Leaves classification

Giovanni Dragonetti, Erica Espinosa, and Luca Mainini

Politecnico di Milano

November 28, 2021

1 Introduction

The problem we had to face is the classification of leaves images. Our dataset is composed by 17.728 images divided into 14 classes: Apple, Blueberry, Cherry, Corn, Grape, Orange, Peach, Pepper, Potato, Raspberry, Soybean, Squash, Strawberry and Tomato. Most of the images have dimensions 256 x 256 and have been isolated on a black background except for *three outliers with different shapes*; we resized them either before feeding them to the network or in the network itself. We first noticed the *unbalanceness of our classes*, as shown in Figure 1. To address this issue, we experimented two different approaches: *class weights* and dataset balancing via *upsampling*. We first built a simple model to check what we could achieve with very few levels, then we experimented the transfer learning technique with some of the most important models in image recognition.

2 Data augmentation

Since some classes contain a limited number of data, we have decided to use the data augmentation technique. We initially tried to figure out how to modify the images, particularly how to set the parameters of `ImageDataGenerator`. Displaying generated images, we could broadly understand the *range of these hyperparameters*: for instance, a brightness parameter superior to 2 generates meaningless images.

We then trained our model on images generated with different parameters saving for each combination of values the model and the score obtained on the validation. Then, we would have liked to test these models on the CodaLab set and then choose the best parameters with respect to the obtained value [`Data_Augmentation_Tuning.ipynb`]. Unfortunately, we had a *too limited number of attempts*, so we leave this tuning procedure as improvements to be done in the future.

We also tried implementing a custom augmentation to simulate a more realistic environment as shown in Figure 2 [`Custom_Augmentation.ipynb`] but this proved unsuccessful.

3 First model

We first consider a first simple model, constituted by *3 increasing-depth blocks of 2 convolutional and activation layers* alternated with layers of *Max Pooling* and, at the top, a *fully connected network* with one hidden layer. To reduce overfitting, we have also added 2 *dropout layers*. The input has been normalized by dividing by 255. The model is represented in Figure 3 and summarized in Table 1.

Initially, we had extracted a small part of the data to use it as a test set. However, the accuracy on this test set was very different from that obtained on the competition test. Then, we have decided to use all the available dataset to carry out training and validation and to test the model through the challenge platform. The performance of this model was not so low (score: 0.64), but extremely improvable especially for the classes Potato, Apple, Cherry and Corn.

4 Two approaches to face classes unbalanceness

At this stage of the project, we were faced with the fact that the classes in our training dataset were not balanced.

4.1 Class weights

In order to solve the problem, we have decided to modify the *cost function* to minimize, giving *more weight to smaller classes* such that the final loss function is a weighted average of each class loss function:

$$L = \sum_{i=1}^{14} w_i \cdot L_i$$

$$w_i = \frac{N}{14 \cdot n_i}$$

where L is the total loss function, N is the total number of images and 14 is the number of classes. The variables w_i, n_i and L_i respectively stand for the weight, the number of elements and the loss function computed in class i. In this way our network will make an extra effort to learn information about the smaller classes.

4.2 Dataset upsampling

Alternatively, instead of relying on class weights, we tried to balance the dataset by upsampling the classes until they all had the *same number of elements*. This was performed by augmenting the images through an `ImageDataGenerator`. Its implementation is given in [`Dataset_Upsampling`].

5 Transfer learning

After this first approach, we decided to use transfer learning to improve the performance of our network. This choice was made because, with the limited amount of data available, training a deeper convolutional section would not have been as effective as we wanted. We have tested with three different models:

- **VGG16**: a well known model for image classification already introduced during the course;
- **Xception**: one amongst the best performing models on the imagenet dataset;
- **MobileNetV3** (large): a very lightweight model with good performances.

We have built our models adding a *maximum of two fully connected layers* at the top of the transferred one with at *most 256 neurons*. Initially we trained only the added layers and then we fine-tuned the whole model *decreasing the learning rate*. We have used both approaches to resolve the class unbalanceness and we have noticed that only the first one, class weights, works properly. Probably, with upsampling, having many modifications of the same few images in most of the classes, the model learned only how to recognize those images.

6 Hyper-parameters tuning

To find the best fully connected network to add on top of the imported convolutional segment, we used Keras-tuner, an hyperparameter tuning tool. The method we decided to use was *Hyperband*, as performing a grid or random search would have been too costly. The base we analyzed was composed by a flattening layer followed by one or more dense layers with *Relu activation* and the output layer; dropout was applied to all except for the output. The hyperparameters we analyzed and their respective ranges were: *number of fully connected layers* (1 or 2), *number of units of such layers* (128 or 256), *dropout rate* (0.1, 0.3 or 0.5) and learning rate (1e-2, 1e-3 or 1e-4). Unfortunately the process was unsuccessful, probably because we limited the number of epochs per combination to 20. The models trained with these hyperparameters had lacking performances but we could not afford to repeat the tuning as it took too much time and resources.

7 Test time augmentations

To make our model more robust we also tried using test time augmentations: feeding the same *image augmented* in various ways to the same network and *averaging its predictions*. The augmentations we used were *flipping, shifting, rotating and cropping* the center of the image; the parameters and number of effects to apply were chosen analyzing the performance of the model on such combinations. [`Test_Time_Augmentations_Tuning`]

8 Final model

After having tested our alternatives we have realized that the best performing one is the *Xception* added with one hidden layers with 128 neurons. The model was trained using the class weights technique on the augmented set. The global accuracy obtained in the final test set is 0.9018867925.
Figure 4 shows training and validation curves.

