REPORT HOMEWORK1 - GROUP TheDataArchitects

- The problem:
For the first challenge we were asked to classify plants that were divided into two classes according to their state of health. This problem is a binary classification problem, so the goal is to predict the correct class label in {0 = healthy, 1 = unhealthy}.
Due to the fact that the problem doesn't specify the relative importance of classifying a plant as healthy or unhealthy we set the cut off equal to 0.5 (images with output < 0.5 are classified as healthy and vice versa).
We used Colab as the main tool to build the networks due to its high flexibility in running the code.

- Handling the dataset:
Our first step has been making data pre-processing, which involves loading the dataset, visualizing it and preparing it for training. Firstly, by printing random images from the dataset, we identified the presence of outliers which included images of Shrek and Trololo. We also observed that the images of Shrek and Trololo were the same across the whole dataset. In addition, we noticed the presence of duplicates even between the correct images. Then, we decided to remove the outliers (Shrek and Trololo) so that they could not negatively affect the training process of our model. We also removed the duplicates so that our estimate of the validation accuracy could not be optimistic due to the presence of copies of images both in the training and validation sets. Consequently, we created a Python script which deleted all the outliers from the dataset by comparing each image with one sample of Shrek image and one sample of Trololo image previously selected and all the duplicates between the remaining images. After those elaborations, the initial dataset of 5200 images was reduced to 4850 images *[1]*. Then, we computed the number of images for each class to see the proportions of the dataset. We discovered that the number of images classified as "Healthy" was 3060 while the number of those classified as "Unhealthy" was 1790 resulting in a slightly unbalanced dataset.
In last place, we converted the string encoded labels into a numerical form assigning 0 equals to Healthy and 1 equals to Unhealthy and we splitted the dataset into training (3350), validation (1000) and test (500) sets. After some days, we decided to delete the test set of 500 images because we considered too small the amount of images available and we wanted to include a higher amount of images in the training set.
Also, because we noticed that the accuracy score obtained by predicting our test set was always more optimistic than the accuracy computed by Codalab.
During the first 8 days of the Development phase we didn't address the problem of unbalancing of the dataset being able to obtain 0.88% of accuracy on the validation set, but, then, we tried to balance only the training set by using an upsampling technique based of adding some random duplicates of unhealthy images to the training set. With those changes we were able to obtain 0.90% of accuracy on the validation set with the best network trained until that moment.
We, also, tried to use data augmentation as a regularization technique to artificially expand our training set by applying a series of random, yet realistic, transformations to the plant images. At the very beginning the transformations chosen by us were only rotation, vertical and horizontal translation and vertical and horizontal flip because we wanted to avoid using transformations that would not preserve the input label. Obviously, the augmentation, as for the balancing, was performed only after the split and applied only to the training set. During the Development phase, we tried other augmentation techniques such as zoom, contrast and crop but we obtained bad results (probably because they did not preserve the input label) so we continued using only the initial transformations.

- The accuracy estimation
At first we thought about splitting our dataset in three sets: training, validation and test set. In this way we could have gotten an unbiased estimation of the true accuracy by simply computing the metric on the test set. However, we decided to remove the test set so that we could have a larger training set. We then thought about using cross validation to get a less optimistic estimation of the true accuracy than the one given by the single validation set, but this method proved to be infeasible due to the additional computational cost and the scarcity of computational resources available. In the end we resorted to a single big validation set as a way of computing the estimation of the true accuracy, even though it was

optimistic, due to the fact that we used it to derive the values of the hyperparameters and as a way of stopping training using early stopping.

- The develop plan:
We firstly tried to build a network from zero based on the ResNet34 architecture *[2] [3]* but the validation accuracy estimated during the network training was low and unstable. We thought that this behavior was related to the fact that the available dataset was very small and, so, the network tended to overfit the training data. So, we considered that building a network from zero was not the best idea because it would have needed a higher amount of images to be correctly trained.
Consequently, we decided to apply the Transfer Learning technique and, so, to use a model pre-trained on ImageNet to leverage an existing architecture and cope with the absence of enough data to train a model from scratch.

