Artificial Neural Networks and Deep Learning Image Classification Homework

Viola Renne, Alessandro Rossi, Lea Zancani

November 27, 2022

Initial phase We immediately realized that, given the data scarcity, especially for the specie one, one of the main problems would be the data itself. Initially, in fact, using the basic model seen in the laboratory and making some optimization changes, we realized that the accuracy and loss values that we obtained on our validation set were much better than those obtained on CodaLab. With an initial scare, we started to improve and refine our model more and more, but still getting poor results on CodaLab, until we realized that the problem was the data. In fact, the division carried out by ImageDataGenerator, took only the first tot% of the data present in the class and, for many classes, those images were not at all representative of the whole class. As a result, we decided to split them automatically using splitfolder, but then we modified them slightly by hand to obtain a validation set as characteristic as possible of the whole class. Although we finally realized that the problem was associated with data pre-processing (and therefore still related to the data itself), we are pleased to have done this initial step because the results we got locally during the whole development phase were very representative of those obtained on CodaLab. At this point, after the initial fear had passed, we decided to start from the basic model seen in the laboratory and to add the refinements one by one.

Handmade CNN

We started trying to use the code from the lectures as a reference to build a first simple model: 5 convolutional layers with Relu activation function and maxpooling and two fully connected layers (one dense with 512 neurons with Relu activation function and one for the output with 8 neurons and Softmax activation).

Splitting the training, validation and test set The first thing we noticed was that we did not dispose of large amount of data, hence we decided to change the original division of the data between training and validation set: we used an internal test set to make some experiments and switched from the initial partition of 75% training, 15% validation and 10% test to 80%, 10%, 10% respectively. We used another partition into 80%, 15%, 5% in parallel, to perform a double check on the improvements obtained with the following steps. With these divisions we reached a local accuracy of 0.5, that we used as a reference to improve our model.

Data augmentation and model modification The first significative improvement was reached performing data augmentation: without changing any other parameters, after several experiments, this allowed us to get to an internal accuracy of 0.63 We then tried to modify the model, and initially noticed we reached better performances keeping the 2 dense layers and lowering the convolutional layers to 4. Eventually we realized the best performances were reached with the combination of 5 convolutional layers and 3 dense layers, which we kept for the final model.

Class weights At this point, we decided to manually flip the images in the training set. This way, we obtained a larger training set to feed to the neural network, after performing the regular automatic data augmentation on it. As for what concerns the imbalances between the number of images in the classes, we dealt with the problem weighting them. We tried to use different functions to compute the weights, as using the reciprocal of the number of images in the class or a logarithmic function.

Optimization We then added batch normalization layers after every non-linear layer of the model and dropout layers to avoid overfitting. We added a GAP layer, which led to some improvement. On the other hand, we also did some experiments with the regularizer, which we eventually decided not to use. We then switched from Relu to Leaky Relu activation function, and doing some research on the subject we discovered there is an open discussion on whether it is more useful to put the batch normalization layer before or after the non linear layer. In our case, after trying both combinations, we decided to put first the batch normalization layer and then the Leaky ReLU.
Thanks to all these changes, we reached an accuracy of 83% in the develop phase, which we were quite satisfied with, and 80% on the final phase.

