Deep Learning for Autonomous Vehicles

Course project - Milestone 2 - Report

Students :

Sylvain Pietropaolo
Yann Martinson
André Gomes

May 22, 2020

Contents

Introduction

The project detailed in this report tackles the implementation of a re-identification code framework for the MSMT17 dataset.

The re-identification task consists in an algorithm able to identify an object and to tell if another image is the same object or not. In this project, the different objects will be some persons that the algorithm should distinguish and re-identify on another image.

For this purpose, an existing code repository [1] will be adapted for the *MSMT17* dataset and different configurations for the training will be tested to analyse their impact on the results. First, a brief overview of the method used is presented. Then, the impact of the changes in parameters is discussed. Finally, the results of the best trained model are shown.

The workload has been distributed as following among the team members. The search of an existing method and its understanding is done by all the members together. Then André tried different backbone architecture, Yann did the droprate tests and Sylvain dealt with the stride parameter. Finally, the parameters for the best model are chosen according to these results by mutual agreements of all the team members.

1 Method overview

This section aims at exposing briefly the different steps involved in the implementation of the chosen re-identification method. Overall, the algorithm is trained to extract the relevant features that will be used to distinguish the different persons one among the others. As every machine learning algorithm, it is basically constituted by 2 steps : training and evaluation. They are both based on a convolutional neural network (CNN) architecture.

For the training, the output of this CNN feeds a fully connected layer to classify the images and to update the weights according to the known label of the image. Then, the evaluation consists in using the CNN model from the training and feed it with a reference image from a *query* set. This image is the persons to re-identify in a set of images called the *gallery*. Then, the evaluation task is a comparison of the images of the gallery to the the query in order to detect if the person is the same or not. The evaluation is the same for all the re-identification methods but there is various way to do the training.

1.1 Training

As there is many features that can be extracted from an image, the algorithm needs to select the relevant ones allowing to distinguish the persons between them. That's what the training is for.

Basically, the trained weights are filters parameters of the CNN. They should be tune such that the network will extract the right features allowing to distinguish the different persons. However, the CNN is not able to do that alone as there is nothing that says if the extracted features are relevant or not. That's why fully connected layers are added at the output of the CNN. Depending on the feature at the input, the last layers will classify the images and the weight will be updated accordingly to have the best classification accuracy.

There is different ways to feed the fully connected layer depending on the training method used. Indeed, the input can either be directly the features but also a feature comparison

[1] Available under MIT license at: `https://github.com/layumi/Person_reID_baseline_pytorch`

between different images. For the first case, the last layer will classify the input images and update weights accordingly. In the second case, the label used to update the weights is not anymore the class of the input images but if the inputs images are the same person or not. The first approach is implemented on the models used here. The input will be an image and the training is done according to a classical classification, not by an image comparison. This choice was made for simplicity regarding the time available to implement such a method and to train the model from a large dataset. Indeed, a CNN followed by a fully connected is a classical classification method which is easier to use and implement.

1.2 Evaluation

After the training, the CNN model is trained to extract the relevant features allowing to distinguish people. Once these features are extracted, the goal is then to use them smartly to do the re-identification. The evaluation dataset is split into 2 set of images : *queries* and *gallery*. The aim is then to re-identify the queries inside the gallery. For this purpose, the features of each query are compared to the features of all the gallery images. Depending on how they match, each image on the gallery is given a score. A mAP metric is then computed, relating how the images in the gallery that have the same label as the one in query are positioned in the scoring list. Finally all the mAP of the queries are averaged to lead to a global mAP, which gives an indication of the accuracy of the model for the re-identification task.

2 Parameter changes

2.1 CNN backbones

The CNN backbones tested are either based on Resnet50 or Densenet121. In both the cases, a pre-trained model on ImageNet is loaded from Pytorch to fasten up the computation. In order to compare results and to find a best fitting one, the following models are tested:

ResNet-50
Resnet50 is the default model of the implemented code. It has been proven very effective while being trained on the COCO dataset. It is based on a residual layers architecture. It is also a backbone that acts as base for other models.

PCB
PCB is another model based on ResNet-50. PCB stands for Part-based Convolutional Baseline and has been performing very effectively in person retrieval[2]. The part-based convolutional baseline consists in reshaping the backbone network (here ResNet-50) with slight modifications. The structure before the original global average pooling layer is maintained exactly the same as the backbone model. The difference is that the pooling layer and what follows are removed and replaced by the PCB approach. Finally, a classifier is implemented with a fully-connected layer.