- Studied architectures and hyperparameters tuning:
Among the architectures available on keras applications we tried the following ones:

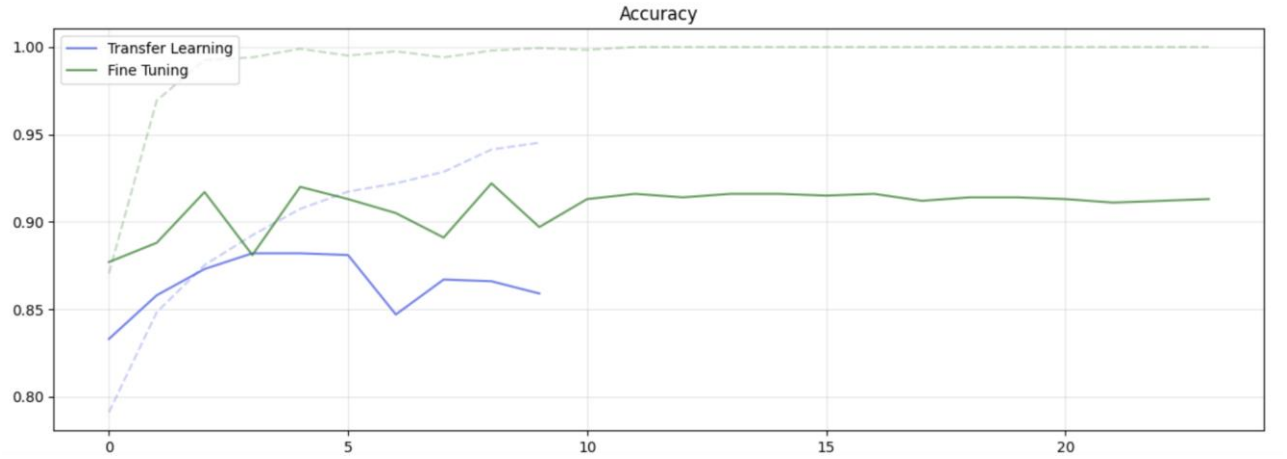
- VGG16
- MobileNetV2
- EfficientNet B0, B2 and V2S;
- InceptionResNetV2
- ConvNeXtBase, ConvNeXtTiny, ConvNeXtSmall, ConvNeXtLarge

We found out that the best architectures, in terms of trade-off between size (MB), validation accuracy and convergence speed (number of epochs before early stopping was applied) were the ones belonging to the EfficientNet family *[4]*. However, the ConvNextBase was the best architecture in terms of validation accuracy which was also stable along epochs during fine-tuning. In the end we chose to submit an ensemble of five ConvNextBase models. The details about the ensemble are at the end.
Each Keras Application pretrained model expects a specific kind of input preprocessing. From the keras.applications library's documentation we found that for ConvNeXtBase preprocessing is already included in the model and it expects their inputs to be float or uint8 tensors of pixels with values in the [0-255] range, which was compatible with our data. Still, we decided to apply different resizing to the images before feeding them into the pretrained model in order to understand if this could positively affect the validation accuracy.
On top of the pretrained model we designed a classifier in the following way: we considered different configurations in terms of dense layers, number of units for layers, drop-out rate and presence or absence of batch normalization. Our final choice, accordingly to the results on the validation set, has been to set these hyperparameters to the following values:
2 dense layers, 64 units for the first one and 32 for the second, dropout rate of 1/8 so that a fraction of input units were set to zero at each update during training and a batch normalization layer between each dense layer and the corresponding ReLu activation.
In the end, regarding the training algorithm parameters, we made the following decisions:

- We used two different Adam optimizers for transfer learning and fine-tuning. The learning rate for transfer learning is higher (0.01) to allow for rapid adaptation of the randomly initialized weights of the network classifier on top of the pretrained model to the new task. In the fine-tuning phase, we decided to set around 60% of the total layers of the pretrained model to remain frozen and to set the learning rate to 0.001 so that the training would not have ruined the weights learned on ImageNet of the pretrained network;
- We considered early stopping and LR scheduler as callbacks to prevent overfitting and adjust the learning rate based on validation accuracy, ensuring efficient and effective training. The number of epochs was set to 10 for transfer learning to adapt the pre-trained model to the new task, and 200 for fine-tuning, allowing the whole model to refine its understanding of plant health.

