Artificial Neural Networks and Deep Learning

Report - Challenge 1

Alessandro Ferri	Laura Colazzo	Ignazio Neto Dell'Acqua	Zanubia De Pasquale
10578145	10705522	10704869	10668387

A. Y. 2023/2024

1. Introduction

The primary goal of our initial assignment was to develop and train a Convolutional Neural Network (CNN) for a binary image classification task, leveraging a provided dataset. This dataset comprised labeled images of plants categorized as either healthy or unhealthy. This report dives into the iterative process of designing and evaluating the models that eventually led to the submission of our final model.

The structure of the report unfolds as follows: Section 2 provides insights into the preliminary phase, including the analysis, cleaning, and preprocessing of the input dataset. In Section 3, we go deeper into the details of our first classification attempt, employing a handcrafted CNN, yet facing challenges as the results fell short of expectations. Section 4, on the other hand, shifts focus to the effectiveness of pretrained models in tackling the classification task. This involves exploiting transfer learning and fine-tuning techniques, emphasizing the selection of models that exhibited the most promising results. The results of our explorations are presented in Section 5, where we detail the model that outperformed others on the competition test set.

Through these sections, we aim to provide a comprehensive understanding of our approach, challenges faced, and the ultimate success achieved in the pursuit of an optimal plant classification task.

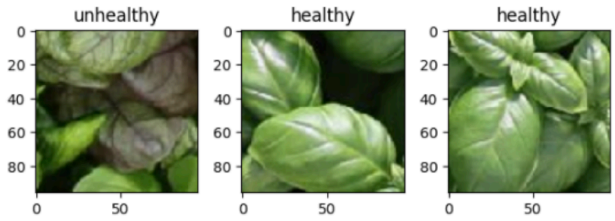

Figure1: Samples from the input dataset

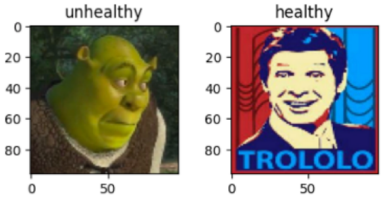

Figure 2: Classes of outliers

2. Data inspection and input preprocessing

The dataset comprised a collection of plant pictures exhibiting details of the leaves, categorized as either healthy or unhealthy (see Figure 1). However, it exhibited class imbalance, with uneven representation across the two classes. Additionally, there were two distinct types of outliers present in the dataset, as illustrated in Figure 2.

2.1. Preprocessing

The state of the dataset required some preprocessing interventions.

2.1.1. Clustering

To eliminate outliers from the input, a clustering algorithm was employed. This facilitated the partitioning of images within the dataset into three clusters: the plant images cluster and the two outlier clusters. Subsequently, all outliers were removed.

2.1.2. Class weighting

The class imbalance became apparent in the initial classification results, by looking at the confusion matrix, where models struggled to accurately classify unhealthy plants. To address this issue, we balanced the classes by assigning customized weights: the underrepresented class (unhealthy plants) received a higher weight (1.31), while the more prevalent class received a lower weight (0.88).

While experiments involving class weighting did not result in a drastic improvement in performance, the technique proved beneficial in mitigating the aforementioned bias.

2.2. Data augmentation

A good practice, especially with a limited dataset, is to implement data augmentation. This prevents the neural network from learning patterns in the data that do not represent useful features of the object being classified, thereby enhancing its ability to generalize to unseen data. Our experimentation primarily involved geometric transformations (RandomFlip, RandomTranslation, RandomZoom, Random-

Rotation) as well as other transformations related to lighting (RandomBrightness, Random-Contrast). However, while the former augmentations appeared to enhance the model's predictive capabilities, the latter were scarcely tolerated by the network. Consequently, the final decision was that of applying geometric transformations only.

3. Hand Crafted models

One of the initial attempts to design the network involved creating a model from scratch for training. The considered networks comprised a sequence of convolutional, dense, and dropout layers arranged in various combinations. Different hyperparameter values, including learning rate, batch size, and the number of layers, were tested. Techniques to mitigate overfitting, such as Early Stopping and Learning Rate Scheduling, were also employed.

However, this simple network failed to achieve a satisfying performance on the external test set. Post-submission on CodaLab, the accuracy did not exceed 63%. This outcome indicated that the network's structure was not sufficiently complex for the task, encouraging the exploration of more refined network architectures.

4. Pre-trained models

Several experiments were conducted using pre-trained networks. All members of the group simultaneously tested various models available in Keras Applications to identify the one best suited for the given task.

The models included in our experimentation were Xception [1], VGG16 [2], ResNet152V2 [3], and ConvNeXtBase [4].

In conjunction with the transfer learning technique, fine tuning was applied to the pre-trained networks to further enhance predictions. Various strategies for unfreezing the pre-trained layers were explored, yielding different outcomes across different networks. However, it was observed as a consistent trend that applying more than two fine-tuning steps did not significantly impact performance. Consequently, the fine-tuning technique was iterated at most twice.

