Generalization in Metric Learning: Should the Embedding Layer be the Embedding Layer?

Nam Vo
Georgia Tech
namvo@gatech.edu

James Hays
Georgia Tech
hays@gatech.edu

Abstract

Many recent works advancing deep learning tend to focus on large scale setting with the goal of more effective training and better fitting. This goal might be less applicable to the case of small to medium scale. Studying deep metric learning under such setting, we reason that better generalization could be a big contributing factor to improvement of previous works, as well as the goal for further improvement.

We investigate using other layers in a deep metric learning system (beside the embedding layer) for feature extraction and analyze how well they perform on training data and generalize to testing data. From this study, we suggest a new regularization practice and demonstrate state-of-the-art performance on 3 fine-grained image retrieval benchmarks: Cars-196, CUB-200-2011 and Stanford Online Product.

1. Introduction

We study deep metric learning from the perspective of generalization. Research in the field of computer vision has focused mostly on test time performance pushing state-of-the-art. However analyzing training performance ould lead to new insight; for example He at al [7] showed deeper network under-perform on training data and proposed residual connection to overcome that obstacle.

Deep learning has helped to advance many computer vision tasks, including fine grained image retrieval: identifying reference images semantically similar to the input. Through multiple layers of linear and non-linear operations, a deep convnet can transform the input image into a feature vector in a semantically meaningful high dimensional space; we can learn such a representation by training on a (or multiple) semantic task. Frequently in order for it to be optimal for direct similarity comparison, a distance metric learning loss is employed as the main objective during training; these approaches are called embedding learning or deep metric learning (DML). Note that many times cosine/dot-product is used as similarity measurement instead, usually for preference reason; we consider them collectively the same task in this paper.

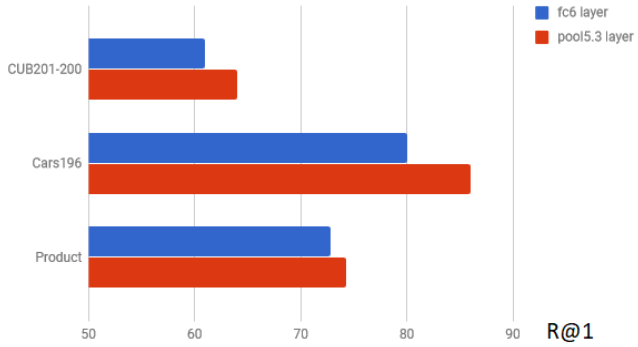

Figure 1. Recall at rank 1 performance on 3 benchmarks of the embedding layer (fc6) vs the layer before it (pool5.3)

Given a trained network (for any task, not necessary DML only), the output of any layer can be used as image feature. DML works often use the "embedding" layer: the last one in the network that is the input of the loss function. We train a network using DML and decide to measure the performance of other layers; interestingly the layer before the embedding layer often outperform it, as shown in Figure 1. Upon further investigation, it is found that while the last layer perform the best on training data, it doesn't generalize well.

Motivated by above observation, we present here a study of generalization in the context of small to medium scale DML; common practice is to finetune a network that is pretrained on large scale data. In such scenario, it is very easy to fit the training data and so the generalization ability of the network is important.

Given Figure 1 result, we explore the following hypotheses:

1. The embedding layer is not pretrained: so it does not generalize as well penultimate layer pool5.3.

2. Pool5.3 layer is shallower: shallow networks fit training data worse, but generalize better than deeper networks.

3. Distance from the loss layer: the closer a layer to the loss, the better its feature fits training data and the worse it generalize to test data.

We show, from our analysis, how to train a simple DML pipeline that outperforms state-of-the-art on 3 fine grained image retrieval benchmarks CUB-200-2011, Cars-196 and Stanford Online Products. Our finding can be potentially applicable to previous works to improve their performance.

Related works are discussed in the next section. We describe our DML system baseline in Section 3 and perform the ablation study in Section 4. We provide more detail about our experiments in Section 5 and conclude the paper in Section 6.

Note that when we make comparison statements, unless stated otherwise it is about image retrieval performance, more specifically in term of recall at rank 1, the commonly used metric in the literature.

2. Related works

Deep metric learning in computer vision has many applications, from generic image search [31] to sketch-based search [21], image geolocalization [29, 28], people re-identification [4, 8] and face recognition [22, 18].

