Table of Content

Proxy Anchor Loss for Deep Metric Learning

Omar Moured[1]
[1]Electrical and Electronics Engineering

3230 Words Student ID:2102911
Keywords: Metric Learning, Pair-loss , Proxy-loss 27th January, 2021

Abstract

Metric learning is the task of learning a distance function over objects [1]. These trained functions can be used in many computer vision applications including, but is not limited to, classification, matching image correspondences, optical flow and multiple objects tracking. However, the critical and challenging question is that, what loss function should be used to teach these models to discriminate between the fine-grained semantic relations ? In this project I will focus on a very recent promising loss proposed in CVPR2020 by S.Kim et al [2].

1 Introduction

Metric learning aims to measure the similarity among samples while using an optimal distance metric for learning tasks [3]. In recent years, we are facing highly non-linear complex data which require utilizing fine-grained semantic relations. Thanks to neural networks and activation functions we can cope with the before-mentioned issues. Utilizing these tools with metric learning is called deep metric learning. Existing metric learning losses can be categorized into two classes: pair-based and proxy-based losses. The former class can leverage fine-grained semantic relations between data points, but slows convergence in general due to its high training complexity. In contrast, the latter class enables fast and reliable convergence, but cannot consider the rich data-to-data relations. More details will be illustrated later in this report. In this project, I am studying Proxy-Anchor loss [2], a new proxy-based loss that takes advantages of both pair- and proxy-based methods and overcomes their limitations.

2 Project Outcomes

Here is the major points I learnt from this project :

- Writing a professional Dataset-loader (for various datasets such as [4]) for metric learning tasks and reducing its memory-computation complexity.
- Studied various sampling techniques, to reduce the training time and memory requirements for low-RAM GPUs.
- Get to know, how Computer Vision researcher organize and manage ML experiments to track performance (such as using wandb).

3 Applications of Deep Metric Learning

Before diving into the paper, I would like at a glance, to go over few applications of DML networks.

3.1 Clustering

Utilizing the networks trained on deep metric learning can provide a robust distance measurements between all images in the dataset which can be then used to fill the affinity matrix. Figure 1, shows an example of utilizing Siamese Network to form a graph between MNIST handwritten data. Then, the images which are close to each other (up to a threshold) are combined in one cluster.

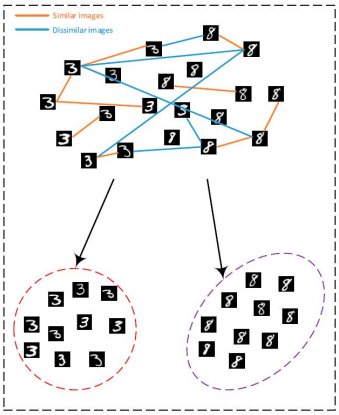


Fig. 1. Clustering MNIST images by using Siamese Network, trained with metric loss [3].

3.2 Visual Tracking

In Visual tracking, we are interested in associating detected targets from previous frames to the current one. To achieve that, the most similar objects pair are found by using DML network as in Figure 2, the network is trained to match similar pedestrians pairs [5].

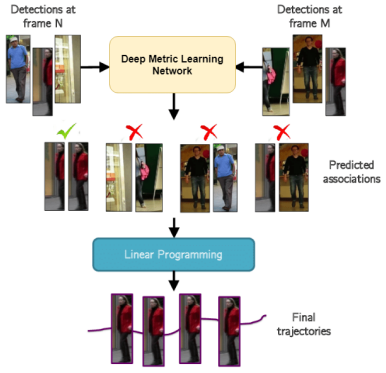


Fig. 2. Multiple object tracking with learned detection associations [5].

4 Datasets

In this section, I will go through the popular dataset used to measure the performance of DML methods. The first four are the ones used in the paper [2].

4.1 Caltech Birds Dataset [6]

It is part of Caltech Vision Dataset Archive. It consists of 200 different birds categories, in separate folders, each with various images. In total it has 11,788 images. The authors also provides 15 Part Locations, 312 Binary Attributes, 1 Bounding Box. But in this project, we will only focus in the image as a whole (match its category, and not the bird attributes). Figure 3 illustrate in example images from the dataset.

Fig. 3. Multiple birds images from [6] dataset. Images of the same row represent same category but different binary attributes

.

4.2 Cars-196 Dataset [7]

The Cars dataset (maintained by Stanford university) contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images, where each class has been split roughly in a 50-50 split.

Fig. 4. Sample images from Car-196 dataset [7].

