Homework 1 - Image classification

The following project aims to build a Deep Learning model able to differentiate healthy and unhealthy plants based on pictures. As this challenge is related to supervised training and binary classification, one decided to implement a Convolutional Neural Network (CNN) architecture widely acknowledged in the literature as one of the most performing types of neural network for image classification. The efficiency of the model would be assessed by the analysis of its performances (accuracy, precision, recall, and F1 scores).

I. Dataset overview and preprocessing steps.

The dataset was composed of 5200 squared RGB pictures of plants with a resolution of 96x96 pixels. Background removal was already applied to the pictures, which are partitioned into 2 classes : Healthy and Unhealthy leaves. A significant imbalance in label distribution was observed: “Unhealthy” data class represented 38% of total data.

A manual inspection of the dataset helped detect the presence of outliers. As there were no clear parameters such as mean pixel value to distinguish outliers from clean data in an automatic way, the outlier removal was made by hand. Outlier images represented about 4% of the dataset, and were equally partitioned between the 2 classes as shown in ***Fig. 1***. Removing the outlier did not change the unbalance in class population.

To solve the imbalance between the number of images in the two classes, we artificially created new samples in the minority class (unhealthy) by using image augmentation techniques. Random spatial transformations (translation, flip, zoom) were applied to the unhealthy images of the dataset to generate new samples from the same class. The number of freshly generated ‘unhealthy’ samples was decided in order to balance the class population of the 2 classes. After this step, both classes of the dataset are equally represented (see ***Fig. 2***).

Categorical labels “healthy” and “unhealthy” were then converted into binary vectors, ensuring the network's compatibility with softmax activation functions and improving its capacity to discern and classify diverse classes.

During the whole development phase, a partition of the dataset was used with training/validation (resp. 70% and 15% of the initial re-balanced dataset). The last 15% were preserved for testing the model at the end of every development phase.

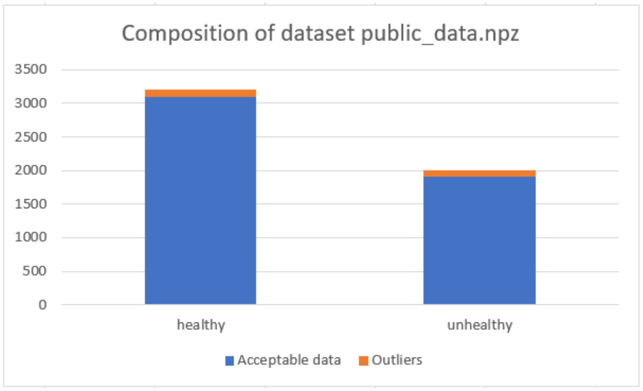


***Figure 1** - Inspection of public_data.npz*

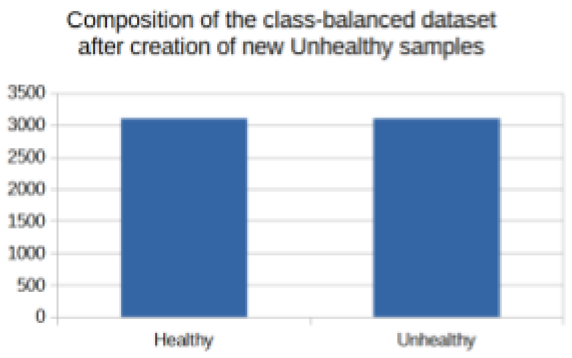


***Figure 2** - Representation of the population of each class after doing class-targeted image augmentation on the Unhealthy samples to rebalance the dataset*

II. Choice of the model

The next step towards the building of a classifier was the choice of the model to implement. The selection of the models was made based on several criteria such as accuracy and number of operations to perform[1]. A few models have caught our attention.

[1] Boracchi et al., “Famous CNN Architectures and CNN Visualization” (2023), slide 96.