The popular approach is to use a Siamese [2, 3] or triplet network [35, 31, 29]: training examples are put in the form of pairs or triplet, with the label being whether they are similar or not; and either the contrastive loss or triplet hinge loss is used for training, it encourages similar images to have similar features and dissimilar images to have different features. Most recent advances of DML focus on example mining strategy and/or better loss function.

The Multibatch method [26] performs metric learning on all possible pairs from the mini-batch and show that it reduces the variance of the estimator and significantly speed up the convergence rate. In [29], a similar trick called mini-batch exhausting improves retrieval performance by simply extending the learning to all possible triplets in the mini batch. In [8], it is studied under the name of batch-all and batch-hard. Kihyuk Sohn [23] used a similar batch construction strategy and proposed the multiclass n-pair loss which improves upon the triplet loss.

In [13], Harwood et al proposed a smart mining procedure to select effective samples from the whole training data; this help the training to converge faster. Huang el al [10] proposed position-dependent deep metric unit to adaptively select hard examples taking into account the local neighborhood. In [34], Wu et al propose to sample uniformly w.r.t. their relative distance; this distance weighted sampling is shown to be more effective than random or the common semi-hard sampling.

Wang et al [32] proposed a new loss that constraints the angle of the triplet triangle and demonstrate favorable properties over the traditional distance based hinge loss for triplet. Different from previous works, the proxy based loss [15] behaves like a classification loss; each training example is compared against learnt proxies, not with each other in a pair, triplet or cluster, hence eliminate the need of example mining.

In [17], multiple embeddings are learnt at the same time, later one would focus more on examples that are deemed hard to previous ones; the output embedding is a concatenation of all learnt embedding. In a similar spirit, Yuan et al [36] propose to learn a set of embeddings organized in a cascaded manner, in which easy training examples are filtered out sequentially and only the hardest one reach the last embedding loss.

3. Our choice of a deep metric learning system

Our thesis is that generalization is an important factor (we will support the argument with experimental result in the next section). With that in mind, we describe the DML design that we choose to work with. Note that we are not proposing a new method, all components here already exist or can be easily implemented; though we will release the source code and models from this study.

3.1. Base architecture

We use the VGG-16 architecture with conv layers only (fc6, fc7 and fc8 are removed), followed by a global pooling layer. This architecture is popular for scene image retrieval task [1, 27, 19, 5]. Although more sophisticated pooling method can be better (NetVLAD [1], R-MAC [27] or GeM [20]), we use the simple MAX pooling.

We employ BatchNorm [11] in our system, which is known to make training efficient and also provide regularization effect. Greff et al [6] shows, with ResNet and Highway network making optimization easier, the role of BatchNorm becomes purely regularization (higher training loss, lower testing error). Dropout is also experimented with but not as effective as a regularizer.

We use PyTorch as the framework for experimenting, which comes with VGG-BathNorm pretrained on ImageNet.

3.2. Feature extraction layers

A new fully connected layer fc6 is added at the end of the network, this is the traditional output layer or embedding layer. In this work we will experiment with using, not only this last layer, but also other layers before it as the feature extraction layer.

In the experiment we will describe other variants where we further add fc7 and fc8. All of them have the same output size of 512.

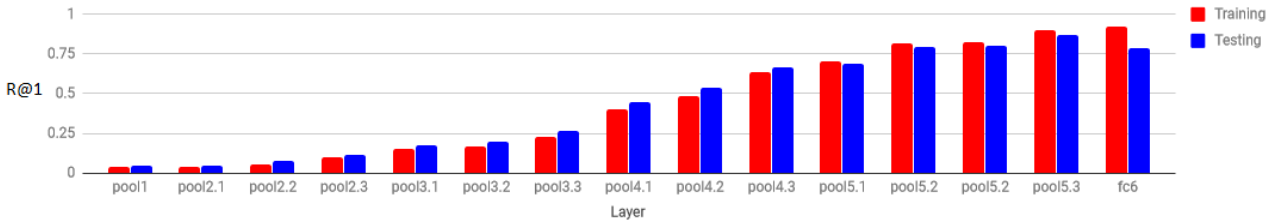

Figure 2. R@1 performance of different layers on training and test set of Cars-196.