4.3 Stanford Online Products Dataset [8]

120k images of 23k classes of online products for DML tasks. It is considered as the largest dataset for metric learning. This is why, I couldn't run experiments on this dataset.

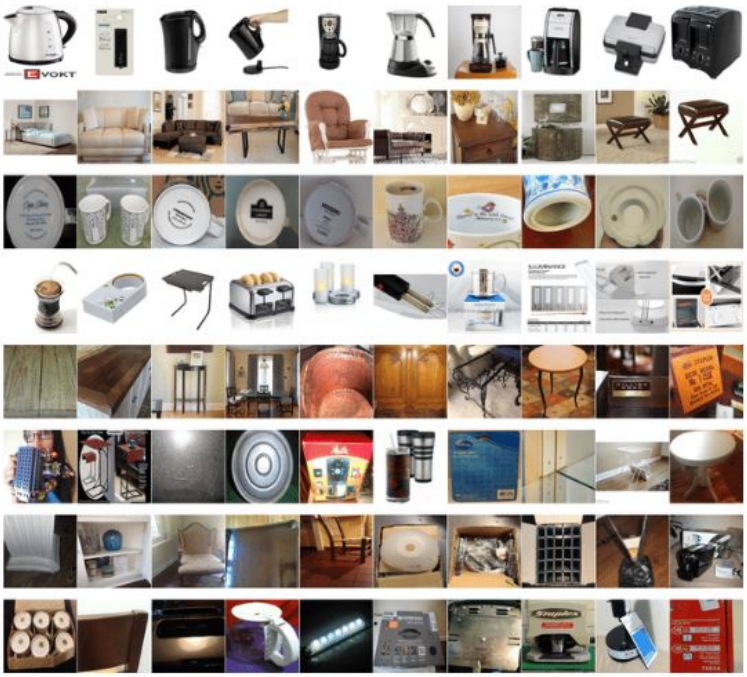


Fig. 5. Sample images from Stanford Online Products dataset [8].

4.4 Large-scale Fashion Database [9]

DeepFashion contains over 800,000 diverse fashion images ranging from well-posed shop images to unconstrained consumer photos, constituting the largest visual fashion analysis database. It consists of 50 categories, 1,000 descriptive attributes, bounding box and clothing landmarks.

Fig. 6. Sample images from Deepfashion Database [9].

4.5 Caltech-256 dataset

It is another dataset provided by Caltech Vision Dataset Archive. The Caltech 256 is considered an improvement to its predecessor, the Caltech 101 dataset, with new features such as larger category sizes, new and larger clutter categories, and overall increased difficulty. There are 30,607 images in this dataset spanning 257 object categories. Object categories are extremely diverse, ranging from grasshopper to tuning fork.

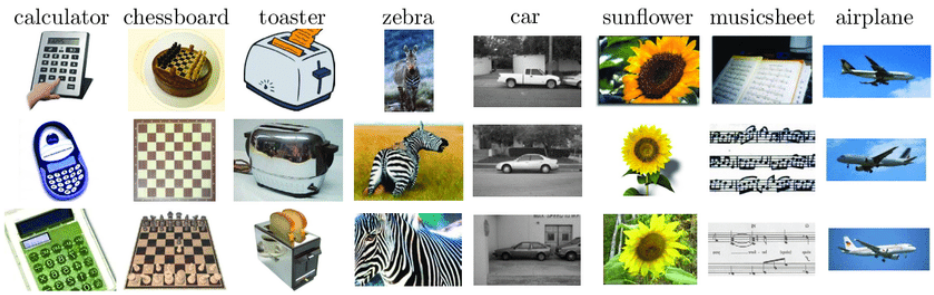


Fig. 7. Sample Images from Caltech-256 dataset.

5 DML Losses

In this section I will discuss how DML work, and what are the state-of-the-art cost functions we are trying to minimize. This section and the following ones will cover 5 different loss functions, namely, Triplet, N-pair, Lifted Structure, Proxy-NCA, and the proposed Proxy-Anchor loss [2].