In our search for a suitable CNN model, a DenseNet architecture was considered for its high accuracy in comparison studies[1]. DenseNet is characterized by its short connections of the convolutional layers and the connection of each layer to every other. A DenseNet-169 model was implemented layer by layer.

The quasi-VGG9 architecture was tested. As a sub-model of the VGG, it is known for its smaller filters and deeper network. It is characterized by its high amount of filters and use of global average pooling on each channel, which make it a straightforward and robust CNN architecture[2].

Because of the high computational time for training from scratch the former models, Transfer Learning was investigated for improving our results. ResNet and Inception V3 appeared to be the best compromises between accuracy and number of operations. Indeed, both of them are supposed to have on average an accuracy of 0.77 for 7 G-ops (for ResNet) and 12 G-ops for (Inception V3)[3], with a reasonable number of parameters (less than 25 millions). Both models were implemented using transfer learning and pre-trained weights from imageNet.

Eventually, the results in accuracy were compared for all the 4 models. ResNet50 with transfer learning was giving the best results (see ***Fig. 3***) for shorter computational time, and hence was selected as the only model for the rest of the competition.

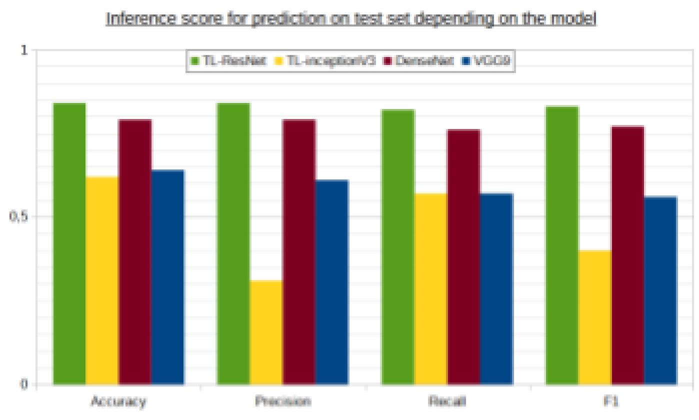
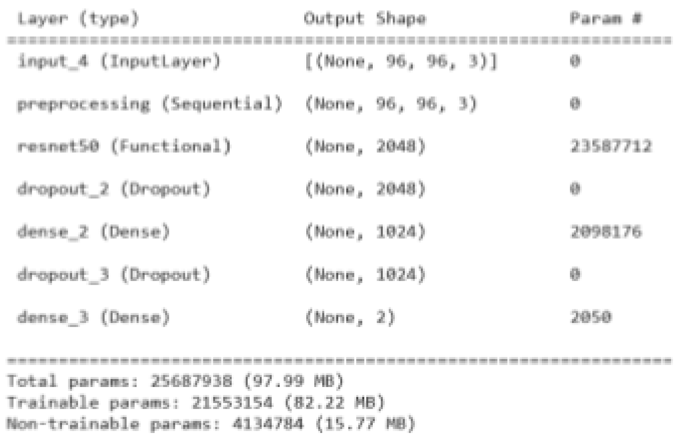


Figure 3 *- Inference score on testing set for several models.*



```
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 96, 96, 3)]       0

 preprocessing (Sequential)  (None, 96, 96, 3)         0

 resnet50 (Functional)       (None, 2048)              23587712

 dropout_2 (Dropout)         (None, 2048)              0

 dense_2 (Dense)             (None, 1024)              2098176

 dropout_3 (Dropout)         (None, 1024)              0

 dense_3 (Dense)             (None, 2)                 2050

=================================================================
Total params: 25687938 (97.99 MB)
Trainable params: 21553154 (82.22 MB)
Non-trainable params: 4134784 (15.77 MB)
```

Figure 4 *- Summary of our final ResNet model implemented with transfer learning*

III. Inductive transfer learning and fine tuning

Tensorflow.keras.applications library features several models to perform transfer learning, including ResNet50, ResNet101 and ResNet152. When using one of them, the reference database for weights values have to be entered. The transfer learning implemented for this task is relying on the weights of ImageNet database.

