# Image classification

## Task

The objective of this assignment is to build a classifier capable of distinguishing between images of healthy and unhealthy plants. This is essentially a binary classification task, where for each image we must assign a label from two possible categories: "healthy" or "unhealthy". To achieve this, we will be training a Convolutional Neural Network (CNN) on a dataset of appropriately labelled images.

## Data inspection

The first step of our work consisted in analyzing the provided dataset.
Upon a more detailed examination, we identified the presence of outliers and duplicate images. Consequently, our first course of action was to clean the dataset by eliminating these outliers and duplicates, ensuring a more accurate and reliable dataset for our analysis. We then converted the labels using one-hot encoding.


*Figure 1 Outlier "Shrek"*


*Figure 2 Outlier "Trololo"*

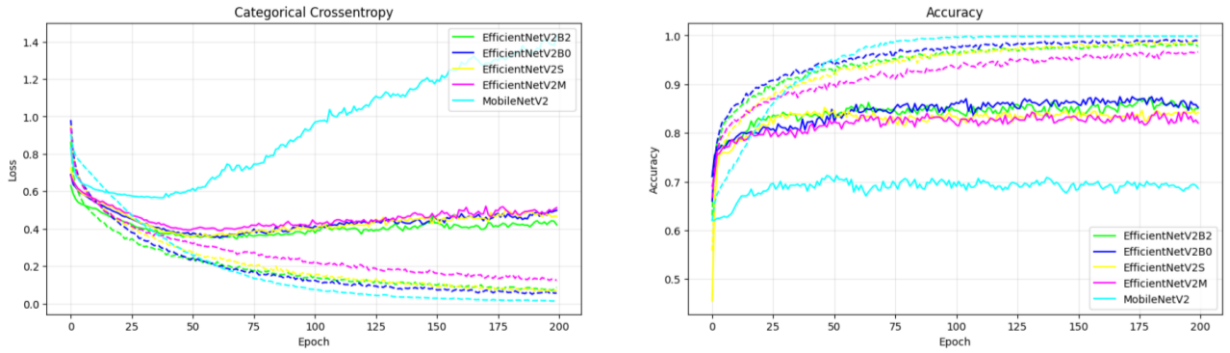

*Figure 3 Duplicated images*

## Unbalanced class instances

When examining the provided dataset, it became evident that there was a significant imbalance in the sample sizes of the classes. This disparity became even more pronounced after the removal of outliers and duplicates, resulting in a ratio of 1 to 1.7 between the two classes. To mitigate this imbalance, we implemented class weights within the `fit` method. This approach is designed to reduce bias towards the more heavily represented class and ensure a more balanced model performance.

## Network selection

For the implementation of our CNN, we employed a Transfer Learning strategy. This involved choosing a robust pre-trained network and adapting it to our specific needs. We achieved this by removing the existing fully-connected (FC) layers at the end of the network and replacing them with custom-designed FC layers, tailored to our classification problem. During this process, we evaluated various networks and experimented with different configurations for the final layers to find the most suitable match.
The effectiveness of these adaptations is demonstrated in the subsequent plot, which compares the results obtained using different networks.




*Graph 1 Comparison of networks (training in dotted line, validation in continuous line)*

The graphical analysis clearly indicates that EfficientNetV2B2, EfficientNetV2B0, and EfficientNetV2S emerge as the most effective networks regarding crossentropy performance. Moreover, EfficientNetV2B2 and EfficientNetV2B0 demonstrate superior accuracy. Based on these evaluations, we have chosen to implement either EfficientNetV2B2 or EfficientNetV2B0 in our frameworks.
Further research on the Keras website [1] confirmed that the number of parameters of these two models aligns well with our available training samples. This is a crucial factor, since an inappropriate number of parameters during the fine-tuning phase could lead to overfitting or underfitting when adjusting the model weights.
The choice of the right number of layers and neurons has been driven by multiple testing, also depending on factors like the type of network and the augmentation techniques adopted.

## Augmentation

To improve the performance of our CNN, we explored several augmentation techniques.

Augmentation techniques help to create a diverse and robust dataset, allowing the neural network to generalize better to various scenarios, thus reducing overfitting. Our initial experiments focused on exploring geometric transformations such as rotation, flipping and translation, which proved to be beneficial.
We also resized and reshaped the images to better fit each network requirements, leading to better performances.

