A Metric Learning Reality Check

Kevin Musgrave[1], Serge Belongie[1], Ser-Nam Lim[2]

[1]Cornell Tech [2]Facebook AI

Abstract. Deep metric learning papers from the past four years have consistently claimed great advances in accuracy, often more than doubling the performance of decade-old methods. In this paper, we take a closer look at the field to see if this is actually true. We find flaws in the experimental setup of these papers, and propose a new way to evaluate metric learning algorithms. Finally, we present experimental results that show that the improvements over time have been marginal at best.

Keywords: Deep metric learning

1 Metric Learning Overview

1.1 Why metric learning is important

Metric learning attempts to map data to an embedding space, where similar data are close together and dissimilar data are far apart. In general, this can be achieved by means of embedding and classification losses. Embedding losses operate on the relationships between samples in a batch, while classification losses include a weight matrix that transforms the embedding space into a vector of class logits.

In cases where a classification loss is applicable, why are embeddings used during test time, instead of the logits or the subsequent softmax values? Typically, embeddings are preferred when the task is some variant of information retrieval, where the goal is to return data that is most similar to a query. An example of this is image search, where the input is a query image, and the output is the most visually similar images in a database. Open-set classification is a variant of this, where test set and training set classes are disjoint. In this situation, query data can be classified based on a nearest neighbors vote, or verified based on distance thresholding in the embedding space. Some notable applications of this are face verification [32], and person re-identification [14]. Both have seen improvements in accuracy, largely due to the use of convnets, but also due to loss functions that encourage well-clustered embedding spaces.

Then there are cases where using a classification loss is not possible. For example, when constructing a dataset, it might be difficult or costly to assign class labels to each sample, and it might be easier to specify the relative similarities between samples in the form of pair or triplet relationships [42]. Pairs and triplets can also provide additional training signals for existing datasets [4]. In both cases, there are no explicit labels, so embedding losses become a suitable choice.

Recently, there has been significant interest in self-supervised learning. This is a form of unsupervised learning where pseudo-labels are applied to the data during training, often via clever use of data augmentations or signals from multiple modalities [25,12,2]. In this case, the pseudo-labels exist to indicate the similarities between data in a particular batch, and as such, they do not have any meaning across training iterations. Thus, embedding losses are favored over classification losses.

In the computer vision domain, deep convnets have resulted in dramatic improvements in nearly every subfield, including classification [18,13], segmentation [21], object detection [28], and generative models [9]. It is no surprise, then, that deep networks have had a similar effect on metric learning. The combination of the two is often called deep metric learning, and this will be the focus of the remainder of the paper. The rest of this section will briefly review the recent advances in deep metric learning, as well as related work, and the contributions of this paper.

1.2 Embedding losses

Pair and triplet losses provide the foundation for two fundamental approaches to metric learning. A classic pair based method is the contrastive loss [10], which attempts to make the distance between positive pairs (d_p) smaller than some threshold (m_{pos}), and the distance between negative pairs (d_n) larger than some threshold (m_{neg}):

$$L_{contrastive} = [d_p - m_{pos}]_+ + [m_{neg} - d_n]_+ \tag{1}$$

(Note that in many implementations, m_{pos} is set to 0.) The theoretical downside of this method is that the same distance threshold is applied to all pairs, even though there may be a large variance in their similarities and dissimilarities.

The triplet margin loss [41] theoretically addresses this issue. A triplet consists of an anchor, positive, and negative sample, where the anchor is more similar to the positive than the negative. The triplet margin loss attempts to make the anchor-positive distances (d_{ap}) smaller than the anchor-negative distances (d_{an}), by a predefined margin (m):

$$L_{triplet} = [d_{ap} - d_{an} + m]_+ \tag{2}$$

This theoretically places fewer restrictions on the embedding space, and allows the model to account for variance in interclass dissimilarities.

A wide variety of losses has since been built on these fundamental concepts. For example, the angular loss [39] is a triplet loss where the margin is based on the angles formed by the triplet vectors. The margin loss [43] modifies the contrastive loss by setting $m_{pos} = \beta - \alpha$ and $m_{neg} = \beta + \alpha$, where α is fixed, and β is learnable via gradient descent. More recently, Yuan *et al.* [45] proposed a variation of the contrastive loss based on signal to noise ratios, where each embedding vector is considered signal, and the difference between it and other vectors is considered noise. Other pair losses are based on the softmax function and LogSumExp, which is a smooth approximation of the maximum function. Specifically, the lifted structure loss [24] is the contrastive loss but with LogSumExp applied to all negative pairs. The N-Pairs loss [34] applies the softmax function to each positive pair relative to all other pairs. The recent multi similarity loss [40] applies LogSumExp to all pairs, but is specially formulated to give weight to different relative similarities among each embedding and its neighbors. The tuplet margin loss [44] also uses LogSumExp, but in combination with an implicit pair weighting method. In contrast with these pair and triplet losses, FastAP [1] attempts to optimize for average precision within each batch, using a soft histogram binning technique.

