Artificial Neural Networks & Deep Learning
Challenge 1

Denaldo Lapi and Fabio Raffaeli

Team: fabio_dena

June 22, 2022

1 Introduction

In this document we report the development of our CNNs by dividing the explanation in the two main approaches used: a completely custom model made from scratch and a transfer learning model based on a pre-trained one.

A detailed description of the dataset and the choices we made about data loading and data preparation are provided in the notebook "dataLoading".

2 Model from scratch

The first operation we did was splitting the dataset between train and validation (80% train and 20% validation) by preserving the balance among the classes in the two subsets (see "dataLoading" notebook for the details).

We started by designing a very trivial CNN: we did this on purpose in order to get a starting base from which to apply new techniques and be able to sequentially improve the model according to the obtained results. This very first model was composed by 5 convolutional layers (with an increasing number of filters in each layer going from 16 to 256) and 2 fully connected layers (512 and 256 neurons), and no regularization of any kind was applied. The chosen loss function, since this is a multiclass classification problem, was the categorical cross-entropy, while the optimizer was Adam's, with a learning rate of 0.001. We did not apply early stopping. This model overfitted very quickly after few epochs and had a very poor performance on both validation and test. Still, we used it as a baseline for our subsequent experiments.

Then, we applied early stopping based on the validation loss in order to prevent the model from overfitting, and dropout for regularization. The performance on the validation set was decent but on the test set it was still poor. We tried different configurations for the network by changing the number and size of the layers, the learning rate, the batch size and also the dropout value but we only got minor improvements.

The wide distance between validation and test performances of the models we tried suggested us that there must have been a considerable difference on the dataset we used for training and validation and the test one, not available to us. This found a possible explanation in the fact that the training dataset is highly unbalanced.

To face the problem, we first tried to apply corrective weights to the classes. The weights were computed in an inversely proportional fashion with respect to the number of samples of the considered class and applied in the training phase to balance the relevance given to each class (see notebook "dataLoading" for the details of this procedure).

This approach did not work and, if anything, it worsened the performance of the models. Another attempt was the replacement of the categorical cross-entropy with the focal cross-entropy function (implemented in keras as *SigmoidFocalCrossEntropy*), which allows to differently weight the samples of the various classes. This did not improve the results.

Thus we switched to the application of oversampling, consisting of duplicating samples from the less represented classes until they match in size the most represented one(s). We re-splitted our dataset by using oversampling only over the training set as explained in the notebook "dataLoading". This method brought us the first notable improvement in both validation an test performances, thus we kept using it in all the following experiments.

We have to specify that the drawback of this approach was the massive increase in the size of the dataset, which in turn led to much slowlier trainings of the CNNs.

We observed a second major improvement in performance after the introduction of data augmentation, which allowed to obtain more robust neural networks. Indeed, data augmentation increased the variety of images in terms of exposure, brightness, angle and distorsion. It also further mitigated the problem of the unbalanced dataset, which was only partially solved by oversampling since many of the images were duplicates, thus did not provide enough variety.

We decided the type and intensity of the transformations to apply by trying to match the potential test set, being cautious not to exaggerate. At this point we were using a dropout factor of 0.4 in order to appropriately handle the large dataset and limit the increasing risk of overfitting.

The following focal step in the improvement of the model was the addition of batch normalization layers after each of the convolutional and fully connected layers (before the application of the activation function). This achieved us an increase of 0.2 in the accuracy on the test set.

Given the good results brought by data augmentation, we tried a heavier version of it by adding new types of transformations and making them stronger. This modification granted us another improvement and allowed to achieve our best result with the custom model of 85.47% of accuracy on the first phase test set and 84.71% on the second phase test set. The loss and accuracy curves of this model can be seen in figure 1. The detailed structure of the final model is described in the "notebookFinal".

We add that in the comparison of models, since the original dataset is unbalanced, accuracy is not very informative, even after the application of oversampling. For this reason we used other metrics like precision, recall and F1 score.

