Homework 1

Davide Rizzotti, Filippo Lazzarin, Simone Tognocchi, Enrico Marinelli

December 26, 2023

1 Introduction

The problem consists in a binary image classification problem. Our training set is composed by plants images divided into two categories: healthy and unhealthy. The goal is to predict the correct class label in 0: healthy and 1: unhealthy.

1.1 Approach used by the team

After the first phase of preprocessing and cleaning of the dataset, we decided to start working in parallel, so that each of us could experiment with their own ideas and different models. Throughout this individual exploration, we maintained an ongoing feedback loop, sharing results and discoveries. This approach facilitated extensive experimentation, enabling us to derive optimal insights from our collective efforts. We found this way of collaboration to be the best, since from each result we outlined the response to different initializations, hyperparameters and architectures, in order to gain experience to build the most performing choices.

2 Dataset Preprocessing

2.1 Data Visualization

Before building the model, we started by visually exploring the datasets containing images of both healthy and unhealthy plants. At first glance, it was not easy to spot a clear difference between the two classes. However, we found that in some instances the leaves exhibited distinct color variations for unhealthy plants, showing a mix of yellowish and brownish tones. These observations are essential for our preprocessing approach, especially when choosing data augmentation methods.

Figure 1: healthy samples

Figure 2: unhealthy samples

2.2 Outliers removal

While visualizing data, we have found several set of outliers composed by two pictures repeated for hundreds of samples in the dataset. They were composed by an image of Ėduard Chil', in his known character of Mr. Trololo, and an image of the popular animated movie Shrek (Figure 3). As a matter of fact we have mapped out the first two pictures of their kind, searching them manually and storing their pixels data as a variable. Luckly this data was the exact same for every outlier. That led us to remove them with a python script based on the variables previously defined, applied as a *numpy mask* on the *public_data* dataset. Hence we have defined a new dataset as *cleaned_dataset.npz* once the mask has been applied, that would have result in keeping all the clean, informative and necessary data while discarding the outliers that would have affected negatively both the training and prediction phases. In the end the new dataset has been saved as a file and loaded in every new model script, without the need of preprocess the outlier data again.

Figure 3: Left: Shrek, Right: Mr. Trololo

3 Model development

3.1 Introduction

Since the task to be solved is image classification, we decided to use Convolutional Neural Networks. Models of this kind are able to learn and extract from images discriminative features at higher and lower level.

3.2 Train, Validation and Test sets

The dataset was partitioned as follows: 80% for the training set, 10% for the validation set, and 10% for the test set. We decided to assign a significant portion of the dataset to the training set due to the limited availability of data.

3.3 Pretrained models

Given that training a big CNN from scratch is both computational expensive and a very complex task, we decided to use pretrained models available in the Keras.applications library: in this way we could benefit from the large and various set of features extracted from a very big dataset with a lot of different images, achieving better results. We decided to use different EfficientNet and ConvNeXt models which are trained on the ImageNET dataset (containing more than 14 millions images). We started using smaller models to understand the initial performances of these models, then we moved on using more powerful models with a higher number of parameters, while keeping an eye to possible overfitting issues.

3.4 Transfer Learning

Transfer learning is exactly the process of replacing the last fully connected layers of a network with other layers in order to adapt it to another task. In our case, we want to replace the classification output layers of the pretrained network with a layer containing only two neurons, since we are addressing a binary classification problem. Finally, we retrain the network by freezing all the weights in the pretrained network (e.g. we force this weights not to change during training). In this way we are able to fully exploit the power of the pretrained network to solve our task.

3.5 Fine Tuning

With transfer learning just the "top" of the network (the layers involved in classification) is learned during training. The idea of fine tuning is to boost the performances of the pretrained network, by retraining with a small learning rate the last convolutional blocks of the network: thanks to this process the network becomes even more specialized on the specific task by refining, or literally "fine tuning", the latest features extracted by the model. In general, we have assessed that a fine tuning procedure lead to up to a 5%-7% upgrade in accuracy performances.

3.6 Training

For the training process, we used Adam with weight decay as the optimizer, in order to apply regularization and hence reduce the risk of overfitting. A batch size of 64 was used as it was seen to be the best trade off between performances and training time. Moreover, again to avoid overfitting, early stopping with a patience of 20 epochs was enabled. Finally, an interesting technique that allowed our models to perform better was to train the model again on almost the whole dataset (leaving out some samples for a validation set, used by earlystopping) to leverage learning on as much data as we could.

