1st ANNDL Challenge: Image Classification Report

Balzarini Filippo - 10719101, Benelle Francesco - 10727489, Cavicchioli Michele - 10706553
TheCEOs

1 Abstract

The goal of the challenge is the creation of a Neural Network model to perform binary classification on a set of healthy and unhealthy leaves. We started facing the problem with a mindset that was already oriented toward the use of transfer learning with a pre-trained model, at that time, we did not have any experience in solving an image classification problem we decided it was better to start experimenting with a model with a good trade-off between the number of parameters and top-1 accuracy, having said this we chose to use *MobileNetV2*. Chronologically, the second thing we did was to inspect the dataset to see what kind of images were provided by the professors. In this phase we encountered some outsiders, images that were not leaves; so we took them out from the provided dataset and saved a cleaned version, the number of outsiders was 196, so the length of the dataset decreased from 5200 to 5004. Another aspect that we noticed is the difference in the number of unhealthy labels concerning the number of healthy images, more or less 60% of the samples are labeled as healthy (the delta is more or less 1000 images).

2 Operations on Dataset and Augmentation

2.1 Dataset Specification

The dataset provided for this challenge was made of a total of 5200 images of leaves, divided into 2 classes: healthy and unhealthy. At first, we tried to print some images to get a better view of this binary classification problem, but we noticed the presence of two kinds of outliers. Then we proceeded by eliminating them from the dataset, obtaining a cleaned one made of 3101 healthy and 1903 unhealthy leaves, for a total of 5004 96x96 RGB pictures.

2.2 Dataset Augmentation

During our tests, since the number of samples was exiguous, we implemented various techniques of data augmentation, some of them directly on the training set like the use of operations from the *KerasCV* [1] libraries such as *CutMix*, *MixUp* [2], or *RandAug*.[3]

2.3 Data Augmentation Layer

The first approach to data augmentation included a preprocessing layer concatenating data augmentation methods like *RandomFlip*, *RandomTranslation*, *RandomContrast*, or *RandomZoom*. [4]

2.3.1 Class weight

Using *sklearn* [5] library we tried to fix the different distribution of the labels in the dataset with the *compute_class_weight* function but once obtained the weights and computed them in the training, no real advantages were registered from the network, so we decided to discard this option on the training of the final model.

3 First Network Models and Experiments

3.0.1 Transfer Learning

When we tried to implement our first model, we decided to try Transfer Learning with *MobileNet*, since we thought that the network was a good compromise between a small number of parameters and a good enough top-1 accuracy. Noticing that we couldn't go further than a score of 0.72, we started to take into account other Models like *ResNet*, *Xception*, and *InceptionNet* but all of those choices led to scarce results, but we had our turning point when we tried the *EfficientNetV2* model, which provided us good results in terms of accuracy. Having found a Transfer Learning model which we felt confident with, we started trying fine-tuning and trying to unlock different blocks of the network, this led to a significant upgrade to our model's accuracy. Then we experimented a bit with different Optimizers, such as *Adam*, *AdamW*, *Nadam*, or *Adamax*. One thing that is common in every attempt is that we had as a final output layer a Dense Layer with a *softmax* and with a size of 2 since we are using *Categorical Crossentropy* along with the one-hot encoded labels for the two classes.

3.0.2 Fine Tuning

To optimize the pre-trained weights of *imagenet* to our specific use case we chose to perform fine-tuning to our network. Of the pre-trained model we chose to keep frozen only the first 189 layers, which is the number that gave us the best results at the end of testing. Every time we trained a network, regardless of whether it was the first training or Fine Tuning, we calculated the *best epoch* as the difference between the final training epoch and the patience to retrain the model for the perfect number of epochs merging the training dataset and the validation one.

3.0.3 K folding

One approach we have tried during our development is K-folding, with which we tried to train our network with different portions of the dataset in train and validation set to obtain the average best epoch of training trying to generalize as possible the result such that it was not influenced by the portion of the dataset. We tried to use K-folding with $k = 5$ but since this method is time-consuming, and the results obtained were quite similar to the one without this optimization, this approach has been excluded from the final model.