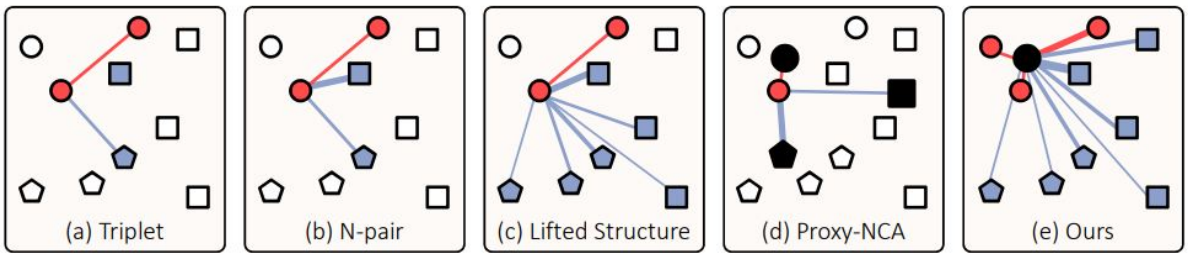


Fig. 8. Comparison between popular metric learning losses and the paper one. Small nodes are embedding vectors of data in a batch, and black ones indicate proxies; their different shapes represent distinct classes. The associations defined by the losses are expressed by edges, and thicker edges get larger gradients. (e) represents Proxy-Anchor loss [2].

As mentioned in the Introduction section, there are two types of DML loss. Anchor- and Proxy-based ones. In the next subsections, each will be illustrated along with its advantages and limitations.

5.1 Anchor-based Losses

The pair-based losses are built upon pairwise distances between data in the embedding space. Only images are forwarded through deep learning models to collect features and then measuring how far are they from each other to find the best match. In other words, the network is trained to project images onto an embedding space in which images from same class are semantically near and grouped to each other. However, since they take a tuple of data as a unit input, the losses cause prohibitively high training complexity, $O(M2)$ or $O(M3)$ where M is the number of training data, thus slow convergence. Other disadvantage is that, **some tuples** do not contribute to training or even degrade the quality of the learned embedding space. That's why some proposals use intelligent **sampling methods** to cope with this problem.

5.1.1 Triplet loss [10]

This loss takes a pair of embedding vectors (obtained from the trained neural networks) as input, and aims to pull them together if they are of the same class and push them apart otherwise. The graphical representation in Figure 8(a) can be also represented by the mathematical equation :

$$triplet_{loss} = \sum_{i}^{M} [||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha] \tag{1}$$

Where x_i represent a single training data out of M sample points. The exponent $a, p, and n$ represent wither the training sample is anchor (query), positive (match), negative one respectively. The third component α is a bias term which acts as the threshold. The aim is to minimize $triplet_{loss}$ from equation 1. Meaning, reduce the distance between $f(x_i^a)$ and $f(x_i^p)$ with pushing away $f(x_i^n)$. By using smart sampling techniques [10], the complexity of Triplet loss can be reduced to $O(M2)$, that is by only utilizing hard triplets (approximated nearest neighbor index). Even with that, the complexity is still quite high. It is nice to notice that Triplet loss is not utilizing all the data within the batch (3 data only).

5.1.2 N-pair [11] & Lifted Structure loss [12]

Both N-pair and Lifted Structure losses reflect hardness of data. N-pair uses N negative samples from different classes to be pushed away. While Lifted Structure loss uses N negative samples from every opposite class. With similar intelligent sampling techniques, complexity can reach $O(M3)$. Both losses show similar performance in complex datasets.

5.2 Proxy-based Losses

Proxy-based metric learning is a relatively new approach that can address the complexity issue of the pair-based losses. A proxy means a representative of a subset of training data and is estimated as a part of the embedding network parameters. Such as computing the mean of all embedding for every class. The common idea of the methods in this category is to infer a small set of **proxies that capture the global structure of an embedding space** and relate each data point with the proxies instead of the other data points during training. Since the number of proxies is significantly smaller than that of training data, the training complexity can be reduced substantially.

5.2.1 Proxy-NCA [13]

Proxy-NCA [21], which is an approximation of Neighborhood Component Analysis (NCA) [8] using proxies. In its standard setting, Proxy- NCA loss assigns a single proxy for each class, associates a data point with proxies, and encourages the positive pair to be close and negative pairs to be far apart, as illustrated in Figure 2(d). in [13], the proxy is randomly initialized along with the neural network weights. Proxy-NCA can be also represented by the mathematical equation :

$$\mathcal{L}(X) = \sum_{x \in X} -\log \frac{e^{s(x,p^+)}}{\sum\limits_{p^- \in P^-} e^{s(x,p^-)}} \tag{2}$$