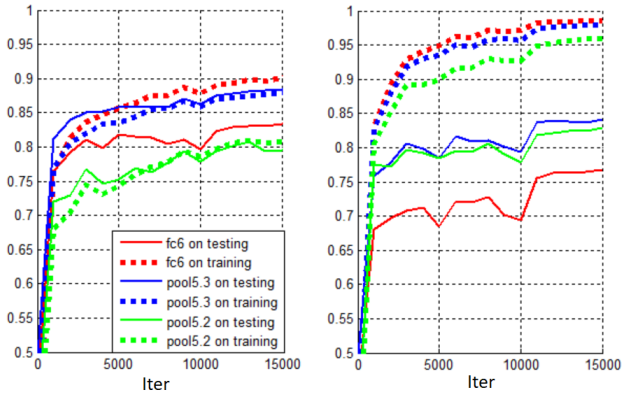

Figure 3. R@1 performance of different layers on training and test set of Cars-196. Left: our baseline network trained with DML loss; Right: the variant trained with classification loss.

3.3. Training mini-batch construction

While DML loss often involve looking at pair or triplet as example, recent works construct batch of images as input instead, and only form pairs/triplets/clusters at the loss layer. Hence the mini-batch can be constructed randomly in any way as long as a lot of similar/dissimilar pairs/triplet can be formed (for example [29, 23]).

In case the data comes with class label, we can randomly sample p classes ($p > 1$), k images per class ($k > 1$) resulting in a batch size of $m = pk$. Under fixed m, maximizing p will increase the diversity leading to more stable gradient; even though the number of possible triplets, $pk(k-1)(pk-k)$, is reduced.

3.4. Loss function and example mining

Aside from [19] and [13] that actually perform hard example mining on the whole training data, most mining strategies operate within mini-batch, which is constructed randomly uniform. Hence all these approaches can simply be formulated as a loss function operating at mini-batch level looking at all possible pairs/triplets/clusters and weight them differently. For instance, the recently proposed focal loss for classification task [14] quickly diminish the contribution of easy examples.

Hence we are not doing any explicit hard example mining. Given the batch b of output image features, our loss is defined as

$$L(b) = \frac{1}{M} \sum_{triplet(a,p,n) \subset b} tripletloss(a,p,n)$$

where $M = \sum_{triplet(a,p,n) \subset b} 1$, the total number of valid triplets in a batch; (a, p, n) is a triplet of the anchor, the similar and the dissimilar instance respectively. We follow [29, 23, 8] and use the smooth loss function (which shown to be better than the traditional hinge loss):

$$tripletloss(a,p,n) = log(1 + exp(d(a,n) - d(a,p)))$$

Where d is the squared euclidean distance (negative dot product can also be used instead). Note that it is sensitive the the scale of the image feature. In our implementation we normalize the image feature to have unit magnitude and then scale it by 4, as suggested by [29]. An alternative is to not do normalization and let the network learn the scaling, as suggested [8]; which also works in our experience, though one has to be careful with the initialization to avoid numerical instability at the beginning.

4. Studying How Well a Layer Generalizes

Using the system described in the last section as the baseline, we experiment on the small scale dataset Cars196 [12] (more experiment detail and result on other datasets will be reported in the next section). In this section we describe the ablation study.

For retrieval task, the common metric Recall at rank 1 (R@1) is used, which is the percentage of test images whose first nearest neighbor (in the feature space) is of the same class.

Using the ImageNet-pretrained model, we train the system on the training set, then measure the performance on the test set using either the embedding layer (fc6) or the one before it (pool 5.3) for feature extraction; the surprising result is that pool5.3 is better (Figure 1).

In Figure 2, we show the performance of all layers in our network on training data and testing data (we perform max pooling so that all features have similar dimensionality of 512, the raw feature outputs of early layers are too big to be

