

# AN2DL - Image Classification - Report

Gattinoni Valentina 10583105

Manzoni Claudio 10580671

Recalcati Andrea 10578352



November 29, 2021

## 1 Objective

The goal is to create an architecture which is able to classify some leaves images over 14 different classes, assigning the correct label. The implementation is done with a Convolutional Neural Network (CNN).

## 2 Dataset

We were given a labelled training set, composed of 17728 images, whereas the test set was unknown (accessible only by submitting trained models on *Codalab*).

Before starting, we created an **overview** of the dataset (see the notebook `Dataset_Inspection.ipynb`) in order to analyze its main features. By way of example here we report an image present in the dataset with its main characteristics. From this output of the `get_next_batch` function non-trivial considerations can be drawn: images are stored as  $256 \times 256 \times 3$  tensors and they are not always centered and in parallax.

```
(Input) image shape: (8, 256, 256, 3)
Target shape: (8, 14)

Categorical label: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
Label: 13
Class name: Tomato
```




Figure 1: Random extraction of an image from the dataset

But we do not want, for example, *Tomatoes* leaves to be learned only as rotated in the above specific way. This suggests that we should try, starting from existing data, to create new synthetic data and adding slightly modified copies of existing images. This will surely act as regularizer and will help to reduce overfitting when training a model and to avoid low performances on strange and unclear datasets.

## 3 Development

First of all, using the command `ImageDataGenerator` and its option `validation_split` we split the dataset in training and validation set with a 0.8/0.2 ratio.

Moreover, over the training set, we exploited some data augmentation techniques to generalize the training as much as possible. No preprocessing has been done over the validation set, to simulate an unknown test set.

### 3.1 Architecture built without Transfer Learning

We started by implementing a simple architecture, inspired by the one done in Lab classes. We used this simple net (and its "fastness" to be trained, even if not so precise...) to study some important **hyperparameters** and preprocessing parameters, ranges and boolean variables (of the data augmentation technique). In particular:

- Regarding the **input shape**, we decided  $256 \times 256$ , due to the fact that it is the size of all the images in the dataset;
- After some trials on the configuration of the **data augmentation ranges**, we found the values minimizing the loss on the validation set: we adopted these optimal values for all the future networks.

We then developed a more complex architecture trying to improve the performance with a *Trial and Error* method, acting on the number of convolutional layers and neurons, and therefore increasing its complexity. As complexity increased, more interventions were needed to avoid overfitting: for this purpose we exploited dropout layers (very important before dense layers) in the top part of the CNN. Also *Maxpooling* and *Flattening* blocks turned out to be useful to increase the performance of the network on the validation set.

You can find the detailed implementation of the resulting CNN in the attached notebook `No_TL_best.ipynb`. In the table below we report the accuracy on training and validation set, and the loss (Categorical Cross-Entropy) after 33 epochs (the whole training process took 43 epochs but we had set the *patience* of *early stopping* to 10).

Loss	Accuracy	Val_Loss	Val_Accuracy	Score (1 <sup>st</sup> phase)	Score (2 <sup>st</sup> phase)
0.1681	0.9465	0.1915	0.9376	59.43%	59.05%

Table 1: CNN No TL: results for training, validation and test (*Codalab* score)

After having developed this classification model, we tried to improve it changing the batch size but we did not achieve any better result.

At this stage, we tried to attach the problem of **class imbalance**. Our dataset, indeed, contains far more instances in some classes than others (for example, *Tomatoes* leaves represent 1/3 of the whole images). This imbalance in the training set can lead to poor average precision during the machine learning classification. For this purpose, instead of using as loss the *Categorical Cross-Entropy* we implemented our own version of *Weighted Categorical Cross-Entropy*, assigning inversely proportional weights with respect to the classes size. The implementation is in the Notebook `Weighted_loss_CNN.ipynb`. However, we did not manage to submit the so-trained model, since we received (by *Codalab*) the error "*Unrecognized Loss Function*", even if we inserted it in the `model.py` file.

### 3.2 Transfer Learning architecture

We had no doubt that to improve our performances the only way was Transfer Learning.

We imported the whole VGG16 architecture, slightly modified in the FC part (to solve a 14 classes classification problem) and again, as done in the previous network (without TL) we started searching the optimal hyperparameters (focusing on the top part of the network). In order not to have a too complex architecture, we applied a structural regularizer. For the detailed implementation, see the notebook `VGG16_TLbest.ipynb`; the model summary of this network is reported below.

