1, 5

[34] R. Negrel, A. Lechervy, and F. Jurie. MLBoost Revisited: A Faster Metric Learning Algorithm for Identity-Based Face Retrieval. In *Proc. BMVC*, 2016. 2

[35] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep Metric Learning via Lifted Structured Feature Embedding. In *Proc. CVPR*, 2016. 1, 2, 3, 4, 5, 7, 8

[36] O. M. Parkhi, A. Vedaldi, and A. Zisserman. Deep Face Recognition. In *Proc. BMVC*, 2015. 1, 2, 4

[37] O. Rippel, M. Paluri, P. Dollar, and L. Bourdev. Metric Learning with Adaptive Density Discrimination. In *Proc. ICLR*, 2016. 2

[38] 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*, 115(3):1–42, 2015. 5

[39] 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. 3, 7

[40] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. In *Proc. CVPR*, 2015. 1, 2, 4

[41] C. Shen, J. Kim, L. Wang, and A. v. d. Hengel. Positive Semidefinite Metric Learning using Boosting-Like Algorithms. *JMLR*, 13:1007–1036, 2012. 2

[42] 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. 1, 4

[43] 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. 1

[44] K. Sohn. Improved Deep Metric Learning with Multi-Class n-Pair Loss Objective. In *Proc. NIPS*, 2016. 2, 3, 7, 8

[45] 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. 2, 5

[46] R. Tao, E. Gavves, and A. W. Smeulders. Siamese Instance Search for Tracking. In *Proc. CVPR*, 2016. 1

[47] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv*, abs/1605.02688, 2016. 5

[48] E. Ustinova and V. Lempitsky. Learning Deep Embeddings with Histogram Loss. In *Proc. NIPS*, 2016. 1, 2, 3, 4, 5, 8

[49] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 2, 5, 6, 7, 8

[50] E. Walach and L. Wolf. Learning to Count with CNN Boosting. In *Proc. ECCV*, 2016. 3

[51] G. Waltner, M. Opitz, and H. Bischof. BaCoN: Building a Classifier from only N Samples. In *Proc. CVWW*, 2016. 1

[52] K. Q. Weinberger and L. K. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *JMLR*, 10(2):207–244, 2009. 1, 2

[53] P. Wohlhart and V. Lepetit. Learning Descriptors for Object Recognition and 3D Pose Estimation. In *Proc. CVPR*, 2015. 1, 2

[54] B. Yang, J. Yan, Z. Lei, and S. Z. Li. Convolutional Channel Features. In *Proc. CVPR*, 2015. 3

[55] Y. Yuan, K. Yang, and C. Zhang. Hard-Aware Deeply Cascaded Embedding. In *Proc. ICCV*, 2017. 2, 8

[56] W. S. Zheng, S. Gong, and T. Xiang. Reidentification by Relative Distance Comparison. *PAMI*, 35(3):653–668, 2013. 1