AN2DL - Competition 1 - Report

Bergonzoli - Bonfanti - Giovannini

1. Introduction

Our case study is an Image Classification problem, in which we have to build a model to recognize different types of leaves. The dataset is made of 17758 images, belonging to 14 labels. The number of pictures in each label is largely different, indeed tomates are almost five times the others.

2.1. Solution: CNN

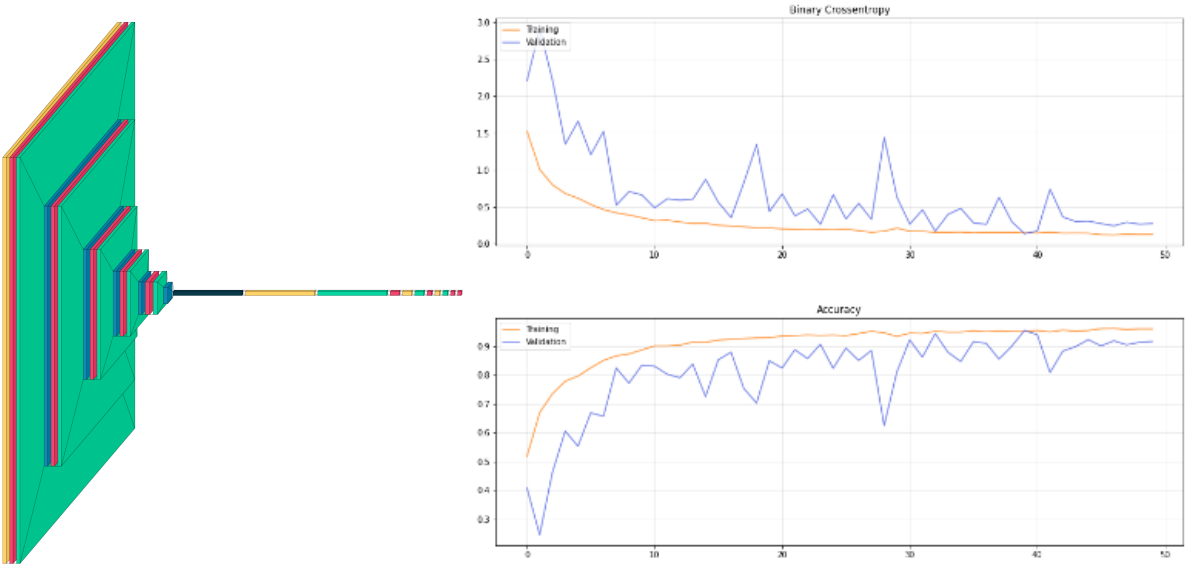
We started with a trivial model, just to really understand the problem. To have the input of the CNN between 0 and 1, we did the rescale in the ImageDataGenerator and then we proceeded with small changes, like adding or cutting some layers, and decreasing the size of the input images.

However, we realized that the dataset was a huge limit for our implementation, so we introduced the data augmentation, which gave us more various images to work on, using geometric transformation such as rotation, flip and zoom. We noticed that, in the models with data augmentation, we reached in the test set at least an increase of 30% in the mean accuracy value.

Knowing that there exist different types of pooling layer, we tried to apply them to our model. We changed the size of the *MaxPooling2D* and used also the *AveragePooling2D*, but in the end the best choice was still the *MaxPooling2D* with size (2,2).

Another important step was adding the *BatchNormalization* in every layer, which normalizes data over the batch, obtaining a regularization effect.

At this point of our analysis we focused on the fully connected layers and we tried two ways to improve the CNN. First, we ran a model with 1024 units in the dense classifier layer, but the number of parameters became huge and not adequate for the type of problem (it reached 17milions of parameters). So we switched to adding more dense layers, with a smaller number of units. This was actually a good choice, which gave us the best score in the test set, until this point.



2.2. Solution: Transfer Learning, Fine Tuning

We approached the problem in a different way: the transfer learning.

We started implementing the network using VGG16, which we'd already seen during lectures, keeping the structure of the fully connected layer of the best CNN trained before.

We wondered if the images in input were already handled enough, thanks to the preprocess associated with the VGG16 network, so we decided to remove the *BatchNormalization* layer, leaving only the *preprocess_input* one.

To control the oscillations of the validation loss and the validation accuracy, the learning rate was decreased to 1e-5. Moreover, we added the fine tuning method and in this way we observed a significant improvement in the mean accuracy of the model.