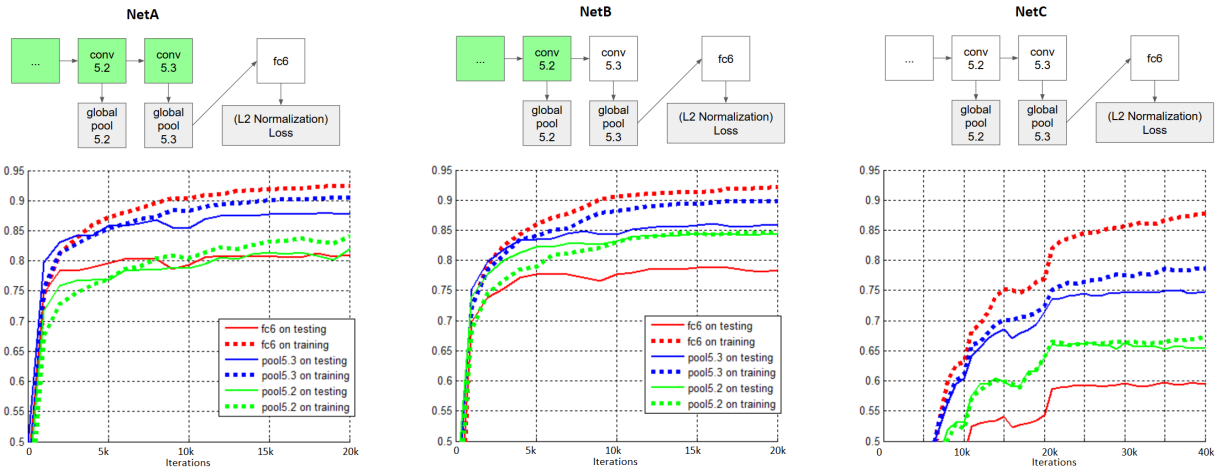

Figure 4. R@1 performance of different layers on training and test set of Cars-196. Green box: pretrained layer, white box: initialized from scratch layer, gray box: parameterless layer. Best viewed in color.

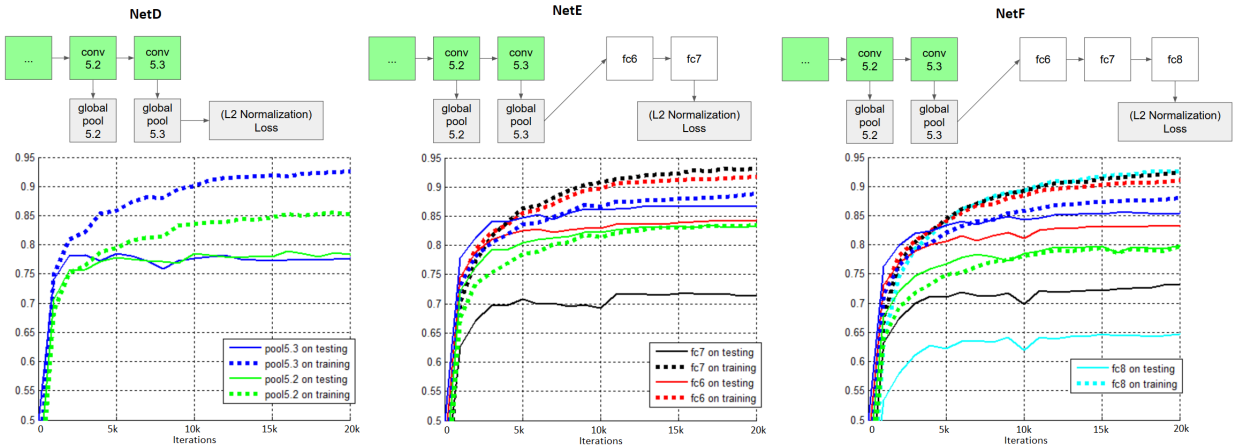

Figure 5. R@1 performance of different layers on training and test set of Cars-196. Compared to NetA in Figure 4, the position of the loss function is changed. Best viewed in color.

practical and actually perform worse). In Figure 3-left, we plot the performance of the last 3 layers pool5.2, pool5.3 and fc6 changing as the network is trained, reviewing generalization is the explanation. Although fc6 improves and surpass pool5.3 on training data as expected, the gap between training and testing gets bigger quickly, resulting in worse test time performance. This goes back to the fact that small to medium scale training is more susceptible to over-fitting than under-fitting.

4.1. The role of better loss and training strategy

Our system has no problem over-fitting such small scale training data if trained long enough (for example over 90% R@1 can be achieved in 100 epochs). We speculate that it is similar for other systems from related works. This might contradict with the thesis of these works: improvement is achieved because the novel loss can fit better, or the new strategy can select more effective examples for training. We propose a complementary reasoning: another important factor here is the generalization ability of network, which is affected by the loss function, training and architecture choice, or regularization techniques.

We provide another example to support this argument: we train the same baseline network, but replace the DML loss with classification-output-fc7 and classification-loss (Softmax-CrossEntropy); the result is shown in Figure 3-right. Under this loss, the fitting and generalizing behavior is quite different: all 3 layers fit better, but generalization is worse, especially for fc6, which even pool5.2 now outperforms.

