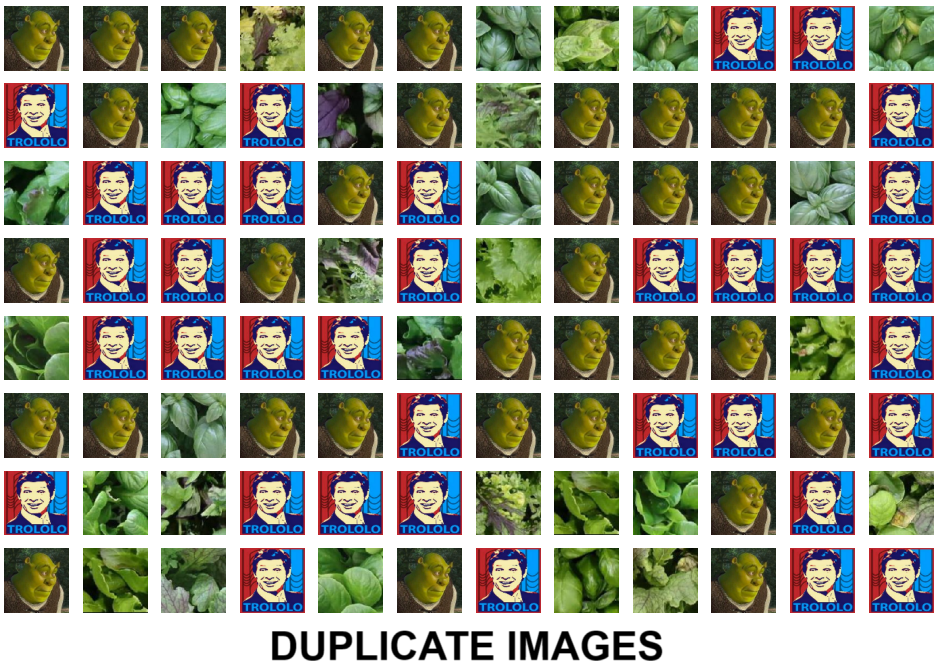
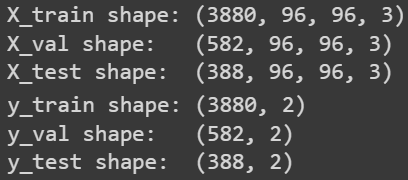
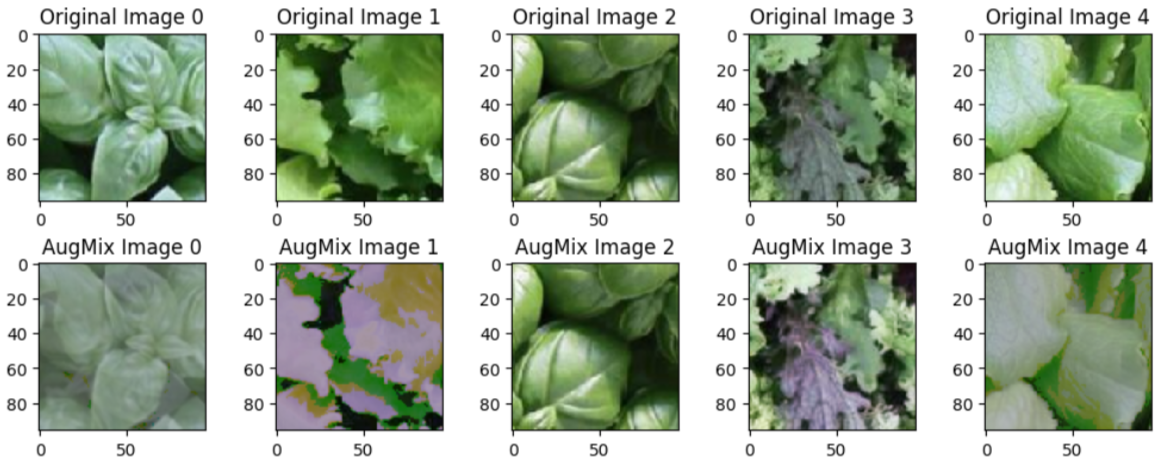
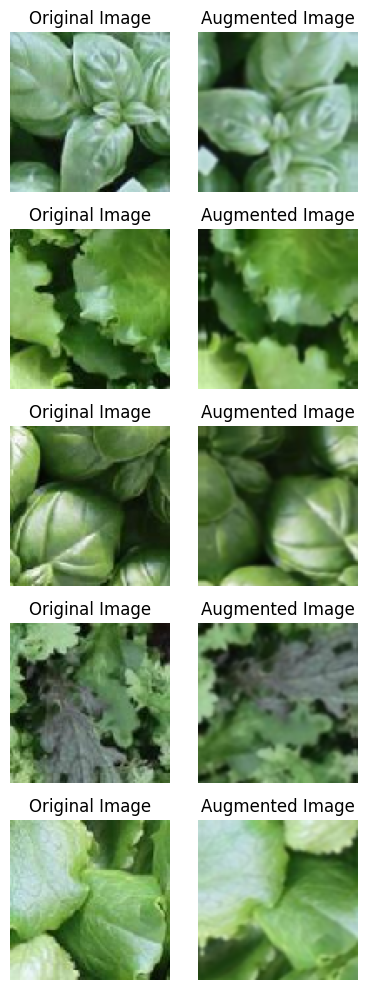
REPORT ANN FIRST HW

