Transfer Learning and Custom Fine Tuning Heath-State Classification of Leaves

Gianluca Canzii, Leonardo Cesani, Matteo Colella, Leonardo Spinello
gianluca.canzii@mail.polimi.it, leonardo.cesani@mail.polimi.it, matteo1.colella@mail.polimi.it, leonardo.spinello@mail.polimi.it

"*Fine tuning is like cutting your hair when you're depressed: you think it's going to make everything better but actually it just makes things worse.*"

— Some user on the ANNDL's group chat.

***Abstract*—This report aims to show the work of the group *Backbropagation* carried on to provide a solution for a binary classification problem in the context of natural images. Due to the lack of a big dataset, techniques such as transfer learning, fine tuning and data augmentation have been exploited to propose a model that achieves high accuracy over the test set.**

I. Introduction to the Problem and First Analysis

The project, developed under the first homework provided by the "Artificial Neural Networks and Deep Learning" course offered by Politecnico di Milano, aims to build a model to classify healthy and unhealthy leaves of different types of plants (Fig. 1). The provided dataset is composed of 5200 RGB images of size 96×96 pixels, 3199 of them labeled as "healthy" and 2001 labeled as "unhealthy".

Fig. 1. Comparison of healthy and unhealthy leaves

A. Dataset Analysis

The first analysis that we performed was the outliers and duplicates analysis. We decided to build a python script in order to identify all the duplicates in the dataset, and the images that do not match the specifications of the problem. More specifically, we decided to detect the images that showed imbalance in the red or in the blue channel with respect to the green one. This was performed by writing a function that checks for each image if *amount* is larger than parameter *threshold*; *amount* is the number of pixels that show a difference between blue (or red) channel and green that is greater than parameter *difference*. Performing some tests, we obtained the correct parameters for our scope. After removing the outliers, which were images not related to the scope of the task, we detected and deleted the duplicated images.

B. First Experiments and Baseline

The first model we built was a *quasi-VGG* model with three convolutional layers, a global average pooling layer and two dense layers connected with two neurons in the output layer. We trained it with a split version of the new dataset, but after assessing poor performance over the test set we decided to focus on *transfer learning* and *data augmentation* techniques.

II. Transfer learning and Choice of the Model

In order to select the best model for our application, we have built a script that allows us to iterate over different models in order to select the best one according to the validation accuracy. After a cyclic train over the different models our choice ended up on *EfficientNetV2S* and *ConvNeXtLarge*.

A. The EfficientNetV2S

We started analyzing the EfficientNetV2S, proposed by Tan et al. in 2021 [3], due to its high performance compared to execution time. This advantage was given by the simplicity of the net that, given the small size of the (at the time) poorly augmented dataset, helped us to reach higher validation accuracy valued. The low amount of weights, however, limited our result to 87%[1] in test accuracy, leading us to further improvements with the use of more complex nets trained with more complex datasets.

B. The ConvNeXtLarge

The ConvNeXtLarge [1] model proposed by Liu et al. in 2022, also known as Convolutional Neural Networks with External Memory (ConvNeXt), incorporates

[1]All the results before section VI have been obtained in the first phase of the competition

external memory for improved performance. It consists of convolutional layers for initial data processing and feature extraction, max pooling layer for dimensionality reduction, long short-term memory (LSTM) layer for processing sequential data.

1) Custom Layers: We decided to add on top of the ConvNeXtLarge some layers to specify the model to the task requested in the project. We added the layers shown in Fig. 2.
In particular:

- tfkl.GlobalAveragePooling2D: this layer was used to reduce the spatial dimensions of our 3D tensors to 1D tensors, preserving the batch size and channels but removing the height and width.
- tfkl.Dropout: this layer randomly sets a fraction rate of input units (specifically 1 out of 5) to 0 at each update during training time, which helps to prevent overfitting.
- tfkl.Dense: this layer is a regular densely-connected neural network layer, which is used to change the dimensions of our vector.
- tfkl.BatchNormalization: this layer normalizes the activations of the previous layer.