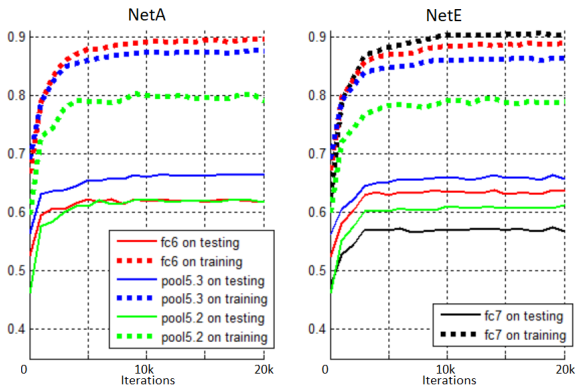

Figure 6. R@1 performance on CUB-200-2011.

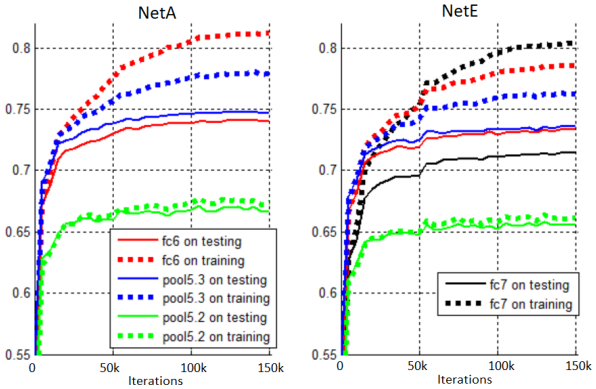

Figure 7. R@1 performance on Stanford Online Product

Note that classification loss has been previously shown to help improving retrieval performance in multi-task/co-training, especially in large scale training setting where fitting data really well is the more important factor [18, 33]. Even though the experiment result is different from ours, Horiguchi et al [9] argue that classification loss is advantagous when the size of training data (samples per class) is large. In [28], it is argued that learning with classification is more effective than metric learning because training data is very diverse/noisy.

4.2. Is pretrained layer is better for embedding

Now we know that pool5.3 can generalize better than fc6; the first possible explanation for that is: conv5.3 is a pretrained layer while fc6 is initialized from scratch. We compare 2 cases in Figure 4: NetA, the same baseline we are using, and NetB, where both conv5.3 and fc6 are initialized from scratch; for reference we also include the case of NetC where the whole network is initialized from scratch.

It can be seen that with one or more layers initialized from scratch, the performance of most degrades accordingly. Except fc6, directly under loss layer, can continue fit the training data better if trained long enough.

The training performance of fc6 is better than pool5.3 which in turn is better than pool 5.2. This is expected, as the feature obtained from a layer that is closer to the loss layer work better on training data.

Performance on test data is more interesting:

- NetB:pool5.2 is much better than NetA:pool5.2. It could be conv5.3 being initialized from scratch has a regularization effect. So pretraining a layer might be good for layers after it but has a "bad effect" for layer before it?

- Pool5.3 performs the best, whether it is pretrained or not. In fact NetB:pool5.3 outperforms both NetB:fc6 and NetA:fc6. It is also better than the NetB:pool5.2 (where conv5.2 is pretrained).

So while conv5.3 being pretrained does indeed help improve performance, this alone does not explain why pool5.3 is better than fc6.

4.3. The depth and the loss layer's position

We explore other potential explanation: layer's depth and the position of loss function. We move the loss layer up/down and introduce 3 new cases, shown in Figure 5

- NetD: similar to NetA except we remove fc6, so the loss layer is now directly after pool5.3

- NetE: similar to NetA except we add another fully connected layer fc7.

- NetF: similar to NetA except we add 2 fully connected layer fc7 and fc8.

All new layers have the same output size 512 with ReLU activation between them (not shown in the figure).

We look at the test performance of a layer when it is directly before the loss layer and when it is 2 layers away from the loss layer: pool5.3 in NetD is significantly worse than NetA (even though it's slightly better on training data). Interestingly, we observed similar result for fc6 layer (NetA vs NetE) although the difference is much smaller; and for the case of fc7 (NetE vs NetF) the difference is not significant.

