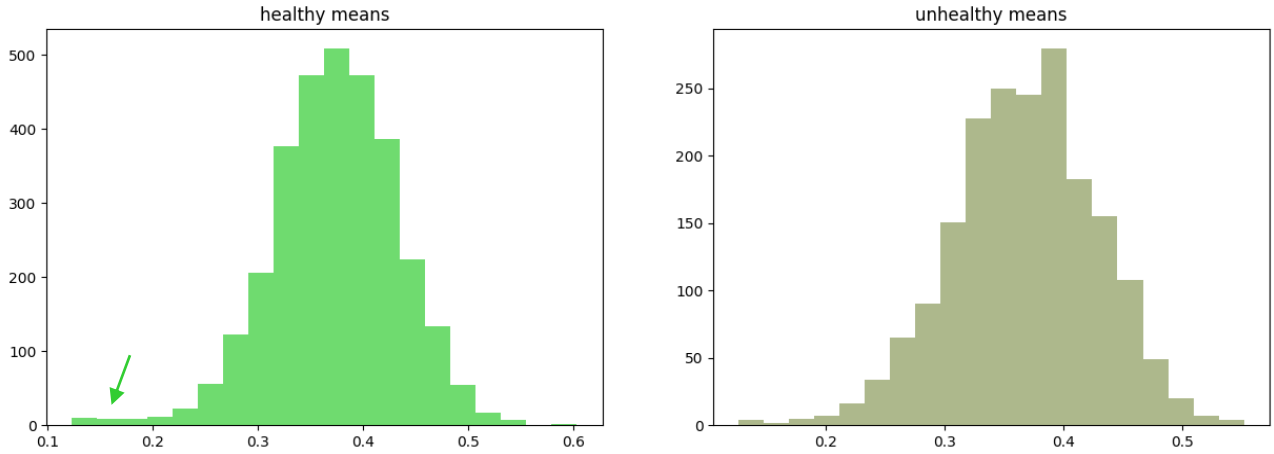
# AN2DL Challenge 1

*Group:* Learning in the Deep *Components:* Giulia Mezzadri, Federico Angelo Mor

## 1. Explorative data analysis

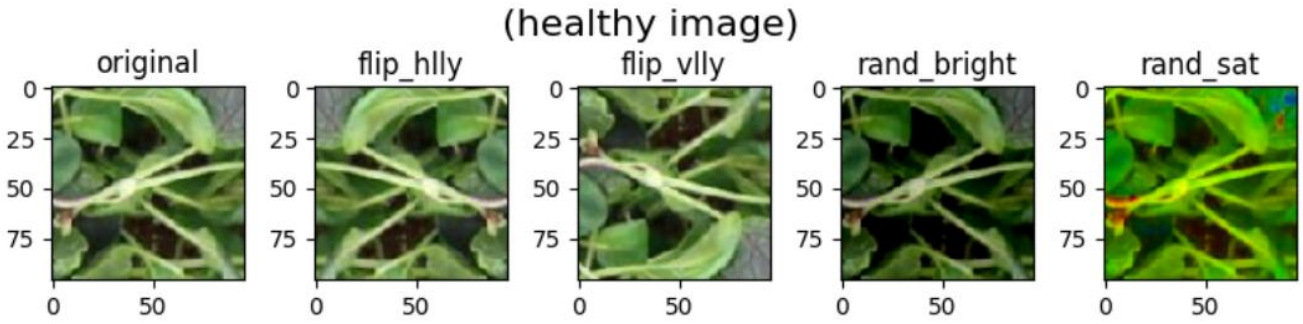
*Data Inspection and Filtering* We started loading the data and inspecting them to see which kind of images there were. During the inspection we saw some "weird" images (indeed not plants), which we defined as "shrek" or "meme". We decided to remove all of them, both because they didn't concern the task (classifying healthy or unhealthy plants) and especially because they didn't have constant labels: for example, the shrek ones were sometimes labelled as healthy and sometimes as unhealthy, so we interpreted this as a clear attempt to confuse us and our models.
We found 196 of such images, which we removed from the dataset using a *for* cycle passing all images and keeping only the ones acceptable, through a comparison with a shrek and meme sample with the *np.array_equal* function. In this way we reduced from 5200 to 5004 images.
At this point we had a "clear" and clean dataset, which we could start to work on. However, we thought of two other steps we could do before starting to train Neural Networks (NNs), to possibly enhance their performance. These steps were further outliers detection and data rebalancing.

