POLITECNICO
DI MILANO

AN2DL: Challenge 1

Authors:
Gabriele De Santis - 10820992
Giulia Maria Sarcina- 10581803
Michela Capasso - 10786184

November 2021

1 Preparation

The aim of this challenge is to create a Convolutional Neural Network to classify images of leaves contained in a dataset.
First of all, we analyzed the dataset. It is composed of 14 classes divided into labelled directories; each of these directories contains images of leaves of a certain type with black background. Some of these leaves have brown spots, they have different colors and shapes and some of the images have different brightness. Besides the preprocessing phase, we tried to modify and underline all these characteristics through the augmentation methods trying to avoid overfitting. For this reason we used the *ImageDataGenerator* class for generating batches of tensor image data with real-time data augmentation and preprocessing. We have choosen sundry parameters for the augmentation such as *rotation, height-shift, width-shift, zoom, horizontal and vertical flip and reflect fill mode*.
Moreover we noticed that the dataset is unbalanced because some classes contains definitely too many images than the others (the biggest one has approximately 5500 images while the smallest one 500). We investigated how to deal with the class imbalance and we decided to implement *class_ weight* from *sklearn.utils* library with the 'balanced' option.
After these considerations, in order to start the learning phase we divided the dataset into two directories, one for the train set (90% of the dataset) and the other one for the validation set (10% of the dataset). Moreover we defined the *create_ callbacks* function to trace the learning phase thanks to *TensorBoard*, periodically save the model to disk and apply some learning techniques like early stopping and adaptive learning rate.

2 Custom Model

The first model we made was (to be clear, we'll represent our models as follows)

`Model1: (Conv2D + MaxPool) X 5 + Flatten + (Dropout + Dense) X 2`

The Conv2D layers had an incremental number of filters. The first one had 16 filters, the second one had 32 filters and so on until 512. The validation accuracy reached was around 0.9 and because of that we decided to submit this model. Unfortunately, the test accuracy dropped to 0.48.
Because of that, we decided to start again with smaller models. We tried reducing the repetitions of *Conv2D + MaxPool* but they gave us lower validation accuracy.

Only repeating *Conv2D + MaxPool* for 4 or 5 times the model reached a good validation accuracy.
As we said before, we used *class_ weight* to deal with class imbalance. However we thought that giving "less importance" to the richest classes in the dataset would have led our model to a lower accuracy. As we expected, the model with *class_ weight* had a lower validation accuracy than the same model without *class_ weight*. We decided to submit both the models with and without *class_ weight* and we noticed that the model without class weight had lower generalization skills.

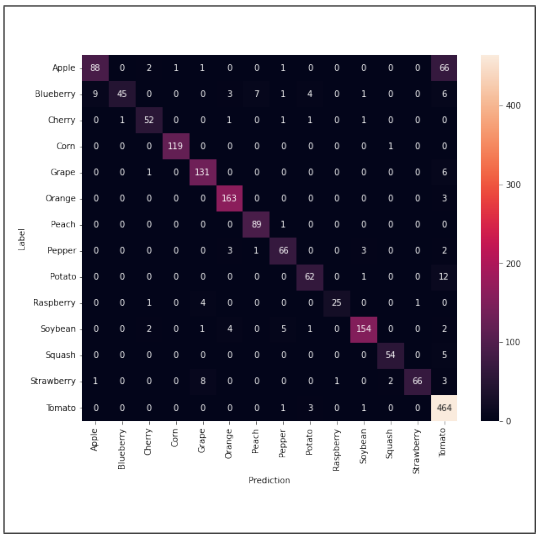

Figure 1: Confusion matrix of model1 with 4 repetitions of Conv2D + MaxPool and class weight

Although the confusion matrix of the last model showed good results (as you can see above in the Fig. 1), the low test accuracy was probably due to the fact that test images had very different features compared to our dataset images. So we tried to better generalize.
In order to do that trying not to reduce the loss of information we decided to double the Conv2D layers before each MaxPool and to give it a try to batch normalization. Our new model was

`Model2: ((Conv2D) X 2 + MaxPool + BatchNorm) X 5 + Flatten + (Dropout + Dense) X 2`

The validation accuracy was still around 0.91 and the test accuracy went up to 0.61.
We also tried different approaches that were:

- Using a *Gaussian Noise layer* at training time;
- Populating the dataset with more images taken from the *Plantvillage dataset*. We though this could be a good strategy in order to have more samples but it was not worth the trouble.

Since we wanted to try also transfer learning, we decided to just be happy with the results achieved. Because of that, we trained the Model2 without the validation phase in order to have more images. That made us achieving a test accuracy of 0.71 with our custom model.

