Challenge 1 Report - ANNDL 2023/2024

(1) Andrea Federici, 10621924
(2) Umberto Salvatore Ficara, 10660214
(3) Alberto Pirillo, 10667220

The purpose of this document is to show the results and explain the passages of our study that were adopted to solve the first ANNDL challenge. Firstly, an introductory description of the task and the solution adopted to solve it is presented. After that, the analysis focuses on the data preprocessing operations, which include dataset balancing and data augmentation techniques. Then we discuss the various reasoning we went through when choosing the models' architecture, together with a brief history of our attempts that ends with the final model submitted for phase 2 and its predictions analysis. Finally, considerations on the classification problem are given to show what knowledge we gained from this work.

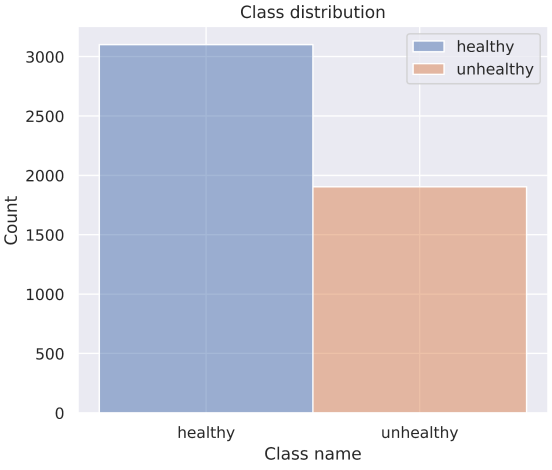
Problem description

In this challenge, we tackle an image recognition problem, a sub-field of computer vision that involves building Machine Learning models capable of interpreting and understanding the semantic information contained in an image. The dataset we were given contains two types of images: healthy and unhealthy plants. Our objective is to be able to categorize the health condition of a plant using only a picture. Therefore, we are working on a binary classification problem.

There are various techniques to address an image recognition problem: we decided to adopt a Deep Learning solution, namely Convolutional Neural Networks (CNNs) which are very suitable due to their ability to automatically extract hierarchical features from the image. CNNs are a type of deep neural network designed for processing and analyzing visual data such as images and videos. They are composed of a feature extraction part in which multiple convolutions between the input image and filters that can be automatically learned from data are performed to extract features and a fully connected part used to classify such features.

Dataset Inspection and Preprocessing

Our work began by inspecting the provided dataset. By plotting all the images in the dataset we noticed the very obvious presence of *outliers*, given that some of the images were not representing plants. These images were removed by hand looking at the plots. We also investigated the class distribution and found that the dataset was imbalanced towards the *healthy* class. Labels were then one-hot encoded.



At this point, we followed the common practice of *splitting the data* into train and validation sets and by doing so, we paid attention to the class distribution by applying *stratified sampling*. We also considered preparing a test set but then we decided to use the hidden test set on CodaLab instead.

With respect to *data augmentation techniques,* we implemented translation, rotation, zoom, flip, contrast and brightness diversification to data by means of a keras Sequential stack of layers. Each layer has a probability of being active or not for each batch given as input. We made the augmentation pipeline part of the model so that it would be automatically turned off during inference. Concerning translation, rotation, and zoom layers, we noticed that particular combinations of them could produce images that were very different from the original ones. We feared that this might lead to the removal of important features from the original images. We found that using these layers with very small modifier values gave us the best results.

The Models Journey

Initially, we created some very basic models, inspired by the LeNet and VGG architectures to have a baseline on the accuracy. Given that we are working on a binary classification problem, the output of our network is given by a Dense layer with 2 neurons and we are using the *CrossEntropy* loss. In this phase, the best model's accuracy was 0.80 on the validation set, while maintaining the training accuracy at 0.84.

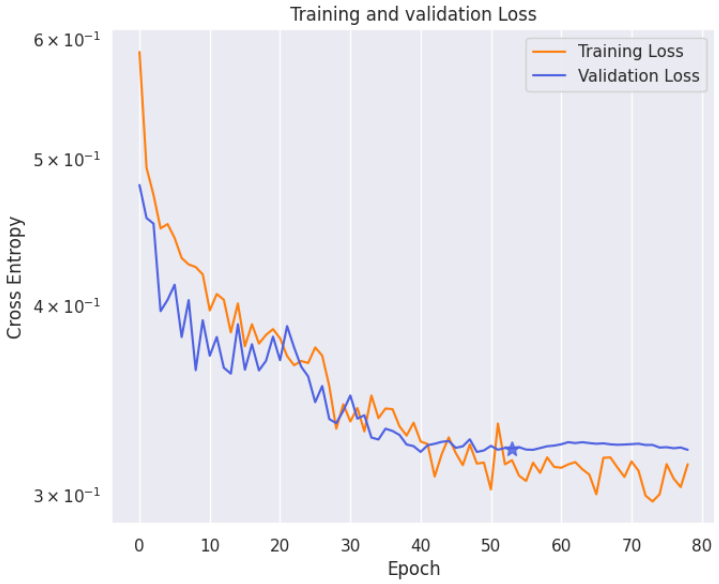
To reduce *overfitting* we used techniques like *Dropout* and *Batch Normalization* layers after every *Conv2D* block and as an optimizer we used *AdamW* which includes a *weight decay* term.
These tweaks were kept for all the following models, together with the addition of class weights to compensate for the imbalanced dataset.