*Outlier Detection* About the outliers we saw that, at a glance, healthy plants were lighter than unhealthy plants, which appeared instead to be darker. We were confident that brightness was a possible feature that our models could use (together with many others, like maybe patterns in the leaf veins, leaf edges, colours, and so on), and so if we included too many dark healthy plants, this could have misled our NNs.




We plotted a histogram of the mean value of the pixels for the plants, and indeed we found some possible outliers for the healthy case. Also considering the fact of being in a "natural" setup, where data commonly follow a gaussian distribution, we were then more convinced to remove those images (the ones which had a mean value less than 0.2 in the histogram) that lied in the far tail of the gaussian expected trend. For the unhealthy images we saw a more regular histogram, there were again slightly heavy tails in the lower values, but we decided to keep all the images: both because we had few unhealthy data, and because those dark unhealthy plants may not be necessarily considered outliers but just a stronger sign of unhealthiness.
Still, we considered that these "outliers" could help prevent overfitting and increase the number of data, and even be correct samples actually; so we tested the models also with keeping them, and we noticed slightly different performances.

*Data Rebalancing* About data rebalancing, we saw that initially the proportion of healthy and unhealthy images was not the same but there were more healthy than unhealthy images. We therefore decided to act on this disequilibrium by sampling at random some images from the unhealthy class, applying a transformation, appending them to the old dataset, and finally apply a random permutation to shuffle the so obtained dataset (as well as the corresponding labels). In this way we managed to rebalance the data distribution, resorting to a 50-50 ratio of labels, starting from the original ratio of 62-38.
An interesting point was also choosing carefully the transformations to apply, as we had to be sure they would have been non-affecting for the classifying task.