The last important observation is that: when initialized from scratch, and not directly under the loss layer, pool5.3 (NetB) and fc6 (NetE) performance are actually not too different. When directly under the loss layer, pool5.3 (NetD) can be slightly worse than fc6 (NetA).

The position of the loss layer w.r.t. the feature extraction layer seems to play an important role explaining why one layer is better than the other. Hence to get the best possible test time performance, one should consider using other layers beside the last layer; another option is to distance the

Figure 8. Some nearest neighbor (NN) retrieval examples on the 3 datasets, we show cases in which using feature from different layers results in different NN.

intended output layer from the loss layer so that it has a less direct influence, therefore reduce overfitting.

As best practice, we suggest to use NetA:fc6 and NetA:pool5.3 respectively in the cases of less over-fitting to more over-fitting. In small scale setting, NetE:pool5.3 can also do well; and NetE:fc6 can be used when one can not control the pretraining step but want to define the embedding dimension. Note that when considering these options, it should be taken into account that their behavior can be different with not only different dataset/task, but also different loss function and training strategy.

5. Experiments

Here we provide more detail about our system implementation, and the experiment set up on 3 datasets:

- Cars196 [12]: consist of 16k images of 196 different classes of cars. Following the typical protocol, we train on the first 98 classes and test on the other 98 classes.

- CUB-200-2011 [30]: has around 12k images of 200 bird species. The first 100 classes (5864 images) are used for training and the rest for testing.

- Stanford Online Product [25]: is bigger than the other 2 with 120k images of more than 22k products from ebay. Similarly the first half (11318 classes with a total of 59551 images) are used for training and the remaining classes are for testing.

We use stochastic gradient descent with momentum weight 0.9, weight decay factor 5e-4, learning rate 0.01 (decreasing by 10 times during training). Random scaling, cropping and flipping is used for data augmentation; a single center crop is used during test time. We use batch size 32, train for 20k iterations in case of Cars-196 and CUB-200-2011, and 200k iterations in case of Online Product.

Table 1. R@k performance on 3 benchmarks.

Dataset	Cars-196						CUB-200-2011						Product			
k	1	2	4	8	16	32	1	2	4	8	16	32	1	10	100	1000
NetA:pool5.3	87.8	92.7	95.6	97.5	98.6	99.2	66.4	77.5	85.4	91.3	95.2	97.1	74.8	88.3	95.2	98.5

Table 2. R@1 performance on 3 benchmarks, compared to previous works.

Method	Network	dim	Cars-196	CUB-200	Product
Lifted structure [16]	GoogLeNet	64/64/512	53.0	47.2	62.5
N-Pair loss* [23]	GoogLeNet	64/64/512	71.1	51.0	67.7
Facility [24]	Inceptionv1BN	64	58.1	48.2	67.0
Angular loss [32]	GoogLeNet	512	71.4	54.7	70.9
HDC [36]	GoogLeNet	128x3	73.7	53.6	69.5
PDDM Quadruplet** [10]	GoogLeNet	128	57.4	58.3	69.5
BIER [17]	GoogLeNet	512	78.0	55.3	72.7
Margin [34]	ResNet50	128	**79.6**	**63.6**	72.7
Proxy-NCA [15]	InceptionBN	64	73.2	49.2	**73.7**
Ours	VGG-Conv & BN				
NetA	– pool5.3	512	**87.8**	**66.4**	**74.8**
	– fc6	512	81.0	61.7	74.1
NetE	– pool5.3	512	86.7	65.8	73.6
	– fc6	512	84.3	63.7	73.3
	– fc7	512	71.4	56.8	71.4

*: [23] uses multiple random crops during test time
**: [10] uses object bounding box cropped images.

5.1. Result

In Figure 6, we show NetA and NetE performance on CUB-200-2011. This dataset is much harder to generalize even though its size is similar to that of Cars-196. Still pool5.3 is the best performing layer; and fc6 does better if it is not next to the lost layer.

In Figure 7, we show performance on Stanford Online Product. Similar to previous cases, fc6 is outperformed by pool5.3; however due to this dataset's bigger size, there's some differences: (1) it takes much longer to train, (2) NetA seems to fit the training data slightly better/faster than NetE, resulting in better test time performance too. We speculate that when training at even larger scale, bad generalization will be a less significant concern.

We show some qualitative retrieval examples in Figure 8.