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14714688
conv2d (Conv2D)	(None, 6, 6, 64)	294976
global_average_pooling2d (GlobalAveragePooling2D)	(None, 64)	0
dense (Dense)	(None, 14)	910
Total params: 15,010,574		
Trainable params: 7,375,310		
Non-trainable params: 7,635,264		

Figure 2: Model summary of the network built using the VGG16 supernet

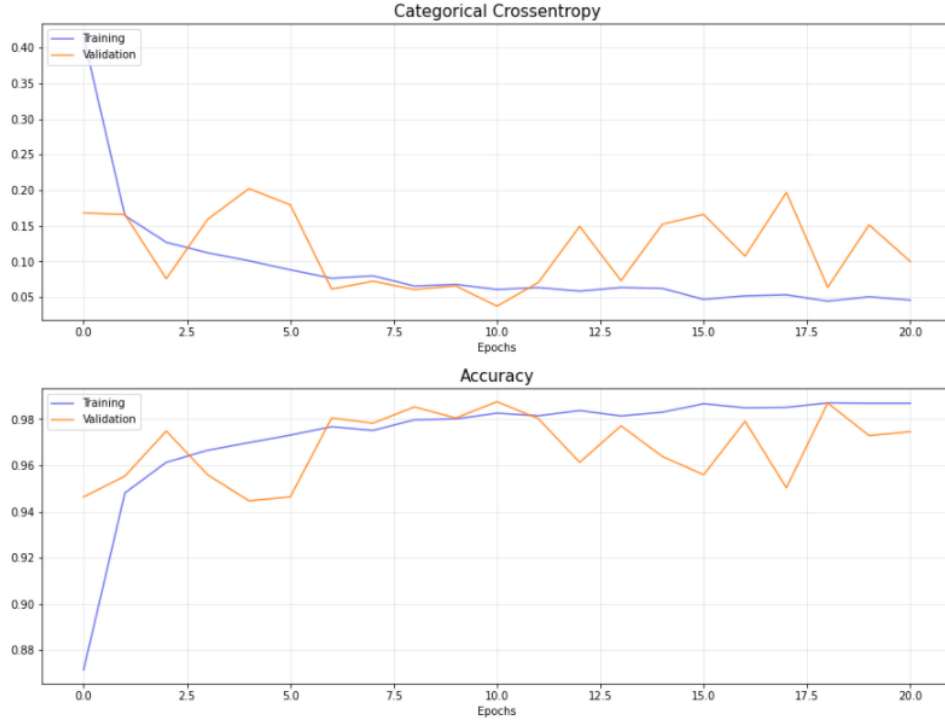


Figure 3: Chart of accuracy and Categorical Cross-Entropy trend as function of the number of epochs on training and validation set (network that exploits VGG16 architecture)

In the following table we report accuracy and loss on training and validation set and the score on the test set.

Loss	Accuracy	Val Loss	Val Accuracy	Score (1 <sup>st</sup> phase)	Score (2 <sup>st</sup> phase)
0.0605	0.9826	0.0371	0.9876	83.01%	82.83%

Table 2: Network exploiting VGG16 architecture: results

**With respect to the model without TL**, in this one the loss has decreased a lot (both on training and validation set), and also the performance on the test set has significantly improved ( 82.83% on the 2<sup>nd</sup> phase test set): the psychological 80% accuracy threshold has been crashed!

Moreover, this model is robust and stable with respect to changes on the test set (we passed from 83.01% to 82.83%) and this confirms the goodness of the model.

Since **supernet-based models** usually learn faster than simple ones built by beginners (like us!), we tried to speed up the training further to see if the preprocessing (data augmentation) and the monitoring (early stopping with 10 epochs of patience) were superfluous. We lowered the patience of early stopping and removed the data augmentation. With this configuration (implementation in the notebook **VGG16\_NO\_AUG**), the model performed worse on the validation set and also got a lower score on the test set (as regards the latter, 75.66% instead of 83.01% in the 1<sup>st</sup> phase of the competition, 77.73% instead of 82.83% in the second one); for this reason we consider as the "best" model the one in which we augmented the images with appropriate transformations and ranges (tuned as explained in section 3.1).

## 4 Conclusions and future developements

As further development, if we had more computational capacity to speed up hyperparameter tuning and network training, we would tune the learning rate and increase also the number of kernels of the network.

Furthermore, another way to correct class imbalance could be to divide the class of *Tomatoes* into  $n$  groups, thus generating  $n$  different datasets, containing each a smaller number of *Tomatoes*, so that the  $n$  datasets are balanced; then an average of the weights made on the  $n$  trainings can be used as the resulting weights, in this way obtaining a more robust method.

Nevertheless, we are really satisfied to have ventured into a real-life task and performed a good result.