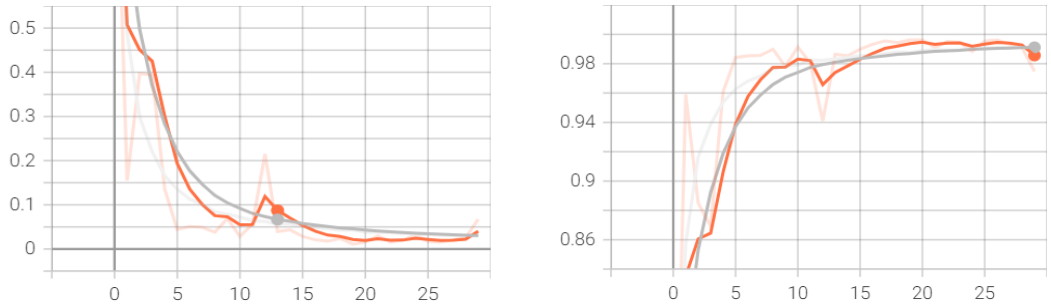

Figure 1: Loss (left) and accuracy (right) in train (grey) and validation (orange) of the model from scratch

3 Transfer Learning model

After getting very good results from the model built from scratch, we decided to move to the use of transfer learning. There are various pre-trained models available on the internet that can be used for this purpose. The ones we tried are VGG16, InceptionV3, InceptionResnetV2 and Xception. We remind that we always kept using oversampling and data augmentation in this phase of the project, since we had already proved their utility.

For each of the aforementioned CNNs we performed the same experiment: we used them as feature extractors and placed a flattening layer followed by a fully connected layer of 512 neurons on top. We then trained the obtained models by freezing the first part and letting only the classifier layers learn (with a learning rate of 0.001). Among the models realized in this way, the best performing one was the one based on Xception, therefore we kept it for the following experiments.

Since the flattening layer at the end of the feature extractor contributed to an extremely large number of trainable parameters and large training times, we replaced it with a global average pooling layer, which also helped to limit overfitting. Then we made other experiments in order to tune the learning rate and the structure of the classifier head, whose results brought us to a model with a learning rate of 10^{-4} and the output dense layer of 14 units immediately after the global average pooling, without any other layer in between. This new configuration allowed to increase a better the accuracy on the test set.

3.1 Fine Tuning

Once we identified the Xception-based model as the best to start from, we applied fine tuning to it by unfreezing some of the layers of the pre-trained model and retraining it with a lower learning rate. We tried different values of the learning rate and different unfreezing points for the network.

We obtained the best performance from a model with a learning rate of 10^{-5} and the Xception layers unfreezed starting from the 126th. The test accuracy was of 92.08% on the second phase test set. The loss and accuracy curves of this model can be seen in figure 2.

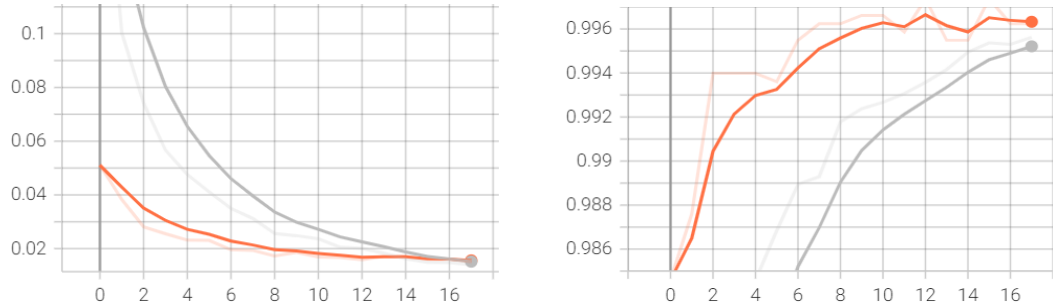

Figure 2: Loss (left) and accuracy (right) in train (grey) and validation (orange) of the transfer learning model