**Data quality and preprocessing:**

We started the challenge with data exploration of the public\_data dataset. Taking a simple plant image as reference we looked for outliers, that are those which are most dissimilar from it (using MSE), so we found many pictures of “Trololo”. Obviously they have to be removed from the dataset because they introduce bias, so we exploited a function that calculates the hash values of each image and stores the indices of images with the same hash values and then we proceed to remove duplicates in the data (350 images). Visually analysing duplicate images we discovered also “Shrek” pictures that are mostly green, so were not classified as outliers at first. The next step involved the split of the data in train, validation and test set respectively with the following proportions: 80, 12 and 8.

**Data Augmentation:** 

Our first approach to data augmentation was to apply it on the training data and store it in a file. We defined a pipeline that consists in: balance the training data by flipping 1020 images (the difference in number between healthy and unhealthy labels), use AugMix to apply multiple augmentations at the same time, lighten darker images and darken lighter images and add them to the previous ones and then shuffle the data. This approach proved to be ineffective, probably because it was generating too many correlations between images.

Even so those images have been reused as some sort of test set that somehow can be used as an indicator of how well our models were able to generalise.

For this reason we thought that augmenting images in place during training would be the best solution. At first we used the ImageDataGenerator from Keras, with good results but since it seems to be deprecated we opted for a more flexible solution still offered by keras that consists in placing augmentation as layers directly after the inputs.

These layers are then automatically deactivated when the model is in inference mode.

We are convinced that this is the best solution for our task thanks to its flexibility and its less impact on computational resources and required training time.

On the left it’s possible to observe an example of augmented images used to train our best models (in particular using: flip rotation zoom translation).

**Balancing Dataset:**

Before going deep into the training of the models we discussed data balancing, since the distribution of the labels isn’t equal. Our first concerns were in fact to have a representative of the real-world distribution and to avoid introducing bias in the model's training process.

To ensure fairness, and generalisation capabilities of our models we tried mainly 3 different solutions in this order:

1. Balance the training data with image augmentation;
2. Undersampling healthy plant images;
3. Using class\_weight while training.

The first solution didn’t work well as previously stated for the same reasons, the second one was discarded since the training data available isn’t so much, whilst the last one might be the simpler and most efficient solution because it permits to use all available images and to avoid the problem of balancing the classes distribution.

**Keras Neural Networks architectures:**

We first tried to realise custom neural networks but we quickly realised the limits of this approach so we decided to try with models pretrained on the imageNet dataset.

Browsing the Web, we found that EfficientNetB0[5] has been used to perform a similar task[1].

EfficientNetB0 is revealed to be a very versatile architecture giving us valuable results (see the delivered notebook).

Approaching EfficientNetB0 training we used transfer learning in the most classical way, that is to say download the model with pretrained weights on imageNet dataset, freeze all its layers and add a small network with a few dropout and dense layers at the end.

Next we tried a different way of performing transfer learning, we cut the EfficientNetB0 network in block6, and then connect it to our small network.

By doing this we popped out the block7 that since is one of the last layers is also one of those that capture more specific features that may be irrelevant for our task.

Once we made the most of it, we passed from EfficientNetB0 to a more complex network that performs really well in most classification tasks such as ConvNeXt (we used the Base one since it has an appropriate number of parameters).

We replicated what we have done with EfficientNetB0 also for ConvNeXtBase with a good improvement on our test data.

**Run into a Problem:**

During the competition we found out a relevant difference between the local results on our test set and remote results on codalab. At the beginning we didn’t know if there was an issue in our models or on the preprocess of the dataset. We tried to modify our networks thinking about overfitting and by weighting the number of dense layers. A particular model did not present this huge difference in local and remote in the first phase. It was our best model in the first phase obtaining 86% accuracy on codaLab. It did perform a mean subtraction in the different channels as normalisation (see notebook section “Transfer Learning of EfficientNetB0 first phase”). This did make us think that the problem could be possibly in the image data loading and preprocessing.