1.3 Classification losses

Classification losses are based on the inclusion of a weight matrix, where each column corresponds to a particular class. In most cases, training consists of matrix multiplying the weights with embedding vectors to obtain logits, and then applying a loss function to the logits. The most straightforward case is the normalized softmax loss [20,47], which is identical to cross entropy, but with the columns of the weight matrix L2 normalized. ProxyNCA [23] is a variation of this, where cross entropy is applied to the Euclidean distances, rather than the cosine similarities, between embeddings and the weight matrix. A number of face verification losses have modified the cross entropy loss with angular margins in the softmax expression. Specifically, SphereFace [20], CosFace [38], and ArcFace [5] apply multiplicative-angular, additive-cosine, and additive-angular margins, respectively. (It is interesting to note that metric learning papers have consistently left out face verification losses from their experiments, even though there is nothing face-specific about them.) The SoftTriple loss [27] takes a different approach, by expanding the weight matrix to have multiple columns per class, theoretically providing more flexibility for modeling class variances.

1.4 Pair and triplet mining

Mining is the process of finding the best pairs or triplets to train on. There are two broad approaches to mining: offline and online. Offline mining is performed before batch construction, so that each batch is made to contain the most informative samples. This might be accomplished by storing lists of hard negatives [33], doing a nearest neighbors search before each epoch [11], or before each iteration [35]. In contrast, online mining finds hard pairs or triplets within each randomly sampled batch. Using all possible pairs or triplets is an alternative, but this has two weaknesses: practically, it can consume a lot of memory, and theoretically, it has the tendency to include a large number of easy negatives and positives, causing performance to plateau quickly. Thus, one intuitive strategy is to select only the most difficult positive and negative samples [14], but this has been found to produce noisy gradients and convergence to bad local optima [43]. A possible remedy is semihard negative mining, which finds the negative samples in a batch that are close to the anchor, but still further away than the corresponding positive samples [32]. On the other hand, Wu *et al.* [43] found that semihard mining makes little progress as the number of semihard negatives drops. They claim that distance-weighted sampling results in a variety of negatives (easy, semihard, and hard), and improved performance. Online mining can also be integrated into the structure of models. Specifically, the hard-aware deeply cascaded method [46] uses models of varying complexity, in which the loss for the complex models only considers the pairs that the simpler models find difficult. Recently, Wang *et al.* [40] proposed a simple pair mining strategy, where negatives are chosen if they are closer to an anchor than its hardest positive, and positives are chosen if they are further from an anchor than its hardest negative.

1.5 Advanced training methods

To obtain higher accuracy, many recent papers have gone beyond loss functions or mining techniques. For example, several recent methods incorporate generator networks in their training procedure. Lin *et al.* [19] use a generator as part of their framework for modeling class centers and intraclass variance. Duan *et al.* [6] use a hard-negative generator to expose the model to difficult negatives that might be absent from the training set. Zheng *et al.* [48] follow up on this work by using an adaptive interpolation method that creates negatives of varying difficulty, based on the strength of the model. Other sophisticated training methods include HTL [8], ABE [16], and MIC [29]. HTL construct a hierarchical class tree at regular intervals during training, to estimate the optimal per-class margin in the triplet margin loss. ABE is an attention based ensemble, where each model learns a different set of attention masks. MIC uses a combination of clustering and encoder networks to disentangle class specific properties from shared characteristics like color and pose.

1.6 Related work

Recently, Fehervari *et al.* [7] addressed the problem of unfair comparisons in metric learning papers, by evaluating loss functions on a more level playing field. However, they focused mainly on methods from 2017 or earlier, and did not address the issue of hyperparameter tuning on the test set. Concurrent with our work is Roth *et al.* [30], which addresses many of the same flaws that we find, and does an extensive analysis of various loss functions. But again, they do not address the problem of training with test set feedback, and their hyperparameters are tuned using a small grid search around values proposed in the original papers. This is important, as we find that effective hyperparameter tuning significantly minimizes the performance differences between loss functions.

1.7 Contributions of this paper

In the following sections, we examine flaws in the current literature, including the problem of unfair comparisons, the bad practice of training with test set feedback, and the weaknesses of commonly used accuracy metrics. We propose a training and evaluation protocol that addresses these flaws, and then run experiments on a variety of loss functions. Our results show that when hyperparameters are properly tuned via cross-validation, most methods perform similarly to one another. This opens up research questions regarding the relationship between hyperparameters and datasets, and the factors limiting open-set accuracy that may be inherent to particular dataset/architecture combinations. As well, by comparing algorithms using proper machine learning practices and a level playing field, the performance gains in future research will better reflect reality, and will be more likely to generalize to other high-impact fields like self-supervised learning.

2 Flaws in the existing literature

2.1 Unfair comparisons

In order to claim that a new algorithm outperforms existing methods, its important to keep as many parameters constant as possible. That way, we can be certain that it was the new algorithm that boosted performance, and not one of the extraneous parameters. This has not been the case with metric learning papers.

One of the easiest ways to improve accuracy is to upgrade the network architecture, yet this fundamental parameter has not been kept constant across papers. Some use GoogleNet, while others use BN-Inception, sometimes referred to as "Inception with Batch Normalization. Choice of architecture is important in metric learning, because the networks are typically pretrained on ImageNet, and then finetuned on smaller datasets. Thus, the initial accuracy on the smaller datasets varies depending on the chosen network. One widely-cited paper from 2017 used ResNet50, and then claimed huge performance gains. This is questionable, because the competing methods used GoogleNet, which has significantly lower initial accuracies (see Table 1). Therefore, much of the performance gain likely came from the choice of network architecture, and not their proposed method. In addition, papers have changed the dimensionality of the embedding space, and increasing dimensionality leads to increased accuracy. Therefore, varying this parameter further complicates the task of comparing algorithms.

