Experimenting Loss Function Strategies on the Re-Identification Task

Michele Tessari
University of Trento
Computer Science
michele.tessari@studenti.unitn.it

Leonardo Lovato
University of Trento
Artificial Intelligence Systems
leonardo.lovato@studenti.unitn.it

Abstract

This paper describes our solution for Univerity of Trento's 2020/2021 Deep Learning course final project. We will present two neural networks to classify people attributes and identify them starting from a well know dataset. In the first section we describe both tasks, then we present our approach to the solution and finally we report the experiments we made and the results we achieved.

1. Introduction

For the final exam of the 2020/2021 Deep Learning course offered from the University of Trento, we students were asked to build one or more neural networks to classify people attributes and identify them using a version of the Market-1501 dataset [6]. The provided data consists of images of people taken from 6 different surveillance cameras and a list of 27 attributes (such as age, gender, clothes color, ecc...) for each person.

For the classification task we aim at generating correct labels of the 27 different attributes annotated in the dataset, this istance of the problem reflects both the multi-label and multi-class classification cases.

Re-identification task aims at finding the identity of a query person from a large collection. Specifically, in our case we have a test directory and a query directory and we are asked to retrieve, for each query image, all the images (from the test directory) depicting the same person. Additionally, the test directory contains junk images that should never be matched to a person.

To implement our solution we used the PyTorch [2] framework in a Google Colab [1] notebook.

2. Classification task

Driven by the availability of models in PyTorch and the advantages of using an ImageNet pre-trained models, we decided to fine-tune one of them for our task.

2.1. ImageNet and fine-tuning

ImageNet is a very large dataset composed of millions of images depicting more than 20.000 different categories of objects where each image is hand-annotated and comes with bounding boxes and labels for the objects in it.

The PyTorch library offers a nice set of neural network models, starting from AlexNet up to the most recent ones. Most of them comes eventually pre-trained on ImageNet, meaning that the weights of the network are the result of the training on the mentioned dataset. This is a great features due to the fact that training a model on a generic dataset make it possible to acquire "good" weights which allows for a better and faster results on the specific dataset and task.

2.2. Model implementation

We tried 3 different architectures to test the accuracy and training time, we also tried to train the networks from scratch to see the properties of fine-tuning. The first model is ResNet18, the second is ResNext50[5] while the last one is EfficientNet [3], a recent model architecture with similar results compared to ResNet but a lower number of parameters. For each of them we made adjustment to the last layer as the images from ImageNet and the ones from our dataset have different dimensions and the number of possible output classes are not the same.

2.3. Dataset preparation

We splitted the 12.989 available annotated images in training and validation sets in a 70/30 manner, respectively. To avoid having images of certain people in both sets, at first we ordered the images by person id and took them up to the 70% of the total to create the training set and the remaining for the validation. Lately, we decided to randomly take the ids of 70% of the depicted people (the are in total 751 different person) and use all the related images to create the training set, the remaining were used for the validation set. Furthermore, we applied data-augmentation (a well known technique used to increase the dimension of the dataset) trying 2 different ways to augment the data: the first one consist of taking an image and add an augmented version of it with a 15ish% probability. For the second one, instead, we counted the number of times each attribute appeared in the training set and augmented those images in which the depicted attributes were the minority, in this way we tried to make the dataset more balanced so that the network could see enough examples (for each label) and hopefully be able to better generalize. Finally, for some images none of the label regarding the color of the upper or lower clothes were positive, therefore we added two new labels (”upmulti” and ”downmulti”) to include those cases.

2.4. Training details

We trained each different architecture for 25 epochs using the Adam optimizer, the cross entropy loss for the age attribute and binary cross entropy loss for all the others using accuracy as metric. We implemented an early stopping technique to stop the training of the network if no improvements over the accuracy were made for 10 consequent epochs. When improving the accuracy, instead, we save the model directly to Google Drive to avoid losing the data due to the limit of GPU usage in Google Colab. For each epoch we computed the loss, the accuracy for each label and the average accuracy, storing those information using Tensorboard.

2.5. Main modules

In the delivered classification notebook we can find 4 classes and few different functions:

- **SurveillanceDataset:** inherited from the torch.utils.data Dataset class, we use it to preprocess the images and make them ready for the training
- **MyModel:** the network class that prepares the model given the input configuration
- **MultiLabelLoss:** a simple class used to apply the correct loss given the label
- **ComputeAccuracies:** another simple class used to compute each accuracy and the average
- **get_data:** a function used to get the training and validation DataLoader used to train the network
- **main:** wrapper function that retrieve the data and train the network for all the specified epochs

3. Re-identification task

For the re-identification task we decided to use the backbone of the best model from the classification task to extract good features of the images. Consequently, we fine-tuned this model in order to produce features that are more distant between images with different person ids. We tested 3 different losses and we tried the re-ranking [7] technique to improve the final results.

3.1. Dataset preparation

We prepared the dataset by grouping the images with the same id and adding new images with data augmentation if the total number of images with some id did not reach a minimum threshold.