Where x is a single sample from X **embedding vectors**. p^+ represent the positive proxy which correspond to the current x. While P^+ represent all the negative ones. $s(.)$ denotes the cosine similarity between two vectors. To understand how the loss in equation 2 work we need to consider it's derivative :

$$\frac{\partial \mathcal{L}(X)}{\partial s(x,b)} = \begin{cases} -1 & \text{if } p = p^+, \\ \frac{e^{s(x,p)}}{\sum\limits_{p^- \in P^-} e^{s(x,p^-)}} & otherwise \end{cases} \tag{3}$$

From the derivative 3 we can understand that minimizing the loss encourages x and p^+ to be pulled to each other by a **constant gradient**, while pushing away P^- negative samples relative to their cosine similarity. Meaning, harder negative proxies are pushed stronger. As we pointed in the Introduction, Proxy based methods have lower complexity than anchor based ones. For Proxy-NCA [13], the complexity is $O(MC)$ where C is the number of classes. In other words every image within the batch will be compared with C number of proxies. This loss can be understood better with the visual representation 8. We will discuss the disadvantages of this loss in the following section (Proxy-Anchor).

5.2.2 Proxy-Anchor

The proposed loss in [2] is a modification of the Proxy-NCA [13]. The main idea is to include the proxy as a sample data within the batch (anchor). That is to preserve data-to-data interaction along with proxy-to-data one, as shown in 8.

$$\mathcal{L}(X) = \frac{1}{|P^+|} \sum_{p \in P^+} \log(1 + \sum_{x \in X_p^+} e^{-\alpha(s(x,p)-\gamma)}) + \frac{1}{|P|} \sum_{p \in P} \log(1 + \sum_{x \in X_p^-} e^{\alpha(s(x,p)+\gamma)}) \tag{4}$$

The authors proposed the loss function 4 shown above. Where $\gamma > 0$ is a margin, $\alpha > 0$ is a scaling factor, P indicates the set of all proxies, and P^+ denotes the positive proxies of data in the batch. For each proxy p, a batch of embedding vectors X is divided into two sets: X_p^+ and $X_p^- = X - X_p^+$.

How it works: It is easy to notice that the loss is pulling and pushing all embedding vectors in the batch, **but with different strength** relative to their hardness (cosine similarity). To better compare the previous loss and this one, let's consider the gradient.

$$\frac{\partial \mathcal{L}(X)}{\partial s(x,b)} = \begin{cases} \frac{1}{|P^+|} \frac{-\alpha h_p^+(x)}{1 + \sum_{x' \in X_p^+} h_p^+(x')} & \forall x \in X_p^+, \\ \frac{1}{|P|} \frac{\alpha h_p^-(x)}{1 + \sum_{x' \in X_p^-} h_p^-(x')} & \forall x \in X_p^- \end{cases} \tag{5}$$

Where $h_p^+(x) = e^{-\alpha s(x,p)-\gamma}$ and $h_p^-(x) = e^{\alpha s(x,p)+\gamma}$ are positive and negative hardness metrics for embedding vector x given proxy p, respectively. As shown in the above equations 5, the gradient for $s(x,p)$ is affected by not only x but also other embedding vectors in the batch; the gradient becomes larger when x is harder than the others. In this way, Proxy-Anchor loss enables embedding vectors in the batch to interact with each other (data-to-data) and reflects their relative hardness through the gradients, which enhance the quality of the learned embedding space.

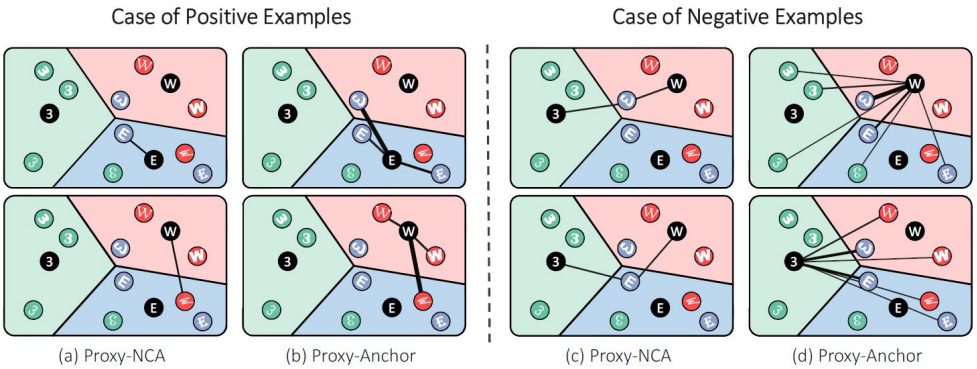


Fig. 9. Differences between Proxy-Anchor and Proxy-NCA [13] in handling proxies and embedding vectors during training.

