AN2DL IMAGE CLASSIFICATION REPORT
TEAM: back_in_black

1. Scope and first contact with data:

The designated task entailed the development of a neural network model capable of proficiently classifying images within the context of a binary classification problem. Specifically, the objective was to discern and categorize leaves based on their health status, differentiating between healthy and unhealthy states.
The initial phase included analyzing the provided dataset. A crucial step involved visualizing sample images to gain a deeper understanding of the dataset's relevance to the specified problem.
Subsequently, meticulous examination revealed the presence of incongruent images within the dataset. To ensure the integrity of feature comprehension, these non-conforming instances were deleted from the dataset, in order to have only coherent images.
The dataset was inspected to establish the number of different labels provided within data, in order to understand whether data were equally distributed. This allowed us to proceed by experimenting with different types of analyzes based on the original dataset and on a balanced dataset via image augmentation through different transformations.

Following the data cleansing process, the dataset has initially been separated into training and validation sets; the allocation ratio was set at 80% for the former and 20% for the latter.
In further attempts, an extra step was taken, creating a test set alongside the training and validation sets, with the aim of verifying the veracity of the newly constructed model.

Proceeding with normalization, the data underwent transformation to adhere to a standardized scale within the [0, 1] range, thereby enhancing model convergence during training. Simultaneously, the label set underwent encoding, transitioning into a categorical format through the application of one-hot encoding techniques.

2. Preprocessing

We employed preprocessing techniques to augment the number of images during training, initially implementing both geometric and optical transformations. In this way every batch during training was subject to random transformations, further reducing overfitting and in order to avoid enlarging the training dataset beforehand.

3. Design of a hand built model

We decided to design a first model, just to get an idea of what results we could reach without any kind of pre-processing, to see what our starting point was. We directly decided to build a Convolutional Neural Network. We tried different combinations: one model with three convolutional layers (32, 128, and 256 three-by-three filters), another with four convolutional layers (32, 64, 128, and 256 three-by-three filters), then five convolutional layers (32, 64, 128, 256 and 1024 three-by-three and five-by-five filters). To each of these convolutional layers we added a Relu activation function, a 2D max pooling and the batch normalization. Of course, we then added an 2 neurons output layer with a softmax activation function, to make the final predictions. We used AdamW as an optimizer as it was the one giving us the best result. We also tried others such as Adam and SGD but without improving the result. The loss function we chose was the BinaryCrossentropy as we are dealing with a binary classification. We then trained these models with 500 epochs, and with an early stopping method with a patience of 30 and a variable learning rate monitored by the accuracy of the validation data predictions. The best of these models turned out to be the one with four convolutional layers, but the results weren't quite good, as the accuracy on the training data reached 80%, and the accuracy on the validation data reached 73%.

4. Transfer learning approach

Afterwards, the approach we decided to follow was to adopt the transfer learning technique. In order to do this, we experimented with different pretrained networks from the keras.applications package. The network we tried are:

- VGG16
- VGG19
- ResNet50V2
- ResNet101V2
- ConvNeXtSmall
- ConvNeXtBase
- ConvNeXtLarge

After the results provided through different attempts, the network we decided to work with has been ConvNeXtLarge and we are gonna explain how we got there.

4.1. Experiments:

Below is a brief description of the initial models that were subsequently discarded in favor of more accurate ones. Each model was configured and trained using early stopping with different patience and variable learning to try preventing overfitting.

VGG16 and VGG19

The initial exploration involved the application of transfer learning with VGG16 and VGG19 networks, yielding promising but ultimately challenging results to enhance further. Despite enriching VGG16 with additional fully connected layers, achieving a satisfactory accuracy of 0.8362, efforts to improve proved elusive. The transition to VGG19 produced similar outcomes.

ResNet50V2 and ResNet101V2

Subsequent attempts with ResNet50V2 and ResNet101V2 involved varied strategies such as cyclic learning rate, fine-tuning, and cross-validation. However, none yielded substantial improvements. Implementing a cyclic learning rate and unfreezing the last 20 layers of ResNet resulted in modest results, with an accuracy of 0.6212.