Table 1. Recall@1 of models pretrained on ImageNet. Output embedding sizes were reduced to 512 using PCA and L2 normalized. For each image, the smaller side was scaled to 256, followed by a center-crop to 227x227.

	CUB200	Cars196	SOP
GoogleNet	41.1	33.9	45.2
BN-Inception	51.1	46.9	50.7
ResNet50	48.7	43.5	52.9

Another easy way to improve accuracy is to use more sophisticated image augmentations. In fact, image augmentation strategies have been central to several recent advances in supervised and self-supervised learning [3,36,12,2]. In the metric learning field, most papers claim to apply the following transformations: resize the image to 256 x 256, randomly crop to 227 x 227, and do a horizontal flip with 50% chance. But the official open-source implementations of some recent papers show that they are actually using the more sophisticated cropping method described in the original GoogleNet paper. This method randomly changes the location, size, and aspect ratio of each crop, which provides more variability in the training data, and helps combat overfitting.

Papers have also been inconsistent in their choice of optimizer (SGD, Adam, RMSprop etc) and learning rate. The effect on test set accuracy is less clear in this case, as adaptive optimizers like Adam and RMSprop will converge faster, while SGD may lead to better generalization [22]. Regardless, varying the optimizer and learning rate makes it difficult to do apples-to-apples comparisons.

It is also possible for papers to omit small details that have a big effect on accuracy. For example, in the official open-source code for a 2019 paper, the pretrained ImageNet model has its BatchNorm parameters frozen during training. This can help reduce overfitting, and the authors explain in the code that it results in a 2 point performance boost on the CUB200 dataset. Yet this is not mentioned in their paper.

Finally, most papers do not present confidence intervals for their results, and improvements in accuracy over previous methods often range in the low single digits. Those small improvements would be more meaningful if the results were averaged over multiple runs, and confidence intervals were included.

2.2 Training with test set feedback

The majority of papers split each dataset so that the first 50% of classes are used for the training set, and the remainder are used for the test set. Then during training, the test set accuracy of the model is checked at regular intervals, and the best test set accuracy is reported. In other words, there is no validation set, and model selection and hyperparameter tuning are done with direct feedback from the test set. Some papers do not check performance at regular intervals, and instead report accuracy after training for a predetermined number of iterations. In this case, it is unclear how the number of iterations is chosen, and hyperparameters are still tuned based on test set performance. This breaks one of the most basic commandments of machine learning. Training with test set feedback leads to overfitting on the test set, and therefore brings into question the steady rise in accuracy over time, as presented in metric learning papers.

2.3 Weakness of commonly used accuracy metrics

To report accuracy, most metric learning papers report Recall@K, Normalized Mutual Information (NMI), and the F1 score. But are these necessarily the best metrics to use? Figure 1 shows three embedding spaces, and each one scores nearly 100% Recall@1, even though they have different characteristics. (Note that 100% Recall@1 means that Recall@K for any K>1 is also 100%.) More importantly, Figure 1(c) shows a better separation of the classes than Figure 1(a), yet they receive approximately the same score. F1 and NMI also return roughly equal scores for all three embedding spaces. Moreover, they require the embeddings to be clustered, which introduces two factors of variability: the choice of clustering algorithm, and the sensitivity of clustering results to seed initialization. Since we know the ground-truth number of clusters, k-means clustering is the obvious choice and is what is typically used. However, as Figure 1 shows, this results in uninformative NMI and F1 scores. Other clustering algorithms could be considered, but each one has its own drawbacks and subtleties. Introducing a clustering algorithm into the evaluation process is simply adding a layer of complexity between the researcher and the embedding space. Instead, we would like an accuracy metric that operates directly on the embedding space, like Recall@K, but that provides more nuanced information.

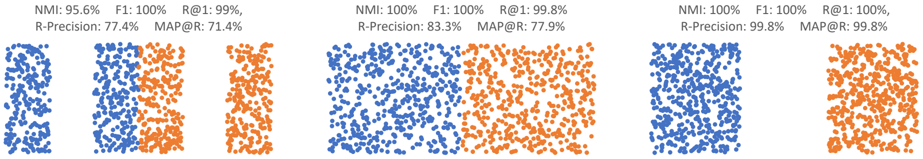


Fig. 1. How different accuracy metrics score on three toy examples.

Finally, here is an interesting scenario to further illustrate why we should specifically not use NMI. Let N be the size of our dataset, C be the number of classes, and $S = N/C$ be the number of samples per class. Now assume we have a random embedding space, such that when we run k-means clustering, each cluster consists of S samples, each of a different class. This is a bad embedding space, because we would like each cluster to contain S samples, all of the *same* class. But the NMI paints a different picture. Let X be the set of classes and Y be the set of clusters. If $S \leq C$,

$$H(X) = -C\left(\frac{1}{C}\log\frac{1}{C}\right) = \log C \tag{3}$$

$$H(Y) = H(X) \tag{4}$$

$$H(X|Y) = C\left(-\frac{1}{C}\left(S\left(\frac{1}{S}\log\frac{1}{S}\right)\right)\right) = \log S \tag{5}$$

$$I(X;Y) = H(X) - H(X|Y) = \log C - \log S \tag{6}$$

$$NMI = 2\frac{I(X;Y)}{H(X)+H(Y)} = 1 - \frac{\log S}{\log C} \tag{7}$$

For datasets where there are many classes and few samples per class (an important scenario for metric learning), the NMI will be high even for completely random embeddings. For example, with 10,000 classes and 4 samples per class, the NMI is 84.9%. This is supported by experiments on random embeddings (Figure 2), and results for standard datasets with randomly initialized convnets (Table 2). Thus, it seems that NMI becomes less informative in few-shot datasets.