global_average_pooling2d (GlobalAveragePooling2D)	(None, 1536)	0	['layer_normalization[0][0]']
dropout (Dropout)	(None, 1536)	0	['global_average_pooling2d[0][0]']
dense (Dense)	(None, 256)	393472	['dropout[0][0]']
dropout_1 (Dropout)	(None, 256)	0	['dense[0][0]']
dense_1 (Dense)	(None, 128)	32896	['dropout_1[0][0]']
batch_normalization (Batch Normalization)	(None, 128)	512	['dense_1[0][0]']
dense_2 (Dense)	(None, 2)	258	['batch_normalization[0][0]']

Fig. 2. The parameters for our classifier

From now on, when referring to the model, we will be talking about the aforementioned one.

2) Hyperparameters Tuning: In order to choose the optimal hyperparameters for the training process, we employed a systematic approach. More specifically, we instantiated a procedure that iteratively probes different combinations of *batch size* and *learning rate* values. After several iterations and analysis of the results, we came to the conclusion that the highest level of validation accuracy could be reached with a batch size of 64 and a learning rate of 0.001.

3) Model 1: A first model was obtained with a tuning over flipped and rotated images, used both while training the dense layers and while fine tuning the net. More specifically, after a first training with the aforementioned hyperparameters, we performed the fine tuning unfreezing all the weights with *learning rate*=0.0001. This operation was performed twice: the first time adding a sequential layer that performed random rotations and flips at every epoch, the second time with a merge of the original training set and of the validation set. We managed to obtain a validation accuracy of 90%. We will refer to this model calling it *Model 1*.

III. Data Augmentation

We realized that to obtain better results we needed to focus on data augmentation. We did this to create a new model with a higher robustness to unseen data and to eventually use techniques such as ensembling. We based our work on the concept of *Class-Adaptive Data Augmentation* proposed by Yoo et al in 2023 [4] , that consists in finding the optimal transformation parameters for each class.

A. Class-wise Augmentation

This innovative technique consists articulates in three phases:

1) Training of an initial model: in order to perform the later operations, it is necessary to have a model trained on original or poorly modified images. We had already trained *Model 1* that was also achieving very satisfying.

2) Optimization of the transformation parameters: we chose to find the optimal parameters for 10 different transformations, that can be seen in Fig 3. Bayesian optimization was exploited to this scope, more specifically we used the python library *BayesianOptimizazion* by Nogueria et al. [2]. We ran the optimization for each of the two classes and for each of the transformations, in order to find the best parameter for each. After declaring the bounds for each parameter, we ran the optimization minimizing the cost function that accounted for the error that the transformation introduced on a modified image (assessed using *.predict* method on the previously trained model), balanced with a term that encouraged the increase of the parameter.

3) Training with new augmented dataset: the last passage was the training of the new net, to which the next part of the report will be devoted.

B. Use of Augmented Data in Training

We realized that some optimal parameters were very similar for the two classes and some were more distant. We decided to use the latter ones in the creation of a new augmented dataset to be able to differentiate between the two classes, while the transformations that could be applied almost independently on the *unhealthy* and

```
Class healthy- Transformation rotate: {'phi': 0.3435657853877308}
Class healthy - Transformation shear: {'phi': 25.151587759761053}
Class healthy - Transformation horizontal_shift: {'phi': 0.22701936955231622}
Class healthy - Transformation vertical_shift: {'phi': 0.14502292379679366}
Class healthy - Transformation zoom: {'phi': 0.13841030691281658}
Class healthy - Transformation brightness_top: {'phi': 0.4291927637480982}
Class healthy - Transformation brightness_down: {'phi': 0.5360504749959921}
Class healthy - Transformation contrast: {'phi': 0.929372775478044}
Class unhealthy - Transformation rotate: {'phi': 0.49462835991615806}
Class unhealthy - Transformation shear: {'phi': 45.0}
Class unhealthy - Transformation horizontal_shift: {'phi': 0.3879495900814421}
Class unhealthy - Transformation vertical_shift: {'phi': 0.3468234569887735}
Class unhealthy - Transformation zoom: {'phi': 0.14492545891532727}
Class unhealthy - Transformation brightness_top: {'phi': 08335860670516785}
Class unhealthy - Transformation brightness_down: {'phi': 0.15270307170959083}
Class unhealthy - Transformation contrast: {'phi': 1.00}
```

