A brief study about Metric Learning and your improvements using HDML approach

Thales Eduardo Nazatto
UNICAMP - Institute of Computing
tenazatto@gmail.com

I. Introduction and objectives

Metric Learning is a Machine Learning approach that aims to estabilish similarities between objects according to your distance. With Deep Learning adoption in academic community studies, Metric Learning methods detect these similarities with more accuracy, and some applications are image retrieval [1], person re-identification [2] and geo-localization [3]. These studies, due to recent discovery of Deep Learning, are only on the beginning, and some improvements are proposed. One of these is Hardness-aware Deep Metric Learning (HDML) [4], who uses additional layers and loss functions to provide data augmentation and feature synthetization, improving Metric Learning results.

This study has the objective to see the Deep Metric Learning properties and your effects on a Convolutional Neural Network, trying to compare two of main loss functions and your HDML-enhanced counterparts. Section II will define the concepts about Metric Learning, your loss functions and HDML, section III will define the study methodology and how the developed program works, section IV will discuss about the experiments and, finally, details about conclusions, suggestions and future work at section V.

II. About Metric Learning

A. Main concepts and Loss functions

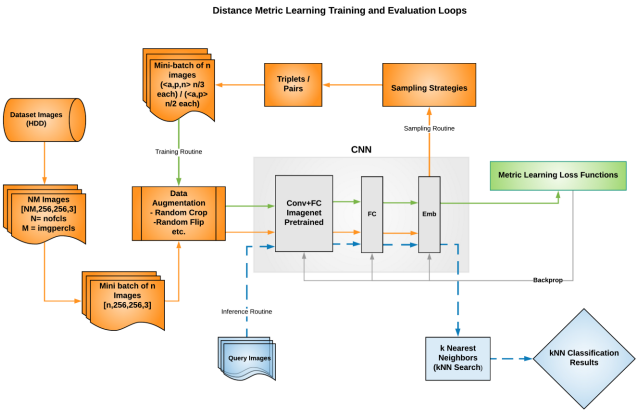
According to [5], Metric Learning is an approach based directly on a distance metric that aims to establish similarity or dissimilarity between objects. While metric learning aims to reduce the distance between similar objects, it also aims to increase the distance between dissimilar objects.

Conventional metric learning methods usually employ the Mahalanobis distance or kernel-based metric to characterize the linear and nonlinear intrinsic correlations among data points [4]. [5] says also that a linear transformation property on Mahalanobis distance makes Euclidean distance in the transformed space equal to it in original space for two samples.

However, despite Deep Metric Learning importing concepts and purpose, your approaches are completely different due to Neural Network properties. In a Deep Metric Learning approach, specified on Figure 1, a mini-batch of images are the input of a Convolutional Neural Network, a loss function is calculated after each training routine and serves as neural network backpropagation. A sampling routine can also improve the model using the samples as feedback to model itself.

In Deep Metric Learning, the main difference and effort is placed on the Loss function and embedding layer instead

Fig. 1. Training and inference routine for Deep Metric Learning



of CNN architecture. This occurs because the main objective of Metric Learning is dependable of these parts: With the embedding layer the Neural Network maps images in an embedding manifold space and with the Loss function explicitly push similar images together in embedding space and pull dissimilar images away from each other during the training phase [6]. The main Loss functions used are:

- **Contrastive Loss**: Encodes both the measures, similarity and dissimilarity, independently in a loss function. It directly addresses a distance between image features of two similar images, and the distance between two dissimilar image features is included in the loss if they are not separated by a distance margin, α.
- **Triplet Loss**: Takes in a triplet of deep features (x_{ia}, x_{ip}, x_{in}), where (x_{ia}, x_{ip}) have similar product labels and (x_{ia}, x_{ip}) have dissimilar product labels and tunes the network so that distance between anchor x_{ia} and positive x_{ip} is minor than the distance between anchor x_{ia} and negative x_{in} plus a margin α.
- **Lifted Structure Loss**: Minimizes smooth upper bound for stable network training, and your idea is to improve a mini-batch optimization using all $O(B^2)$ pairs, available in the batch, instead of $O(B)$ separate pairs.
- **N-Pair Loss**: Is similar to Lifted Structure loss in the sense that it recruits multiple negative product examples while generating loss term in a given mini-batch and does not suffer from slower convergence like triplet loss and contrastive loss, but it tries to optimize cosine similarity between a positive anchor and negative product samples in a probabilistic way.

