Image Classification

Leaderboard nickname: NotRunningCode
Students: Federica Baracchi, Federico Camilletti, Patricks Francisco Tapia Maldonado

1 Data

1.1 Data augmentation

The first approach to the model was oriented in the analysis of the dataset. The assignment we received was to classify eight species of plants represented in images. We had a total of 3542 images divided into 8 classes. However, we found that the division wasn't homogeneous since had the following number of images per class:

Species	1	2	3	4	5	6	7	8
Number of images	186	532	515	511	531	222	537	508

It was evident that we had an unbalanced dataset. To avoid getting unoptimized results for the classes that were underrepresented we decided to solve it. As the dataset was small we decided to not go for under-sampling. Instead, we choose over-sampling using data augmentation. It was achieved mainly using the Keras library, in particular with ImageDataGenerator. Applying data augmentation we have seen two different kinds of alterations that we could apply to the images. A hard augmentation that altered the images combining rotations and reflections to fill empty spaces and a light augmentation that did not alter the plants' structures.

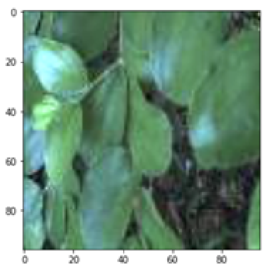


Figure 1: Original

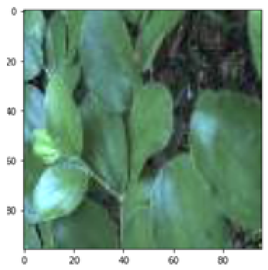


Figure 2: Light

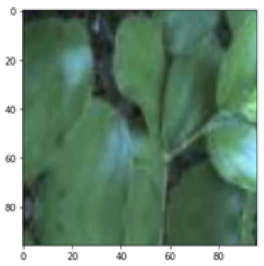


Figure 3: Hard

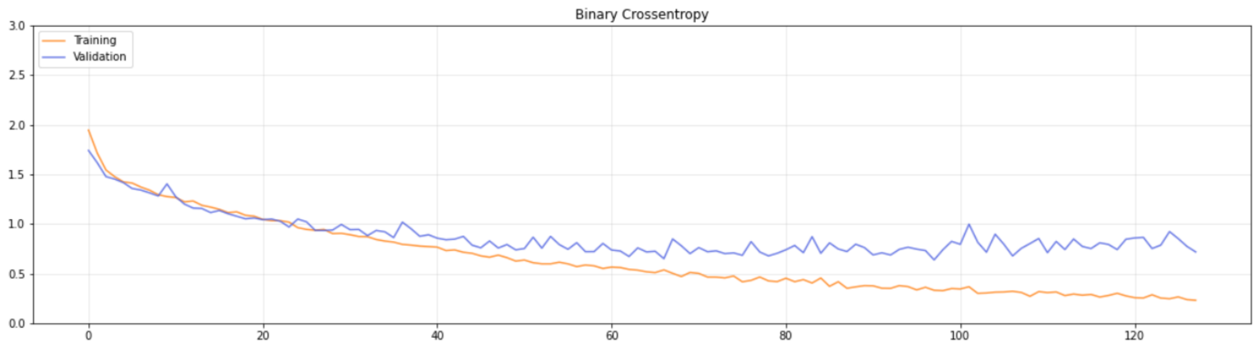
We have tried the two different approaches with various networks and since the performance of the light method in terms of accuracy on the test set was better than the hard one we decided to use it. The results of data augmentation showed greater robustness, however, we still had unoptimized results on the underrepresented species. For that reason, we decided to use the Keras class-weight option to give the right importance to the underrepresented classes. This modification didn't change the results. What solved the issue was adapting the receptive field of the neural network to reach optimal performance.

1.2 Splitting the data set

During the solution of the unbalanced data set issues, we split data into 80 percent for the training set and 20 percent in the test set. In particular, we divided the training set from the test set creating two different folders. The validation set was achieved by using 20 percent of the training set thanks to the ImageDataGenerator constructor. This division showed to work well in training, but also the results we got in our test set were close to the ones we got in the hidden set. For these reasons, we decided to keep this division also for the rest of the experiments.