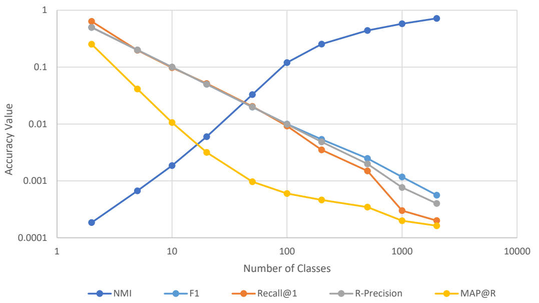


Fig. 2. We generated 10,000 random embeddings of dimensionality 64, and randomly assigned class labels. This plot shows the value of each accuracy metric (vertical axis) on those random embeddings, as the number of classes varies (horizontal axis).

Table 2. NMI of embeddings from randomly initialized convnets. CUB200 and Cars196 have 50-100 samples per class, while SOP has on average less than 10 samples per class.

	CUB200	Cars196	SOP
GoogleNet	23.6	19.1	81.2
BN-Inception	18.5	13.7	73.1
ResNet50	21.3	16.7	80.8

3 Proposed evaluation method

To do a proper evaluation of loss functions, we created the following settings, which address the problems described in the previous section.

3.1 Fair comparisons and reproducibility

- All experiments are run using PyTorch [26], using an ImageNet [31] pretrained BN-Inception network [15], with output embedding size of 128. BatchNorm parameters are frozen during training, to reduce overfitting.
- The batch size is set to 32. Batches are constructed by first randomly sampling C classes, and then randomly sampling M images for each of the C

classes. We set $C = 8$ and $M = 4$ for embedding losses, and $C = 32$ and $M = 1$ for classification losses.

- During training, images are augmented using the random resized cropping strategy. Specifically, we first resize each image so that its shorter side has length 256, then make a random crop that has a size between 40 and 256, and aspect ratio between 3/4 and 4/3. This crop is then resized to 227x227, and flipped horizontally with 50% probability. During evaluation, images are resized to 256 and then center cropped to 227.
- All network parameters are optimized using RMSprop with learning rate 1e-6. We chose RMSprop because it converges faster than SGD, and seems to generalize better than Adam, based on a small set of experiments. For loss functions that include their own learnable weights (e.g. ArcFace), we use RMSprop but leave the learning rate as a hyperparameter to be optimized.
- Embeddings are L2 normalized before computing the loss, and during evaluation.
- Configuration files will be publicly available for easy reproducibility with our benchmarking tool.

3.2 Hyperparameter search via cross validation

- To find the best loss function hyperparameters, we run 50 iterations of bayesian optimization. Each iteration consists of 4-fold cross validation:

    - The first half of classes are used for cross validation, and the 4 partitions are created deterministically: the first 0-12.5% of classes make up the first partition, the next 12.5-25% of classes make up the second partition, and so on. The training set comprises 3 of the 4 partitions, and cycles through all leave-one-out possibilities. As a result, the training and validation sets are always class-disjoint, so optimizing for validation set performance should be a good proxy for accuracy on open-set tasks. Training stops when validation accuracy plateaus.
    - The second half of classes are used as the test set. This is the same setting that metric learning papers have used for years, and we use it so that results can be compared more easily to past papers.

- Hyperparameters are optimized to maximize the average validation accuracy. For the best hyperparameters, the highest-accuracy checkpoint for each training set partition is loaded, and its embeddings for the test set are computed and L2 normalized. Then we compute accuracy using two methods:

    1. **Concatenated (512-dim)**: For each sample in the test set, we concatenate the 128-dim embeddings of the 4 models to get 512-dim embeddings, and then L2 normalize. We then report the accuracy of these embeddings.
    2. **Separated (128-dim)**: For each sample in the test set, we compute the accuracy of the 128-dim embeddings separately, and therefore obtain 4

different accuracies, one for each model's embeddings. We then report the average of these accuracies.

- We do multiple training reruns using the best hyperparameters, and report the average across these runs, as well as confidence intervals. This way our results are less subject to random seed noise.

3.3 Informative accuracy metrics

- We measure accuracy using Mean Average Precision at R (MAP@R), which combines the ideas of Mean Average Precision and R-precision.
- R-Precision is defined as follows: for each sample, determine the number of other samples, R, that are of the same class. Find the R nearest neighbors, and count the number, r, that are the same class as the sample. The score for the sample is $\frac{r}{R}$.
- One weakness of R-precision is that it does not account for the ranking of the correct retrievals. So we instead use MAP@R, which is simply Mean Average Precision, but with the number of nearest neighbors for each sample set to R. The benefits of MAP@R are that it is more informative than Recall@1 (see Figure 1), it can be computed directly from the embedding space (no clustering step required), it is easy to understand, and it rewards well clustered embedding spaces.
- In the results tables, we present R-precision and MAP@R. For the sake of comparisons to previous papers, we also show Precision@1 (also known as "Recall@1" in metric learning papers).