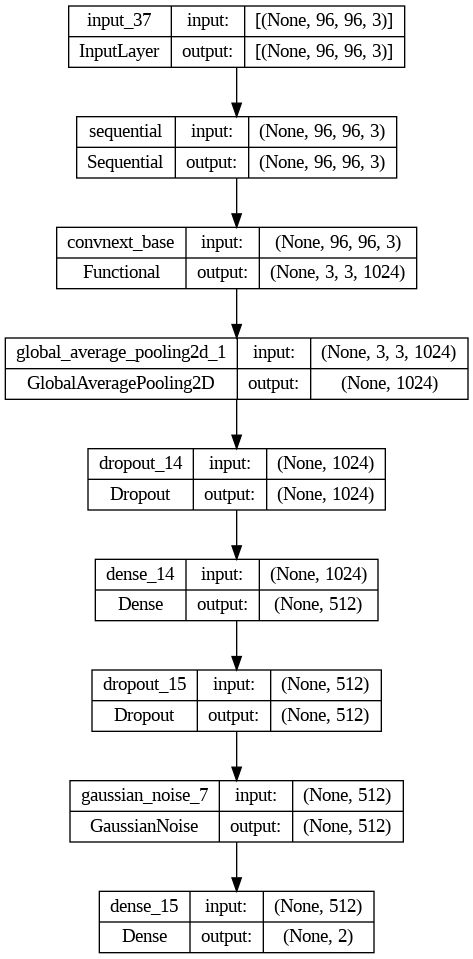
After this, we inspected our data preparation and splitting in order to find an erroneous acquisition of the channels from the dataset. While this error did cost us time to debug, it also highlighted the importance of the acquisition, preprocessing and data quality assessment phase.

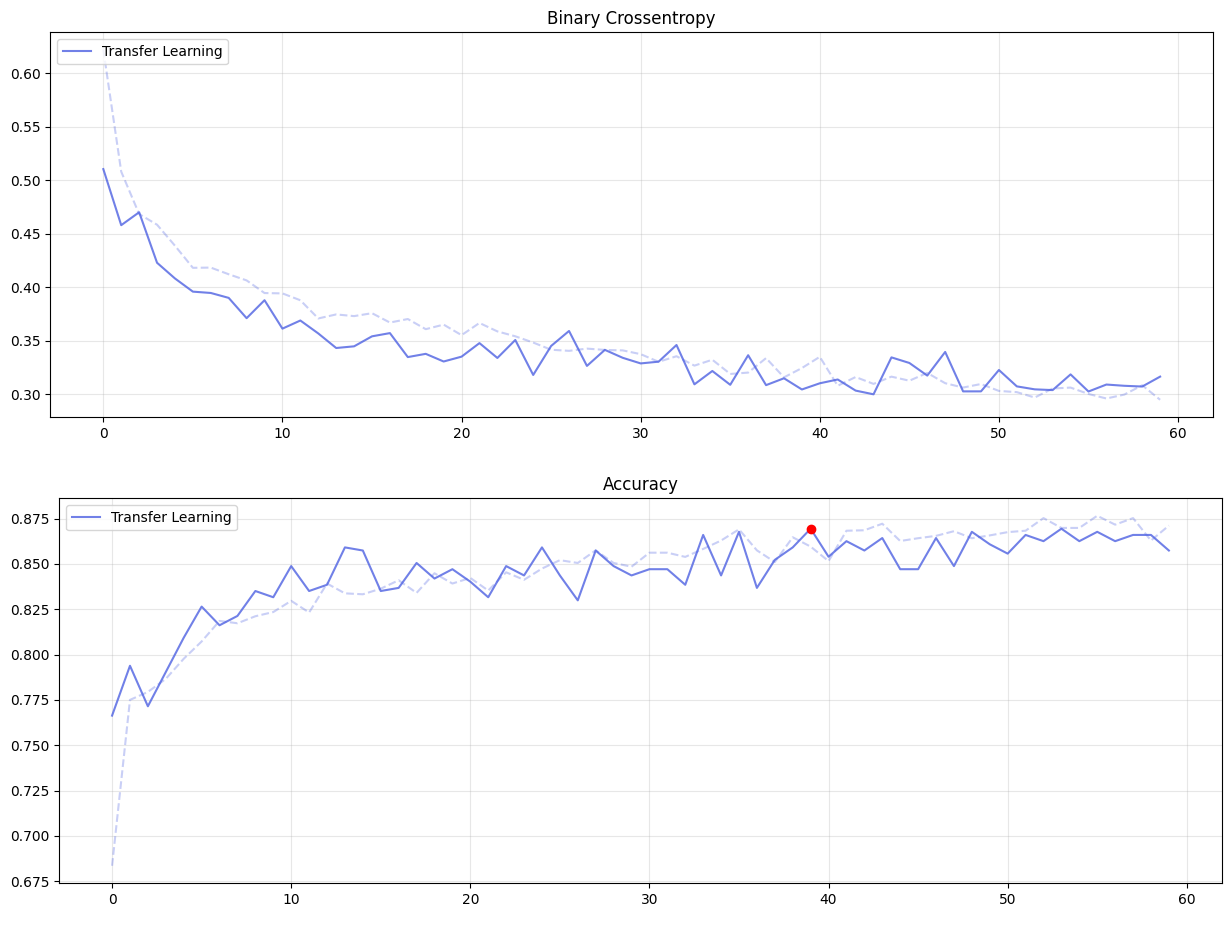
**Our Best Model:**

The process that brought us to our best model is quite linear and reflects what we underwent throughout the last weeks.