- **Angular Loss**: Proposes to encode a third-order relation inside the triplet triangle in terms of an angle at the negative edge. It motivates a push for negative feature vector away from the positive cluster and drags the positive points closer to each other, and can be combined with another loss functions to boost the overall performance.
- **Divergence Loss**: Is a regularizing term, that can be added to another loss functions for joint-supervision. It increases the distance between features of an image learned by different ensemble modules.

B. Tuning and Improving: The HDML Framework proposal

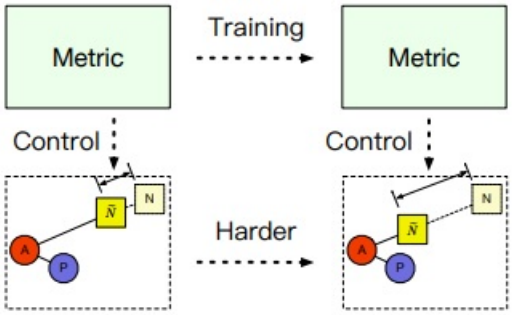
Hardness-aware Deep Metric Learning, presented on [4], is a framework that uses Deep Metric Learning and Hard Negative Mining, a technique that progressively selects false positive samples to will benefit training the most, to synthesize hardness-aware samples as complements to the original ones. It can be divided in 3 parts: Hardness-Aware Augmentation, Hardness-and-Label-Preserving Synthesis and Hardness-Aware Deep Metric Learning.

Hardness-Aware Augmentation, illustrated on Figure 2, gets negative pairs ($\mathbf{z}$,$\mathbf{z}^-$) and construct an augmented harder negative sample $\hat{\mathbf{z}}^-$ by linear interpolation, being represented as:

$$\hat{\mathbf{z}}^- = \begin{cases} \mathbf{z} + [e^{-\frac{\alpha}{J_{avg}}} d(\mathbf{z},\mathbf{z}^-) + (1 - e^{-\frac{\alpha}{J_{avg}}}) d^+] \frac{\mathbf{z}^- - \mathbf{z}}{d(\mathbf{z},\mathbf{z}^-)} \\ \text{if } d(\mathbf{z},\mathbf{z}^-) > d^+ \\ \mathbf{z}^- \\ \text{if } d(\mathbf{z},\mathbf{z}^-) \leqslant d^+ \end{cases} \quad (1)$$

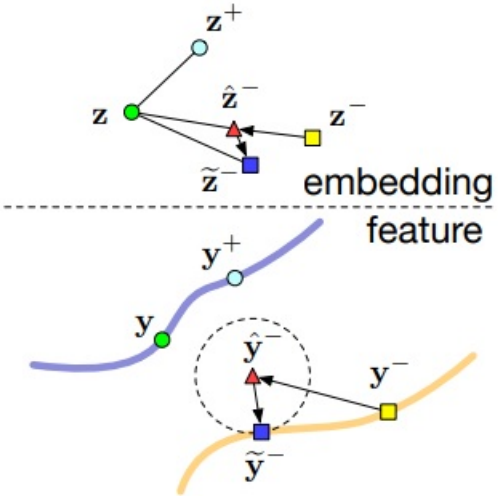
Where J_{avg} is the average metric loss over the last epoch, α is the pulling factor used to balance the scale of J_{avg}, and $d(\mathbf{z},\mathbf{z}^-) = ||\mathbf{z}^- - \mathbf{z}||^2$. This augmentation is made because in the early stages of training the embedding space does not have an accurate semantic structure, and the data could not be meaningful. As the training proceeds, the model is more tolerant, so harder and harder synthetics should be generated to keep the learning efficiency at a high level.

Fig. 2. Hardness-Aware Augmentation proposal



Hardness-and-Label-Preserving Synthesis, illustrated on Figure 3, gets these synthetic data and projects on the embedding space to guarantee that all $\hat{\mathbf{z}}^-$ augmented sample shares the same label with $\hat{\mathbf{z}}^-$. This is possible due to encoding the feature vector $\mathbf{y}$ generated on CNN to original embedding $\mathbf{z}$, doing the hardness-aware augmentation to sythesize augmented embedding $\hat{\mathbf{z}}$ and adding generator layers after augmentation phase who encodes maps $\mathbf{z}$ and $\hat{\mathbf{z}}$ to $\mathbf{y}$' and $\tilde{\mathbf{y}}$ respectively, and compute additional loss functions J_{recon}, J_{soft}, and J_{gen}, where $J_{recon} = ||\mathbf{y} - \mathbf{y}'||^2$ and J_{soft} is the softmax loss function. J_{gen} is a sum of J_{soft} multiplied by a pre-defined parameter λ and J_{recon}.