[2] `https://arxiv.org/abs/1711.09349`

Densenet-121
Densenet121 is another implemented model in the code repository used. It is based on dense blocks, that differentiate the weight propagation from a standard CNN.

Results
All these models are tested with the same hyper-parameters in order to evaluate the impact of the backbone choice on the results. The goal is to select one backbone with which hyper-parameters will be tuned to improve the results. This approach is not the most recommended to test different architectures. There is a high probability that the best set of hyper-parameters for the backbones are not the same. However, for a project with a short timeline, this approach gives a quick overview of what should be expected from the different backbones. Additionally, an overlook on how these different architectures have performed with other datasets can further confirm that the selected model is indeed the most appropriate for the task at hand. The Table 1 summarizes the common parameters used.

Batch size	Learning rate	Optimizer	Drop rate	Stride
32	0.01	SGD	0.5	2

Table 1: Training parameters from the backbone tests

The Table 2 shows the obtained results.

Backbone	ResNet-50	PCB	Densenet-121
mAP	0.26	0.36	0.29

Table 2: Training results from the backbone tests

The PCB backbone performs better than the others by around 7% with the selected hyper-parameters. This CNN architecture is therefore selected as best model for further tuning.

2.2 Batch sizes

The batch size states the number of images used in order to compute the loss and back-propagate the gradient at each iteration during one epoch.

As discussed in the previous Milestone report, there seems to be a consensus and studies tend to prove it that increasing to larger batches leads to decrease quality of the model, with less ability to generalize. The GPU available allows to use a maximal batch size of 64. In addition to the default 32, a model is therefore trained with this limit of 64 to have a quick confirmation of the discussed tendency. The impact of the batch size is tested with the parameters shown in Table 3. The Table 4 summarizes the obtained results.

Backbone	Learning rate	Optimizer	Drop rate	Stride
PCB	0.01	SGD	0.5	2

Table 3: Training parameters from the batch size tests

Batch size	32	64
mAP	0.36	0.34

Table 4: Training results from the batchsize tests

The results indicate that a batch size of 32 leads indeed to better results than 64.

2.3 Strides

The CNN is basically a way to learn the filters to convolve to an image in order to extract relevant features used afterwards according to the algorithm purpose. The convolution of a filter with an image involves different parameters impacting the results like the size of the filters, the padding around the initial image and many others. Among these parameter there is the stride which is the number of pixels the center of the filter is moving after each convolution. Different values are tried for this parameter as it may allow to detect more precise features by having smaller strides. The precision of the feature may have a non negligible impact on the ability of the algorithm to re-identify the different person. For these tests, the Table 5 summarizes the parameters used for the stride tests.

Backbone	Batch size	Learning rate	Optimizer	Drop rate
ResNet-50	32	0.01	SGD	0.5

Table 5: Training parameters from the stride tests

The Table 6 summarizes the obtained results.

Strides	1	2	4
mAP	0.28	0.27	0.26

Table 6: Training results from the stride tests

As expected, having a lower stride seems to improve the accuracy of the model. Indeed, as the extracted features are more precise, the model may use features that are more different from an image to another, which allows to have better classification.

As a drawback, having a lower stride increases significantly the computational time as it augments the size of the output feature map leaving the CNN.

Finally, a stride of 2 is used for the best model (PCB) as it was already computed before the stride tests (ResNet-50). Another model with a stride of 1 will be too long to train in our timeline for an improvement of only 1%.

2.4 Dropout and learning rate

The dropout rate is used in the training process in order to reduce the risks of over-fitting. The dropout rate value is a probability, which therefore lies between 0 and 1, that drops at each iterations a neuron and therefore its corresponding connections with the probability. By using this technique, at each pass the network is reduced to a smaller portion of the full one and update the weights as these neurons exist. On the next iteration, the dropout rate is applied again which results in a new network for the iteration.

Backbone	Batch size	Optimizer	Stride
PCB	32	SGD	2

Table 7: Training parameters from the dropout tests

The dropout rate comparison fell partially through due to a server interruption, leading the model trained with a 0.8 stopped at the epoch 39/59. The results are indicated as follows.