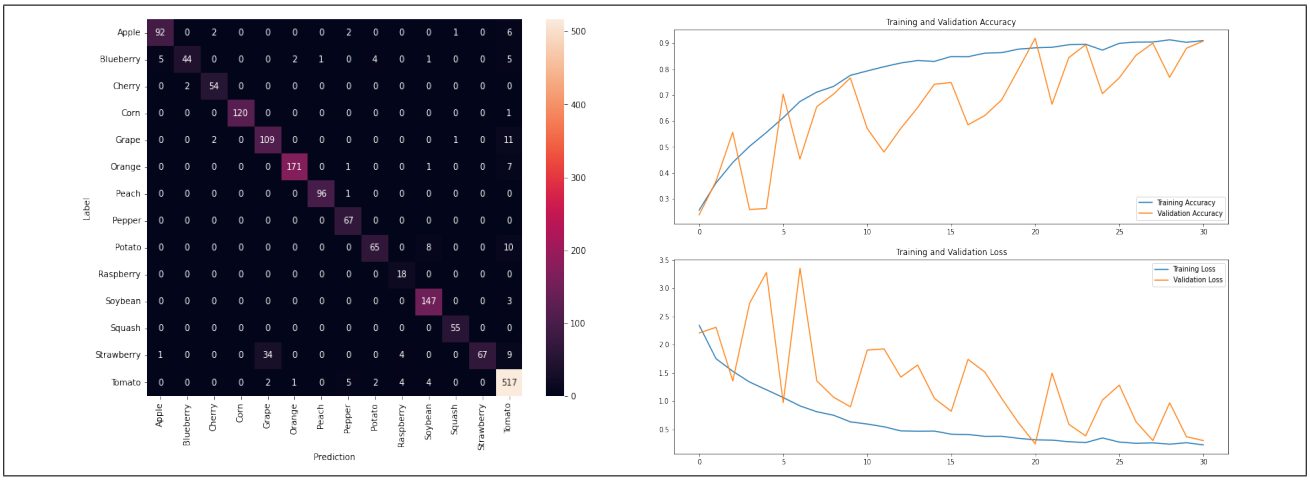

Figure 2: Confusion Matrix, Accuracy and Loss of the Model2

3 Transfer Learning and Fine Tuning

Our final approach to the problem was the *Fine Tuning* method. To apply it we firstly looked for the existing type of pretrained networks in literature. We considered in particular *VGG16* model and *Xception* model, both trained on ImageNet images. To use these networks we applied *Fine Tuning* by freezing the bottom weights of the convolutional base and cutting the top layers to adapt the nets to our problem. In order to correctly apply these nets we imported the same preprocessing used during the training of the net.

Regarding the VGG16 net, we trained it first freezing the first 14 bottom layers and then just freezing the first 10 bottom layers.
Here it is the Fully Connected Network we made

`Model3: Preprocess + VGG16 + BatchNorm + GAP + (Dropout + Dense) X 3`

We also evaluated the performance adding a Gaussian layer with 0.1 of standard deviation, useful to mitigate overfitting. As it is a regularization layer, it is only active at training time. The results of this training phase were satisfactory so we decided to retrain the model with the complete set of images and submit it. In this phase we used the best number of epochs and we decided to keep freezed only 10 layers. The accuracy we obtained on the test was 0.8.
After that we applied the same methodology to the Xception architecture by first freezing 113 layers; at the beginning we obtained less satisfactory results and, because of lack of time, we decided to continue to work on the VGG16 to improve it.
In this phase we tried using an adaptive learning rate through the *ReduceLROnPlateau* method. The results obtained were similar to the previous ones.

```
Model4: GaussianNoise + Resizing  + Xception +
        BatchNorm + GAP + (Dropout + Dense) X 3

Model5: GaussianNoise + Resizing  + VGG16 +
        BatchNorm + GAP + GaussianNoise + Dropout + Dense
```

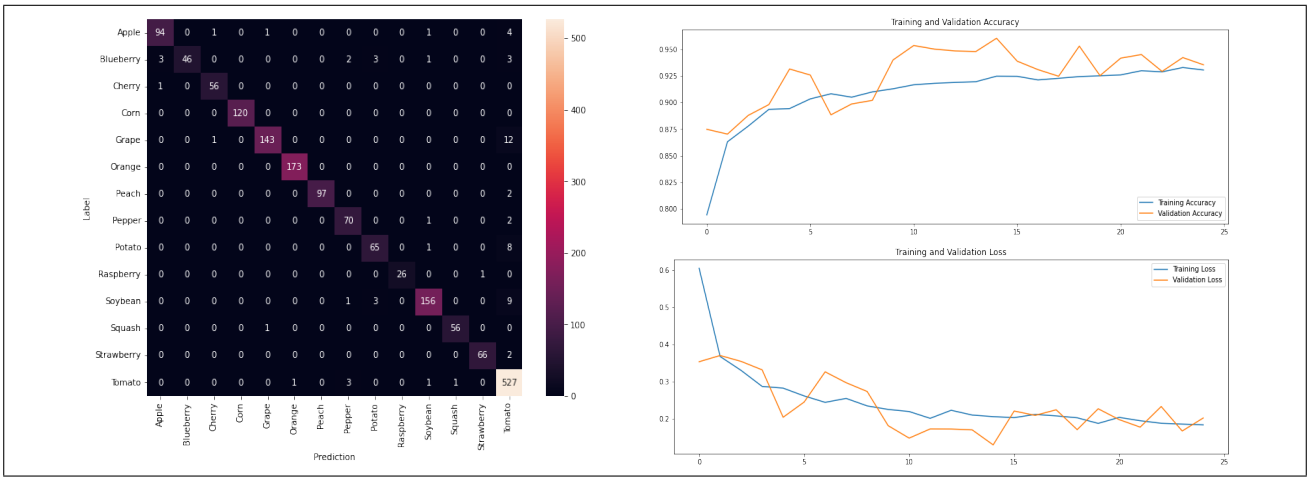

Figure 3: Confusion Matrix, Accuracy and Loss of the Model3

4 Conclusion

At the end of our analysis we decided to present as the best model the *model3*. It gave us the highest and balanced accuracy on both the training and the validation sets. It also proved to be consistent having shown coherent test scores in both the competition phases.
Nevertheless the best score is about 0.8321, probably because the 'Wild Mean Accuracy' obtained is much lower than the others.
In order to improve our model we would have tried to get better results by trying to increment our dataset with the GAN technique (Generative Adversial Maner) or by trying to initialize our classifier through autoencoders.