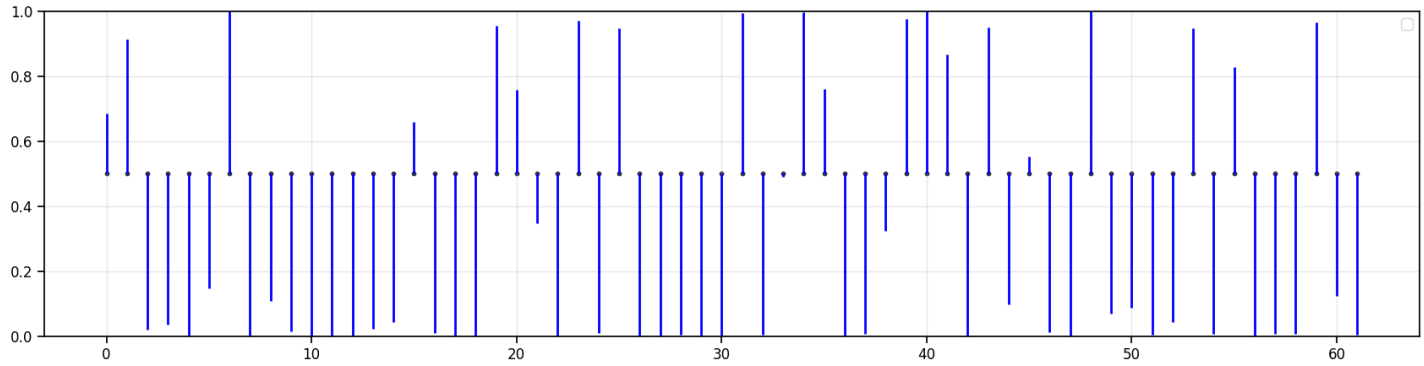
Other details are present in the notebooks associated with this report.
The following image shows the plot of the model accuracy for the ConvexNextBase over epochs for the two phases of training. In the transfer learning phase, the solid blue line represents the accuracy on the validation set, while the dashed blue line the one on the training set. We observe an increase in accuracy

over epochs, which then plateaus. This is expected as the model begins to fit the data well. After the transfer learning phase, fine-tuning is applied where several layers are unfrozen. The solid green line represents the validation set accuracy, which starts higher than the transfer learning phase, indicating that the model has already learned a significant amount from the previous phase. The dashed green line represents the training set accuracy.



- Observation about the errors:

The following image shows the predictions, in terms of probability, given as output of a ConvNextBase model trained with the parameters previously described. These predictions are only the ones relative to misclassified samples in the validation set. From the plot it is clear how the error when misclassifying is relatively large, in the sense that the network seems convinced of a specific (wrong) label giving very high probability or low probability. One of the reasons could be that the network is not capable of learning all the features necessary to decide whether a plant should be classified as healthy or unhealthy.



- Ensemble model:

In the end we decided to submit an ensemble model, consisting in five models, each one with the same classifier described above at the top of the network architecture, but with different model as base:

- A ConvNextBase network *[5]* that takes as input the original images, all with shape (96,96,3);
- Four ConvNextBase networks *[5]* that take as input the images which are first resized with a resizing layer, and then augmented as described in the dataset section. The differences between these four networks is in the way in which the images are resized before feeding them to the augmentation layers : 96x96 (no resize at all), 128x128, 200x200 and 250x250.

When making inference the predicted label of each of the three models is considered and the final predicted label is the one chosen by three or more of the models in the ensemble (majority). In this way it's like we chose to give the same importance to the five models. This is not an issue since the performances in the validation set of the five models were relatively close.

The objective of the ensemble was to increase the overall accuracy, exploiting the different ways in which the five models learn the relevant feature required for classifying correctly the images.