### Mixup

We extended our augmentation strategy to include the Mixup augmentation [2], a domain-independent technique. For the implementation, we randomly selected pairs of images and labels. The output is the combination of the two images, computed using a beta distribution. This mixed sample, along with its mixed label, is then used for CNN training along with the original dataset.

### Test Time Augmentation

We also experimented with Test Time Augmentation (TTA), where we applied rotation, flipping, and translation to test images to create different views and perspectives. Our CNN then produced predictions for each augmented image and the final prediction for the original image was obtained by aggregating the predictions of all augmented versions.
Eventually, we decided to abandon this data augmentation technique given the large computational power required and the negligible improvements obtained in the prediction phase.

## Reducing Overfitting

In order to reduce overfitting, different techniques have been applied.
We decided to adopt both Early Stopping and Learning Rate Scheduling, which respectively allow to interrupt the training when it does not produce positive results and reduce learning rates when a plateau is reached.

### Dropout

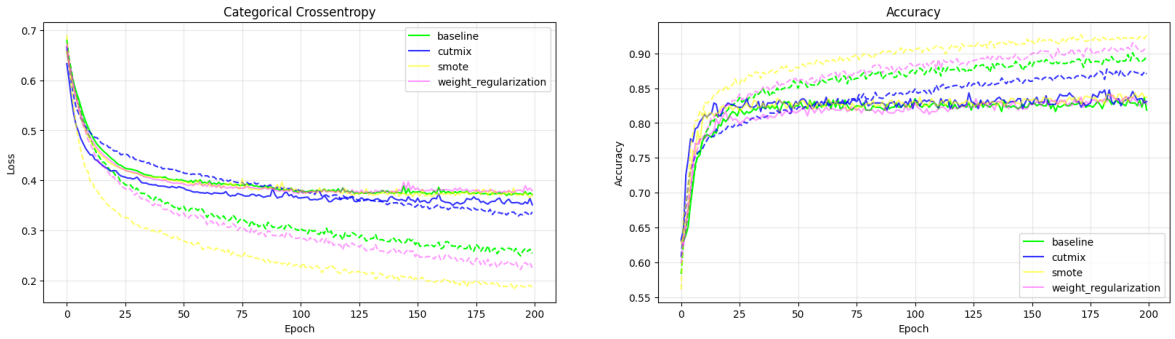
Dropout layers randomly turn off some neurons, forcing the network to learn an independent feature and preventing hidden units to rely on other units.
We adjusted the dropout rates based on the degree of overfitting observed, which was influenced by factors like the quantity of dense layers, the number of units within these layers, and the extent of fine-tuning applied.

### Batch Normalization

Batch normalization in neural network training is a technique that enhances convergence and stability by normalizing the input of each layer across mini-batches, mitigating internal covariate shift and promoting more efficient and accelerated learning. This proved to be beneficial in our specific case, so we adopted it.

## Failed Experiments

During our research, we explored multiple approaches that, even though promising, did not provide a significant performance increase in the context of our problem, as shown in the graph below.
Among these, the most relevant ones have been: Contrastive Learning, a Self-Supervised Learning technique which exploits similarity between pairs of samples [3]; CutMix, a function that combines two samples within a mini-batch by replacing a rectangular region from one image with a corresponding region from another; Synthetic Minority Over-sampling TEchnique (SMOTE) [4], a technique which addresses imbalanced datasets by generating synthetic minority class samples through interpolating feature vectors of existing instances; and Weights Regularization, which introduces a regularization term that penalizes large weights, since small values for the weights have shown to improve generalization of neural networks [5].