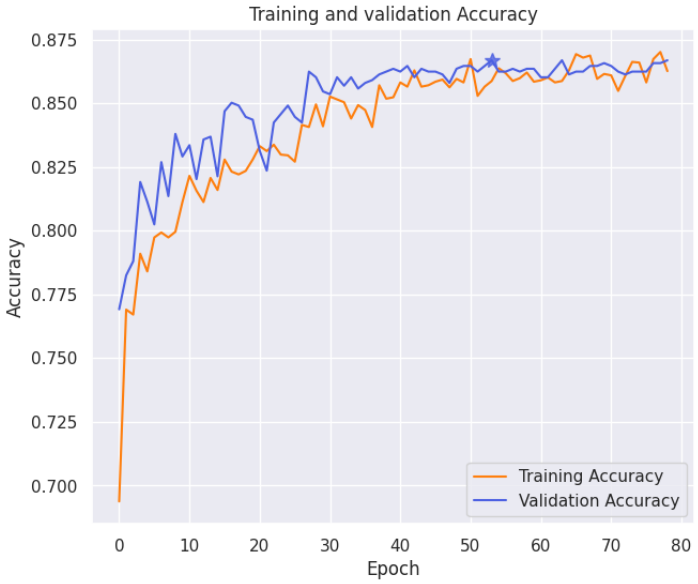
	precision	recall	f1-score	support
healthy	0.9010	0.8806	0.8907	310
unhealthy	0.8131	0.8429	0.8278	191
accuracy			0.8663	501
macro avg	0.8571	0.8618	0.8592	501
weighted avg	0.8675	0.8663	0.8667	501

Despite the addition of class weights, by looking at the classification report generated by *ScikitLearn*, we noticed that the F1 score of the *unhealthy* class was much worse than the one of the *healthy* class, so we decided to act on this by balancing the dataset by *undersampling* the healthy class. Training a model on this balanced dataset resulted in a very similar F1 score for the two classes, but a lower overall performance, due to the fact that the model was being trained on fewer samples. Eventually, given that the metric used in the competition is just the accuracy, and since we noticed that the class distribution of the hidden test set was the same as our training set, we decided not to use undersampling in the next steps.

In order to obtain substantial improvements, we went for *transfer learning*. Each group member worked on different architectures, both to parallelize the training task and to compare the model performances. The most promising results on this classification task were obtained by *EfficientNetV2B0*, *EfficientNetV2S, EfficientNetV2L,* and *ConvNeXtBase* architectures.

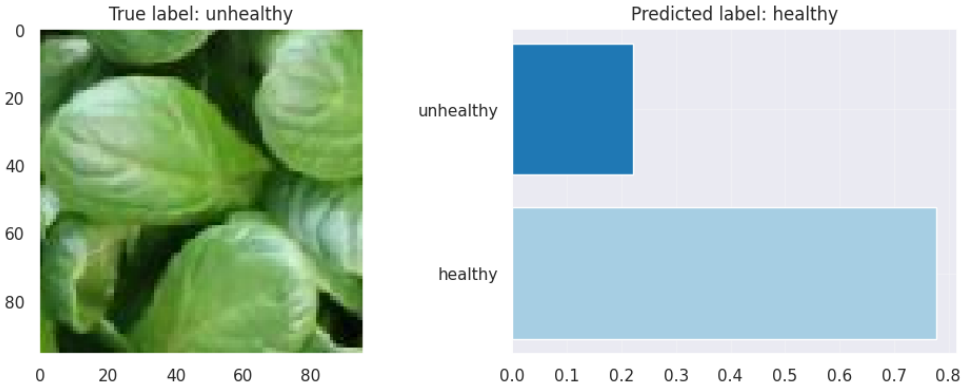
The feature extraction part of these networks was taken as is, and a custom classifier was built on top of each one respectively. In order to build the final classifier, we are using a GlobalAveragePooling layer to extract the latent representation of the base network. The classifier's hyperparameters have been chosen properly both to improve as much as possible the validation set accuracy and according to the shape of the latent representation, in order to avoid having a "too large jump" (in the number of features) between the output of the base network and the final classifier output (which consists of only two values). Each *Dense* layer is preceded by a *Dropout* layer and followed by a *Batch Normalization* layer. Concerning activation functions, we used both *Swish* and *LeakyReLU*. The following graphs show examples of how we monitored performances and overfitting/underfitting during the training phase.





