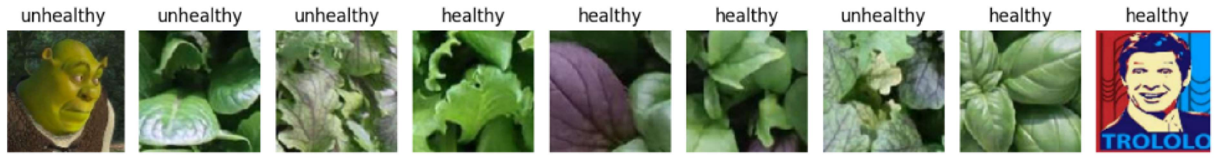
## Introduction

In this challenge, it was required to classify plants that are divided into two categories according to their state of health. It was a binary classification problem. The train dataset contained 5200 RGB images of 96x96 size. As first step, the inspection of the data was done, and after this it turned out that the train data contained *outlier* images. The *outlier* images were all exactly the same throughout the whole dataset, so the *outlier* images were deleted by just comparing pixel values of all other images with pixel values of these 2 unique *outlier* images.



Then, the class distribution of the train data was inspected and it turned out that there were 3101 *healthy* and about a thousand fewer *unhealthy* images. The imbalance was tackled by generating new *unhealthy* images with augmentations of available images. We did random flips, rotations and translations of *unhealthy* images. We ended up with the data which had 3101 *healthy*" and 3101 *unhealthy* images.

Having a balanced dataset is crucial in deep learning as it ensures that the model is exposed to a representative distribution of all classes, preventing bias and enhancing generalization. In a balanced dataset, each class has a sufficient number of samples, allowing the model to learn the underlying patterns of each class effectively, promoting robustness and reliability on unseen data.

Additionally, before the input of every model at training time was placed a data augmentation pipeline. Image augmentation plays a crucial role in deep learning by enhancing the diversity and cardinality of training datasets, thereby improving the generalization and robustness of neural networks. Our image augmentation pipeline involves applying rotations, flips, zooms, and changes in brightness or contrast. Only for the latest trained models additional geometrical and color-affecting image transformations were used as posterization, solarization, equalization, shear transformations and many more. This process helps prevent overfitting and allows to increase the complexity of the network. Image augmentation enables the deep learning model to learn invariant features and patterns, enhancing its ability to recognize and classify images under different conditions, ultimately contributing to the model's overall performance and adaptability.

## Approaches

### Building up a Model from Scratch

At the beginning, an intuitive model is implemented as a first try to gain a better understanding of the level of the task. The network consists of multiple blocks, each comprising a depth-wise separable convolution layer followed by a point-wise convolution layer and a max-pooling layer. The motivation for using this kind of architecture is to improve the generalization of the network while hindering overfitting. The resulting feature maps are globally averaged at each block. The global average pooling outputs from all blocks are concatenated, creating a consolidated feature vector. Dropout layers are applied to prevent overfitting, and two fully connected dense layers with "ReLU" activation are employed for feature transformation. The final layer uses a "softmax" activation function to produce classification probabilities for two classes. This Model is trained on a training dataset of size '4503' samples which resulted in an accuracy of almost 69% on the "CodaLab" test set.

### Transfer Learning & Fine-Tuning

Due to the insufficient accuracy obtained in the previous stage, more advanced and powerful techniques must be utilized for feature extraction and classification tasks. To this end, robust pre-trained models such as EfficientNets, ResNet, and so on are exploited to tackle the problem.

Transfer learning involves leveraging pre-trained models on large datasets for a particular task and then adapting them to a new, related task with a smaller dataset. It typically requires substituting the classification layers on top of the network with newly initialized layers for the new task, train them with high learning rate but freezing the feature extraction part of the pre-trained model. Fine-tuning is a further step in transfer learning, where the pre-trained model is adjusted on the target task's dataset. This process refines the model's parameters to align more closely with the specific features and nuances of the new dataset, allowing the model to specialize and excel in the desired task.

This methodology was applied to a plethora of models. Firstly, a deep classifier was trained on top of the frozen pre-trained model. Regarding the fine-tuning process, it was done iteratively unfreezing more layers at each iteration in order to find the optimum training depth for a given architecture. In this way, it was possible to reach 83% accuracy on Codalab test set during phase 1.

## Self-supervised Learning

The latent representation of a CNN constitutes a compact and expressive encoding in a lower-dimensional space, encapsulating the image's essential characteristics. Is the latent representation derived from a fine tuning process the best latent representation for a classification problem? Is it possible to train a network to produce the latent representation we need to solve a problem?

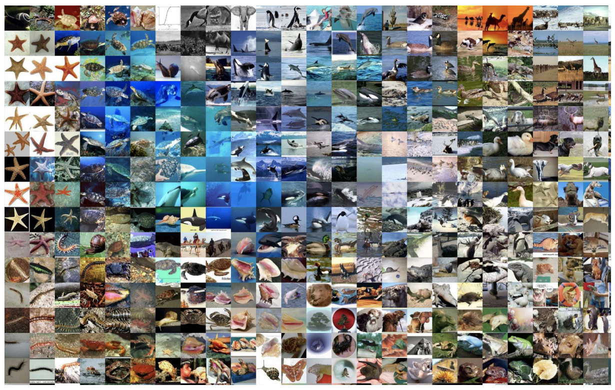


Figure 1: Example of clustered images using their latent representation

In order to answer those questions the team developed two additional architectures (only the second one is described in this report) whose key idea revolves around the assumption that similar inputs should have a close vectorial representation, while dissimilar inputs should be far apart in the embedding space. The common elements between the two architectures are the use of more than one input and a network processing them separately to create a vectorial representation . It is important to underline that there is only one encoding network which is applied to each input. Then a similarity metric is computed in order to train the architecture in such a way that the distance between the representations of similar inputs is minimized, while the distance between representations of dissimilar inputs is maximized. After a working encoding network is obtained, a classification network is built on top of it to solve the classification problem.