Firstly we used EfficientNet[5] to realign our models after the error discovery and also to test the final phase new test set and performance assessed to 76%, not bad for a transfer learning model but this can be easily improved by fine tuning or by using a more complex architecture.

That said we switched architecture in favour of ConvNext and started to produce a solid baseline model for fine tuning exploiting **transfer learning** as we did before.

Here are the ingredients of our **baseline model**:

* **class\_weights** = {0: 0.7918367346938775, 1: 1.3566433566433567}
* **data\_augmentation** (flip, rotation, zoom and translation)
* **Small network** at the end of the ConvNeXtBase pretrained model:
  + **convNeXtBase** pretrained model
  + **GlobalAveragePooling2D**() (that we preferred to the Flatten one because it is able to preserve more spatial features);
  + **Dropout**(0.2) (in order to reduce overfitting and improve model robustness);
  + **Dense**(512, activation='relu')) (in order to capture complex relationships between the input features and the target variable);
  + **Dropout**(0.2)
  + **GaussianNoise**(0.1) (by introducing noise improve the model's ability to learn robust and generalised representations and helps reducing overfitting);
  + **Dense**(2, activation='sigmoid')(x) (making class predictions) (It’s worth noting that we preferred to use “sigmod” as activation function in the last Dense layer instead of “softmax” because it suits better for binary classification tasks).
* **Learning rate scheduler** exploiting cosine decay (learning rate slowly decreasing from 1e-4 to 1e-5 in 100 epochs);
* **Early stopping** (restore best weights with max validation accuracy)
* **AdamW** optimizer (weight decay=1e-4)

The use of an optimizer such as AdamW[4] greatly improved the performances of our models and helped stability and convergence of the training process.

Having a solid performance of 80,0% on the CodaLab test set we were able to start the **fine tuning** process[2].

We **unfreeze** all the layers of the ConvNeXtBase network except for the **batch normalization** ones (also because the network was running in inference mode since we passed training=False when we built the model) and then start the training with the following parameters:

* **class\_weights** = {0: 0.7918367346938775, 1: 1.3566433566433567}
* **Learning rate scheduler** exploiting cosine decay (learning rate slowly decreasing from 1e-5 to 1e-8 in 100 epochs);
* **Early stopping** (restore best weights with max validation accuracy)
* **AdamW** optimizer (weight decay=1e-3)

**Conclusions:**

A small recap on how we approached the challenge:

The first few days we explored the data to better understand the task and how to deal with it and then we proceeded to recover lab sessions and browse the web in order to build a solid knowledge on the subject and on the various approaches to accomplish it.

The next step consisted in a trial and error procedure combined with the knowledge we had.

**Team contributions:**

* **Simone Callegarin:** Worked on almost every section of the project, in particular on the data preprocessing part and transfer learning, also discovered the error. He focused especially in the testing of many keras architectures, the transfer learning process of EfficientNetB0 and ConvNeXt with the greatest results and the report.
* **Sabrina Azzi:** Dataset preprocessing, Dataset balancing, transfer learning on ConvexNeXT, ResNet, VGG19, fine tuning of ConvNeXT, EfficientNet, report.
* **Gianvito Caleca:** Data processing, tested variations of many augmentation techniques, transfer learning on EfficientNetB0, MobileNet, VGG16, fine tuning on ConvNextBase, EfficientNetB0 with great results.
* **Nicolò Caruso:** Tried splitting and balancing data in different ways, tested some more simple keras architectures (MobileV3small, Xception…), tried fine tuning and custom networks.

**References:**

*[1]* [*https://iopscience.iop.org/article/10.1088/1742-6596/1693/1/012148*](https://iopscience.iop.org/article/10.1088/1742-6596/1693/1/012148)

*[2]*[*https://www.tensorflow.org/tutorials/structured\_data/imbalanced\_data?hl=it#calculate\_class\_weights*](https://www.tensorflow.org/tutorials/structured_data/imbalanced_data?hl=it#calculate_class_weights)

*[3]* [*https://keras.io/guides/transfer\_learning/*](https://keras.io/guides/transfer_learning/)

*[4]* [*https://arxiv.org/abs/1711.05101*](https://arxiv.org/abs/1711.05101)

*[5]* [*https://keras.io/examples/vision/image\_classification\_efficientnet\_fine\_tuning/*](https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/)