AN2DL Homework 1

Valeria Amato
10641790

Emanuele Paci
10681377

Andrea Riboni
10699906

1 Introduction

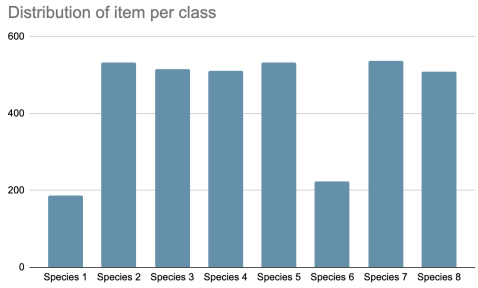
The project's goal was to create an image classification model for classifying leaf photographs based on the plant species to which they belong. In the following report we are going to describe how we handled this problem in three stages:

1. Analysis of the dataset and research of the best data augmentation parameters
2. Creating a CNN from scratch
3. Applying transfer learning and fine tuning

2 Approaches

2.1 Analyzing the dataset

We noticed the dataset is quite small and presents a discrepancy of the number of items per class. Therefore, our next architectural decisions must take this into consideration.



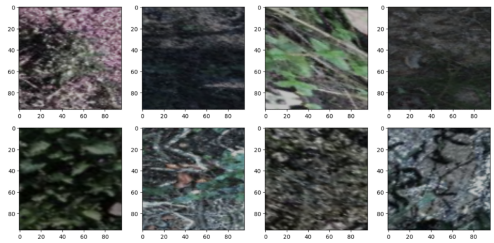
To reduce the discrepancy it's probably going to be necessary to apply various data augmentation techniques in order to create new unseen samples. Another option that might be beneficial is to implement class weights to penalize overrepresented classes and so balance the distribution. We chose accuracy as our primary metric, especially by keeping an eye on its trend on the validation set, in order to make things straightforward.

Furthermore, we experimented different percentages to split our full dataset into training set, validation set and test set: firstly, we began by dividing using the conventional formula: 0.75 – 0.15 – 0.1. However, the dataset was already lacking in images, and we were wasting some of them in our test set, which was already on Codalab, so we decided that having a test set was a pointless notion, even if -for starters- it was a nice indication on what could have been Codalab's score. As a result, we increased the training and validation sets to 0.80 and 0.20, respectively.

Since a bigger training set could end up in an increased chance of overfitting, we adopted the early stopping approach when training the networks.

2.2 Data Augmentation

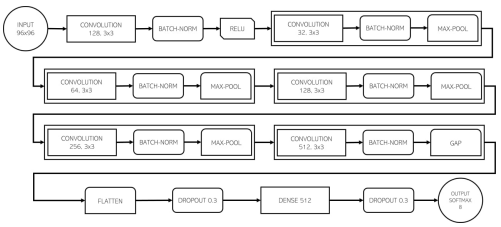
The primary purpose of data augmentation is to provide additional photos to enhance the existing dataset. It is crucial to determine which transformations may be used to achieve this aim, which means that the resulting image should not be excessively deformed; otherwise, it's going to be difficult to correctly classify images. The set of transformations we have found more fitting for our first network ("Standard CNN") consists of rotations, zooms, flips and brightness changes. The second network ("Transfer learning and fine tuning") used a slightly different set of transformations that included the set used for the first network, shears and shifts.



These samples have been generated using data augmentation on the 8 original dataset photos.

2.3 "From scratch" CNN

Creating a convolutional neural network from scratch has been our first approach to the problem, similarly to the ones we have seen during practice lectures. The crucial aspect is defining the network structure: the basic idea was to start with a small network and to deepen it, iteration by iteration. At the end, the model which performed the best was composed of 6 convolutional layers, interveiled with batch normalizations, max pooling layers and global average pooling layers.



The basic idea is to apply a series of macro-layers composed of increasingly deeper convolutions followed by a pooling operation to reduce the dimensionality. The classification part of the network manages to use the drop-out technique in order to reduce the risk of overfitting. This network reached an accuracy of 76% over the public leaderboard. As previously mentioned, data augmentation has been an effective strategy to deal with the tiny dataset size and has helped us to increase our score, whereas class-weights had made it worse.