5.3 Section Summary

In Proxy-NCA loss, the scale of the gradient is constant for every positive example (with value -1) and that of a negative example is calculated by taking only few proxies into account as shown in Eq.3, **the constant gradient scale for positive examples damages the flexibility and generalizability** of embedding networks. In contrast, Proxy-Anchor loss determines the scale of the gradient by taking relative hardness into consideration for both positive and negative examples as shown in Eq.5. This feature of Proxy-Anchor loss allows the embedding network to consider **data-to-data** relations that are ignored in Proxy-NCA and observe much larger area of the embedding space during training than Proxy-NCA. Figure 9 illustrates these differences between the two losses in terms of handling the relative hardness of embedding vectors. In addition, unlike Proxy-Anchor loss, **the margin** imposed in Proxy-Anchor loss leads to intra-class compactness and inter-class separability, resulting in a more discriminative embedding space.

6 Experiments

In this section, I will go through the used metrics in the field of Metric Learning to assist and evaluate the performance of these methods. Next, I will discuss how did I prepared the environment and obtained the same results as the authors (as a confirmation). Finally, my own experiments on different dataset than the one used in their paper. In addition to that, will discuss the ways I tried to enhance the proposed loss performance.

6.1 Metrics and Assessment

As we pointed in Datasets section, the datasets come with huge number of cl assess. A group of these classes will only be used for training and the rest of the classes are used for evaluation. That is, we would like to know, did the network learnt to find the similarity or it over-fits the training classes. The evaluation classes are totally different than the ones for training. For fair comparison between different methods, the split is usually defined in the datasets website. The following bullet points, define the split used in [2]:

- Caltech Birds Dataset [6] : 5,864 images of its first 100 classes for training and 5,924 images of the other classes for testing.
- Cars-196 Dataset [7] : 8,054 images of its first 98 classes are used for training and 8,131 images of the other classes are kept for testing.

6.1.1 Recall@K Metric

To measure the retrieval quality of the network, Recall@K metric is usually used in the literature. First, the query image is forwarded and compared with all other images in the test dataset. The top K images are then sorted according to their distance (descending order). The probability of having a match (class-match) within Top-1 image/s is called Recall@1. The same concept is applied to other K values.

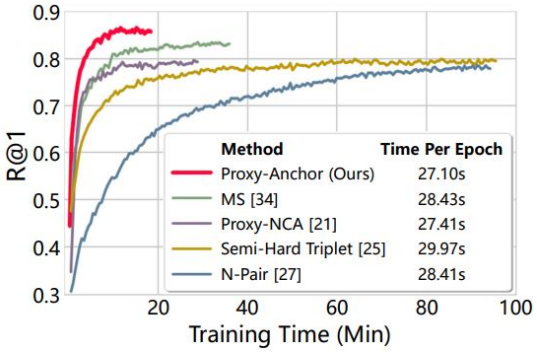
6.2 Environment Preparation

Before integrating the new dataset. I ran group of training session to compare between the claimed results and the one I obtained. In this subsection, I will be showing the paper original results and mine for the same dataset and training settings. In my desktop, I am using **NVIDIA GeForce RTX 2080 Ti** along with **Ubuntu 18.04 Docker**, and **CUDA 10.0**.

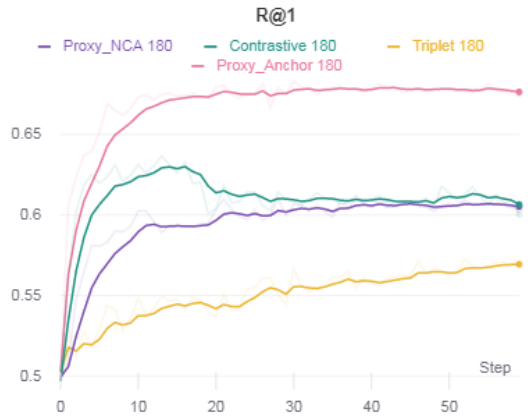
Table 1
My own results compared to the paper.

