Homework 1

Bistacchia, Erica*
Gurrieri, Davide†
Negroni, Sabrina‡

2023/2024

POLITECNICO
MILANO 1863

CONTENTS

1 INTRODUCTION

The goal of this project is to to develop a Convolutional Neural Network architecture for classifying plants health status. The provided dataset is made up by 5200 images of 96×96 pixels in RGB format and each image is labelled as healthy or unhealthy. In the following sections we present the workflow we followed during the project, giving particular attention to the techniques we found to be most interesting and to have yielded to the best results.

2 DATA PRE-PROCESSING

Following a preliminary inspection of the dataset, all outliers were identified and systematically removed to prevent the model from learning wrong patterns. This refinement process resulted in a 5004 samples dataset, in which an imbalance in class distribution was noted: approximately 62% of the images were labeled as healthy and the remaining 38% as unhealthy. Recognizing the potential negative impact of these issues on the model performance, we employed data augmentation techniques to tackle both data scarcity and imbalance. [1]

*erica.bistacchia@mail.polimi.it
†davide.gurrieri@mail.polimi.it
‡sabrina.negroni@mail.polimi.it

Figure 1. Examples of CutMix (left) and MixUp (right) augmentations applied on images of the provided dataset.

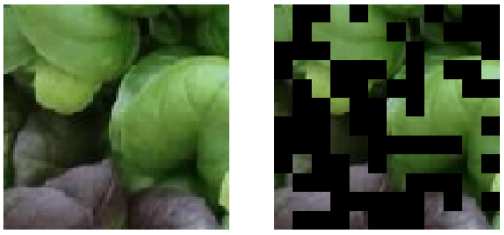

Figure 2. Example of Hide and Seek augmentation technique applied on an image of the provided dataset.

We tested different transformations among the ones available in Keras and we discovered that some of them, specifically random flip, rotation, and zoom, were highly effective for our task. Indeed, we expected that the introduction of these transformation as an augmentation layer in the CNN could prevent from overfitting the data during the training. On the contrary, transformations like random contrast, brightness, and crop didn't lead to the expected results. We interpreted this worsening in performance with a decreased legibility of information: for example the contrast technique resulted in a more darkened and less visible image. In order to incorporate these transformations in the model, we added an augmentation layer, and we tuned the hyperparameters such that each value is coherent with the other: horizontal and vertical flip are both active, rotation factor has a max value of 90 degrees and zoom factor of -0.1.

Additionally, we implemented CutMix and MixUp [2],[3] transformations using Keras CV library: CutMix forces the model to distinguish features from different regions of the image, MixUp , instead, effectively creates new synthetic samples by incorporating information from different images, preventing the model from overfitting to the original training set. This augmentation strategy, in conjunction with the others, yielded to better results in the model performance.

Finally, we have implemented a region deletion strategy called Hide-and-Seek (HaS) [4] which is a type of information dropping technique similar to dropout regularization. Unlike the latter, where the elimination is performed at the feature level, in region elimination, the deletion is applied at the input level. The design of this process allows the network to learn multiple relevant parts of the plants rather than only the highly discriminative parts.

3 NETWORK DESIGN AND IMPLEMENTATION

In this section we describe how our design process evolved and the choices we made to select our best model.

3.1 Phase 1: Custom-Made Networks

The first approach we followed was to design a CNN model from scratch. We began by entirely designing both the convolutional layers and the classifier.

We used some of the models encountered in labs, such as the QuasiVGG9 and VGG-like ones. We tried different settings, in order to explore the capabilities of these networks and to understand whether or not they were able to capture meaningful information. Unfortunately, due to their low complexity these models turn to be ineffective: they were unable to overfit the data during training, and even if validation results appeared promising, they performed poorly in the test phase. The best result we obtained was around 68% accuracy.

3.2 Phase 2: Pre-implemented Networks

In this second stage, we moved on to more complex models since those mentioned above proved not to be powerful enough in terms of accuracy for our task. We replaced the custom-designed convolutional layers with pre-implemented features extractors, taken from the Keras Applications module. Then we added a classical fully connected classifier. Initially we worked on lighter models like Xception, MobileNet and ResNet. We trained the entire network without using pre-trained ImageNet weights. Although we achieved better results than with the prior models, reaching validation accuracy of around 80-85%, the overall performance was still not good enough. A possible reason is that training complex networks from scratch requires advanced techniques that are beyond our capabilities. The accuracy on the test set dropped significantly to 69% with Xception and to 75% with ResNet50.