3 Transfer Learning and Fine Tuning

Following the construction of the Standard CNN models, we chose to use the transfer learning approach to create different models, possibly with a higher accuracy rate. The base models that have been employed are the following:

- VGG16 and VGG19
- Xception
- EfficientNetV2L

We chose to expand up networks in terms of depth during the initial development phase, progressing from VGG16, VGG19 and Xception and to EfficientNetV2L.
This comparison allowed us to see how increasing the number of layers affects performance, and the models produced the following results in the development phase:

Base Model	Score
VGG16 / VGG19	62%
Xception	84%
EfficientNetV2L	86%

As it is possible to see from the table, an higher number of layers increases the score of the our model.
As a consequence, the EfficientNet family provided the foundational models we used in the final step. Although model scaling occurs on numerous dimensions, from depth to input resolution, the EfficientNet design does not exclusively rely on depth scaling in order to achieve higher overall performance.
The EfficientNetV2L base model allowed us to achieve better scores even if increasing the training time with respect to models such us Xception, since the numbers of parameters involved is so much higher.
The previous model were evaluated using the various types of augmentation methods and we determined that data augmentation that had an influence on test-performances was the following:

Type	Parameter
Horizonatal Flip	True
Vertical Flip	True
RandomRotation	0.25
RandomZoom	0.1

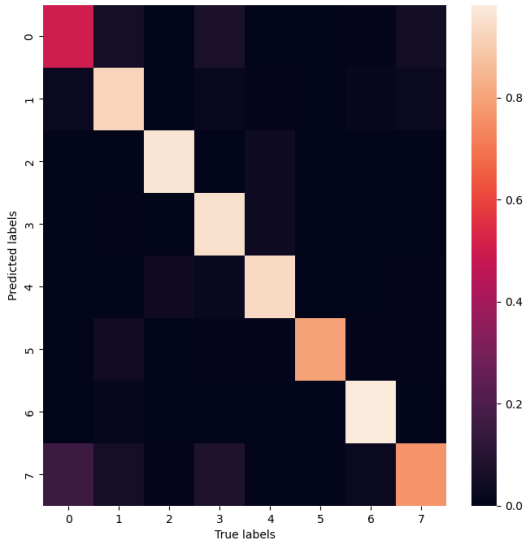
To assess the effects of scaling up the top layer, we used alternative versions of the top layer up to achive this one:

Layer	Activation	Parameter
Flatten	/	/
DropOut	/	0.7
Dense	ReLu	512 units
DropOut	/	0.3
Dense	ReLu	256 units
DropOut	/	0.1
Dense	Softmax	8 units

This has been tried with Xception and EfficientNetV2L base model. The second was slightly better than the first one.
As it is possible to read, the first dropout layer is quite high due to the risk of overfitting that we faced for all creation process.

4 Hyperparameters Tuning

During the development phase, we tested the effects of tuning various hyperparameters on overall performance, paying particular attention to batch size, number of freezed layers, number of units per layer.
Firstly we used KerasTuner, a scalable hyperparameter optimization framework that solves the pain points of hyperparameter search.



The confusion matrix above represents the performance of our network. The first pixel tends towards purple, as it often mixes up 1 with 8.

- Regarding batch size, we noticed that using relatively small batch sizes, such as 16, 32 or 64, increases instabilities during training. Our last model uses 32 as batch size, which increased the accuracy of the gradient. A lower number of the batch size seems to work better with model. However, a low number as 16 makes the training really slow
- We started considering as number of freezed layer $\frac{3}{5}$ of the total. This time, the easiest approach was a grid-search and we determined that freezing 12 layers was the best choice for our model.
- Despite higher amount of units per layer increased the local accuracy, it ended up being an overfitting illusion and most of the time we relied on layers of 2^7 to 2^8 units

5 Conclusions

After the competition's development phase, we tested our models during the final phase, and we achieved our highest score with the ensamble model between Xception and EfficientNetV2L, which received a score of 0.8630. The model statics are the following:

- Accuracy: 0.8630
- F1: 0,8598