4 Experiments

4.1 Losses and datasets

We ran experiments on 11 losses, and 1 loss+miner combination, and prioritized methods from recent conferences (see Table 6). Face verification losses have been consistently left out of metric learning papers, so we included two losses from that domain. For every loss, we used the settings described in section 3, and we ran experiments on three widely used metric learning datasets: CUB200 [37], Cars196 [17], and Stanford Online Products (SOP) [24]. We chose these datasets because they have been the standard for several years, and we want our results to be easily comparable to prior papers. Tables 3-5 show the mean accuracy across training runs, as well as the 95% confidence intervals where applicable. Bold represents the best mean accuracy. We also include the accuracy of the pretrained model, the embeddings of which are reduced to 512 or 128, using PCA.

Table 3. Accuracy on CUB200

	Concatenated (512-dim)			Separated (128-dim)		
	P@1	**RP**	**MAP@R**	**P@1**	**RP**	**MAP@R**
Pretrained	51.05	24.85	14.21	50.54	25.12	14.53
Contrastive	67.21 ± 0.49	36.92 ± 0.28	26.19 ± 0.28	58.63 ± 0.46	31.48 ± 0.19	20.73 ± 0.19
Triplet	64.40 ± 0.38	34.63 ± 0.36	23.79 ± 0.36	55.97 ± 0.32	29.60 ± 0.28	18.80 ± 0.27
ProxyNCA	66.14 ± 0.32	35.48 ± 0.18	24.56 ± 0.18	58.31 ± 0.28	30.60 ± 0.13	19.72 ± 0.13
Margin	65.48 ± 0.50	35.04 ± 0.24	24.10 ± 0.26	56.17 ± 0.37	29.49 ± 0.20	18.62 ± 0.20
N. Softmax	65.43 ± 0.23	35.98 ± 0.22	25.20 ± 0.21	58.54 ± 0.23	31.74 ± 0.20	20.94 ± 0.18
CosFace	67.19 ± 0.37	**37.36 ± 0.23**	**26.53 ± 0.23**	59.83 ± 0.30	32.07 ± 0.19	21.25 ± 0.18
ArcFace	67.06 ± 0.31	37.23 ± 0.17	26.35 ± 0.17	**60.10 ± 0.19**	**32.31 ± 0.09**	**21.42 ± 0.09**
FastAP	63.64 ± 0.24	34.45 ± 0.21	23.71 ± 0.20	55.85 ± 0.33	29.82 ± 0.22	19.14 ± 0.20
SNR	**67.26 ± 0.46**	36.86 ± 0.20	26.10 ± 0.22	58.80 ± 0.25	31.56 ± 0.16	20.75 ± 0.17
MS	65.93 ± 0.16	35.91 ± 0.11	25.16 ± 0.10	58.51 ± 0.18	31.36 ± 0.10	20.58 ± 0.09
MS+Miner	65.75 ± 0.34	35.95 ± 0.21	25.21 ± 0.22	58.17 ± 0.22	31.27 ± 0.19	20.49 ± 0.20
SoftTriple	66.20 ± 0.37	36.46 ± 0.20	25.64 ± 0.21	59.55 ± 0.35	32.10 ± 0.19	21.26 ± 0.18

Table 4. Accuracy on Cars196

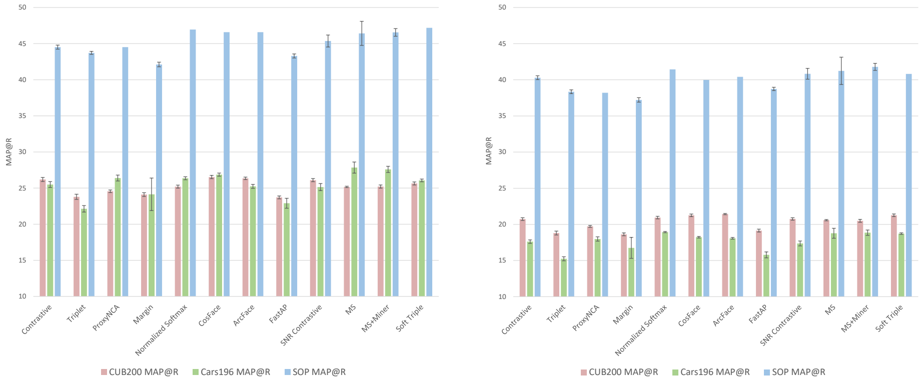
	Concatenated (512-dim)			Separated (128-dim)		
	P@1	**RP**	**MAP@R**	**P@1**	**RP**	**MAP@R**
Pretrained	46.89	13.77	5.91	43.27	13.37	5.64
Contrastive	81.57 ± 0.36	35.72 ± 0.35	25.49 ± 0.41	69.44 ± 0.24	28.15 ± 0.21	17.61 ± 0.24
Triplet	77.48 ± 0.57	32.85 ± 0.45	22.13 ± 0.45	63.87 ± 0.41	26.07 ± 0.32	15.24 ± 0.28
ProxyNCA	83.25 ± 0.37	36.63 ± 0.34	26.39 ± 0.41	70.85 ± 0.63	28.64 ± 0.25	17.98 ± 0.30
Margin	82.08 ± 2.41	34.71 ± 2.17	24.14 ± 2.25	70.95 ± 2.69	27.58 ± 1.50	16.75 ± 1.45
N. Softmax	83.58 ± 0.29	36.56 ± 0.19	26.36 ± 0.21	72.92 ± 0.19	**29.59 ± 0.09**	**18.93 ± 0.09**
CosFace	85.27 ± 0.23	36.72 ± 0.20	26.86 ± 0.22	**74.13 ± 0.21**	28.49 ± 0.14	18.22 ± 0.11
ArcFace	83.95 ± 0.23	35.44 ± 0.26	25.24 ± 0.27	73.67 ± 0.36	28.64 ± 0.13	18.07 ± 0.12
FastAP	78.20 ± 0.74	33.39 ± 0.67	22.90 ± 0.69	64.73 ± 0.57	26.42 ± 0.42	15.78 ± 0.41
SNR	81.87 ± 0.35	35.40 ± 0.44	25.14 ± 0.49	70.17 ± 0.44	27.90 ± 0.35	17.36 ± 0.34
MS	**85.29 ± 0.31**	**37.96 ± 0.63**	**27.84 ± 0.77**	73.73 ± 0.96	29.38 ± 0.60	18.77 ± 0.69
MS+Miner	84.59 ± 0.29	37.70 ± 0.37	27.59 ± 0.43	72.88 ± 0.30	29.46 ± 0.35	18.85 ± 0.37
SoftTriple	83.66 ± 0.22	36.31 ± 0.16	26.06 ± 0.19	72.98 ± 0.16	29.39 ± 0.10	18.72 ± 0.11