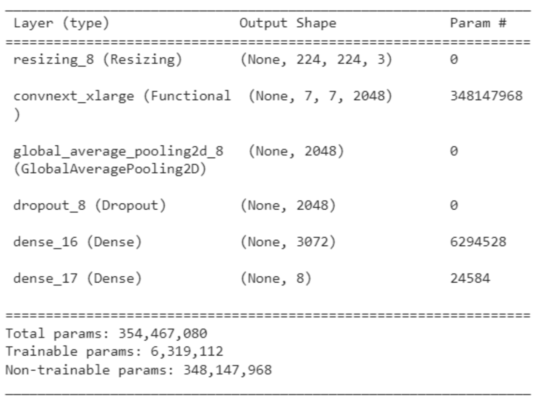
2 Convolutional Neural Networks

The first approach used to classify the plants was based on sequential block of layers made of convolutions, poolings and dropouts. The latter in particular were used in order to avoid overfitting.
Several improvements were produced comparing the accuracy of the different models and supporting us with the output images of every layer.
We have noticed that the average pooling works on maintaining homogeneous brightness, instead max pooling only catches the main features of the image. The best model, in our case, turns out to be the one with only average poolings.
An important mention has to be made on the last layer before the fully connected ones, where we have applied the global average pooling. We have noticed that this change has boosted the accuracy of the model.
Regarding the fully connected layers we avoid to use many hidden layers, in particular this last part was calibrated heuristically.
Finally the best model detected consists on 4 blocks of convolutions + average pooling except the last global average pooling:
$32 \rightarrow 64 \rightarrow 128 \rightarrow 256$.
Regarding the fully connected part, only one classifier was provided made by 512 neurons.
Many other attempts were made, as try double convolution on each block and change the activation function to smoothness the relu one, obtaining worst accuracy.
The best fit on data augmentation in this section was to only use a range for brightness and use the flip of the images.
The relevant parameters to be set were the batch size and the patience. The first one, after several attempt, was set on 32 which gives us the best result and therefore it was maintained for the following models. The second one was set on 30, we have noticed on the error plot, as the one below, that the curve tends to rise and then get back down. Due to this phenomenon the patience was crucial in order to achieve the accuracy of the 83% and on codalab of 81%.



3 Transfer Learning

After building the best model, among those tested, we implemented the famous CNN architectures. We started with the VGG16/VGG19 model, and then we tried different models like RestNet50, various types of EfficentNet, Xception model. Finally, we used the ConvNeXt model and in particular **ConvNeXtXLarge**. Since the best results in terms of accuracy have been achieved by the latter, we have focused on this, modifying it to improve its accuracy and limit overfitting.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resizing_8 (Resizing)       (None, 224, 224, 3)       0

 convnext_xlarge (Functional  (None, 7, 7, 2048)       348147968
 )

 global_average_pooling2d_8   (None, 2048)             0
 (GlobalAveragePooling2D)

 dropout_8 (Dropout)         (None, 2048)              0

 dense_16 (Dense)            (None, 3072)              6294528

 dense_17 (Dense)            (None, 8)                 24584

=================================================================
Total params: 354,467,080
Trainable params: 6,319,112
Non-trainable params: 348,147,968
_________________________________________________________________
```

The steps that have been done for transfer learning:

1. take a pre-trained model (**ConvNeXtXLarge**) since the convolutional part can be seen as a feature extractor

2. remove or edit FC layers as they are very customized, as are used to solve specific classification objectives. In our case, we have chosen the **GlobalAveragePooling** after noticing that it worked well in our case.

3. freeze CNN weights and train only FCN layers

4. **fine-tuning**: the entire CNN is retrained, but the convolutional layers are initialized to the pre-trained model. The model chosen is very large (about 350M parameters) but the number of layers is not too high(about 300). This allowed us to update them all at the end of fine-tuning but it was chosen to reduce the learning rate compared to the one commonly used. The only built-in layer that has non-trainable weights is the BatchNormalization layer so we decided to freeze it.

3.1 Limit overfitting

In building our model, it is necessary to make choices to limit overfitting and therefore allow it to learn and better recognize new images: the **dropout** technique was used, by randomly turning off thirty percent of neurons for each epoch. Looking at the graph of error functions allowed us to recognize whether or not **overfitting** occurred. We noticed that the error function submitted to the validation set, before reaching the minimum, shows a trend similar to that evaluated on the training set (**Early stopping** technique). Not having access to the test set, looking at the errors, we could expect an **accuracy** evaluated on the test not far from that evaluated on the validation set. As expected, the accuracy achieved by species with a lower number of images is lower than the others. We tried a technique to overcome class imbalances, giving weight to the minority class proportional to its under-representation. But this technique was ultimately not helpful in this model.