3.3 Phase 3: Transfer Learning Approach

The fact that over-fitting is not reached during the training in the previous phase suggested us that more complexity needs to be included in the model. Thus, we tested larger and more recent networks, namely ConvNetX-Base, Small and Large.

This time we initialized the weights using the ones already trained for the ImageNet dataset. Then we applied a standard transfer learning approach, freezing all the convolutional layers and training only the classifier with an high learning rate. However, the performance of validation was not outstanding, around 80-85%, but we expected this, since most of the model is trained for a different purpose from our own. This directed us towards fine-tuning techniques.

3.4 Phase 4: Fine-Tuning

In order to optimize the performance of the mentioned models, we try to apply fine-tuning to adjust the pre-trained weights of the convolutional layers. We tried different unfreezing patterns, starting from the last layers blocks and decreasing each time the learning rate.

Despite initial expectations, the models did not demonstrate improvements in performance. Instead, in some cases, there was a decline in accuracy on the validation set.

We justified this behavior by concluding that our task differed significantly from that of the pre-trained network. Consequently, we decided to unfreeze the entire network, so that all parameters could be adapted to our target. This led to a huge increase in validation performance, even reaching 95% with ConvNeXtBase and ConvNeXtLarge networks.

We also applied a variant of the previous method: instead of separately training the classifier with the rest of the network frozen and then training the entire network, we simply initialised the weights of the convolutional part with the pre-trained ones and then trained the entire network, without freezing anything. The only difference with respect to Phase 2 setting is a clever initialisation of the weights. This version also performed very well in validation and allowed us to achieve 89% accuracy on the test set during the first part of the challenge.

4 IMPROVEMENTS

In this section we describe the techniques used to try to improve our best models.

4.1 Learning rate scheduler

We set an adaptive learning rate by calling the callback implemented in Keras `LearningRateScheduler()` for enhancing the performance of the model. In particular we used an exponentially decaying learning rate. This dynamic regularisation aims to refine the training process of the model over time, facilitating better convergence and potentially solving overfitting problems. Despite the successful implementation of this strategy, which resulted to a regularising effect on the model, our experiments revealed a lack of discernible impact in terms of accuracy improvement.

4.2 Dropout and Gaussian Noise

Dropout and Gaussian Noise are two fundamental regularisation techniques [5] used in our models. Dropout acts by randomly setting to 0 the inputs of a layer with an assigned probability, while Gaussian Noise introduces additive random values with 0 mean to all the input of a layer. The first one was fundamental to avoid over-fitting in the network architecture, while the second technique could be seen as a form of data augmentation and thus both of them help preventing over-fitting. However, the most effective one between the two techniques appears to be the incorporation of dropout layers in the Dense Network. Depending on where these layers were placed, we set their parameter to higher or lower values: if placed before a dense layer with few units (less than 100), frequency was set around 0.1, if placed before one layer with more units, frequency was set up to 0.4 (with 1024 units).

4.3 Ensemble

Having explored various configurations for individual models, thereafter we adopted an ensemble approach to further improve the predictive capabilities of our model. Specifically, we used a majority voting strategy on an odd number (n) of previously trained models that had performed successfully. To implement this ensemble, we modified the predict function, by generating a vector of predictions based on the voting mechanism of the models. Practically, ensemble prediction was determined by assigning a value of 1 if at least n/2 + 1 models predicted 1 (”unhealthy”), and 0 (”healthy”) otherwise. This ensemble strategy aimed to exploit the combined power of multiple models, as well as potentially mitigating the individual limitations while improving the overall predictive performance. As a result, the effect of this approach on the model performance was remarkable: the accuracy on the validation set improved, reaching the threshold of 96%.

4.4 Generative Adversarial Networks

We implemented a Conditional Generative Adversiarial Network [6] to combat data scarcity and the unbalancing of the two classes. Our idea was to generate synthetic data belonging to the unhealthy class, taking advantage of the fact that the desired label can be given as input to the model. Unfortunately, the architecture training failed due to our limited resources. In fact, after a considerable number of epochs, the model was

still unable to generate images similar to the real ones. We therefore had to give up.