Fig. 3. Hardness-and-Label-Preserving Synthesis proposal

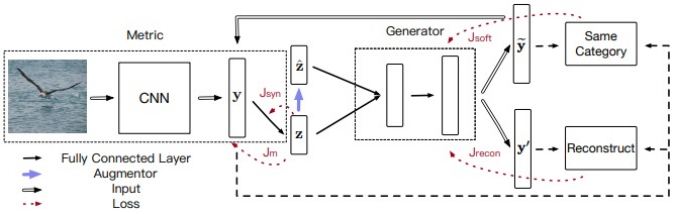


And Hardness-Aware Deep Metric Learning, illustrated on Figure 4, gets both parts described above to calculate a new loss between the embedding space and the synthetic samples to boost deep metric learning performance. For a specific loss J in metric learning, the objective function to train the metric is:

$$J_{metric} = e^{-\frac{\beta}{J_{gen}}} J_m + (1 - e^{-\frac{\beta}{J_{gen}}}) J_{syn} \quad (2)$$

Where β is a pre-defined parameter, J_m is the loss J over original samples, J_m is the loss J over synthetic samples. $e^{-\frac{\beta}{J_{gen}}}$ has a role to balance the weight between original and synthetic samples. In the first iterations, this weight tends to favor J_m, but throughout the process J_{gen} will change and tends to favor synthetic samples, because harder tuples are synthesized on hardness-aware augmentation phase to keep the high efficiency of learning.

Fig. 4. Hardness-Aware Deep Metric Learning proposal



III. Methodology

A. Program creation, execution and usability

Three programs based on code presented in [4] were created. The first one download the datasets to computer, the second one treat the information presented on datasets and put all data on a HDF5 Database file and the third get one of these database files and stream your data to a Deep Neural Network with Metric Learning loss functions. Some modifications are made to get a limited sample of classes in dataset creation. The datasets downloaded to use in experiments are Cars196, CUB-200-2011 and Stanford Online Products, which are widely-used in Deep Learning studies.

To execute the programs, *Python 3* and Linux are required. The programs can't run on Windows because one of libraries used in code (*Fuel*) can't be installed due to a broken dependency on this OS. The main libraries used in this code are:

- *H5py*: Used to read and write HDF5 files.
- *Fuel*: Used to determine Schemas for HDF5 files, and Data Streaming.
- *OpenCV*: Used to manipulate images presented on datasets.
- *Tensorflow*: Used to create Deep Neural Network. The version used was 1.15.2

The download programs are simply executed without any flag, but the dataset and Neural Network use flags to configure some execution parameters. The flags from the dataset program are:

- **-d**: The dataset used to create HDF5 file.
- **-n**: Number of classes sampled on created HDF5 file.

And the main flags used from the Neural Network program are:

- **-LossType**: The dataset used to create HDF5 file.
- **-dataSet**: The dataset HDF5 file used to training.
- **-batch_size**: Images mini-batch size.
- **-batch_per_epoch**: The number of batches per epoch.
- **-max_steps**: The maximum iterations on training.
- **-init_learning_rate**: Initial learning rate on training.
- **-regular_factor**: Weight decay factor.
- **-Apply_HDML**: Use HDML Loss functions if activated.

B. Input and Output data

Since three programs were created to do this process, the output of last executed program is the main input used in the next one. The datasets downloaded on the first program will be used in HDF5 file creation, and the HDF5 file will be used in Neural Network program. The output of the Neural Network program are summary logs updated according to configured batch per epoch size. These logs contains the measures shown at experiments section.