Table 5. Accuracy on SOP

	Concatenated (512-dim)			Separated (128-dim)		
	P@1	**RP**	**MAP@R**	**P@1**	**RP**	**MAP@R**
Pretrained	50.71	25.97	23.44	47.25	23.84	21.36
Contrastive	73.27 ± 0.23	47.45 ± 0.28	44.51 ± 0.28	69.28 ± 0.22	43.39 ± 0.28	40.29 ± 0.27
Triplet	72.94 ± 0.20	46.79 ± 0.21	43.72 ± 0.21	68.03 ± 0.26	41.57 ± 0.26	38.35 ± 0.26
ProxyNCA	73.89	47.34	44.52	68.20	41.13	38.21
Margin	71.53 ± 0.09	45.13 ± 0.34	42.11 ± 0.32	66.92 ± 0.34	40.36 ± 0.31	37.22 ± 0.31
N. Softmax	75.48	49.84	46.95	70.85	44.46	41.45
CosFace	75.59	49.44	46.59	70.13	42.86	39.98
ArcFace	75.76	49.39	46.59	70.55	43.33	40.41
FastAP	72.32 ± 0.26	46.32 ± 0.28	43.29 ± 0.28	67.89 ± 0.24	41.91 ± 0.23	38.74 ± 0.23
SNR	74.04 ± 0.65	48.30 ± 0.80	45.36 ± 0.83	69.95 ± 0.65	43.92 ± 0.72	40.83 ± 0.75
MS	75.01 ± 1.21	49.45 ± 1.67	46.42 ± 1.67	70.65 ± 1.70	44.40 ± 1.85	41.24 ± 1.89
MS+Miner	75.06 ± 0.45	49.56 ± 0.53	46.55 ± 0.53	**70.96 ± 0.40**	**44.95 ± 0.48**	**41.79 ± 0.49**
SoftTriple	**76.10**	**50.03**	**47.18**	70.82	43.71	40.82

Table 6. The losses covered in our experiments.

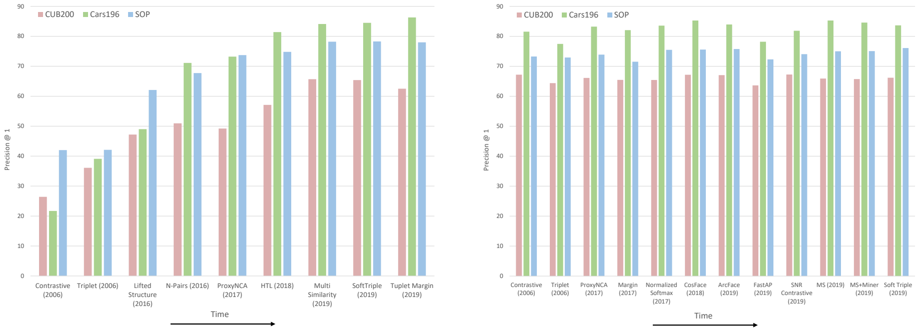
Method	Year	Loss type
Contrastive [10]	2006	Embedding
Triplet [41]	2006	Embedding
ProxyNCA [23]	2017	Classification
Margin [43]	2017	Embedding
Normalized Softmax (N. Softmax) [20,47]	2017	Classification
CosFace [38]	2018	Classification
ArcFace [5]	2019	Classification
FastAP [1]	2019	Embedding
Signal to Noise Ratio Contrastive (SNR) [45]	2019	Embedding
MultiSimilarity (MS) [40]	2019	Embedding
MS+Miner [40]	2019	Embedding
SoftTriple [27]	2019	Classification



(a) Concatenated (512-dim)

(b) Separated (128-dim)