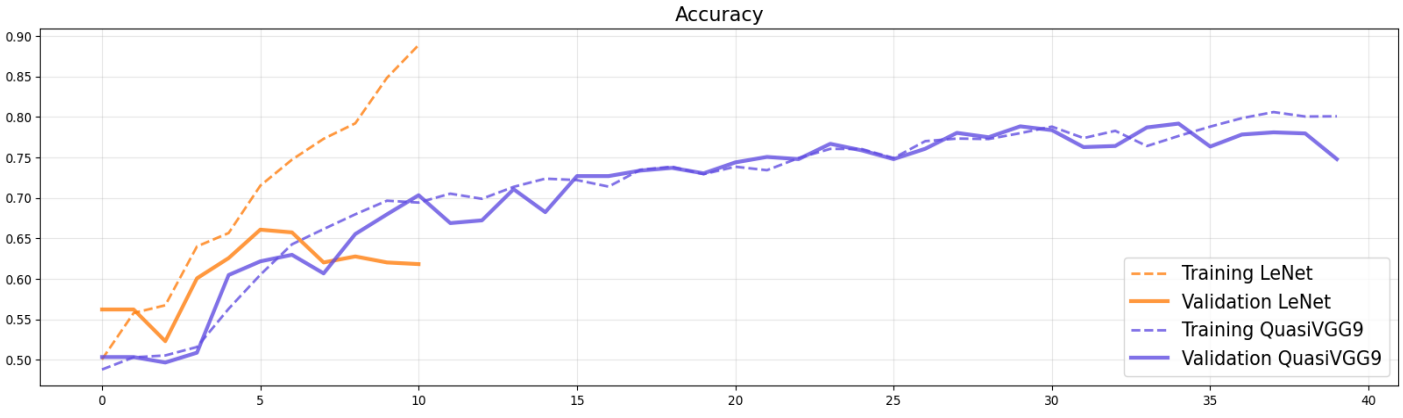



We tried first with random flips, both horizontally and vertically, which we were sure were a safe transformation, in the sense that they couldn't change the health appearance of the plants, as leaves can grow in different orientations, as well as the camera that took the pictures could have been variously oriented.
We then considered applying a random change in brightness and saturation, but we later discarded these options (also after seeing some worse performances in the models). About the reasons behind this removal, we thought that the lighting of images is probably a feature that the NNs consider for this classification task, so adding a brightness variation could alter the learning process and the final prediction. While about the saturation change, we thought that it could produce some artifacts in the image, for example new colors (like the red, and even some purple spots, as we can see in the image above), or again a brightness variation, which will both lead to a wrong (or better, confused) learning process.

## 2. Model definitions

We then approached the problem of building an appropriate neural network by trying both simple hand-built nets and pretrained ones. Actually, the model definition and data processing steps were not independent: we experimented different models but also on different setups of data, like the original filtered data, the filtered and rebalanced data, this last one plus outliers removal, and so on.
This because we wanted to check if that data processing was actually meaningful, i.e. improving our performance rather than worsening them, and in case understand why. In this way we were able to assess the rationality behind the possible modifications in the data.

*Simple Classical Models* With the classical and simple models, written and trained from scratch, we didn't manage to get good performances. We implemented and tested LeNet and QuasiVGG9 models, but during training they quickly showed convergence of the accuracy values at a not-so-high number, or they exhibited clear overfit on the training data. LeNet was the worst one with respect to the overfitting problem, while QuasiVGG9 was working but was not good enough (it sticked to a 0.60ish accuracy on the CodaLab submissions scoreboard).