Comparing to state-of-the-arts: in Table 1 and 2, we report the R@1 performance of different layers of NetA and NetE. As a baseline, our fc6 yields comparable result to some state of the art approaches, and our pool5.3 result outperform all of them. Note that each component (loss function, mining strategy, etc) of each system might not be directly comparable because of difference in network architecture and embedding dimension (and possibly data preprocessing, training hyper-parameters, etc). Moreover we are investigating the generalization effect of different layers, not proposing a new method to replace existing works.

5.2. Analyzing publicly released models

This effect can also be demonstrated on other models, not just ours. Among previous works, LiftedStructure [16] and HDC [36] have released the models used in their papers. LiftedStructure [16] released 4 models for each benchmarks, trained with different embedding size 64, 128, 256 and 512. With the same architecture, HDC [36] model however has 3 different output layers, the system's final output is the concatenation of them. We obtained and tested these models, the result is shown in Table 3.

Compared to other systems, LiftedStructure [16] model fits training data the best on Cars-196, though not on CUB-200 and Product. This is because it was trained with 100 times smaller learning rate on CUB-200 and 10 times smaller number of epochs on Product. Note that the output layer surpasses the penultimate layer on training set only with enough fitting, which might not be the case here. Nevertheless during test time, the output layers are significantly outclassed by the layer before them.

HDC model [36] seems trained to fit training data better than Lifted Structure (except on Cars-196); with output3 layer, trained on the hardest examples, perform better than the other 2 output layers. We speculate that lower performance compared to our system on both training and test set is because of smaller number of training epoches. Still

choosing penultimate layers for feature extraction improves the retrieval performance of this model, same as ours.

One could argue that, different from our case, it is mainly because of bigger embedding size here. However notice that output layers still fit quite well on training data even with smaller embedding size, but generalize worse on testing data. [16] made an observation that the embedding size did not play a significant role in their system. But we do observe that the size is a contributing factor affecting both fitting and generalizing performance, and not only that of the embedding layer but also other layers' as well.

Finally, we observe that our system generalizes better: at similar training performance, our output layer still perform better on test set (refer to Figure 4, 6 and 7); this is also the case for penultimate layers. We speculate the reason to be difference in embedding size, loss function, network architecture and its corresponding pretrained model. Worse generalization might be what prevented previous works from fitting the training data further.

6. Conclusion

In this study we propose that, in small to medium training setting, generalization is a big contributing factor to recent advancements in DML. As we has shown, choosing a different loss or different feature extraction layer can improve test time performance even though it actually underperform on training data, and vice verca. Hence generalization should be kept in mind when designing or analyzing new techniques in the future.

Given that any layer in a network can be used for feature extraction, major part of the this work is a ablation study of how each layer perform differently in the fine grained image retrieval task. Not too surprising, we found that the closer a layer to the loss layer, the better its produced feature is. Different from what people would assume, the last layer is usually however not the best one at test time, due to bad generalization; the second last layer is. Similar effect is also observed in other models released by previous works.

Table 3. R@1 performance of different layers from publicly released lifted structure model. Penultimate denotes the layer before the output layer.

R@1		Cars-196		CUB-200		Product	
Layer	dim	train	test	train	test	train	test
Lifted Structure 4 released models [16]							
model-64	64	97.9	49.7	67.0	42.8	63.5	60.2
penultimate	1024	96.5	**70.8**	67.2	**49.3**	66.6	64.8
model-128	128	**98.0**	50.1	66.6	44.5	63.9	60.8
penultimate	1024	96.9	70.1	**67.3**	49.1	66.8	65.0
model-256	256	97.4	50.0	66.9	45.3	64.4	61.2
penultimate	1024	96.6	68.7	**67.3**	48.8	67.0	65.0
model-512	512	97.7	47.9	66.4	46.3	65.2	62.1
penultimate	1024	97.0	65.1	66.4	47.1	**67.7**	**65.9**
HDC released model [36]							
output1	128	60.6	41.9	66.1	34.3	69.7	61.6
penultimate1	1024	60.9	54.3	67.4	40.3	70.6	64.5
output2	128	71.0	58.0	78.0	44.0	71.3	64.9
penultimate2	1024	68.8	63.8	76.8	48.8	70.9	65.5
output3	128	79.4	71.4	**84.0**	52.7	73.6	67.2
penultimate3	1024	73.6	74.1	79.7	55.8	72.9	68.8
all outputs		**80.4**	73.7	83.7	53.6	**76.5**	69.5
all penultimates		75.5	**75.7**	79.9	**57.2**	75.6	**70.1**
Our NetA							
output (fc6)	512	**92.4**	81.0	**89.6**	61.7	**81.2**	74.1
penultimate	512	90.4	**87.8**	87.6	**66.4**	77.9	**74.8**

References

[1] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5297–5307, 2016.