Preprocessing was applied to the inputs via the preprocessing function specific to ResNet50 available tensorflow.keras.applications. The function was also added to the model when submitting in the model.py file.

The batch size was chosen to be 128 to minimize the needed computational time[4]. In addition to the ResNet50, a dense layer featuring 1024 neurons has been defined in order to investigate a possible improvement of the accuracy of the classifier (see ***Fig. 5***).

Eventually, fine-tuning was used on the transfer-learned model. Indeed, this method has the weights of the pre-trained model on a large data set used as the starting values for training a new network[5]. In our case, 100 layers were frozen in the pre–trained model by preventing it from updating its weights during the fine-tuning process. To choose the amount of layers to freeze, a loop was created to test the accuracy of different percentages of frozen weights. Then, the maximum value of N was taken so the fine-tuning is optimal.

Description	Accuracy	Recall	Precision	F1

[2] Roman V.,"CNN transfer learning and fine tuning" (2020).
[3] Boracchi et al., "Famous CNN Architectures and CNN Visualization" (2023).
[4] Leslie N. Smith, "A disciplined approach to NN hyperparameters"(2018), US Naval Research Laboratory Technical Report
[5] Bareto S., "What is Fine-Tuning in neural network" (2023)

Transfer Learning ResNet50 only	0.7989	0.7989	0.7998	0.7987
Transfer Learning ResNet50 with dense layer (1024 neurons)	0.8156	0.8156	0.8168	0.8154
Transfer Learning ResNet50 with dense layer (1024 neurons) and fine tuning (see ***Fig. 4***)	0.8456	0.8456	0.8525	0.8448

Figure 5 *- Summary of scores obtained for transfer learning with ResNet50 and its versions.*

IV. Preventing Overfitting

In addition to the improvement of accuracy, overfitting was one of the main concerns during this project. Indeed, despite image augmentation, the overfitting phenomenon observed was quite consequent. Thus, to prevent potential difference in accuracy while submitting the model, early stopping was implemented with a patience set to 10. A learning-rate scheduler with factor = 0.999 and patience of 2 was tested, but did not do any improvements so this was discarded.

Dropout layers with a rate of 0.6 have also been added after ResNet50 and the dense layer to prevent overfitting. With dropout, after a certain epoch, the difference in accuracy between training and validation would remain constant, as it used to keep on increasing without dropout.

V. Conclusion and possible improvements

For this project, different CNN architectures were tested (quasi-VGG9, DenseNet, ResNet, Inception V3), and ResNet50 was implemented via transfer learning. Even though the efficiency ought to be improved, the accuracy remains quite satisfying (0,8456). Fine tuning, overfitting prevention and data augmentation were added to the work and showed a slight influence on the increase of accuracy. A lot can still be done to improve it. An idea would be to increase the number of neurons, even though this might increase computational cost. Getting more training data could help decrease misclassification on the training set. In addition, Datasets better suited to the problem can be used for transfer learning. Furthermore, additional dense layers at the end of the model could help improve the trainability of the model. Finally, other CNN models available in tensorflow.keras.applications such as ConvNext or VGG19 could be more performant on this type of task.

Contributions	**Contributor**
Outlier removal, class re-balancing, preprocessing, test for improving final model, prepared model.py for submission. Optimization of the final model by tuning parameters. Writing the report.	Madeleine
Project management, previous tests on models for transfer learning (Inception V3), implementation of transfer learning ResNet50 (classic and with extra dense layer), overfitting prevention (dropout, early stopping adaptation), report writing.	Alexandra
Implementing the model VGG9. Testing ResNet50 with an extra dense layer. Working on fine tuning and data augmentation (research and implementation). Writing the report.	Diana
Preprocessing of the data set. Implementing the test and evaluation structure for local tests. Implementing the DenseNet model. Working on fine tuning and data augmentation (research and implementation). Writing the report.	Ibrahim Ait El Hend