3.7 Data Augmentation

Up to this point, using only transfer learning and fine tuning, we were able to achieve performances around 85% percent of accuracy: we knew we had to experiment with something else. We decided then to introduce data augmentation in our training pipeline. First of all we tried individually all the methods of Keras Image Augmentation Layer. In this way we evaluated our models trained with one preprocessing layer with a single augmentation method and we observe that the methods that increased the most our performance were random translations and random flip, that improved the validation accuracy of 4 and 3% respectively. For all the models developed afterward we used a single preprocessing layer with this two augmentation methods implemented sequentially, obtaining test accuracy improvement of 3% and more balanced predictions.

3.8 Hyperparameter tuning

Along with data augmentation, also hyperparameter tuning techniques have been used to understand what are the best choice for the parameters of the network, in particular they have been useful to:

- choose the number of neurons in the last dense layers, the dropout rate for dropout layers, and the learning rate for the training process

- understand if adding some layers led to better performances or not (in particular for the dropout layer)

It's important to remark that this process could be done only for smaller networks (such as EfficientNetV2M), because it requires a lot of different trials: with just 5 parameters to tune it becomes reasonable to try even 30 different combinations, which requires a lot of time.

3.9 Final models architecture

Our models were structured with the following architecture:

- Input layer of dimensions 96x96x3
- Data augmentation layer with random flip and translation transformations
- The pretrained model, with the top of the network removed
- Sequence of dense, dropout and batch normalization layers
- Dense layer with 2 neurons at the output layer, corresponding to the 2 distinct classes.

3.10 Ensemble of models

By means of data augmentation and hyperparameter tuning we were able to achieve around 90% accuracy with both EfficientNet and ConvNeXt models, but what helped us to get into the top and reach 94% of accuracy, making more reliable and accurate predictions, was the idea of creating an ensemble of the best models created by us. In the first ensemble, we used a ConvNeXtXLarge, a ConvNeXtBase and an EfficientNetV2M, obtaining a performance of 93%. Then we created an ensemble with 4 models: an EfficientNetV2M, an EfficientVNetV2L, a
ConvNetXtBase and a ConvNetXLarge. The prediction was done through majority voting, in which each network has a weight of 1 vote, except for the ConvNetXLarge, that was the most performing on CodaLab, with a single accuracy of 90%, that has a weight of 2 votes. This new ensemble network obtained an accuracy on the test set of CodaLab of 94%, a precision of 97.06%, a recall of 86.84% and a F1-score of 91.67%.

4 Conclusions

Upon analyzing the obtained results, we deduced that a degree of overfitting occurred in the test set during the development phase. Our ensemble model, which initially achieved a 94% accuracy, experienced a decline in performance, reaching 86.50% accuracy in the final phase. While the results remains reasonably good, it's evident that certain modifications in the model have resulted in a slight overfitting to the Codalab test set.

5 Contributions

- Davide Rizzotti: Development and training of EfficientNet models, both L and M versions. Hyperparameter tuning. Most of contributions in the Introduction and Model Development parts of the report.
- Simone Tognocchi: Data Augmentation, development, training and hyperparameter tuning of the ConvNetXtXLarge model. Creation of the python script to embed the different models. Contributed in writing and revising the report.
- Filippo Lazzarin: Preprocessing of the data, including data visualization and removing the outliers in order to obtain a clean dataset. Tried different models, including EfficientNet and ConvNet, and tried different configurations. Contributed in writing and revising the report.
- Enrico Maria Marinelli: Training of ConvNeXtXLarge and Efficient-Netv2 models, with Hyperparameter Tuning. Contributed in writing and re-vising the report.

Model	Accuracy	Precision	Recall	F1-score	Phase
EfficientVNetV2L	89%	84.62%	86.84%	85.71%	Development
ConvNeXtBase	89%	88.57%	81.57%	81.57%	Development
ConvNeXtXLarge	90%	91.18%	81.58%	86.11%	Development
Ensemble	94%	97.06%	86.84%	91.67%	Development
Ensemble	86.5%	84.90%	78.42%	81.53%	Final