Fig. 3. The optimal transformations parameters

healthy data were applied randomly at run-time through a sequential layer at the beginning. Some operations were added such as grey scale, green channel removal and flip, that we weren't able to optimize due to time constraints. The dataset was created taking into account that some transformations have a more aggressive effect on the image than others: hence, we decided to split them into *light* and *heavy* ones, and to apply one of each category to each augmented image. It is worth noticing that the creation of augmented images allowed us to balance the dataset, by transforming twice each unhealthy image. In Fig 4 an sample of the said augmented images can be seen.

Fig. 4. A sample of augmented dataset

IV. Fine Tuning

As previously mentioned for *Model 1*, fine tuning procedure has been applied to both the model we came up with. This consists in a further training phase performed with a smaller learning rate and unfreezing some convolutional layers (or all, as in our case), with a different (or even the same) dataset, in order to make the net more robust to unseen data. It is particularly useful to adapt the pre-trained weights learned over *Imagenet* to a specific use case.

V. Final Results and Conclusions

A. Model 2

We created a model with the same structure as *Model 1*, but trained using different techniques and data. Indeed, we initially trained the feature classification part with the original dataset (without outliers and duplicates). After, reducing the learning rate by a factor 10, we fine tuned 20 million weights using both original and augmented data (both class-wise and at run-time, as previously mentioned). A last tuning procedure was performed over the whole model with the same data, using a further decreased learning rate. This model, to which we will refer with *Model 2*, managed to obtain the same 90% accuracy that we had already got, but gained some points in *precision*.

B. Ensemble and final results

We decided to use *Model 1* and *Model 2* to exploit the ensemble method. This technique consists in mixing the predictions generated by multiple neural networks, in order increase the performance. Although it would be possible to weight the outputs, we came up with a satisfying 92% of accuracy over the unknown test set.

VI. Conclusions

A. Final Evaluation

Finally we evaluated the ensemble model over the second test set [2] obtaining a classification accuracy of 87.6%. This result shows the model to be quite good in generalizing over unknown data, since the classification accuracy lost only around 4% between the two phases. Nevertheless, we realized that despite not overfitting the first test set (we did not train the model on it), we took decisions based on the results obtained inferring on it, leading to a tailor made model that achieved better performance on that specific set of images.

B. Discussion

Our method to classify healthy and unhealthy leaves shows good performances over the test set, even if we know that it is possible to improve them even further by implementing other techniques such as custom augmentation, testing different models for transfer learning and advanced fine tuning techniques. In order to focus on the problems of our classifier, it is possible to implement the receptive field's analysis to identify which are the features that induce the model to fail.

[2]Using the unknown data in the second phase of the competition

VII. Contributions

Team members' focus summarized here:
Gianluca Canzii: data analysis and preprocessing, data augmenting
Leonardo Cesani: data analysis and preprocessing, data augmenting, model implementation
Matteo Colella: model implementation, hyperparameters tuning, testing
Leonardo Spinello: data preprocessing, model implementation

References

[1] Zhuang Liu et al. *A ConvNet for the 2020s*. 2022. arXiv: 2201.03545 [cs.CV].

[2] Fernando Nogueira. *Bayesian Optimization: Open source constrained global optimization tool for Python*. 2014–. URL: https://github.com/fmfn/BayesianOptimization.

[3] Mingxing Tan and Quoc V. Le. *EfficientNetV2: Smaller Models and Faster Training*. 2021. arXiv: 2104.00298 [cs.CV].

[4] Jisu Yoo and Seokho Kang. *Class-Adaptive Data Augmentation for Image Classification*. 2023. DOI: 10.1109/ACCESS.2023.3258179.