With respect to *fine-tuning*, we had a hard time optimizing most large models, but in the end, we managed to boost the validation accuracy by imposing a smaller learning rate, more regularization and unfreezing only the very last part of the feature extractor.

We also tried to inspect the predictions by hand, to understand if there was some evident pattern in misclassified samples, but we soon discarded the idea because of the lack of domain knowledge. The image on the right is taken from the Prediction Inspection notebook.

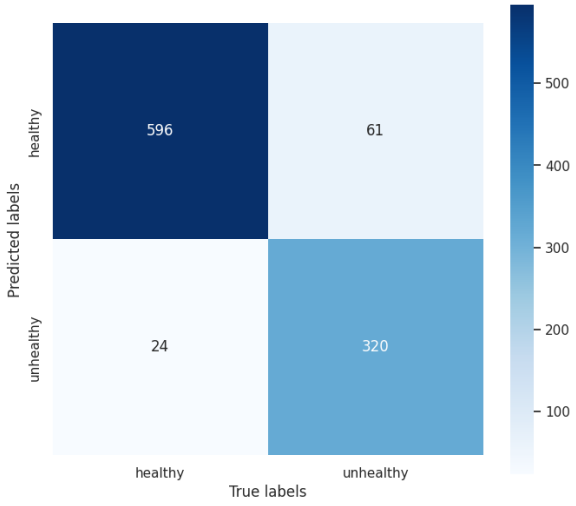


Ensemble of models

Ensemble methods are techniques that combine multiple models in order to achieve better performance. At the end of our work, we decided to use *bagging*, which consists of training multiple models on different data and combining their predictions. Bagging relies on bootstrap sampling to generate an arbitrary number of independent datasets starting from the original one. However, this technique ends up reducing the variance inside of a single dataset since many samples are replaced with duplicates. We noticed that in our case, the datasets generated by bootstrapping did not contain enough samples to train CNNs with good performances. Therefore, we decided not to use bootstrap sampling and trained from scratch the 3 best architectures identified in the previous step.

Then, there are multiple ways to combine the predictions of the model, namely adding together, multiplying together or taking the maximum of the outputs of the individual models. Another possibility is to compute a linear combination of the outputs, using weights determined on the basis of the model's accuracy. We explored all of these techniques and found a great performance improvement. The add and maximum techniques were performing similarly (both with and without weights), whereas the multiply technique was not that effective.

Another technique we tried is called stacking. It consists of using an additional model to learn the optimal weights to be used to combine the outputs of the individual model. In the case of neural networks, this additional model can be a simple *Dense* layer, which can be trained with regular backpropagation by freezing the rest of the ensemble network.



Conclusion

The best model was obtained using the add ensemble on the *EfficientNetV2B0*, *EfficientNetV2L*, and *ConvNeXtBase*. The model provided a 0.89 accuracy score on the phase 1 test set and 0.86 on the phase 2 test set.
The image on the left shows the confusion matrix of the model against the validation set.

We noticed that in order to solve this task we could have also used a different approach: *anomaly detection*, which consists of a set of ML techniques aimed at identifying patterns or data points that deviate significantly from the others. In our scenario, we could have considered the healthy class as the "normal behaviour" and the unhealthy class as an anomaly. It is possible that using recent DL anomaly detection techniques like *Autoencoders* would have led to a significant performance improvement w.r.t. modeling the problem as image classification and using regular CNNs. However, we did not investigate this technique because we realized the feasibility of this approach only in the final days of the competition, and because we had not yet covered this topic during the course's lectures.

Contributions

- **Andrea Federici**: Data Preprocessing, Model Selection (EfficientNet, InceptionNet architectures)
- **Umberto Salvatore Ficara**: Data Preprocessing, Model Selection (ConvNeXt architectures)
- **Alberto Pirillo**: Data Preprocessing, Model Selection (EfficientNet, NASNet, ResNet architectures), Ensembles building and training

With respect to data preprocessing and augmentation, most of the work was carried out together.
As specified before, each group member worked on training different network architectures to understand which one provided the best performance. All notebooks were shared among all group members.