C. Code approaches

The download program wasn't modified at all in relation to [1] code, because it only downloads a zip file. The dataset creation program was improved on your architecture and gets a sample number of classes. All of three datasets has its own class that makes an inheritance to **DatasetIngestion** class, who has these main methods:

- **start**: Begin the creation process. It gets all classes from text/Matlab files, gets a sample of it, decompress downloaded file and creates the new HDF5 file using *Fuel* dataset template.
- **getClasses**: Method written by child class used to get classes from text/Matlab files.
- **decompress**: Method written by child class decompress downloaded file.
- **getTestSamples**: Method written by child class used to get random samples of classes presented on dataset.

The Neural Network program has been improved for better code readability. As Tensorflow starts the training on **tf.app.run()** method, all code before it is only configuration parameters. It was divided on these methods:

- **placeholders**: Raw input (images, labels, training and learning rate) as Tensorflow placeholders.
- **configClassifier**: Config Tensorflow classifier to calls GoogLeNet [7] model as CNN feature extractor with an additional layer as the embedding projector.
- **configLoss**: Config Tensorflow loss to do the configured main Metric Learning loss function.
- **configHDML**: If **Apply_HDML** flag is activated, config Tensorflow with additional HDML layers and loss functions.
- **configTrainSteps**: Config Tensorflow training iteration.
- **configTfSession**: Config Tensorflow session to update loss functions in each iteration and write results in summary logs.

Parameters presented on **FLAGS.py** and HDML calculation files hadn't been modified.

IV. Experiments and Discussion

A. Hardware and parameters

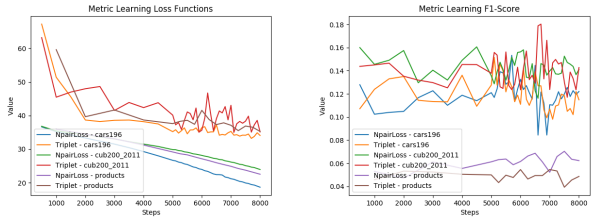
The hardware used to do the training was an Core i7 with 16 GB Memory, a SSD and 2 GB GeForce 940MX GPU. Approximately only 10% of all dataset are used on experiments (20 classes on Cars196 and CUB-200-2011 and 2300 classes on Stanford Online Products), and division between train and test are set to 50%-50% proportion. Cars196 and CUB-200-2011 used an image mini-batch size of 15 in Triplet function and 16 on N-pair function, an epoch batch of 500 iterations up to 5000 iterations and an epoch batch of 100 after that. Stanford Online Products, due to your additional classes, used an image mini-batch size of 18, an epoch batch of 1000 iterations up to 5000 iterations and an epoch batch of 250 after that. The initial learning rate was set as $7 * 10^{-5}$ and weight decay factor was set as $5 * 10^{-3}$

B. Results and discussion

The experiment on this study tries to compare two Metric Learning Loss functions (Triplet and N-Pair) to your HDML counterparts. However, unfortunately none of these functions

worked with HDML enhancements on hardware specified with any dataset and any parameter configuration. Using Triplet J_{soft} returns a NaN on your first iteration and affect the other losses on next iterations causing the model not work properly and be discarded right away. A NaN value occurs because of a division by zero problem or the value is too large, and even changing some softmax and pulling parameters this value persists during every execution. Using N-Pair with Cars196 and CUB-200-2011 samples, HDML can't work because the image mini-batch size should be a divisor of 128 to do the calculations and should be minor than the total number of classes presented on the dataset. Trying with Stanford Online Products samples, the maximum image mini-batch size that worked on this hardware was 18 and more than this value results on GPU memory overflow.

Because of that reasons above, the experiment was simplified to compare differences in Triplet and N-Pair loss behavior. The first point noted was about the loss functions: The N-Pair loss function had lower values and uniform behavior during Neural Network epoch iterations, and a first hypothesis about that measure is that the N-Pair loss will achieve better results. Looking about the F1-score, who measures a balance between precision and recall, it was proven that the N-Pair loss was better on 3 datasets tested, as could be seen on Figure 5.



