Convolutional Neural Network: Healthy *vs* Unhealthy Plants Classification

Artificial Neural Networks and Deep Learning - A.Y. 2023/2024

Luigi Pagani*, Flavia Petruso† and Federico Schermi‡

M.Sc. Computer, Mathematical, and High-Performance Computing Engineering Politecnico di Milano - Milan, Italy

*Email: *luigi2.pagani@mail.polimi.it, †flavia.petruso@mail.polimi.it, ‡federico.schermi@mail.polimi.it*

*Student ID: *10677832, †10544566, ‡10776811*

Codalab Groupname: "RandNet"

1. Introduction and project goal

The provided dataset is composed of 5200 images, representing healthy and unhealthy **plants** (leaves). Each image is a 96*96*3 tensor, with the first two dimensions indicating the pixel size and the third the RGB channel. The project goal is to train a **classifier** that is able to discriminate between the two classes.

2. Data preprocessing

The first step we carried on was data inspection and cleaning. This led to the identification of two types of **outliers**, for an overall number of 196 outliers that were removed from the working dataset. After this step, we had 3101 healthy and 1903 unhealthy plants. We then performed a **train-validation** split of 80:20. For both phases of the challenge, we used our validation set as a test set.

3. Data augmentation

To increase the number of images and improve the generalizability of our results, we performed **data augmentation**. After a brief study of the most reasonable transformations to apply to our data, we opted for a mix of **classic transformations** (including shifts, rotation, flip, translation and brightness) within a range that did not distort completely the images or impacted on class invariance, as reported in the dedicated notebook.

4. CNN

4.1. Feature extraction layers

To determine the structure of the feature extraction (FE) layers, after trying some basic CNN structures we decided to opt for a **transfer learning** approach. To choose the network with the best performance, we benchmarked several architectures, namely ConvNeXtXLarge, EfficientNetLarge, VGG16, and NASNetLarge. In all these networks we introduced the **preprocessing** using the appropriate model function. At first, our main focus was the performance of the feature extraction architecture rather than the classification layer. Therefore, we tried to use a simple layer with 512 neurons with batch normalization and `relu` activation function, followed by a dropout layer of 0.3. We set 40 as the number of epochs. In **Table** 1, we report the validation accuracy of the best epoch. In this step, the FE layers were kept frozen, using the pre-trained weights from *ImageNet*.

Model	Maximum validation accuracy
VGG16	0.794
ConvNeXtXLarge	**0.891**
EfficientNetV2L	0.856
NASNetLarge	0.765

TABLE 1. MAXIMUM VALIDATION ACCURACY FOR DIFFERENT FEATURE EXTRACTORS.

In **Figure** 1, we show how the validation accuracy and loss vary over the epochs. From the figures, it appears that the final validation accuracy is already reached within the first 10-20 iterations, and then plateaus around a value for the rest of the epochs.

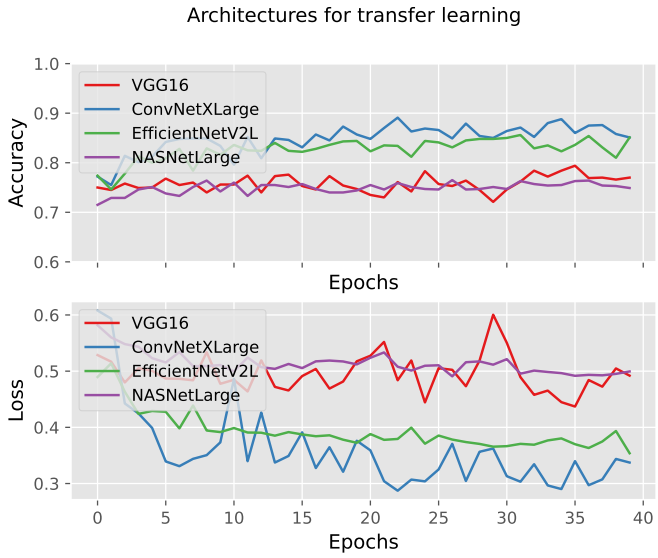

Figure 1. Performance of different feature extractors.

Based on the provided results, we decided to use ConvNeXtXLarge as our feature extractor.

4.2. Classification layers

To **connect** the feature extraction layers to the classification layers, we tried both a `GlobalaveragePooling2D()` and a `Flatten()` + `Dropout()` approach. While the former was faster due to the average operation, the latter gave better validation accuracy, therefore we opted for that one.

We tried to balance the model's complexity and efficiency. After some experimentation and with the help of Keras tuner Hyperband algorithm (see [3]) which repeatedly suggested either 1-3 layers, we opted for 2 dense layers, one of 1024 and another one of 512 neurons. After the second layer and before the output one, we introduced a further `Dropout` layer of 0.3. For both layers, we used the `relu` activation function.

As an **output layer**, we used a single neuron with sigmoid activation, due to the binary nature of the classification task.

In **Table 2**, we report the final configuration of the layers after the FE network.

4.3. Fine tuning

We used fine-tuning to retrain the weights of the FE network. While we kept the same type of optimizer, for better convergence purposes we lowered the learning rate at $5 * 10^{-5}$. As a fine-tuning approach, we decided to unfreeze all the layers except for the `BatchNormalization()`. the model reached a validation accuracy of 94.60%.