*Graph 2 Comparison of multiple techniques (except Contrastive Learning, because of difficulties in plotting)*

## Fine tuning

In our fine-tuning process, we employed a variety of techniques.
Initially, our approach involved unfreezing and fine-tuning each sub-block (such as 6o, 6n, etc.) individually within a larger block, followed by the fine-tuning of the entire block. However, this method proved inefficient for our specific needs, resulting in excessively prolonged durations for fine-tuning a single model without proportionate improvements in performance.
We then explored a recommendation from a Keras guide [6], which advised unfreezing the entire network for simultaneous fine-tuning. Despite its contradiction to our theoretical understanding, we experimented with this approach, only to find it was unsuitable for our objectives.
Ultimately, based on additional guidance [7], we adopted the strategy of unfreezing complete blocks (like block 6, block 5, etc.) and fine-tuning each independently, followed by unfreezing and fine-tuning pairs of blocks. This method significantly accelerated the process compared to the first technique and yielded better results than both previous methods.
For each iteration of fine-tuning, we chose to use the Adam optimizer instead of AdamW, due to the fact we adopted Batch Normalization layers [8], and we set the learning rate to approximately one-tenth of the rate used in previous training or fine-tuning on the same layers.
We also learned from the Keras website [6] that it is essential to keep the Batch Normalization layers frozen (i.e., in inference mode) throughout the fine-tuning process, so we adopted this measure.
Our general strategy involved fine-tuning for a specific number of rounds until performance ceased to improve. Then, we reduced the learning rate ten times more, reaching about one-hundredth of the learning rate of the training, and repeated the fine-tuning until no further improvements were observed. We then proceeded to the next block. When fine-tuning pairs of blocks, we applied a learning rate that was ten times lower than the last rate used in their individual fine-tuning, typically a thousand times lower than the learning rate of the training.

## Ensemble Methods

To make our Convolutional Neural Network (CNN) for plant image classification even more reliable, we explored ensemble techniques.

### Bagging

Our initial exploration led us to implement Bagging (Bootstrap Aggregating) as a technique to reduce the variance of our model. Thus, we trained multiple instances of the same model on different subsets of the training data, drawn with replacement. By aggregating the predictions of these models, we aimed to create a more stable and robust overall prediction. This method did show improvements on our test set, but it instead worsened our results in the Codalab test, so we decided not to adopt it.

### K-Fold Cross Validation

We tried to adopt K-Fold Cross Validation, where we divided the dataset into k parts and trained the model k times, each time using a different part as validation set and the remaining part as training set. This method did not however prove to be effective in our case.

### Other methods- Voting mechanisms

Upon inference, we applied either majority or average voting to the predictions of the individual models. These approaches allowed us to weight the collective wisdom of the ensemble, reducing the impact of bias and uncertainty of the individual models. In the end, we decided to adopt the average vote to account for the confidence scores assigned by each model to the different classes, but while improving performances on our test set, it worsened our results in the Codalab test, so we decided not to adopt it.

## Contributions

Our team distributed the responsibilities evenly, ensuring each member actively participated in every stage of the project. We collectively contributed to the ideation of the models, experimenting with diverse techniques and adjusting parameter values. Since Dario installed Tensorflow and Keras directly on his computer, he leveraged his robust GPU and did not have limits regarding usage time as Gabriele and Alessio did on Google Colab, so he focused more on training the models. To balance the workload, Gabriele and Alessio concentrated on compiling the report, including the relevant code and graphical representations.
In total, each of us dedicated approximately 75 hours to the development of the project.

# Bibliografia

1. Keras.io. *Keras.* [Online] https://keras.io/api/applications/.
2. CutMix, MixUp, and RandAugment image augmentation with KerasCV. *Keras.io.* [Online] 2022. https://keras.io/guides/keras_cv/cut_mix_mix_up_and_rand_augment/.
3. *Weight Rescaling: Effective and Robust Regularization for Deep Neural Networks with Batch Normalization.* Ziquan Liu, Yufei Cui, Jia Wan, Yu Mao, Antoni B. Chan. https://arxiv.org/abs/2102.03497, 2022.
4. *Medium.com.* [Online] 2023. https://medium.com/@fatimazahra.belharar/enhancing-classification-accuracy-for-imbalanced-image-data-using-smote-41737783a720.
5. Kera.io. [Online] https://keras.io/guides/transfer_learning/.
6. *Parameter-efficient fine-tuning of large-scale pre-trained language models.* Ding, N., Qin, Y., Yang, G. et al. s.l. : Nat Mach Intell, 2023. https://doi.org/10.1038/s42256-023-00626-4.
7. *Understanding AdamW through Proximal Methods and Scale-Freeness.* Zhenxun Zhuang, Mingrui Liu, Ashok Cutkosky, Francesco Orabona. 2022.