Convolutional Neural Network: Healthy-Unhealthy leaves classification

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

Carlo Fabrizio[*], Riccardo Malpiedi[†] and Niccolò Perrone[‡]
M.Sc. Computer Science and Engineering - Mathematical Engineering, Politecnico di Milano - Milan, Italy
Email: [*]carlo.fabrizio@mail.polimi.it, [†]riccardo.malpiedi@mail.polimi.it, [‡]niccolo.perrone@mail.polimi.it
Student ID: [*]10747982, [†]10940821, [‡]10707561
Codalab Group: "Neural Net Ninjas"

1. Task Analysis

Given a dataset of 5200 images of plants' leaves, of which 3199 healthy and 2001 unhealthy, the goal was to create a classifier able to classify an unseen image according to its state of health. After removing outliers and duplicates the resulting dataset contained only 4800 images. Three main problems have been faced due to the composition of the dataset: few available samples, unbalanced classes and high level of similarity among the two classes.

To overcome the small size of the dataset , augmentation techniques and Transfer Learning were used. In order to mitigate the gap between classes' size, class weights were included in the created model.

In order to tackle the problem of similarity among the two classes, the choice of a smart network, able to capture small features from an image, was fundamental.

2. Data Augmentation

Data Augmentation techniques have been adopted from the beginning in order to increase the size of the dataset and to allow the classifier to better generalize.

Initially, many "traditional" augmentation techniques were used: *Rotation*, *Flipping*, *Brightness*, *Shifting* , *Contrast*, *Channel switch*, *Zoom out* (Zoom in could delete important features), all of them with high parameters that were gradually reduced in order to achieve an higher validation accuracy. However, our model reached a saturation in the performance and in the ability to generalize, therefore, we adopted more advanced augmentation techniques: MixUp[2] and CutMix.

While MixUp immediately boosted the performance of our classifier, CutMix was not so effective. A combination of CutMix and MixUp was explored (MixUp bacth i with probability p CutMix bacth i with probability $1-p$) but the results were similar if applying only MixUp. Moreover, the idea that CutMix could cancel out some important features from an image, convinced us to only adopt MixUp.

Finally, the idea that maybe Mixup was ruining the information related to the texture of the leaf in the images, conviced us to try a different augmentation pipeline made by: (A*)*brightness change*, *zoom out*, *channel shift*, *flips*, *contrast change*, *rotations (90,180,270 degrees)* (fixed degrees to avoid important features being cut out) and *color degeneration* (better generalize for plants with leaves of different colour).

3. Test-Time Augmentation

Test-Time-Augmentation technique was implemented relatively quickly, using random transformations and producing 10 different random versions of the same image to be classified. This device gave us a small increase in the test performance, on avarage around $1-2\%$ on performance of the classifier (Resnet-50 gave 77% accuracy on the test without TTA and 80% with TTA).

Finally, we also implemented a TTA with the same augmentations described here (*A).

4. Pre-Processing

When we used pretrained models, we did the appropriate preprocessing of the inputs applying rescaling and resizing.

5. Models

5.1. Convolutional Neural Networks from scratch

We started with a baseline model adopting the classical architecture of modern CNNs. As feature extractor we used 3 blocks of 2 `Conv2D` followed by `MaxPooling2D` and a last block of 2 `Conv2D` followed by a `GlobalAvaragePolling`. On the top of the latter feature extractor a `Dense` layer with one neuron with sigmoid activation function was added. Both accuracy and loss function were really unstable, probably due to lack of generalization. This direction was not explored any more due to the fact the the given scenario (small dataset size) was clearly requiring Transfer Learning.

5.2. Pre-Trained Models (Transfer Learning and Fine Tuning)

Since we had a small dataset, we thought we could exploit the knowledge of models pretrained on other image datasets (ImageNet in particular) through transfer learning. This gave us some good results, and results got even better with fine tuning.

5.2.1. Feature Extractor. The choice of the optimal feature extractor was not trivial, many models were showing a very high performance on our local dataset, both on the validation and on the test, with no sign of overfitting. However, just few models were well performing on the hidden test dataset of the competition. The latter problem can be explained due to the fact that those models were extracting wrong features from our dataset. Probably, those models were learning the correlation between the specie of the plant and the state of health -e.g.”$x\%$ of basil images in the dataset was healthy, therefore, if an unseen leaf was similar to basil, with probability $x\%$ was healthy”.

Many families of models were tested: VGG, Resnet, Xception, EfficentNet, Mobilenet, DenseNet and Nasnet, and the ones giving us the greatest results were Resnet-50 and MobileNetV2, 80% and 78% respectively. We also built some ensembles by combining different networks and this leads to reach 85% accuracy on the hidden test. Eventually, we implemented several training techniques (only Trasnfer Learning, Fine-Tuning, ”deep” Fine-Tuning and retrain from zero); however, we did not managed to improve further our results on the hidden test, without TTA.

What gave us a substantial improvement in the performance was the choice of **ConvNext** [1] as feature extractor. Even if it was performing similarly with respect to the other nets on our available data, its first submission gave us 88% on the test data. Not by chance, ConvNext architecture is built ”modernizing” the Resnet architecture, adding many features such as Inverted Bottleneck, a design element popularized by MobilenetV2 architecture, as said in the paper[1]. ConvNext, therefore, fuses elements from Resnet family and elements from MobileNet family, that were the two families of models gaving us the greater performance.