During training we build the training batch by randomly sampling a fixed number ($n\_select = 12$) of images from the training set, creating a minibatch for each person id, and obtaining a batch of the shape: $(batch\_size, n\_select, 3, 128, 64)$. Then the batch is resized to $(batch\_size \cdot n\_select, 3, 128, 64)$ to be compatible with the model, that will gives a tensor of $(batch\_size \cdot n\_select, n\_of\_features)$. Finally we resize the tensor to $(batch\_size, n\_select, n\_of\_features)$ in order to use it for the loss function.

3.2. Training details

We trained the network for 50 epochs with Adam optimizer and compute the mean average precision (mAp) on the validation set with the function provided by the instructors. This time we divided train and validation sets with an 80% ratio taking into consideration that the task is hardest than the classification and that there is not the problem of unbalanced labels. Moreover, also here we saved the model when we detect an improvement of the mAp.

We did not directly take into consideration the problem of junk images as that the features obtained from the classification model should be far away. Anyway, it could be possible to pass the images to a person detector model to skip it if no person was detected. Another idea could be to add junk images in the classification dataset and add an attribute to be predicted in that case.

3.3. Main modules

Similar to the classification task, we used 2 classes to load the data (ReidTrainDataset, ReidValidDataset) and 2 classes for classification/reidentification models (MyModel, ResNetIdentification). Furthermore, we have the following:

- **TripletLoss, CentroidTripletLoss, CustomLoss, MultiLoss:** classes implementing the different losses
- **train_reid, test_reid:** functions used for training/testing the network
- **main:** wrapper function to run the training

Finally, at the end of the file the are classes and functions related to the generation of the file for the query (ImageDataset, generate_dist_matrix, write_prediction_file, show_prediction)

3.4. Proposed approach

We fine tuned the model comparing 3 different loss functions:

3.4.1 Triplet Loss

It compares a reference input (anchor A) with another with the same person id (positive P) and with another with different person id (negative N). It minimizes the distance between the anchor and the positive and it maximizes it between the anchor and the negative, separating positives and negatives with a margin α:

$$\mathcal{L}_{triplet} = [||f(A) - f(P)||_2^2 - ||f(A) - f(N)||_2^2 + \alpha]_+$$

The anchor and the positive are elements of the same minibatch. On the other hand, the negative is chosen within the rest of the batch depending on the strategy used. We tested the training of the model with the following strategies:

- **nearest-id-based:** for a minibatch of features belonging to the same person id, we computed the distance with the remaining features of the batch choosing the minibatch with the nearest mean distance of the features as negative.
- **nearest-feature-based:** similar to the *nearest-id-based* but choosing the n_select nearest features as negative.
- **random id:** a random minibatch with different id of the anchor is chosen as negative.
- **random features:** a random n_select number of features with different id of the anchor are chosen as negative.
- **2 phase based:** *random id* or *random features* strategy during the first 20% of the total epochs, then *nearest-id-based* or *nearest-feature-based* strategy.
- **incremental probability:** for each minibatch, we used *random id* or *random features* with probability $p = 1 - \frac{epoch}{epoch_{max}}$, *nearest-id-based* or *nearest-feature-based* otherwise.

3.4.2 Centroid Loss

It is a variation of the triplet loss [4], where the positives and the negatives are the average of features of the same person ids (G_k):

$$\mathcal{L}_{centroid} = [||f(A) - c_P||_2^2 - ||f(A) - c_N||_2^2 + \alpha_c]_+$$
$$\text{where:} \quad c_k = \frac{1}{|G_k|} \sum_{x_i \in G_k} f(x_i)$$

Also here we tried to use *nearest-id-based*, *random id*, *2 phase based* and *incremental probability* strategies, with the only difference that is used the mean of the selected minibatch as negative centroid.

3.4.3 Custom Loss

We design this new function that maximizes the variance between the means of features with the same person id and it minimizes the variance of the minibatches. With S_k a set of features of images with the person id k:

$$\mathcal{L}_{custom} = \sum_{k \in [0,n]} \mathrm{Var}_k(S_k) - \mathrm{Var}\left(\begin{pmatrix} \mathrm{Mean}_0(S_0) \\ \mathrm{Mean}_1(S_1) \\ ... \\ \mathrm{Mean}_n(S_n) \end{pmatrix} \right)$$

4. Experiments and results

We conducted various experiment to see how each property we used was affecting the results. For each experiment we trained the network 3 times and took the average of the result to draw the conclusions.

4.1. Classification Results

In table 1 we can see the average accuracy and training time for each model mentioned in Sec. 2.2. Each model was trained for a maximum of 25 epochs and we computed the average accuracy according to the results of the best epoch. We can see how ResNet18 gave the best outcome and we can also observe the effects of pre-training on images.

For all the following experiments we used this configuration and train the network for 50 epochs.

In table 2 we show the effects of data-augmentation, it is interesting to see how the number of augmented images drastically impact the results, specifically we observed that augmenting around 15%-18% of the total images was the best practice for this specific case.