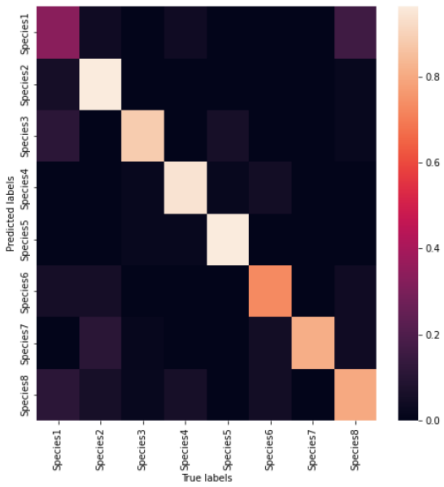


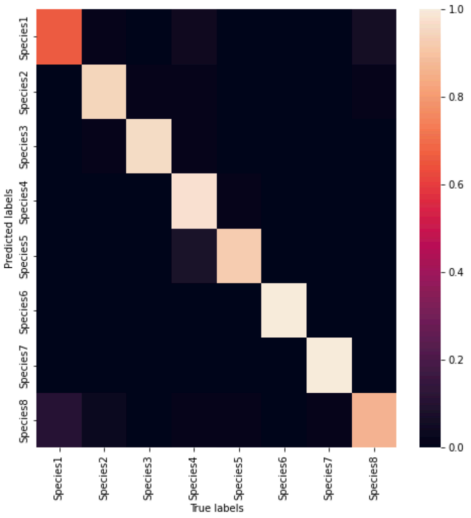
Figure 1: Handmade CNN confusion matrix

Transfer learning and Fine tuning

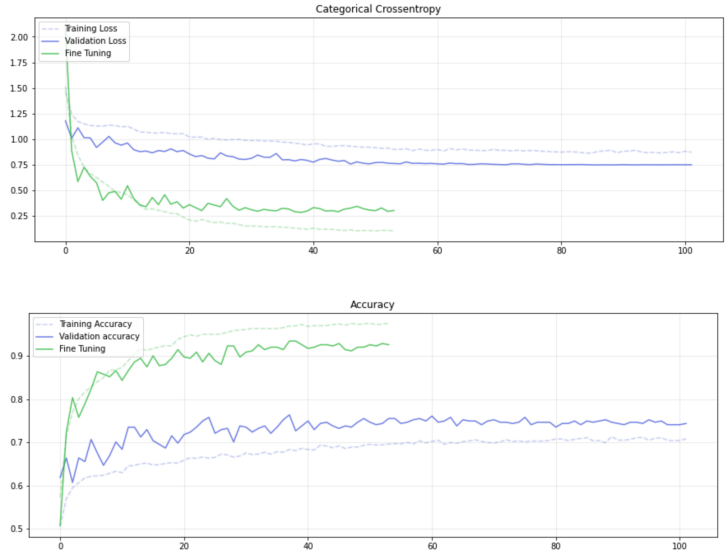
The next step was to try to use transfer learning.

Transfer learning In particular, we started using the VGG-16 weights for the convolutional layers, to which we added 2 dense layers. To fully exploit the supernet potential, we also used the VGG-16 preprocessing function. This way we achieved a local accuracy of 74%, which was quite good but way lower than the results we reached with the CNN we previously built without the supernet. We also tried to use VGG-19 and Xception, but did not manage to get above 73% local accuracy. So, we decided to focus on fine-tuning instead.

Fine tuning We started again with VGG-16 and tried to freeze different numbers of layers, reaching the best performance with 9 frozen layers. We also exploited the 'Reduce learning rate on plateau' method, to lower the learning rate when the learning starts slowing down. This allowed us to solve the problem of tuning the learning rate hyperparameter, which we realized has to be treated with particular care when dealing with supernets. With this model we reached an accuracy of 87%. We tried to push it a little bit further by resizing the images to the dimension used to train VGG, but couldn't obtain any improvement.
We then tried several other supernets as VGG-19 and InceptionV3, but couldn't beat the score we already reached with VGG-16.
The first net we found that gave us better results was DenseNet201: at first it reached a score of 87% as VGG-16, but resizing the images to 244x244x3 (the dimension used to train the model) we reached a score of 89%. Similar results were reached using Xception, which we used for one of the final submissions in the second phase, confirming the score of 89%.

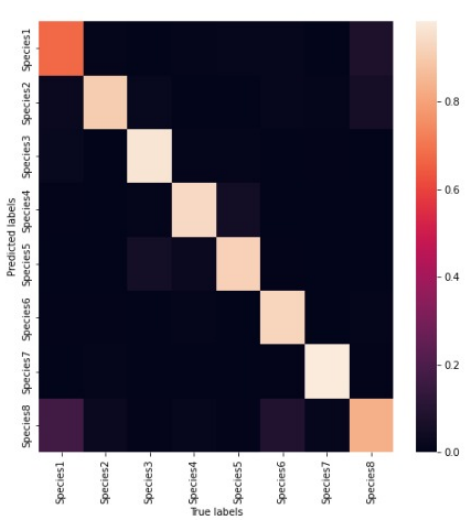


(a) Xception confusion matrix

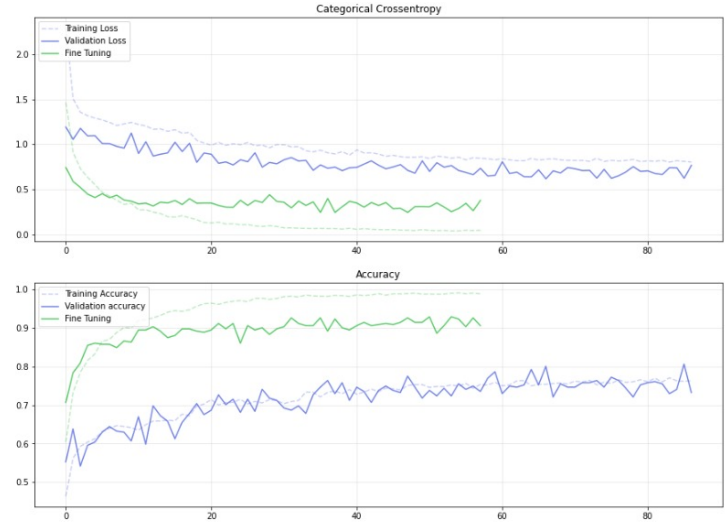


(b) Xception training

We finally switched to the EfficientNet models, and in particular we focused on EfficientNetB4 as our supernet, since it was the best compromise between performance and complexity among the different available EfficientNet versions. Following the same approach we used with DenseNet201, we resized the images to the dimensions used to train the supernet (380x380x3) before the training. This allowed us to reach a local accuracy of 0.79% with transfer learning, and a score of 0.92% with fine tuning on CodaLab. Since this was the best performing model in the first phase, we used it also for the second one, and decided to submit it in two different versions trained with different batch sizes. The final best score in the second phase was 0.915%, which although being a little bit lower than the one we got in the first phase, still made us proud of our work.



(a) EfficientNet confusion matrix



(b) EfficientNet training