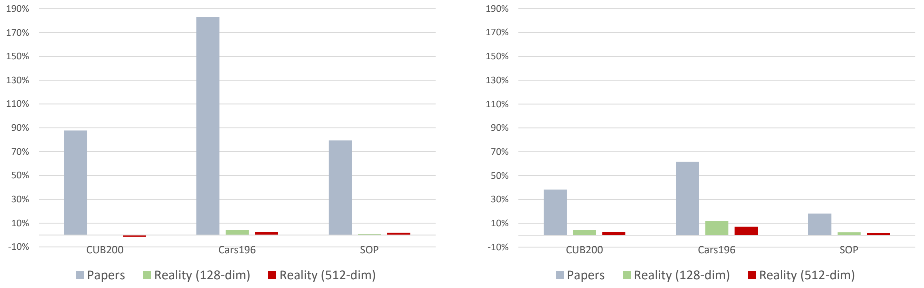
Fig. 3. MAP@R of Concatenated and Separated Embeddings



(a) The trend according to papers

(b) The trend according to reality

Fig. 4. Papers versus Reality: the trend of Precision@1 of various methods over the years.



(a) Relative improvement over the contrastive loss

(b) Relative improvement over the triplet loss

Fig. 5. Papers versus Reality: we look at the results tables of 20 metric learning papers from the years 2017-2019 (2 from 2017, 3 from 2018, 15 from 2019.) First, we note that 5 of the papers from 2019 did not include either the contrastive or triplet losses in their tables. Among the remaining 15 papers, 7 include the contrastive loss, and 12 include the triplet loss. For each paper, we compute the relative percentage improvement of their proposed method over the contrastive or triplet loss, and then take the average improvement across papers (grey bars in the above figures). The green and red bars are the average relative improvement that we obtain, in the separated 128-dim and concatenated 512-dim settings, respectively.

4.2 Papers versus reality

First, let's consider the general trend of paper results. Figure 4(a) shows the inexorable rise in accuracy we have all come to expect in this field, with modern methods completely obliterating old ones.

But how do the claims made in papers stack up against reality? We find that papers have drastically overstated improvements over the two classic methods, the contrastive and triplet loss (see Figure 5). For example, many papers show relative improvements exceeding 100% when compared with the contrastive loss, and exceeding 50% when compared with the triplet loss. This arises because of the extremely low accuracies that are attributed to these losses. Some of these numbers seem to originate from the 2016 paper on the lifted structure loss [24]. In their implementation of the contrastive and triplet loss, they sample $N/2$ pairs and $N/3$ triplets per batch, where N is the batch size. Thus, they utilize only a tiny fraction of the total information provided in each batch. Furthermore, they set the triplet margin to 1, whereas the optimal value tends to be around 0.1. Despite these implementation flaws, most papers simply keep citing the low numbers instead of trying to obtain a more reasonable baseline by implementing the losses themselves.

With good implementations of those baseline losses, a level playing field, and proper machine learning practices, we obtain the trend as shown in Figure 4(b). The trend appears to be a relatively flat line, indicating that the methods perform similarly to one another, whether they were introduced in 2006 or 2019. In other words, metric learning algorithms have not made the spectacular progress that they claim to have made. This brings into question the results of other cutting edge papers not covered in our experiments. It also raises doubts about the value of the hand-wavy theoretical explanations in metric learning papers. If a paper attempts to explain the performance gains of its proposed method, and it turns out that those performance gains are non-existent, then their explanation must be invalid as well.

5 Conclusion

In this paper, we uncovered several flaws in the current metric learning literature, namely:

- Unfair comparisons caused by changes in network architecture, embedding size, image augmentation method, and optimizers.
- Training without a validation set, i.e. with test set feedback.
- The use of accuracy metrics that are either misleading, or do not a provide a complete picture of the embedding space.

We then ran experiments with these issues fixed, and found that state of the art loss functions perform marginally better than, and sometimes on par with, classic methods. This is in stark contrast with the claims made in papers, in which accuracy has risen dramatically over time.

Future work could explore the relationship between optimal hyperparameters and dataset/architecture combinations, as well as the reasons for why different losses are performing similarly to one another. Of course, pushing the state-of-the-art in accuracy is another research direction. If proper machine learning practices are followed, and comparisons to prior work are done in a fair manner, the results of future metric learning papers will better reflect reality, and will be more likely to generalize to other high-impact areas like self-supervised learning.

References

1. Cakir, F., He, K., Xia, X., Kulis, B., Sclaroff, S.: Deep metric learning to rank. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1861–1870 (2019)
2. Chen, T., Kornblith, S., Norouzi, M., Hinton, G.: A simple framework for contrastive learning of visual representations. arXiv preprint arXiv:2002.05709 (2020)
3. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation strategies from data. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 113–123 (2019)
4. Cui, Y., Zhou, F., Lin, Y., Belongie, S.: Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop. In: Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV (2016), `http://vision.cornell.edu/se3/wp-content/uploads/2016/04/1950.pdf`
5. Deng, J., Guo, J., Xue, N., Zafeiriou, S.: Arcface: Additive angular margin loss for deep face recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4690–4699 (2019)
6. Duan, Y., Zheng, W., Lin, X., Lu, J., Zhou, J.: Deep adversarial metric learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2780–2789 (2018)
7. Fehervari, I., Ravichandran, A., Appalaraju, S.: Unbiased evaluation of deep metric learning algorithms. arXiv preprint arXiv:1911.12528 (2019)
8. Ge, W.: Deep metric learning with hierarchical triplet loss. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 269–285 (2018)
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
10. Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). vol. 2, pp. 1735–1742. IEEE (2006)
11. Harwood, B., Kumar, B., Carneiro, G., Reid, I., Drummond, T., et al.: Smart mining for deep metric learning. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2821–2829 (2017)
12. He, K., Fan, H., Wu, Y., Xie, S., Girshick, R.: Momentum contrast for unsupervised visual representation learning. arXiv preprint arXiv:1911.05722 (2019)
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
14. Hermans, A., Beyer, L., Leibe, B.: In defense of the triplet loss for person reidentification. arXiv preprint arXiv:1703.07737 (2017)
15. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning. pp. 448–456 (2015)
16. Kim, W., Goyal, B., Chawla, K., Lee, J., Kwon, K.: Attention-based ensemble for deep metric learning. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 736–751 (2018)
17. Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3d object representations for fine-grained categorization. In: 4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13). Sydney, Australia (2013)

18. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
19. Lin, X., Duan, Y., Dong, Q., Lu, J., Zhou, J.: Deep variational metric learning. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 689–704 (2018)
20. Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., Song, L.: Sphereface: Deep hypersphere embedding for face recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 212–220 (2017)
21. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)
22. Luo, L., Xiong, Y., Liu, Y.: Adaptive gradient methods with dynamic bound of learning rate. In: International Conference on Learning Representations (2019), `https://openreview.net/forum?id=Bkg3g2R9FX`
23. Movshovitz-Attias, Y., Toshev, A., Leung, T.K., Ioffe, S., Singh, S.: No fuss distance metric learning using proxies. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 360–368 (2017)
24. Oh Song, H., Xiang, Y., Jegelka, S., Savarese, S.: Deep metric learning via lifted structured feature embedding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4004–4012 (2016)
25. Owens, A., Efros, A.A.: Audio-visual scene analysis with self-supervised multisensory features. European Conference on Computer Vision (ECCV) (2018)
26. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems. pp. 8024–8035 (2019)
27. Qian, Q., Shang, L., Sun, B., Hu, J., Li, H., Jin, R.: Softtriple loss: Deep metric learning without triplet sampling. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 6450–6458 (2019)
28. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
29. Roth, K., Brattoli, B., Ommer, B.: Mic: Mining interclass characteristics for improved metric learning. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 8000–8009 (2019)
30. Roth, K., Milbich, T., Sinha, S., Gupta, P., Ommer, B., Cohen, J.P.: Revisiting training strategies and generalization performance in deep metric learning (2020)
31. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. International journal of computer vision **115**(3), 211–252 (2015)
32. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 815–823 (2015)
33. Smirnov, E., Melnikov, A., Novoselov, S., Luckyanets, E., Lavrentyeva, G.: Doppelganger mining for face representation learning. In: Proceedings of the IEEE International Conference on Computer Vision Workshops. pp. 1916–1923 (2017)
34. Sohn, K.: Improved deep metric learning with multi-class n-pair loss objective. In: Advances in Neural Information Processing Systems. pp. 1857–1865 (2016)

35. Suh, Y., Han, B., Kim, W., Lee, K.M.: Stochastic class-based hard example mining for deep metric learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7251–7259 (2019)
36. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946 (2019)
37. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The Caltech-UCSD Birds-200-2011 Dataset. Tech. Rep. CNS-TR-2011-001, California Institute of Technology (2011)
38. Wang, H., Wang, Y., Zhou, Z., Ji, X., Gong, D., Zhou, J., Li, Z., Liu, W.: Cosface: Large margin cosine loss for deep face recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5265–5274 (2018)
39. Wang, J., Zhou, F., Wen, S., Liu, X., Lin, Y.: Deep metric learning with angular loss. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2593–2601 (2017)
40. Wang, X., Han, X., Huang, W., Dong, D., Scott, M.R.: Multi-similarity loss with general pair weighting for deep metric learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5022–5030 (2019)
41. Weinberger, K.Q., Blitzer, J., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. In: Advances in neural information processing systems. pp. 1473–1480 (2006)
42. Wilber, M., Kwak, S., Belongie, S.: Cost-effective hits for relative similarity comparisons. In: Human Computation and Crowdsourcing (HCOMP). Pittsburgh (2014), `/se3/wp-content/uploads/2015/01/hcomp-conference-paper.pdf,http://arxiv.org/abs/1404.3291`
43. Wu, C.Y., Manmatha, R., Smola, A.J., Krahenbuhl, P.: Sampling matters in deep embedding learning. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2840–2848 (2017)
44. Yu, B., Tao, D.: Deep metric learning with tuplet margin loss. In: The IEEE International Conference on Computer Vision (ICCV) (October 2019)
45. Yuan, T., Deng, W., Tang, J., Tang, Y., Chen, B.: Signal-to-noise ratio: A robust distance metric for deep metric learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4815–4824 (2019)
46. Yuan, Y., Yang, K., Zhang, C.: Hard-aware deeply cascaded embedding. In: Proceedings of the IEEE international conference on computer vision. pp. 814–823 (2017)
47. Zhai, A., Wu, H.Y.: Classification is a strong baseline for deep metric learning. arXiv preprint arXiv:1811.12649 (2018)
48. Zheng, W., Chen, Z., Lu, J., Zhou, J.: Hardness-aware deep metric learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 72–81 (2019)