People Classification and Re-Identification

Deep Learning Course Project Report

Davide Piva
2nd year CS Master Degree @ University of Trento
davide.piva-1@studenti.unitn.it

Davide Bulbarelli
2nd year CS Master Degree @ University of Trento
davide.bulbarelli@studenti.unitn.it

Abstract

In this document are reported the solutions that have been implemented in order to solve the two requested tasks with an acceptable accuracy and loss. In the first chapter is presented the approach used to solve the first task, with a particular attention to the technical solution that has been implemented and how the training and the evaluation phases have been addressed. Then, in the second chapter is presented the approach used to solve the second task exploiting the already learned model during the first task with a particular attention to the technical solution adopted and how the training and evaluation phases have been tackled. At the end of each chapter are attached the Tensorboard [12] graphs that show the obtained results from the point of view of loss function and accuracy.

1 CLASSIFICATION TASK

The first task of this project is to design, train and evaluate a multi-class classification model that must be able to predict a set of attributes for each pedestrian image given in input. These attributes are listed here below:

- gender
- hair length
- sleeve length
- length of lower-body clothing
- type of lower-body clothing
- wearing hat
- carrying backpack
- carrying bag
- carrying handbag
- age
- color of upper-body clothing
- color of lower-body clothing

The results produced using the test samples are attached in the submission and named as "classification_test.csv".

1.1 Technical solution

In this section, is described the implemented solution used to solve the above-mentioned task. First of all, in Section 1.1.1 is detailed the data preprocessing method applied before the overall training phase and, secondly, in Section 1.1.2 is motivated the reason why ResNet50 [10] has been chosen with respect to other neural networks that are available in PyTorch.

1.1.1 Data Preprocessing. Before proceeding with the training phase the entire dataset has been preprocessed in two different stages described here below:

(1) first of all, to each image is associated an one-hot encoding vector that represents the target prediction. In particular, this vector has size 56 that is the number of different labels multiplied by the number of value each label can assume. Hence, at training time the neural network will receive an image as input and the one-hot encoding vector in order to compute the loss and update its weights;
(2) in the second place, the full training set is split in 80% training set and 20% validation set in order to evaluate the learned model at each epoch.

1.1.2 Fine-tuning ResNet50. In order to solve the task, several neural networks available in the PyTorch framework have been trained and evaluated. Then, the results have been compared in order to pick the model with the best performance. In particular, using the same preprocessing methods detailed in Section 1.1.1, the following neural networks have been trained and tested: AlexNet, GoogLeNet, ResNet34, Resnet50, ResNet152, WideResNet and ResNext. Each network has been instantiated with the pre-trained version and the final output layer has been replaced with a sequential layer composed by the following pieces:

- linear layer: N input features, 1024 output features (where N is the number of features produced in output by the previous layer);
- linear layer: 1024 input features, 512 output features;
- linear layer: 512 input features, 56 output features;
- sigmoid

The network whose performance revealed to be the best is Wide ResNet with a training accuracy of 88,12 and a validation accuracy of 86,47. The main problem with this network is that the overall model is very heavy (more than 200MB) and each forward pass takes several seconds. Given the objective of using the same trained network in both project's tasks and knowing that for the second task Wide ResNet will be too slow, a tradeoff between accuracy and model complexity has been performed: ResNet50 has been identified as the most suitable neural network for the project's tasks. This choice has been taken also given the considerations detailed in Section 2.1.2.

1.2 Training and evaluation

In order to train and evaluate the ResNet50 neural network, the optimizer and the cost function need to be decided. The optimizer that has been chosen for the network training is Adam [2]: this choice has been taken empirically after training the same network with the same data split and cost function using Adam and then SGD [11]. Although SGD has produced better generalization performance, the difference between the one obtained with Adam was very small and Adam is known to converge faster with respect to SGD. For these reasons, SGD was discarded. In the second place, the most suitable cost function for this particular task has been identifiend in the Binary Cross Entropy function [3] since it is a commonly used loss function for multi-class classification tasks.

Furthermore, some data augmentation methods have been implemented in order to enlarge the available data. The main problem of using these techniques is that they have not produced any advantage to the overall accuracy of the model. Moreover, for ResNet50 and ResNext the overall accuracy has dropped. Due to these reasons and given the aim to use a larger batch size having as limit the eight gigabytes of GPU memory in Google Colab, data augmentation has not been used during the training of ResNet50.