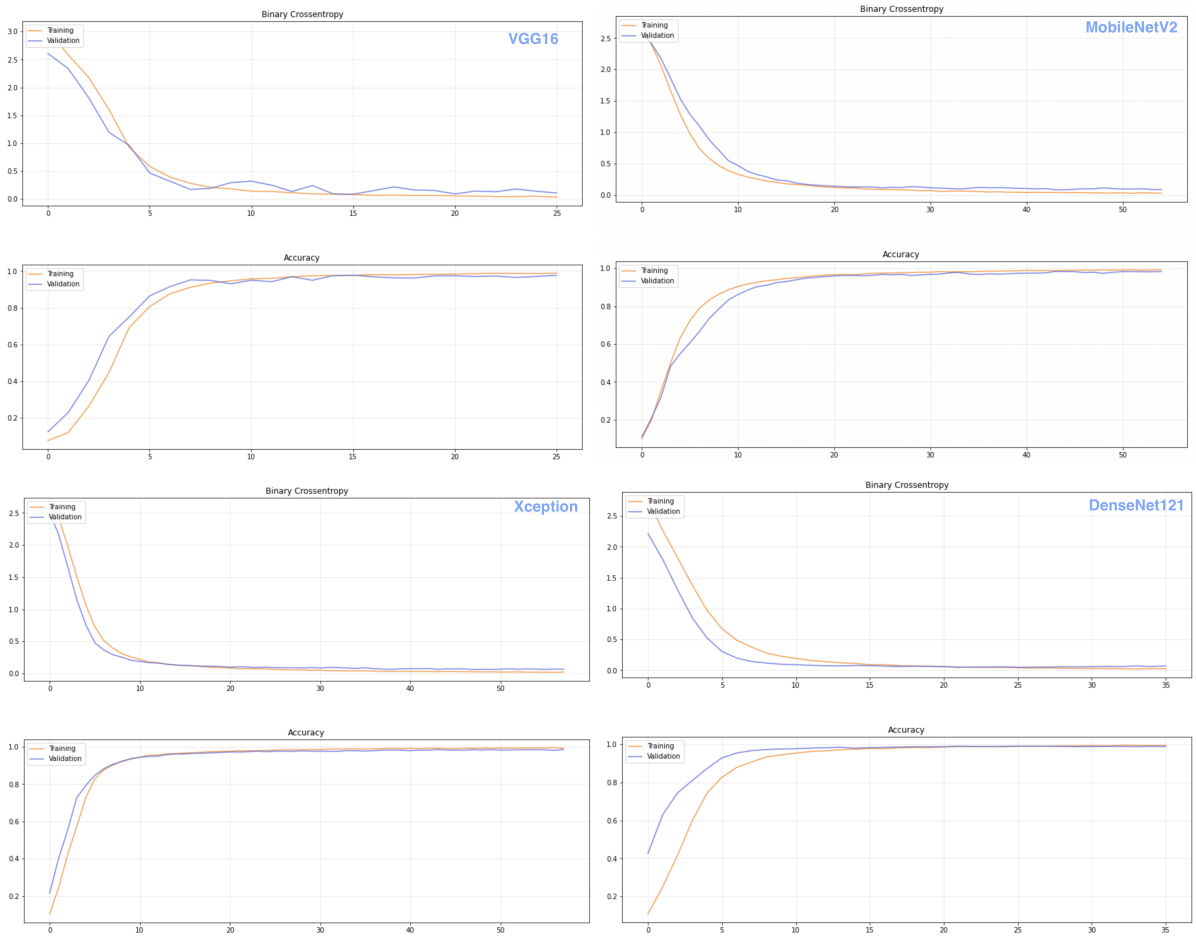
In order to get a better network, we shifted our attention to the dataset that, as we pointed out before, was not equally divided in the 14 classes. For this reason, we followed two different paths. We implemented a function which computes proper class weights relying on the number of images in each class. On the other hand, we manually modified the dataset in order to obtain 1000 pictures in every folder of each class. We noticed that the second method was giving us better results, so we replaced the dataset with the new one.

The last attempt to make progress was the expansion of the input size of the images to 512, but in the end it was not a significant improvement.

At this point, we wondered if we could radically change the model by choosing a different deep learning network, so we investigated MobileNetV2. We chose this Keras application because it's characterized by a small number of parameters with respect to the others; but still, adding the dense layer with 256 units, the trainable params were too much. Therefore we replaced the *Flatten* layer with a *GlobalAveragePooling* one, and since it improved the network we kept it also for the following versions.

Encouraged by the latest upgrades, we inspected other deep learning models, such as Xception and DenseNet121. We reached similar values of mean accuracy, but the best one of all was the last net. Paying attention to the output scores, the column associated with the wild label always had a small value: the most performing one on this issue, in the first test set, was MobileNetV2.

However, in the second test set the model with Xception was the one that reached highest results both on the mean accuracy and the wild mean accuracy.



3. Conclusion

To sum up, the model that carried out the maximum mean accuracy on both test sets was the one with DenseNet121 (first: 0,9260; second: 0,9057).

On the other hand, considering that even the Xception model reached at least 90% of accuracy in both test sets (first: 0,9038; second: 0,9057), and that its wild accuracy is over 10% bigger than the one of DenseNet121, we decided it should be our best model.