The second architecture is an extension of the first one and incorporates triplets of samples during training. Instead of comparing pairs of samples, the network works with triplets, consisting of an anchor sample, a positive sample (similar to the anchor), and a negative sample (dissimilar to the anchor). The network is trained to learn a representation space where the anchor-positive pairs are close, and the anchor-negative pairs are far apart. For this architecture there is no need for labels as long as the triplet is correctly chosen during training. Denoting A, P, N the anchor, positive and negative samples embeddings the contrastive loss for the triplet network can be defined as:

$$L(A, P, N, m) = \max(0, D(A, P) - D(A, N) + m)$$

where D and m are the distance between the samples (for example Euclidean distance or L1 norm) and m the margin. The values of m seemed only to scale the magnitude of the loss function for the encoding network and to have little effect on the representational power of the network and on the classifier accuracy so it was set in the range [0.5,1].

The encoding network can be arbitrarily designed. Our design choice was to use one or more branches with pre-trained models with final average pooling layer and a projection network to the vectorial dimension desired. The class of 'EfficientNetV2' was used as pre-trained models. The architectures were divided in blocks and only the last block was fine-tuned leaving batch normalization layers as frozen. In case of several branches the vectors coming from each of them were summed together. The addition of more branches showed an increase in the representational power of the network and an improved clustering of the embeddings of the input images. In order to train the encoding network, the Adam optimizer was used with learning rate equal to 1e-4. Moreover, since employing pairs of large pre-trained networks or even triplets increased the computational effort and time needed for training, it was decided to employ a mixed-precision computation. In deep learning models, parameters and activations are typically stored using 32-bit floating-point numbers (single precision, or float32). However, using lower precision, such as 16-bit floating-point numbers (half precision, or float16), can lead to significant speedup in training without sacrificing too much model accuracy.

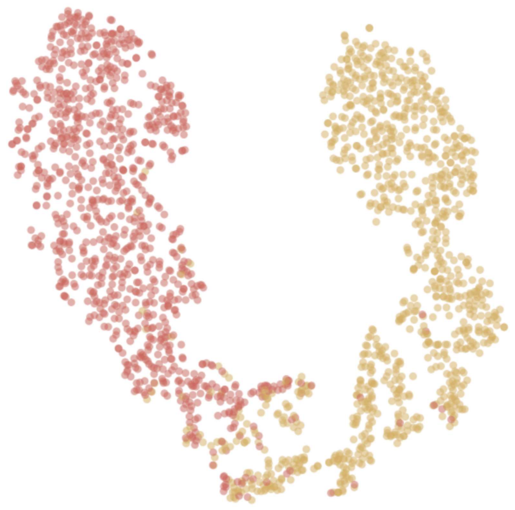
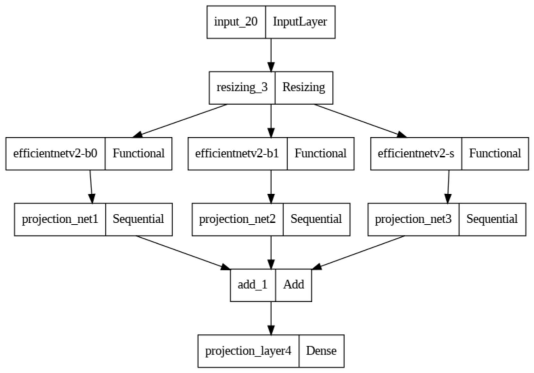


Figure 2: TSNE representation of the learned 20 dimensional embedding vectors for 2000 samples of the training set


input_20 | InputLayer
resizing_3 | Resizing
efficientnetv2-b0 | Functional
efficientnetv2-b1 | Functional
efficientnetv2-s | Functional
projection_net1 | Sequential
projection_net2 | Sequential
projection_net3 | Sequential
add_1 | Add
projection_layer4 | Dense
Figure 3: Example of encoding network. The output of the last layer is used to compute the distance metric

For the final classification layer, we noticed that if the training of the encoding network was successful it was enough to add a couple of dense layers, dropout to prevent overfitting and as output the two softmax neurons. Deeper and more complex classification architectures were trained giving only a little performance gain at expense of increased computational resources and time. As instance, a feedforward network made up by m dense layers repeated N times where the m-th layers are connected to each other through skip connections in a DenseNet-like way was analyzed. For this architecture the library keras-tuner was used to tune the number of m layers, their N repetitions and the number of hidden neurons used in each layer. Additionally, in order to have an overfitting free solution and completely rely on the embeddings representation a classification with the k-nearest neighbors algorithm has been tried. For this solution every data we had and the test dataset have been mapped in the embedding space through an encoding network or summing the vectorial representations of an ensemble of encoding networks and the labels of each test sample have been assigned as the majority label in their neighbors. This architecture with a shallow top classifier achieved 89% precision on Codalab during phase 1.