4.5 Other techniques for imbalance

We made further attempts to solve the problem of the imbalance of the two classes. We applied a simple oversampling technique of the unhealthy class by randomly sampling some images and applying Gaussian noise. We then tried applying contrast and brightness augmentations to the unhealthy sampled images in order to differentiate them from the original ones and highlight plant defects. Unfortunately, this technique has not led to any improvement. We tried to manage the imbalance even at the model level by adjusting the weights of the loss function. Specifically, we used the `compute_class_weight` function from Scikit-Learn to penalize errors more heavily on unhealthy images.

5 FINAL MODEL

During the initial phase of the challenge, the model that exhibited the higher test accuracy was based on ConvNeXtBase architecture. However, evaluating the models accuracies in validation, we realize that the ones based on ConvNeXtLarge outperform the others. Furthermore, by observing the decimal digits of the leaderboard results, we realized that the test dataset in the first part consisted of only 100 elements, leading to significant exposure to variability. In contrast, the dataset in the second phase comprised 1000 samples, promising more stable results. Therefore we decided to rely on ConvNeXtLarge as main architecture, implementing refinements to its original structure with the aim of optimizing overall performance. We report below the final structure of our model.

Our chosen augmentation strategy includes random flipping both horizontally and vertically, random rotation with a factor of 0.3 and fill mode set to reflect, and random zoom with a height factor of 0.1. In addition we applied CutMix or MixUp to every image, choosing one or the other strategy with a probability of 0.5, since applying both the augmentation on each image worsened the performance.

We used transfer learning and fine tuning techniques as explained in the previous section. We did a first training freezing the ConvNeXtLarge convolutional layers. Then a second training was performed, unfreezing the entire network.

We initially performed the two training using a validation dataset and early stopping with a patience of 15, in order to understand the optimal number of epochs. Unfortunately we could not do a k-fold cross validation due to limited GPU resources. We then ran again both training sessions using all the data.

Hyperparameter Settings:

- **Validation Proportion:** 0.2
- **Batch Size:** 64 samples
- **Max Epochs:** 15 epochs for the first training 70 epochs for the second training.
- **Optimizer:** Adam Optimizer
- **Learning Rate:** 1×10^{-4} for the first training, decreased to 5×10^{-5} for the second training (fine-tuning) to avoid losing previously learned features.
- **Loss Function:** Categorical Cross Entropy with label smoothing parameter equal to 0.1.When using MixUp, label smoothing is highly recommended.

After the flattening layer, the customised part of the network is structured as follows:

- Dropout layer with a frequency parameter set to 0.4.
- Dense layer with 1024 units, ReLU activation function, HeUniform initializer, and L1L2 kernel regularizer(1e-3).
- Dense layer with 512 units, ReLU activation function, HeUniform initializer, and L1L2 kernel regularizer(1e-3).
- Dropout layer with a frequency parameter set to 0.3.
- Output layer with 2 units, Softmax activation function, and GlorotUniform initializer.

This latter model achieved the best performance in the second phase, reaching an accuracy of 84.5%.

In the end, we opted to train a similar model by only modifying certain parameters, such as the number of epochs, and incorporating average global pooling instead of flattening. Subsequently, we assembled the two models into an ensemble based on the element-wise maximum of their respective outputs. At the time of the deadline, we were still waiting for the accuracy results so unfortunately we were not able to write it here.

6 CONTRIBUTIONS

All the three group members Erica, Davide and Sabrina contributed to the work equally, doing great team work. In particular Erica focused her work on the Hide-and-Seek technique, Davide on the Keras-CV augmentation and Sabrina on the ensamble model. The rest of the models were tested by all the team members trying different structures and anti-overfitting techniques. All the work done for this challenge can be found on the following github repository: plants-classifier

REFERENCES

[1] Alhassan Mumuni and Fuseini Mumuni. Data augmentation: A comprehensive survey of modern approaches. *Array*, 16, 2022.

[2] Seong Joon Oh Sanghyuk Chun Junsuk Choe Youngjoon Yoo Sangdoo Yun, Dongyoon Han. Cutmix: Regularization strategy to train strong classifiers with localizable features. *arXiv*, 2019.

[3] Yann N. Dauphin David Lopez-Paz Hongyi Zhang, Moustapha Cisse. mixup: Beyond empirical risk minimization. *arXiv*, 2018.

[4] Aron Sarmasi Gautam Pradeep Krishna Kumar Singh, Hao Yu and Yong Jae Lee. Hide-and-seek: A data augmentation technique for weakly-supervised localization and beyond. *Arxiv*, 2018.

[5] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[6] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.