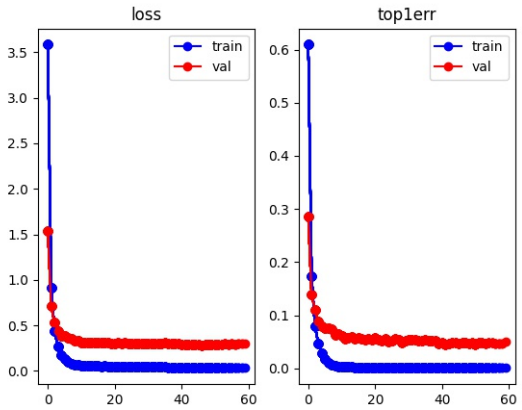
Learning rate	0.01	0.005	0.005
Dropout	0.5	0.5	0.8
mAP	0.36	0.32	0.24

Table 8: Training results from the dropout tests

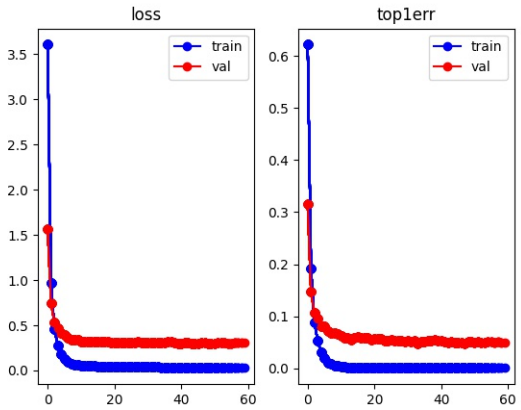
Although the second model stopped early, looking at the overall progression showed that a dropout rate of 0.8 gives an important decrease in the mAP of the model. At the same time, decreasing the learning rate from 0.01 (default) to 0.005 (used for dropout rate tests) also decreased the quality of the results.

2.5 Tests convergence and accuracy

In order to see if the model converges, the loss and top1-error curves are computed at each epoch for both the training and the validation set. They are plotted in the following figures, with the loss and top 1 error on the training set in blue and the same metrics on the validation set in red. One see that both tend to a constant after around 20 epochs; the model has thus converged.