The subsequent discussion provides insights into the pre-trained networks used to address the task, with particular emphasis on Xception and ConvNeXtBase neural networks. These two models received particular attention by the team members due to their noteworthy performance.

4.1. Xception

Before discovering ConvNeXtBase, the most promising network we encountered was Xception.

Xception's computational efficiency, achieved with fewer parameters, proved well-suited for our classification problem compared to other networks.

After determining the optimal number of layers to unfreeze for fine-tuning (106 yielded the best results) and identifying the best hyperparameters (learning rate of 10^{-3} without fine-tuning and 10^{-5} with fine-tuning, batch size of 32 samples, patience for early stopping of 50 epochs on validation loss), we achieved an accuracy of 82% on CodaLab. Despite our efforts, subsequent attempts to improve this score proved unsuccessful.

After several days without any progress, we began to question the model's ability to learn the task effectively. Indeed, throughout our attempts, the model consistently failed to overfit the training set. While not inherently negative, this observation hinted that the model might have lacked the necessary complexity for the task at hand. This suspicion motivated the exploration of other pre-trained networks.

Xception + Fine Tuning + Early Stopping (CodaLab)

Accuracy	0.8200
Precision	0.7567
Recall	0.7368
F1	0.7466

4.2. VGG16

After the work on Xception, experiments were conducted on VGG16, chosen for its simpler architecture (fewer layers) that seemed adaptable to the given dataset and easier to train.

However, while the network initially yielded promising results on the validation set (locally better than Xception), it was later found to be prone to overfitting and incapable of generalizing to unseen data. This limitation became evident when submitting the model on CodaLab, resulting in a lower performance.

Accuracy VGG16 (CodaLab)

Fine-tuning + dropout	0.7300
Fine-tuning + dropout + regularization	0.7900

4.3. ResNet152V2

Alongside VGG16, some efforts were devoted to experimenting with ResNet152V2, aiming to test a network with a different architecture. Nonetheless,

ResNet152V2 (CodaLab)

Accuracy	0.7200
Precision	0.6190
Recall	0.6842
F1	0.6500

despite trying numerous techniques involving a lot of hyperparameter tuning, unsatisfactory results led to the decision to discontinue further exploration.

4.4. ConvNeXtBase

ConvNeXtBase, the last network we experimented with, demonstrated the highest level of performance among the tested networks.

Even without the use of fine tuning, the network exhibited comparable performance to Xception on our local test set. However, in contrast to Xception, ConvNeXtBase demonstrated a tendency to easily overfit our dataset, necessitating the implementation of mechanisms to mitigate this phenomenon. To address this, we introduced a dropout layer before the fully connected (FC) layer and incorporated an L2 regularization parameter. Additionally, we modified the metric monitored by the early stopping callback from validation accuracy (the one used for VGG16 and ResNet152V2) to validation loss, aiming at halting the training when overfitting tendencies emerged.

These adjustments collectively contributed to achieving an accuracy of approximately 92%, both locally and on the online test set.

Accuracy ConvNeXtBase (CodaLab)

Base model	0.8200
Base + dropout	0.8500
Fine-tuning + dropout	0.9200
Fine-tuning + dropout + L2	0.9300

4.3. General design choices

4.3.1. Learning Rate: The learning rate was chosen through a systematic search, starting with a moderate value and adjusted based on the observed convergence behavior. The chosen rate strikes a balance between fast convergence and avoiding overshooting. Best results were achieved with learning rate values of 10^{-3} and 5×10^{-3}.

4.3.2. Number of Epochs: A validation set was employed to monitor the model's performance, and training was stopped when the validation accuracy (or validation loss) plateaued, preventing overfitting and unnecessary computational expenses. With the implementation of early stopping callback, the number of epochs never overcame 200 epochs.

4.3.3. Batch Size: The batch size was selected to balance computational efficiency and model generalization. Empirical testing indicated that a moderate batch size achieved both fast convergence and generalization. Reasonable values were always comprised between 32 and 128.

4.3.4. Regularization Techniques: Dropout and L2 regularization were implemented to mitigate overfitting, and the parameters were chosen according to empirical tests. For example, a good dropout rate was found to be 0.3, while a good regularization term was around 2×10^{-4}.

5. Final model

Although more than one model proved to have high accuracy in local tests, the one which showed the best performance when submitted on CodaLab was ConvNeXtBase. Indeed, in the final phase of the competition, the performance achieved was still reasonably high.

ConvNeXtBase (CodaLab - Final Phase)

Accuracy	0.8540
Precision	0.8196
Recall	0.7894
F1	0.8042

7. Bibliography

[1] https://keras.io/api/applications/xception/

[2] https://keras.io/api/applications/vgg/#vgg16-function

[3] https://keras.io/api/applications/resnet/#resnet152v2-function

[4] https://keras.io/api/applications/convnext/#convnextbase-function