4 Final Model

4.1 The Network

Our final model consists of a **ConvNeXt Base** model, at which we added a Preprocessing Layer at the top and a combination of Dense Layers, Batch Normalization Layers, and Dropout Layers at the bottom. The final summary of the model is as follows:

Summary of the Model		
Layer	Output Shape	Parameters
Input Layer	(96, 96, 3)	0
Preprocessing Layer	(96, 96, 3)	0
ConvNeXtBase	(1024)	87566464
Dropout(0.3)	(1024)	0
Batch Normalization	(1024)	4096
Dense Layer	(512)	524800
Dense Layer	(256)	131328
Dropout(0.3)	(256)	0
Batch Normalization	(256)	1024
Dense Layer	(128)	32896
Dense Layer	(64)	8256
Dropout(0.3)	(64)	0
Dense Layer	(32)	2080
Dense Layer	(2)	66

```
randAugment = kcvl.RandAugment(
  value_range=(0, 255),
  augmentations_per_image=3,
  magnitude=0.2,
  magnitude_stddev=0.2,
  rate=0.5
)
```

4.2 The Preprocessing

We chose to perform preprocessing both on the original dataset and on the input Layer of the FEN. The final train dataset consisted of a set of images transformed by the functions *MixUp* and *RandAug*, [2]. In the Preprocessing Layer, instead, every epoch the images could undergo random transformations, including flip, zoom, rotation, or the addition of contrast.

4.3 The Hyperparameters

- **Batch size**: we opted for a batch size of 1024 images to speed up as much as possible the training.
- **Dataset Split**: having an initial dataset of 5004 images, we chose to split it keeping 10% of the dataset as a validation set and another 10% as a test set.
- **Total Number of Epochs**: the total number of epochs was 300 for both the Transfer Learning Training and the Fine Tuning, despite that the final epoch was never reached.
- **Early Stopping Patience**: 30. After the first training, the patience's value is used to calculate the *best epoch* and retrain the whole model for that number of epochs merging the validation and training set.
- **Fine Tuning Layers**: the number of layers that are kept frozen during the fine-tuning is the first 189, corresponding to the end of the sixteenth block of the second stage of the ConvNeXt Base network.
- **Optimizers**: after the tests on multiple optimizers, we recognized that the best performances came with Nadam.
- **Learning Rate**: we kept the default Learning Rate (0.001) for the first training phase, lowering it to 0.0001 during the fine-tuning to let the network just adapt and not change the weights too drastically.
- **Regularizers**: added the L1L2 regularizers to the dense layers to prevent overfitting and improve the generalization of models.
- **Dropout Rate**: 30% to reduce overfitting as much as possible.

5 Comparison between the models

Accuracy comparison between the tried models

Model	Accuracy	Test	Online Submission
MobileNetV2 [No Preprocessing]	0.6800	0.5010	Not Submitted
InceptionResNetV2 [No Preprocessing]	0.5200	0.6000	Not Submitted
Xception [No Preprocessing]	0.7700	0.5800	Not Submitted
EfficientNetV2S [6]	0.9184	0.8875	Not Submitted
EfficientNetV2M	0.9373	0.8770	Not Submitted
ConvNeXtBase	**0.9588**	**0.9421**	**0.8840**
ConvNeXtLarge	0.9158	0.9020	0.8570

6 Contributions

All the members of the team contributed equally to the project. Here is a list of the features that each member contributed the most to:

- **Filippo Balzarini - filomba**: Clearing the dataset, testing of new models and Hyperparameters, K-folding, class weight.
- **Francesco Benelle - Benels**: Testing of new models and layers combinations and Augmentation techniques on both the dataset and the preprocessing layer.
- **Michele Cavicchioli - el_miki**: Dataset balance, Augmentation techniques, Hyperparameters optimization (layers to freeze, Optimizer, Learning rate)

References

[1] Luke Wood et al. *KerasCV*. `https://github.com/keras-team/keras-cv`. 2022.

[2] Hongyi Zhang et al. *mixup: Beyond Empirical Risk Minimization*. 2018. arXiv: `1710.09412 [cs.LG]`.

[3] Ekin D. Cubuk et al. *RandAugment: Practical automated data augmentation with a reduced search space*. 2019. arXiv: `1909.13719 [cs.CV]`.

[4] Mingle Xu et al. "A Comprehensive Survey of Image Augmentation Techniques for Deep Learning". In: *Pattern Recognition* 137 (May 2023), p. 109347. ISSN: 0031-3203. URL: `http://dx.doi.org/10.1016/j.patcog.2023.109347`.

[5] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[6] Sedrick Scott Keh. *Semi-Supervised Noisy Student Pre-training on EfficientNet Architectures for Plant Pathology Classification*. 2020. arXiv: `2012.00332 [cs.CV]`.