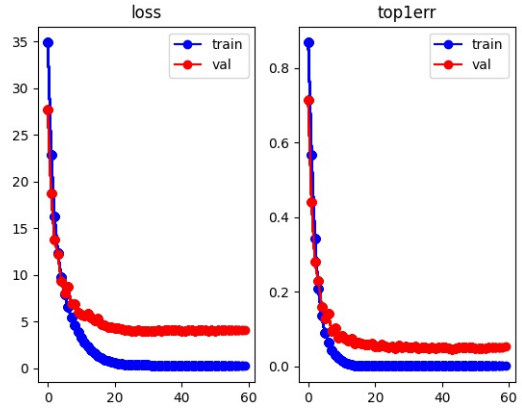


(a) Densetnet 121, mAP: 0.29

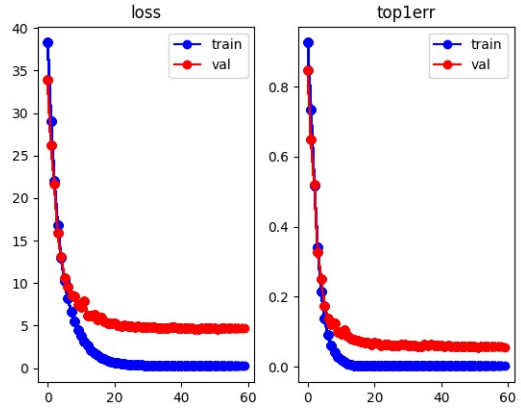


(b) Resnet50, mAP: 0.26

Figure 1: Model comparison, lr: 0.01

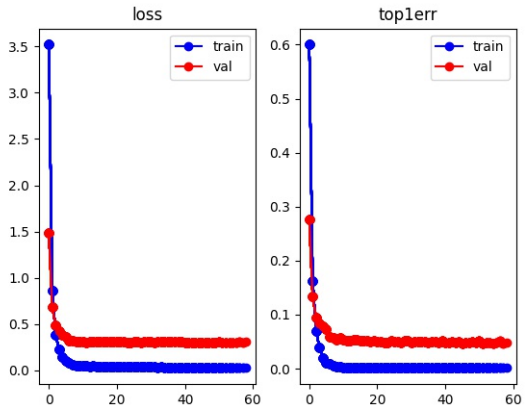


(a) batchsize: 32, , mAP: 0.36

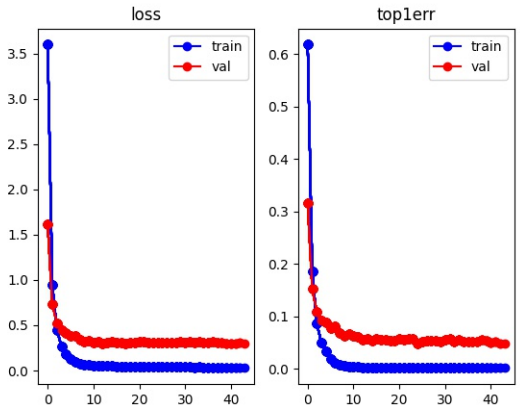


(b) batchsize: 64, mAP: 0.34

Figure 2: PCB batchsize comparison, lr: 0.01

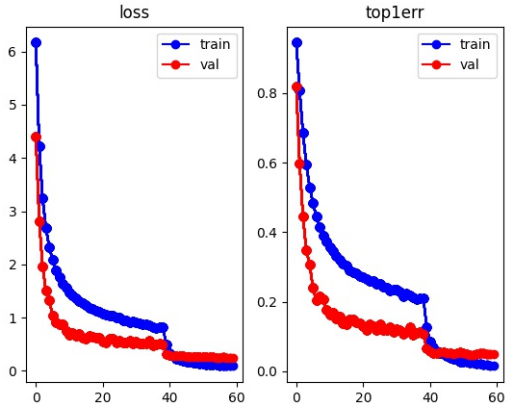


(a) stride: 1, mAP: 0.28

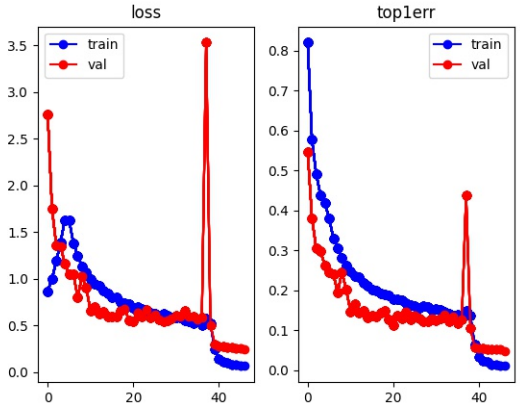


(b) stride: 4, mAP: 0.26

Figure 3: PCB stride comparison, lr: 0.01



(a) dropout: 0.5, mAP = 0.32



(b) dropout: 0.8, epoch 39, mAP:0.24

Figure 4: PCB dropout comparison, lr: 0.005

3 Best model

Finally, all the previous tests lead to choose a model with the parameters shown in the Table 9.

Backbone	Batch size	Learning rate	Optimizer	Drop rate	Stride
PCB	32	0.01	SGD	0.5	2

Table 9: Training parameters from the stride tests

This model gives a mAP of 36% on the re-identification task of the MSMT17 dataset. One should keep in mind that the parameters selected here have been chosen by doing tests on each parameter independently of the others. But a hyper-parameter optimal for a certain configuration may not be optimal for another one. In order to be more precise, a grid search should be performed. It has not been done here in order to explore more diverse hyper-parameters in the available time.

Conclusion

In the implementation of a re-identification algorithm, the training is mainly a classical CNN followed by fully connected layers, but the testing is a quite different. Indeed, instead of feeding the layer with an image and compare the classification with the label to compute the errors, the re-identification task uses a distance between the features extracted from 2 images in order to be able to distinguish 2 persons. Therefore, the algorithm does not really classify but outputs the images which are more likely to be the query. The user decides afterward what is the criterion used to decide if 2 images represent the same person. It could be a simple threshold on the distance between the features extracted by the CNN or something more complex, taking account for example of how previous queries have related to that same image of the gallery set.

The major limitation faced is the strategy used to find the best configuration. Indeed, the one chosen is not optimal as it do not take into account the dependency of a parameters to the others. Thus, there is probably different configurations that would lead to better results. But in a short timeline this approach seems to be the most straightforward to find a relatively good set of hyper-parameters. In addition, it is also difficult to select a suitable method easy to implement among the dense documentation available. Thus, the method which seems the quicker to adapt and tune while already providing good results with other datasets is selected, again for time consideration. It might be necessary to try other methods for the training like the Triplet loss methods, the verification loss method or the contrastive loss method in order to increase the accuracy.