References:

[1] The dataset without outliers and duplicates called ‘clean_collection.npz’ and used in almost all the notebooks created by us can be seen and downloaded by following the link contained in the README file.
[2] Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow, Aurélien Géron, O’Reilly.
[3] The notebook of the model ResNetFromZero is included in the zip file.
[4] The notebook used, during the development phase of the challenge, to create the best model between the ones tried based on pre-trained models of the EfficientNet family is included in the zip file.
[5] The notebook used to create and train the 5 models based on the pretrained model ConvNextBase is included in the zip file.

Contributions:

Antonio Napolitano:

In the initial brainstorming for data preprocessing, I had some ideas on how to handle this phase. After realizing there were outliers (shrek and trololo), I thought about whether it was possible or not to make the removal process automated. Because of the specifics of our dataset, we ended up simply removing those outliers by hand. After realizing there were identical images, I wrote a code to identify the indices of those images.
Concerning the building of convolutional neural network models, I first tried to build by hand a network without using any transfer learning technique, which resulted in a model having a not satisfying accuracy on the validation set. Therefore, after trying several modifications of the hyperparameters of this model, which resulted in small changes in the outcome, I worked on the transfer learning and fine-tuning approach with MobileNetV2. I gave some insights when building networks with other architectures.
Finally, I spent time thinking about all the steps we made, wrote part of the report and helped in refining the rest of it.

Carlo Dell’Orco:

My contribution to the project encompassed various aspects. At the project's inception, during a brainstorming session, our group identified outliers in the dataset provided by Codalab’s competition.
In order to address this issue, our initial approach involved manually inspecting 1300 images to scrutinize the entire dataset. Upon discovering that the same two images consistently posed outliers, I personally devised a small script.
This script, subsequently refined and strengthened by my colleague Federico Caroli, effectively eliminated the outliers from the dataset.
Following this data preprocessing task, I embarked on designing neural networks, given that my team members were already exploring the development of networks based on EfficientNetB2, I undertook the creation of EfficientNetV2-S which yielded our first commendable performance score.
Despite my extensive experimentation with various parameter configurations for the Feed-Forward design, I encountered challenges in elevating the network's evaluation score beyond the 0.88/0.89 threshold.
Notably, the design employed in my EfficientNet was also implemented in the ConvNextBase network that we ultimately submitted.
In addition to these contributions, I also programmed several other networks that were not incorporated into the ensemble, including one based on ConvNextTiny and ConvNextSmall.

Federico Caroli:

I, personally, made experiments, by applying Transfer Learning, Fine Tuning, on the EfficientNetB2 pretrained model and the VGG16 pretrained model. The experiments involved multiple combinations of hyperparameters. However, I didn’t obtain good results and, so, no one of these models have been included in the final ensemble.
Furthermore, I had the task to try multiple and different augmentation techniques (for example contrast, brightness and zoom) to try to identify the ones that let us obtain the best results. Indeed, differently from the lab lectures, I even tried to use the class ImageDataGenerator to perform augmentation, but with poor performance. To conclude, we obtained the highest results either by avoiding to apply augmentation techniques or by applying only rotation, translation and flip techniques.

In addition, I wrote the python script used to delete the outliers and the duplicates from the original dataset (the script is included in the zip file) and the code used to rebalance the training set by adding duplicates of the unhealthy images and I contributed to write the report.

Gabriele Farace:
In the initial phase of the homework I thought about all the possible ways in which we could have obtained a good estimation of the generalization accuracy and the right way of splitting the dataset, proposing the options described in the accuracy estimation section present above. I then built a model from scratch, based on the ResNet architecture (ResNetFromZero Notebook). After trying out different combinations, I switched to a transfer learning framework by trying out the InceptionResNetV2, EfficientNetB0, EfficientNetB7, ConvNeXtBase and ConvNeXtLarge architectures. In this stage I tried different configurations of hyperparameters for each architecture to see what changes could have possibly improved the performance, both in terms of speed and accuracy, finding the optimal way to set the learning rates and the parameters of the callbacks.
Near the end I decided to analyze the errors committed by the best networks when misclassifying samples. Lastly I proposed to use an ensemble of models based on majority voting, writing the script to compute the final predictions of our model, and I gave my contribution in writing the report.