In Figure 1 are shown the obtained performance once ResNet50 has been trained for five epochs using the above-mentioned tools. This model has reached a training accuracy of 85,51 and a validation accuracy of 85,27.

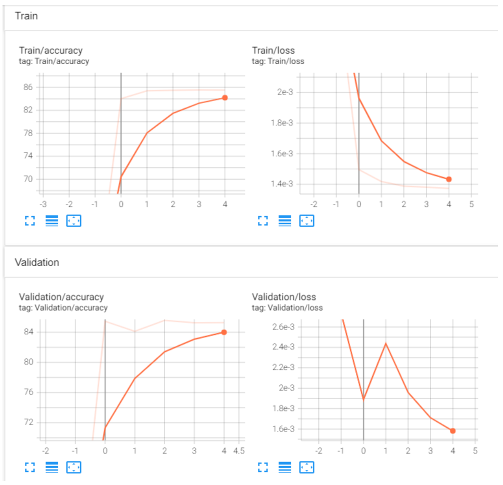


Figure 1: Train and validation loss-accuracy of ResNet50

The number of epochs to perform has been chosen after training the network for 20 epochs, as seen in Figure 2, after 5 epochs the performance starts to degrade on the validation set, while on training set the network starts overfitting.

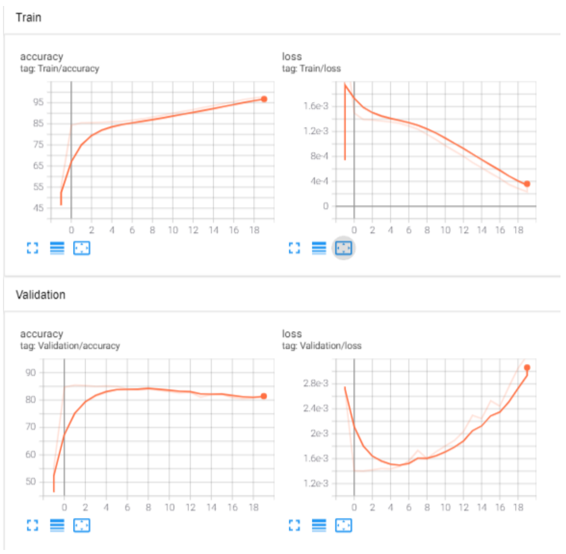


Figure 2: Train and validation loss-accuracy of ResNet50 after 20 epochs

1.3 Source code

The code of the classification task is available in the following Github repository: DeepLearningProject.

In particular the files relevant for this task are:

- DeepLearningProject_classification_learning.ipynb: Python notebook used to train the ResNet50 for classification
- ModelEvaluation.ipynb: Python notebook used to evaluate the trained ResNet50 for classification and output the results file "classification_test.csv".

2 PERSON RE-IDENTIFICATION TASK

The second task of this project is to desing, train and evaluate a neural network able to perform person re-identification. More formally, for every image in input the network is asked to produce in output a list of images that it believes correspond to the same person identity depicted in the input image. As will be explained in Section 2.1.2, the goal is to reuse the model learned during the first task and adapt it in order to perform well also in this one. The results produced using the query and test samples are attached in the submission and named as "reid_test.txt".

2.1 Technical solution

In this section, it is described the implemented solution used to solve the above-mentioned task. In Section 1.1.1 is explained the data preprocessing procedure applied before the overall training phase. Furthermore, in Section 2.1.2 is explained in detail the designed learning model.

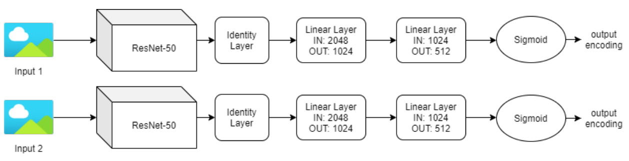
2.1.1 Data Preprocessing. Before proceeding with the training phase of the model, the full training set has been split in 80% training set and 20% validation set. This operation has been performed such that all the images of the same person are in the same set. This technique has been implemented because otherwise the obtained validation performance will be higher than the test performance. The split has been implemented exploiting the person identification number written in the file name.