(a) Loss functions results (b) F1 Scores

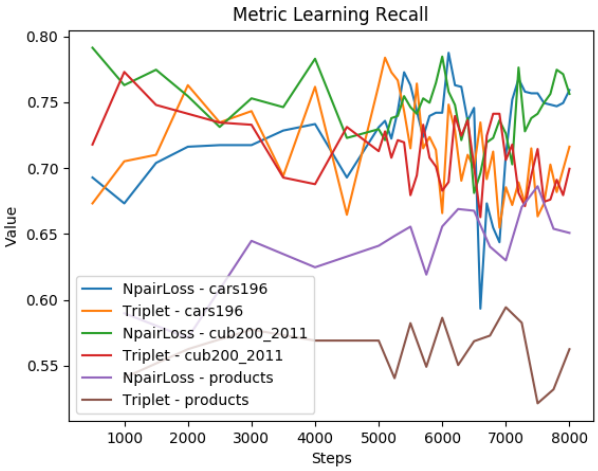
Fig. 5. F1 Score and Loss functions results

This was far from expected because, according to [6], the N-Pair loss function achieve better results on datasets with a larger number of classes. But, unfortunately and despite the small dataset, the low F1 score results worry whether this is the right measure to evaluate. Since HDML synthetize some elements to create more meaningful elements to measure, Metric Learning could prioritize relevancy and the next hypothesis is about the recall measure. As seen on Figure 6, the N-Pair loss also acheieve better results on 3 datasets tested, confirming the hypothesis for that study parameters.

V. Conclusions and Future Work

The first conclusion we've seen in this study is that any improvement made in the loss function, computationally speaking, is high costly. It demands a great amount of memory due to using big datasets to get satisfactory results, and the new loss functions, layers and constraints presented on HDML maximize that need. We also could suppose that the time spent

Fig. 6. Recall results



to process all dataset will increase due to additional calculus used in these new loss functions and additional layers.

The second conclusion is, depepending for the chosen loss function, the behavior of metrics could be completely different. The N-Pair loss had better performance in this situation, but could be less efficient on another approach. Different conditions should be tested to confirm this, like larger datasets, more Neural Network architectures and iterations and another loss function approaches.

As future work, it's possible to suggest a port of HDML using Tensorflow 2. The reasons about this is because the main HDML framework uses the outdated Tensorflow 1.x, and a port from Tensorflow 2 open a possibility to use Google Colab and/or Kaggle features, which is updated to use that library version and your free versions provides a Nvidia Tesla hardware with at least 4 to 10 times more floating point performance and 8 times more memory than the hardware tested [8]–[11]. A Python code who uses libraries whose dependencies are compatible with Linux and Windows would also be well regarded. As this study objective is only to see the Deep Metric Learning properties and your effects, no suggestion will be listed about the Neural Network architecture.

References

[1] Song, H. O; Xiang, Y; Jegelka, S; Savarese, S. Deep metric learning via lifted structured feature embedding. URL: https://arxiv.org/pdf/1511.06452.pdf

[2] Shi, H; Yang, Y; Zhu, X; Liao, S; Lei, Z; Zheng, W; Li, S. Z. Embedding deep metric for person re-identification: A study against large variations. URL: https://arxiv.org/pdf/1611.00137.pdf

[3] Vo, N. N; Hays, J. Localizing and orienting street views using overhead imagery. URL: https://arxiv.org/pdf/1608.00161.pdf

[4] Zheng, W; Chen, Z; Lu, J; Zhou, J. Hardness-Aware Deep Metric Learning. URL: https://arxiv.org/pdf/1903.05503.pdf

[5] Kaya, M. Deep Metric Learning: A Survey. URL: https://www.mdpi.com/2073-8994/11/9/1066/htm

[6] Patel, J. Digging Deeper into Metric Learning with Loss Functions. URL: https://towardsdatascience.com/metric-learning-loss-functions-5b67b3da99a5

[7] Szegedy, C; Liu, W; Jia, Y; Sermanet, P; Reed, S. E; Anguelov, D; Erhan, D; Vanhoucke, V; Rabinovich, A. Going deeper with convolutions. https://arxiv.org/pdf/1409.4842.pdf
[8] Hale, J. Kaggle vs. Colab Faceoff — Which Free GPU Provider is Tops? URL: https://towardsdatascience.com/kaggle-vs-colab-faceoff-which-free-gpu-provider-is-tops-d4f0cd625029
[9] Nvidia GeForce 940MX Specs. URL: https://www.techpowerup.com/gpu-specs/geforce-940mx.c2797
[10] Nvidia Tesla K80 Specs. URL: https://www.techpowerup.com/gpu-specs/tesla-k80.c2616
[11] Nvidia Tesla P100 Specs. URL: https://www.techpowerup.com/gpu-specs/tesla-p100-pcie-16-gb.c2888