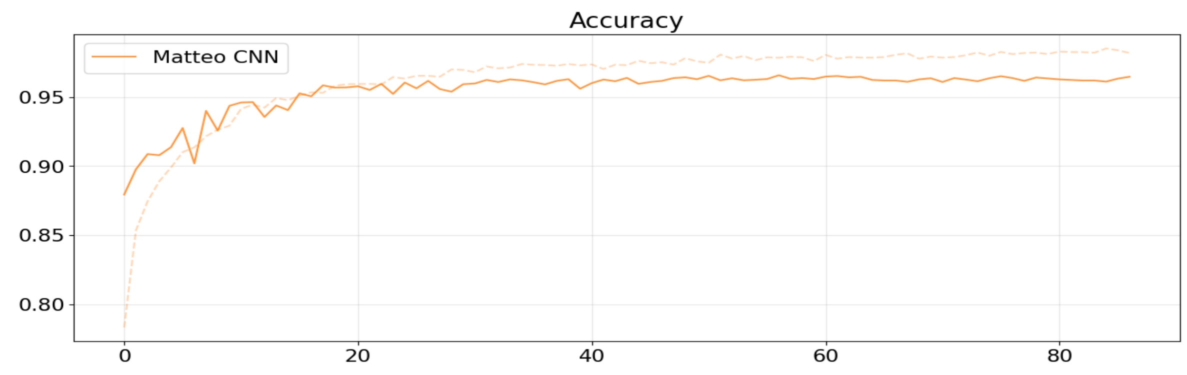
ConvNeXtBase

The ConvNeXtBase neural network emerged as a more effective starting point, employing a simple convolutional neural network (CNN) that demonstrated efficient learning and generalization during training. While achieving a high accuracy of 0.9687, the validation accuracy (val_accuracy) of 0.8891 indicated some potential overfitting even with the use of dropout after each dense layer. Additionally, challenges arose during the inference phase, where the model struggled to recognize unhealthy leaves, as evidenced by the confusion matrix (Figure 1).

4.2. Final Model

ConvNextLarge

Seen the positive results obtained by the ConvNextBase model used for transfer learning, we aimed to improve the accuracy of our model by trying to use a larger and more complex network. For this reason, we tried implementing the Large model of the ConvNext family.
Before starting to train it, we knew we would need more data as the “Large” model is more complex so we used the ImageDataGenerator to generate a larger dataset with augmented images generated from the originals.
In order to compensate for the unbalanced dataset, we also introduced a dictionary containing class weights which proved to be quite useful in the training of our model.
We also added Batch Normalization in order to stabilize the optimisation process and we kept the Dropout in order to better prevent overfitting.
Finally, we tuned the hyperparameters in order to find the best suit for this model. After some trial, we got better results with a dropout rate of 0.4 and a batch size of 32. Patience rate for the early stopping was set at 30.

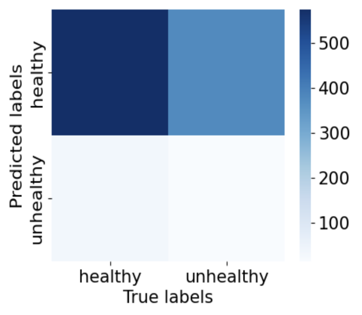


We can see better results with no overfitting and the confusion matrix improved compared to the previous model (Figure 2).

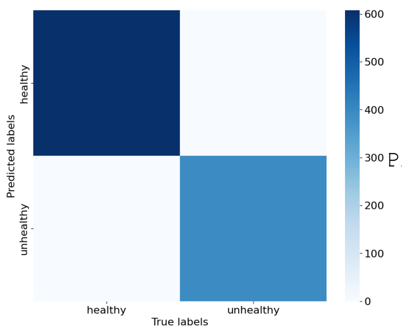
5. Fine tuning

Once we had found a network giving good enough results (both in the local environment and on the codalab test) we decided to start to fine tune it to improve it even further. We lowered the learning rate used to 1e10^(-5) and started to try to unfreeze an increasing amount of layers until we found an optimum. Said optimum was found with the first 160 layers of the network freezed and the others trainable, which resulted in a total of 507.7 MB of trainable parameters, over the total of 776.08 MB. The network, tuned with these parameters, has obtained the following results:

```
loss:  0.0012  -  accuracy:  0.9992  -  val_loss:  0.0712  -  val_accuracy:
```



```
Accuracy:0.5
Precision: 0.4579
Recall: 0.4919
F1: 0.4024
```



```
Accuracy: 0.999
Precision: 0.9992
Recall: 0.9987
F1: 0.999
```

Figure 1

Figure 2

Contributions:

Simone Zacchetti: Focused on cleaning the dataset from non coherent images, standardization and preparation of data for the model. Working on ResNet50V2, ResNet101V2 and ConvNeXtSmall models.

Matteo Zamuner: Exploring different types of images augmentation and optimization on the dataset. Trying a different supernet on which doing transfer learning. Testing different layers configuration on our model and tuning hyperparameters to get the best possible result.

Lorenzo Auletta: Exploiting transfer learning, trying different configurations mainly focused on ResNet50V2, working on preprocessing with data cleaning and augmentation, cross-validation and hyperparameter tuning.

Lorenzo Torsani: Focus on transfer learning and comparing the results of the various imported Networks, trying different types and quantities of new FC layers. Improving the imported network via fine tuning and finding and optimum with regard to the quantity of layers to retrain.