2.1.2 Siamese Network Implementation. The proposed solution implements a Siamese Neural Network that bases its business on the ResNet50 model learned for the first task. In other words, the approach for the training phase is to compute the forward pass of two images of the training set, compare the obtained results using the choosen loss function and finally update the weights. This design choice has been made because it has been demonstrated in several works that the usage of ResNet50 in a siamese-network approach can reach high performances both in speed and accuracy [6] [7] [8] [9].

More formally, the siamese network is composed by the following layers:

- the ResNet50 model learned in order to solve the first task where the latest linear layer has been replaced with an identity layer;
- linear layer: 2048 input features, 1024 output features;
- linear layer: 1024 input features, 512 output features;
- sigmoid

A visual representation of the architecture can be seen in Figure 3



Input 1
ResNet-50
Identity Layer
Linear Layer IN: 2048 OUT: 1024
Linear Layer IN: 1024 OUT: 512
Sigmoid
output encoding
Input 2
ResNet-50
Identity Layer
Linear Layer IN: 2048 OUT: 1024
Linear Layer IN: 1024 OUT: 512
Sigmoid
output encoding

Figure 3: Siamese Architecture

The defined forward method will take in input two images to pass in each branch of the network, then it will return the inferred tensor as output.

2.2 Training and evaluation

In order to train and evaluate the Siamese Neural Network based on ResNet50, the optimizer and the cost function need to be defined. As in the first task, the optimizer that has been chosen is Adam [2]: in order to decide the best optimizer have been tested also SGD [11] and Adagrad [1] but, even if the one that has produced the best result was SGD, as described also in Section 1.2, Adam produces almost the same result but it converges faster. Secondly, as cost function has been implemented the Constrastive Loss [4] with margin fixed to 2. This loss function computes the euclidean distance between the two outputs and then returns a loss value. This value will be low when positive samples are encoded to similar (closer) representations and negative examples are encoded to different (farther) representations.

Furthermore, also for this task have been implemented data augmentation techniques but, as reported for the previous task, they have not produced any valuable improvement in the accuracy. Due to this reason and with the constraint of 8 GB of GPU memory in Google Colab, these techniques have not been used in the final training of the siamese network.

In Figure 4 are shown the gained performances once the Siamese Network has been trained for 15 epochs using the abovementioned tools, obtaining a final loss of 0.00407 on the training part and a loss of 0.00219 on the validation part. Furthermore, the Siamese network has reached a mAP score [5] of 0.0964701270592074.

The number of epochs to perform has been chosen after training the network for 50 epochs, in fact, as shown in Figure 5, between epoch 10 and 20 the loss starts to oscillate and degrade both on training and validation sets.

2.3 Source code

The code of the re-identification task is available in the following Github repository: DeepLearningProject.

In particular the files relevant for this task are:

- DeepLearningProject_person_reid.ipynb: Python notebook used to train the Siamese Network based on the previously trained ResNet50.
- DeepLearningProject_person_reid_evaluation.ipynb: Python notebook used to evaluate the trained Siamese Network and find the optimal threshold value used to compute the query-test association file.
- PersonReID_TensorExtractor.ipynb: Python notebook used to produce the query-test association file. This notebook uses the trained Siamese Network along with the found optimal threshold. The notebook offers the possibility to compute all the output tensors and save them. This operation will save an high amount of time when computing the "reid_test.txt" file for different thresholds.

REFERENCES

[1] adagrad [n.d.]. *Adagrad optimizer.* Retrieved August 26, 2021 from https://pytorch.org/docs/stable/generated/torch.optim.Adagrad.html#torch.optim.Adagrad

[2] adam [n.d.]. *PyTorch Adam optimizer.* Retrieved August 26, 2021 from https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#torch.optim.Adam

[3] bceloss [n.d.]. *PyTorch BCE Loss.* Retrieved August 26, 2021 from https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