	0%	15%	20%	80%
Avg. Accuracy	88.12%	89.41%	88.69%	87.40%

Table 2. Varying accuracy over percentage of augmented images

Lastly, in table 3 we have a controversial result with respect to the previous experiment: while for the previous experiment we divided the dataset in the sorted way, for this last we splitted it in the random way add augmented the set by 16%. It appears that data-augmentation is sort-of ineffective regarding the accuracy but we observed that it took the network a lower number of epochs to converge, in fact the early stopping technique always triggered while training with the augmented data whilst all the 50 epochs were computed in the normal version.

	ResNet18	ResNet18 (pretrained)	ResNext50	EfficentNet
Avg. Accuracy	85.28%	88.38%	87.83%	87.71%
Training time	36m25s	35m43s	1h28m13s	33m46s

Table 1. Accuracies and training times (25 epochs)

	Without	Random	Balanced
Avg. Accuracy	88.37%	88.10%	88.56%

Table 3. Varying accuracy over augmenting technique

4.2. Reidentification Results

We have seen that the triplet loss is the one where we get the best mAp on the validation set (Fig. 1), reaching our best pick of 68.6% mAp.

The custom loss is able to train the model to create better features but it decreases rapidly the performances after ~ 7 epochs. We think this happens because it's possible that the loss is increasing the distance between (for example) 2 macro groups of images but reducing the distance inside the 2 groups (the variance between the mean of the features would still increase but overall many ids would end up to stay near). For future experiments, we think this problem could be solved by maximizing the distance between the means instead of maximizing the variance.

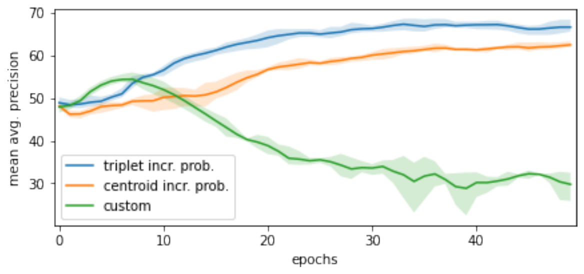

Figure 1. Loss comparisons (mean of 3 runs)

4.2.1 Loss strategies results

The best strategy seems to be the incremental probability feature based (Fig. 2), even if the performances are similar with the 2 phase feature based. An interesting point can be seen by comparing directly the nearest feature approach with the incremental probability: the first approach get high mAp faster than the second but the second reach better results around 25 epochs, when it has a 50% probability to choose the nearest feature. It is even more interesting comparing these 2 methods with the 2 phase feature based: the mAp is stucked around its initial value ($\sim 50\%$) during the "random feature" phase (from epoch 0 to 9) but it reaches better values w.r.t the nearest only method during the "nearest feature" phase. This suggest that using random features helps to push them into better positions during the initial phase of the training even if the mAp on the validation set does not change. This also suggest that after enough epochs, the 2 full random approaches could bring even better results of the other approaches but more experiments should be conducted in order to answer this hypothesis.

Finally, we applied the re-ranking technique to improve the features given by the model. Since its effectiveness is proportional to the accuracy of the model, we computed the re-ranking step only on the validation set of the best model and on the query/test features. From the 68.6% we obtained a mAp of **76.8%**.

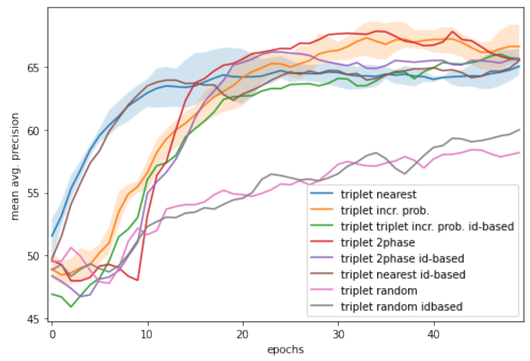

Figure 2. Triplet loss techniques comparisons

5. Conclusions

We presented our solution for the classification and re-identification tasks on the Market-1501 dataset. We achieved overall good results while testing different approaches, moreover we acknowledge the advantages in using techniques we learned during the course by conducting few ablation studies. Working on this project was very interesting and challenging and gave us a great knowledge over the PyTorch framework and the classification task applied to images. Some improvements that could be done may involve using a stronger backbone or applying different losses.

References

[1] Google colab. https://colab.research.google.com/. last access 21/01/2022. 1

[2] Pytorch. https://pytorch.org/. last access 21/01/2022. 1

[3] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019. 1

[4] Mikolaj Wieczorek, Barbara Rychalska, and Jacek Dabrowski. On the unreasonable effectiveness of centroids in image retrieval. *CoRR*, abs/2104.13643, 2021. 3

[5] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. 1

[6] Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. Scalable person re-identification: A benchmark. In *Computer Vision, IEEE International Conference on*, 2015. 1

[7] Zhun Zhong, Liang Zheng, Donglin Cao, and Shaozi Li. Re-ranking person re-identification with k-reciprocal encoding. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3652–3661, 2017. 2