Method	Settings			**R@1 Performance**			
				Own		Paper	
	Model	Batch Size	Embedding Size	CUB-200-2011	Cars-196	CUB-200-2011	Cars-196
Anchor-Scale	bn_inception	180	520	67.5	85.5	68.4	86.1
	resnet	30	520	64.6	83.3	65.9	84.6
Proxy-NCA	bn_inception	180	64	49.2	72.9	49.2	73.2

I used $\alpha = 64$ and $\gamma = 0.1$ in all experiments shown in table 1. We see that my training sessions converged very closely to the paper results. Slight difference due to weights initialization. This issue can be overcome with initializing Pytorch with same seeds (should be provided by the author). Nevertheless, we are ready to integrate our dataset.



(a) The paper overall losses results.



(b) My own overall losses results (generated by wandb).

Fig. 10. Training time vs Recall@1 for CUB dataset for both paper (a) and own runs (b).

I have also generated a plot that combines all different losses to compare it with the one attached in the paper. Figure 10 shows how my training session yielded very close outcomes compared to the original paper for 4 diffrent losses for the Caltech-Birds dataset.

6.3 Experiments on Diffrent Dataset

Now we have shown that we ready to integrate different dataset and check the performance. In this project, I chose Caltech-256 dataset as the different dataset for evaluation. The first 200 classes will be used for training and the other 56 classes are kept for performance evaluation.

6.3.1 DataLoder

The first critical issue to deal with is the Dataloader. For a successful piece of code, the loader needs to utilize the minimum hardware capabilities. For fair comparison, I used the same transformation done on the original datasets.

- Normalize : using 3d-Mean (R, G, B): [104, 117, 128], and 3d-STD (R, G, B): [1, 1, 1].
- RandomResizedCrop : Crop the given image to random size and aspect ratio.
- RandomHorizontalFlip : Randomly flip the images with respect to x axes.
- RBG2BGR : If trained inception weights are used, The image is converted to BGR.

6.3.2 Results

In this section, I will report the best experiments run. I used Batch Normalization Inception Network as it learn the embedding much efficiently compared to others (Same conclusion mentioned in their work). Figure 14 illustrate example outputs for the top-4 retrievals. We can see how the network learnt the embedding invariant to transformation, background and illuminations changes.

Table 2
R@K [1,2,4,8] results for Caltech-256 dataset using 4 diffrent losses.

Loss	Settings						R@K [1,2,4,8]			
	Alpha	Epochs	Embedding-Size	Batch-Size	Learning Rate	Warm				
Proxy-Anchor	32	20	512	180	0.0001	5	0.9235	0.9462	0.9649	0.9786
Triplet							0.913	0.9416	0.9595	0.9746
Constrative							0.9106	0.9398	0.9605	0.9736
Proxy-NCA							0.9046	0.9375	0.9605	0.9745

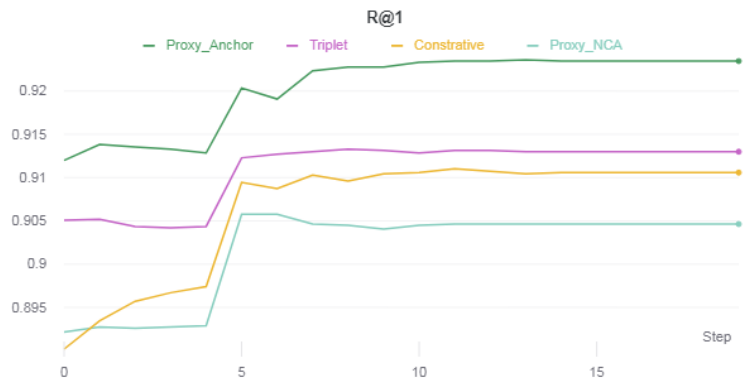


Fig. 11. R@1 Accuracy plot of Caltech-256 dataset. Color code on top. x-axis show epoch.

7 Discussion

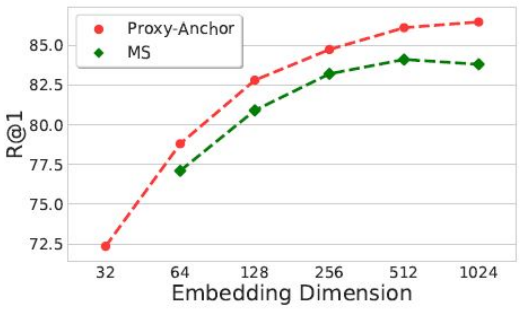
7.1 Impact of Hyperparameters

In Proxy-Anchor proposal, there are 4 important parameters to be discussed:

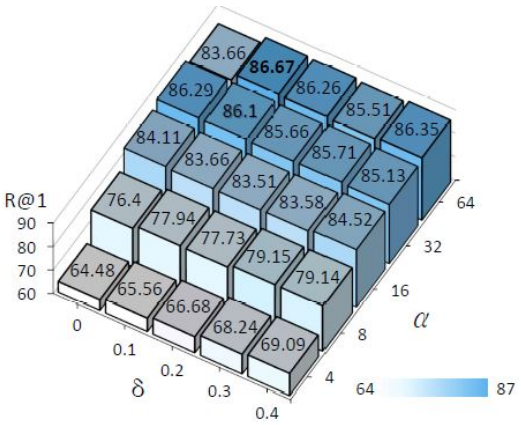
- Batch-Size : this parameter is not very critical for Proxy-Anchor. I have confirmed with my experiments in Table 1 that even with very low batch-size (ResNet) we still get good performance. However, it is relatively hard to learn the data-to-data relations if it is small. One rule I have observed is that, if the dataset classes we are targeting are not similar (e.g. not dog/birds categories dataset) then small batch-size will not harm the over all performance.

- Embedding-Size : Very critical parameter. It is highly sensitive depending on the network (higher with ResNet). The figure 12a below demonstrate the relation of embedding-size and the R@1 accuracy. It is clear that the relation is almost linear. Same previous conclusion can be made here. It all depend on the difficulty of the Target data.

- α and γ : one of the disadvantages of the paper is that there is no approach provided on how to tune these parameters or even how did they arrive to the best values. Some people commits from their GitHub repository found already better values which contradict what mentioned in their paper. Figure 12b below shows how with small change on these two parameters we may observe 10% change.



(a) The effect of Embedding-Size



(b) The effect of α and γ

Fig. 12. Figures indicating the effect of the parameters.

7.2 Room for Improvements

- **Tree Sampling** : one heuristic method to improve the performance is to sample the images in an intelligent way. Instead of randomly picking 180 samples for the batch. We can pick the most representative images at first to boost the training. [14] is one of the earliest work to discuss this matter. In their work, they first form a hierarchical tree similar to Figure 13. This tree can be formed by any approach such as K-Means where L_0 and L_3 represent the farthest and closest classes. But the important part, when sampling, pick the images from closest classes from L_0 level (to learn the discrimination embedding). Unfortunately, the training computation is now higher especially if the tree is chosen to be adaptive (updated every epoch). With experiments, I have concluded that this method is a good to start with if the dataset consist of huge number of classes. However, one may be careful when utilizing this approach because the Data-loader might never sample or sample very few images from a certain far class. Hence, the embedding between those minority become weak. A good solution is to use an adaptive tree. Still there are many recent research trying to cope with this challenge.

- **SimpleShot** : It is always important to ask, is it necessary to be this much complicated to achieve the same performance? one may argue whether it is necessary to use metric losses to achieve high quality embedding representations. [15] is one of the work that illustrate an important remark. In their work, Y. Wang et al k utilized standard classification loss even with very shallow network (4 convolutions) managed to achieve very high performance. The Idea is to train the network as if it is a classification problem. Then, the last layer was removed after and the output feature map is used as the embedding after flattening it. With few tricks training (Centring the data) and post-processing (L_2 normalization) tricks [15] proved that it is not necessary to utilize meta-learning losses even with highly non-linear dataset (mini-ImageNet).

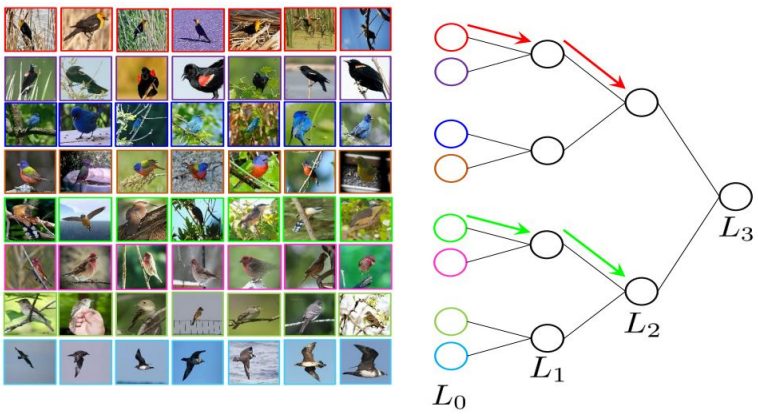


Fig. 13. Hierarchical tree toy example for CubBirds dataset.

7.3 Final Remarks

In this section, I will talk about my final remarks and conclusions. To make it easier, I will list them as points.