[2] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994.

[3] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 539–546. IEEE, 2005.

[4] S. Ding, L. Lin, G. Wang, and H. Chao. Deep feature learning with relative distance comparison for person re-identification. *Pattern Recognition*, 48(10):2993–3003, 2015.

[5] A. Gordo, J. Almazán, J. Revaud, and D. Larlus. Deep image retrieval: Learning global representations for image search. In *European Conference on Computer Vision*, pages 241–257. Springer, 2016.

[6] K. Greff, R. K. Srivastava, and J. Schmidhuber. Highway and residual networks learn unrolled iterative estimation. *arXiv preprint arXiv:1612.07771*, 2016.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] A. Hermans, L. Beyer, and B. Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[9] S. Horiguchi, D. Ikami, and K. Aizawa. Significance of softmax-based features in comparison to distance metric learning-based features. *arXiv preprint arXiv:1712.10151*, 2017.

[10] C. Huang, C. C. Loy, and X. Tang. Local similarity-aware deep feature embedding. In *Advances in Neural Information*

Processing Systems, pages 1262–1270, 2016.

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[12] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[13] V. B. Kumar, B. Harwood, G. Carneiro, I. Reid, and T. Drummond. Smart mining for deep metric learning. *arXiv preprint arXiv:1704.01285*, 2017.

[14] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

[15] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh. No fuss distance metric learning using proxies. *arXiv preprint arXiv:1703.07464*, 2017.

[16] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4004–4012, 2016.

[17] M. Opitz, G. Waltner, H. Possegger, and H. Bischof. Bier-boosting independent embeddings robustly. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5189–5198, 2017.

[18] O. M. Parkhi, A. Vedaldi, and A. Zisserman. Deep face recognition. In *British Machine Vision Conference*, 2015.

[19] F. Radenović, G. Tolias, and O. Chum. Cnn image retrieval learns from bow: Unsupervised fine-tuning with hard examples. In *European Conference on Computer Vision*, pages 3–20. Springer, 2016.

[20] F. Radenović, G. Tolias, and O. Chum. Fine-tuning cnn image retrieval with no human annotation. *arXiv preprint arXiv:1711.02512*, 2017.

[21] P. Sangkloy, N. Burnell, C. Ham, and J. Hays. The sketchy database: learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (TOG)*, 35(4):119, 2016.

[22] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[23] K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in Neural Information Processing Systems*, pages 1857–1865, 2016.

[24] H. O. Song, S. Jegelka, V. Rathod, and K. Murphy. Deep metric learning via facility location. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[25] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[26] O. Tadmor, Y. Wexler, T. Rosenwein, S. Shalev-Shwartz, and A. Shashua. Learning a metric embedding for face recognition using the multibatch method. *arXiv preprint arXiv:1605.07270*, 2016.

[27] G. Tolias, R. Sicre, and H. Jégou. Particular object retrieval with integral max-pooling of cnn activations. *arXiv preprint arXiv:1511.05879*, 2015.

[28] N. Vo, N. Jacobs, and J. Hays. Revisiting im2gps in the deep learning era. *ICCV*, 2017.

[29] N. N. Vo and J. Hays. Localizing and orienting street views using overhead imagery. In *European Conference on Computer Vision*, pages 494–509. Springer, 2016.

[30] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical report, 2011.

[31] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1393, 2014.

[32] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin. Deep metric learning with angular loss. *arXiv preprint arXiv:1708.01682*, 2017.

[33] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *European Conference on Computer Vision*, pages 499–515. Springer, 2016.

[34] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krähenbühl. Sampling matters in deep embedding learning. *arXiv preprint arXiv:1706.07567*, 2017.

[35] P. Wu, S. C. Hoi, H. Xia, P. Zhao, D. Wang, and C. Miao. Online multimodal deep similarity learning with application to image retrieval. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 153–162. ACM, 2013.

[36] Y. Yuan, K. Yang, and C. Zhang. Hard-aware deeply cascaded embedding. *arXiv preprint arXiv:1611.05720*, 2016.