5.2.2. Classifier. Different configurations of classifiers were performed on different networks. In the following, we will focus on the classifier used for our best best, the ConNext.
In order to find the optimal classifier, we started by adding a final dense layer with two neurons and softmax activation at the end of our pre-trained ConvNext. Training this classifier led to a validation accuracy of 0.79%. To improve this result, we decided to adopt an hyperparameter tuning from the KerasTuner library.
The choice of tuner fell on the HyperBand algorithm (random search through adaptive resource allocation) and the use of early stopping. It standed out that for our specific case, a block between the pre-trained model and the two-neurons

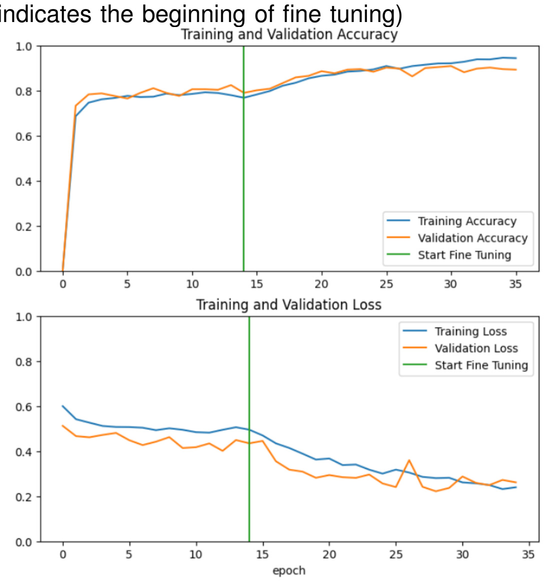
dense layer consisting of a dense layer(128 units) -ReLu activated and a batch normalization, performed better. Furthermore, a dropout with rate 0.5 was added straight after the activation layer to prevent overfitting. Lastly, an L^2 regularization of 0.5 was added to the two dense layers. This final setup improved the validation accuracy to 0.83%.

5.2.3. Training techniques. We expertimented two main training techniques.

Retrain from zero We tried to train some pre-trained models without unfreezing the layers. In order to avoid the chance of overfitting we added a strong regularization to the convolutional layers of the model and to the classifier, plus a strong dropout and a learning rate scheduler with an initial learning rate very low. Some feature extractors gave us the same results as training with fine tuning (RESNET-50: similar validation accuracy and same test accuracy), some others were showing presence of overfitting. therefore, we started fine tuning the pre-trained models.

Fine-tuning To perform this technique, we started by freezing the whole pretrained model and training the classifier. After this first step, we unfroze the last layers of the pretrained model and we resumed the training with a small learning rate in particular. We tried different configurations, we'll report the ConvNext fine tuning in particular since it gave us the best result in phase 1. We trained the classifier for 15 epochs with a 0.001 learning rate (we used Adam optimizer), we unfroze last 37 layers (ConvNext Tiny has 157 layers) and we trained the model for 10 additional epochs with a lower learning rate (0.0001). At last, we unfroze 30 layers more and we continued for other 10 epochs. We stopped the training to prevent overfitting and our model generalized well, indeed we got 90% accuracy on our validation set and 88% accuracy on test set.

Figure 1. Validation accuracy and loss of ConvNext (the green line indicates the beginning of fine tuning)



6. Ensemble

In order to achieve an higher generalization power, different ensemble techniques have been tested. Initially we tried Bagging, many models, trained on different bootstrap versions of the same training dataset were combined. However, The latter implementation was suffering from the fact that many models were correlated either because pretrained on the same dataset (imagenet) or because they were using a similar feature extractor (Resnet-101 with Resnet-50). Therefore, the optimal combination of nets was formed of models belonging to different families of Nets: MobileNetV2 ,Resnet-50, Xception, EfficentNet , DenseNet and NasNet, weighted empirically by our sensitivity in order to balance Precision and Recall, and this ensemble gave us 85% on the hidden Test. A Boosting method (Adaboosting) was tested with no interesting results, starting from a weak CNN as base classifier, it was too much sensitive to changes of the data and the computation required was too high.

7. Final Model

- The following model gave us the best results for both phase 1 and phase 2:
  - Augmentation: *Mixup* + *Flip* + *Rotation* + *Translation*
  - Model: ConvnextTiny+`GAP`+`Softmax`
  - Training: Fine-Tuning (described in 5.2.3)

8. Contributions

We all contributed on training and testing different models, here the main details are reported.

Carlo Fabrizio inspected and cleaned the data (removing ”outliers” and duplicates), then he implemented different data augmentation techniques (mixup, cutmix, geometric transformations). He also did experiments with bagging and boosting.

Riccardo Malpiedi implemented trasfer learing and focused on fine tuning for different models, trying different configurations. He also implemented weighted ensemble to combine different models.

Niccolò Perrone focused on hyperparameter tuning and tried to combine our best models with different ensemble techniques.

References

[1] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.

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