9 Conclusion

The Xception network proved to be the most accurate, this result is expected since it is also the best performing on the Imagenet dataset; close performances were provided also by MobileNetV3 with an accuracy of 0.8868 and VGG16 with 0.866. The fact that the *performance obtained with MobileNetV3 is comparable* with the one of other models (much heavier) is an indication that we don't need too many convolutional layers to extract the proper class features from simple images as ours. To make our model more robust, a future improvement could be to split our biggest category (Tomato) in k subsets S_i such that $|S_i|$ is comparable with the cardinality of the other class sets. Then train k models, one for each S_i and all the other whole categories and finally average them. We couldn't try this approach for time reasons.

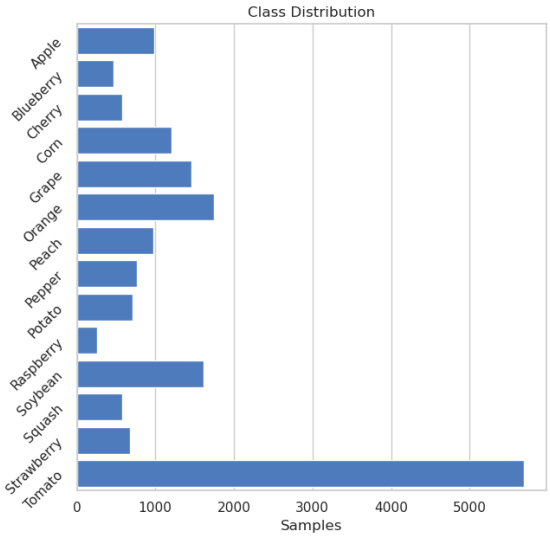

Figure 1: Notice that the cardinality of the Tomato class is more than 20 times superior to the one of the Raspberry's

Figure 2: Custom augentation

Layer (type)	Output Shape	# Param
Input (InputLayer)	(256,256, 3)	
convolution 1	(128, 128, 8)	1184
convolution 2	(128, 128, 16)	6288
convolution 3	(32, 32, 32)	12832
convolution 4	(32, 32, 32)	25632
convolution 5	(8, 8, 64)	51264
convolution 6	(8, 8, 64)	102464
Classifier	512	524800

Table 1: Structure of our first model.

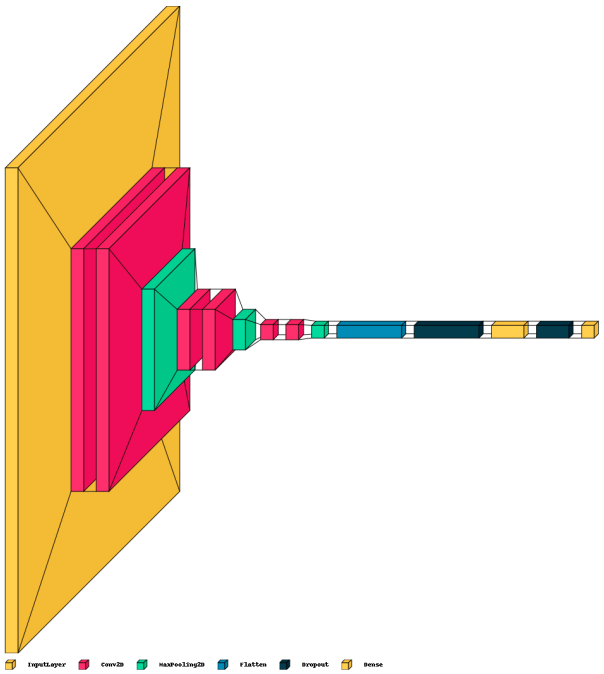

Figure 3: Three increasing-depth blocks of two convolutional layers alternated with layers of MaxPooling. At the top a fully connected layer with one hidden layer.

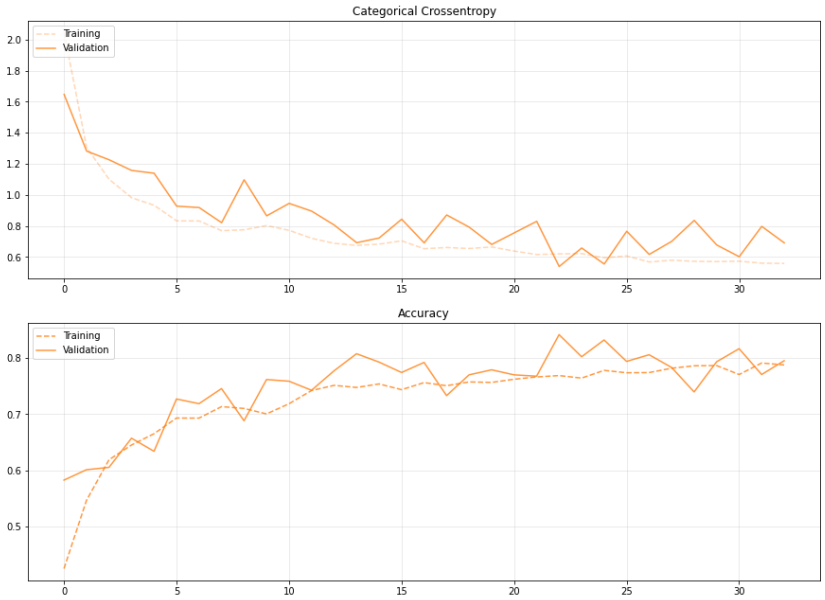

Figure 4: Loss function and accuracy of the best model.