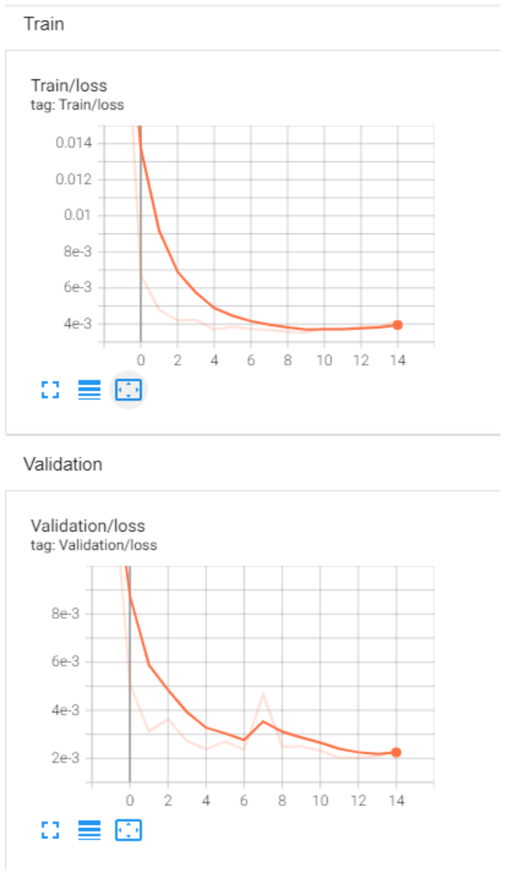


Figure 4: Train and validation loss of Siamese Network

[4] contrastive-loss [n.d.]. *Contrastive loss function.* Retrieved August 26, 2021 from http://yann.lecun.com/exdb/publis/pdf/hadsell-chopra-lecun-06.pdf

[5] map [n.d.]. *Mean Average Precision score.* Retrieved August 26, 2021 from https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

[6] reid1 [n.d.]. *Person re-identification paper.* Retrieved August 26, 2021 from https://ieeexplore.ieee.org/abstract/document/9191079

[7] reid2 [n.d.]. *Person re-identification paper.* Retrieved August 26, 2021 from https://openaccess.thecvf.com/content_ICCV_2017/papers/Chung_A_Two_Stream_ICCV_2017_paper.pdf

[8] reid3 [n.d.]. *Person re-identification paper.* Retrieved August 26, 2021 from https://arxiv.org/abs/1811.07487

[9] reid4 [n.d.]. *Person re-identification paper.* Retrieved August 26, 2021 from https://arxiv.org/pdf/2104.13643v1.pdf

[10] resnet50 [n.d.]. *PyTorch ResNet.* Retrieved August 26, 2021 from https://pytorch.org/hub/pytorch_vision_resnet/

[11] sgd [n.d.]. *PyTorch SGD optimizer.* Retrieved August 26, 2021 from https://pytorch.org/docs/stable/generated/torch.optim.SGD.html#torch.optim.SGD

[12] tensorboard [n.d.]. *Tensorboard.* Retrieved August 26, 2021 from https://www.tensorflow.org/tensorboard

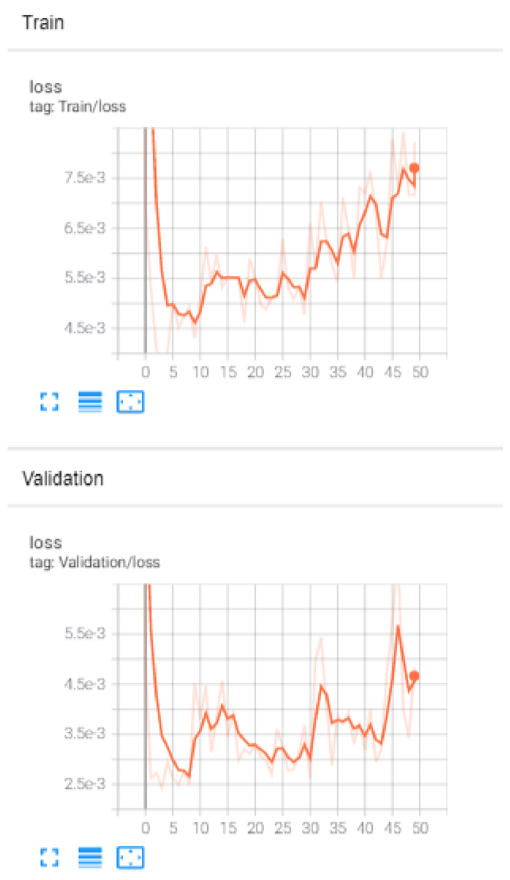


Figure 5: Train and validation loss of Siamese Network after 50 epochs