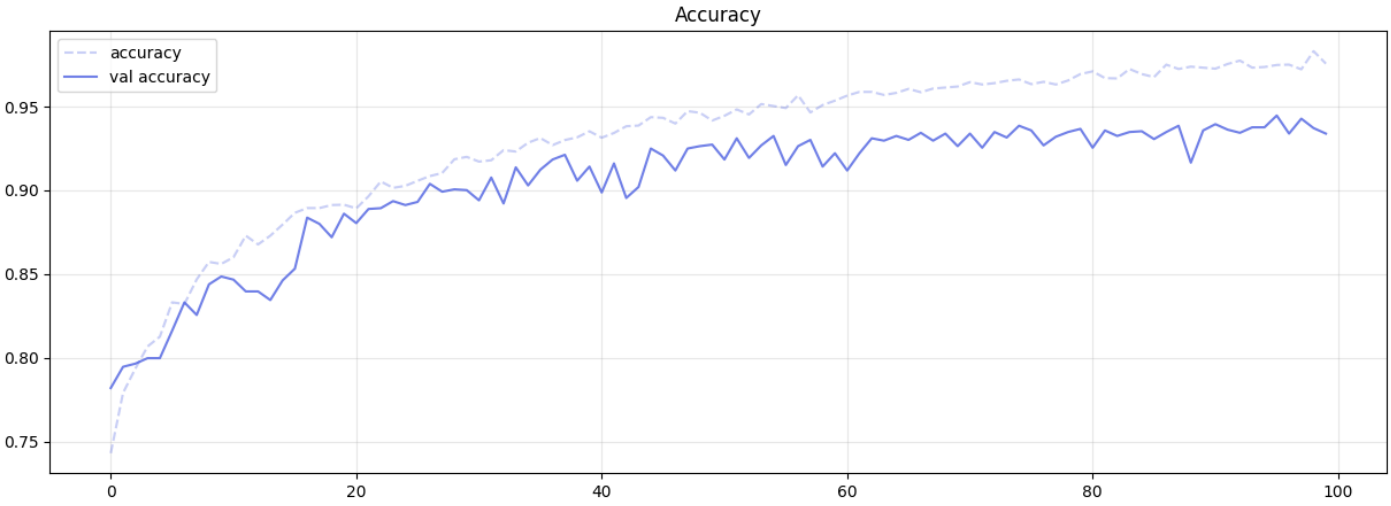

*Transfer Learning and Fine Tuning models* At that point we thought about using more powerful and pretrained NNs on ImageNet, with the transfer learning approach. We made various attempts implementing the following models: MobileNet, VGG19 and Xception; always referring mainly to the keras website for problems and technicalities.
Another attempt was made with EfficientNet, of which we tried more than one version (B0, V2B0, V2BS), but we noticed a weird behaviour concerning validation accuracy: the graph kept oscillating between the values around 0.4 and 0.6, never converging; and even the fine tuning recommended for that library didn't solve the issue.

So in the end we settled on the Xception model, which was deemed the most promising. The base transfer learned model was better than our original best, the simple QuasiVGG9, but to further improve it we chose to implement fine-tuning, and indeed we noticed a substantial upgrade in the validation accuracy. This improvement was probably also due to another tweak: after the first trials we saw a stable graph for the accuracy, and so we tried to reduce the learning rate, in order to maybe explore better the region of parameters where our model "comfortably" was arriving to.
For the model design and test we also included other overfitting-preventing techniques, like early stopping (also by playing with its parameters), but especially dropout showed a significant effect; since the early stopping we started with wasn't enough to completely solve the problem alone. We also added a layer of augmentation by including horizontal and vertical flips plus rotations and translations, of which we tried different parameters in general.

As anticipated in the beginning, the data setup changed over the course of time, and we ran this model on different setups: firstly on the just-filtered data, and secondly on the rebalanced and outlier-removed data (call them *old* and *new* model respectively). The new model showed a slightly better performance on our test set (also on the CodaLab scoreboard of Phase 1, where we moved from 0.80 to 0.82 accuracy), but in the end this difference didn't look significant (in fact in Phase 2 the odds swapped, with a 0.73 to 0.72 accuracy in favour to the old model).
On our test set we saw also an interesting behaviour: the new model had maybe a better accuracy, but tended to misclassify as unhealthy images which were actually healthy (as we can see from the confusion matrices, where rows are predicted labels, columns the true labels, and the order is unhealthy then healthy). We think this was due to the rebalancing, which introduced a lot of new augmented unhealthy plants, and so the net was not able to correctly classify all of them.
Anyway in the end we considered the two models as equivalent; then after some more tweaking we stopped to see meaningful improvements and so we sticked to these models. Here we propose the training behaviour plot for the old one.

```
Old model
Accuracy: 0.9219
Precision: 0.9222
Recall: 0.9219
F1: 0.9219
[[576  40]
 [ 56 557]]

New model
Accuracy: 0.9243
Precision: 0.9253
Recall: 0.9243
F1: 0.9243
[[584  32]
 [ 61 552]]
```




We even tried to enhance a bit the performance by trying to get the predictions not by simply taking the argmax produced by the two-neurons in the output layer with the softmax function, but by thresholding one of the argmax components, with a threshold decided basing on the ROC curve. This idea didn't produce a meaningful change as the predictions values were already concentrated on the edges of the interval [0,1] (as we can see in the histogram on the notebook, showing the values of the column 1 of the predictions vector), so there were not much "indecisive" values on which a tuning of the threshold would have produced more impactful effects.

*Conclusions* We tried a lots of different techniques, both on the data processing and the model test sides, but we ended up with an accuracy of 0.82 in the Phase 1 scoreboard. This suggests that maybe for further improvements we should have tried more "heavy" architectures from the keras package, for example the ConvNeXtXLarge, EfficientNetV2L, or ResNet152V2. Having these nets lots of parameters, with respect to our small Xception net, they would have probably performed better. But we thought that just choosing a huge net and train it with "no thought" was not the real aim of the challenge; so in the end we were still satisfied of having achieved somehow good performance out of a relatively simple net architecture.