- **Meta-Learning is data dependant !** : It is not necessary to go for highly complicated losses and computations if your Dataset includes few number of classes or the targeted classes can be easily discriminated. Such as a Dataset of different animal species. In this scenario, SimpleShot is the option to go with. But if the targeted problem includes **few images per class and the inter-class variants are high** then Proxy-Anchor method is the choice.

- **Lite models** : I found that, It is not necessary to use very deep network if we are using complicated loss. If simple classification loss is utilized then a deep network (e.g. ResNet-101) is useful. Otherwise, the performance is not effected much.

- **Parameters Tuning** : within the same meta-loss you can have various performance according to the initial parameters used. A good start is to use Keras Tuner. Give possible ranges of parameters such as α or γ with a grid of 10 epochs. Then choose the best pool and increase the iterations. In addition to that, Initializing the proxies (for proxy-based losses) with the mean of the embedding of each class (obtained with pre-trained networks) will help to reach local-minimum faster. This idea is similar to transfer learning but without having the network structure limitation or the huge computation.

- **Training & Inference tricks** : Finally, the new publications are discussing the usage of different training and inference heuristics. Meaning, with the same method but with utilizing the L_2 normalization as an example you may boost the performance by a big margin. Centering the input with zero-mean, processing the images in different color space or applying various transformations as discussed earlier are possible solutions.

Fig. 14. Output examples for Caltech-256 dataset (section 6.3). On the left we see the query image an on the right, the top 4 matches (minimum distance).

7.4 Acknowledgment

Thanks to Prof. Aydin Alatan and his assistant, I was able to explore new topics within the field of Computer Vision. Seeing things from mathematical and probabilistic perspective was one of the important outcomes I learnt from the course.

8 References

1. Wikipedia. *Similarity learning — Wikipedia, The Free Encyclopedia* http://en.wikipedia.org/w/index.php?title=Similarity%20learning&oldid=988297689. [Online; accessed 31-December-2020]. 2020.
2. Kim, S., Kim, D., Cho, M. & Kwak, S. *Proxy Anchor Loss for Deep Metric Learning* 2020. arXiv: 2003.13911 [cs.CV].
3. Kaya, M. & Bilge, H. Ş. Deep metric learning: A survey. *Symmetry* **11,** 1066 (2019).
4. Ali, N. & Zafar, B. Caltech101 Image Dataset (Aug. 2018).
5. Leal-Taixé, L. & Canton-Ferrer, C. Learning by tracking: Siamese CNN for robust target association (Apr. 2016).
6. Wah, C., Branson, S., Welinder, P., Perona, P. & Belongie, S. *The Caltech-UCSD Birds-200-2011 Dataset* tech. rep. CNS-TR-2011-001 (2011).
7. 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).
8. Song, H. O., Xiang, Y., Jegelka, S. & Savarese, S. *Deep Metric Learning via Lifted Structured Feature Embedding* in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).
9. Liu, Z., Luo, P., Qiu, S., Wang, X. & Tang, X. *DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations* in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016).
10. Schroff, F., Kalenichenko, D. & Philbin, J. FaceNet: A Unified Embedding for Face Recognition and Clustering. *CoRR* **abs/1503.03832.** arXiv: 1503.03832 (2015).
11. Sohn, K. *Improved Deep Metric Learning with Multi-class N-pair Loss Objective* in *Advances in Neural Information Processing Systems* (eds Lee, D., Sugiyama, M., Luxburg, U., Guyon, I. & Garnett, R.) **29** (Curran Associates, Inc., 2016), 1857–1865.
12. Song, H. O., Xiang, Y., Jegelka, S. & Savarese, S. *Deep Metric Learning via Lifted Structured Feature Embedding* 2015. arXiv: 1511.06452 [cs.CV].
13. Movshovitz-Attias, Y., Toshev, A., Leung, T. K., Ioffe, S. & Singh, S. No Fuss Distance Metric Learning using Proxies. *CoRR* **abs/1703.07464.** arXiv: 1703.07464 (2017).
14. Ge, W., Huang, W., Dong, D. & Scott, M. R. *Deep Metric Learning with Hierarchical Triplet Loss* 2018. arXiv: 1810.06951 [cs.CV].
15. Wang, Y., Chao, W.-L., Weinberger, K. Q. & van der Maaten, L. *SimpleShot: Revisiting Nearest-Neighbor Classification for Few-Shot Learning* 2019. arXiv: 1911.04623 [cs.CV].