Layer
Flatten
Dropout 0.4
Dense, 1024, relu
Dense, 512, relu
Dropout 0.3
Dense, 1, sigmoid

TABLE 2. FINAL ARCHITECTURE OF THE LAYERS FOLLOWING FEATURE EXTRACTION.

4.4. Regularization strategies

To reduce the risk of overfitting, we employed different regularization strategies. First, we performed **data augmentation** as explained in **Section 3**. Then, as a general rule (in this and the other models tried), we decided to add a `Dropout()` layer after the `Flatten()` layer and any dense layer, even if in some cases we decided to remove it based on empirical results. The dropout levels were set as 0.4 and 0.3 respectively (even supported by the results of our analysis performed with Keras tuner). Finally, we used **L1-L2 regularization** (Elastic-net type) of 10^{-3} to introduce a bias in the loss function and reduce the variance.

5. Self Supervised (SimCLR) Learning model

Due to the limited size of the dataset, we decided to try a **self-supervised learning** approach. In particular, we explored the self-supervised *SimCLR* strategy described in [1]. For a general overview of the method and its rationale, please refer to the dedicated notebook. Concerning the parameter choice, we opted for a batch size of 1024 due to memory constraints. While larger batch sizes are generally preferred, compensating with a high number of epochs can mitigate the impact of the relatively small batch sizes. Consequently, we conducted training until convergence to 500 self-supervised epochs. As a projection width, we selected 128 to adhere to our computational constraints.

As the **encoder**, we chose *ResNet-18*, randomly initialized, and implemented image transformations suggested in the original paper. These transformations

include cutout, color deformation, Gaussian blur, and flip.

To set the **temperature** parameter, we relied both on experimentation and what was suggested in [2] (see paragraph B-8). Based on this approach, we finally set a temperature of 0.25.

This network, pretrained via self-supervised contrastive learning, achieved a validation accuracy of 88.70%. Therefore, we retained the previously described convolutional model as our final choice due to its superior performance.

6. Shared parameters, metrics, and approaches

Here we report the main parameters we chose and their rationale.

- **loss function**: due to the binary nature of the classification tasks, we used *binary crossentropy*.
- **metric**: for the same reason as above, we used the *binary accuracy*.
- **optimizer**: after some investigation on *RMSprop* and *Adam* as possible optimizers, the results were similar and we opted for the latter, with 10^{-3} learning rate.
- **precision**: for most of the models attempted we used the the *mixed precision* to reduce memory requirements by using lower-precision data for part of the training in an adaptive way. We did not implement it in the self-supervised model, as in that specific scenario the default settings of TensorFlow did not support the use of contrastive loss in mixed precision.
- **weights**: to deal with the problem of class imbalance, we decided to assign weights to the classes for a **weighted loss function**, in a way that was inversely proportional to the frequency of each class in the dataset. Depending on the model, this step allowed to obtain an increase of the F1 score of the under-represented class and a general increase of 1-2% accuracy.

7. Test time augmentation

After the model was chosen, we relied on test-time augmentation as a strategy to improve the test performance. We used some trial and error efforts, we used 32 as a batch size and set as 50 the number of images generated from each original one. We saw that on average the increase in accuracy was 0.6%.

8. Results and performance

Here, we report the accuracy of our best model from the training and validation set.

Looking into **Table 3**, the drop in accuracy registered on the test set suggests that our model might, while still presenting a good performance, have benefited from more techniques to improve its robustness.

Accuracy of the final model	
Train	99.90 %
Validation	94.60 %
Test	86.70 %

TABLE 3. BINARY ACCURACY ON DIFFERENT DATASETS

We also report the full metrics of the test set.

Performance on the test set	
Accuracy	86.70%
Precision	83.11 %
Recall	81.58 %
F1-score	82.34 %

TABLE 4. PERFORMANCE METRICS ON THE HIDDEN TEST SET.

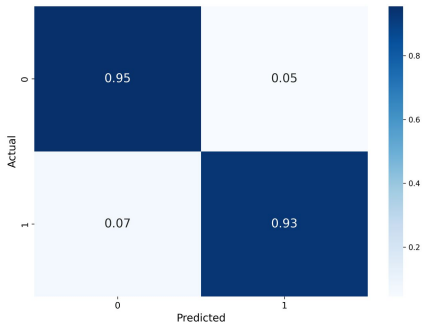

Figure 2. Confusion matrix of the model performance on the validation set.

9. Potential improvements

One approach concerning model training that we did not address consists of finding a good architecture using the train-validation split, and then once the architecture is defined re-train the model using the full training set, setting as early stopping epoch the one identified in the previous runs, based on monitoring the chosen validation metric, as suggested in [2] (Section 7.8).

Another improvement could consist of using more advanced augmentation techniques, *RandAugm*, *Cutmix*, and *CutOut*, to increase model robustness.

References

[1] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[3] Lisha Li et al. “Hyperband: A novel bandit-based approach to hyperparameter optimization”. In: *The journal of machine learning research* 18.1 (2017), pp. 6765–6816.

Contributions

- **Luigi Pagani**: model conceptualization, model development (data inspection and preprocessing, choice of the classification layers, hyperparameter tuning, self-supervised model), report writing.
- **Flavia Petruso**: model conceptualization, model development (data augmentation, benchmarking of the FEN, hyperparameter tuning, choice of the classification layers, test-time augmentation), report writing.
- **Federico Schermi**: model conceptualization, model development (outliers removal, data augmentation study, choice of the classification layers, hyperparameter tuning, fine-tuning, test-time augmentation), report writing.