Homework_01 - Report

Authors: E. Broggini, 10670120; E. Cabiati, 10698131; L. I. Pagliochini, 10707169; F. Zhou, 10679500.

---

1. Introduction

Given a set of 5200 images of plant leaves of dimensions 96x96x3 labeled either as healthy or unhealthy, build a model able to classify plants among the two classes.
Different approaches were considered to tackle the problem at hand: even though, in the end, we were not able to achieve the results we set out to, we would like to present a few models that particularly convinced us of their validity, as well as the path that we followed to get there.

2. Data Inspection & Cleaning

We started off our work by inspecting the given data, plotting random samples out of the dataset. In the process, we soon came to realize that the data was filled with nonsensical images, "outliers", to call them so, together with a bunch of other duplicates of "coherent data".
So, *we proceeded by not only eliminating the outliers, but also all the other duplicates of leaves we could find*.
Nonetheless, *we assumed the remaining data to be valid, that is, to be correctly classified*.
We then moved onto considering the distribution of classes: we noticed a relatively slight unbalance towards the healthy class, with respect to the unhealthy ones.
This was, in general, overlooked during our first analysis, to be later taken into considerations in some models proposed, by performing some non-canonical oversampling on the unhealthy instances through simple geometric transformations.

3. Dataset Partitioning

We opted for a **90:10** *split between training and validation*. This was *to make use of all possible data* we could get for the training phase of the models. Some slight variations were performed, especially for those cases where the plotted graphs of loss and accuracy showed signs that the validation was not representative for the training set generated.
When occasionally using our own test set, extracted from the given data, we spotted a recurrent gap between the accuracy achieved on it and the one based on the hidden test set. This was the first hint we got to further improve the generalization property of our nets.

4. Data Augmentation - General Approach

Regarding data augmentation, our team chose to employ mainly *geometric transformations* instead of intensity-related ones, related to color augmentation. For instance, we utilized a majority of rotations, flip, zoom and translations by little factors, to not spoil excessively the dataset. This was done to avoid possible chances of misclassification: in our opinion, in many cases, the colors associated with healthy samples were generally brighter than those associated with unhealthy ones. As a result, we mainly stuck to geometric transformations, aware of the risk that this might not introduce robustness of the model for this kind of intra-class variability.

5. First steps and Hand-Made CNNs

The rationale behind building hand-made models was to have some sort of reference on how different components of a CNN could affect the learning process, as well as checking whether the behavior of the training was consistent with the theory.
The base architecture we came up with consists of five convolutional layers with an increasing number of filters at each level. Each layer consists of: *Convolution2D*, *ReLU activation*, *Batch Normalization*, *Average Pooling* and, optionally, also *Ridge* regularizer along with *Dropout*. The adopted kernel size was constantly set at 3x3 in order to achieve *more nonlinearity* in the model (as, clearly, the separating hyperplane we are looking for is not linear) while having less parameters to train. After the convolutional part, a *GAP* layer was inserted for further dimensionality reduction. Lastly, the output layer consists of a single neuron with sigmoid activation function.
By using this architecture as a benchmark, we performed some *a priori estimates* for the complexity of the model required by the given problem and the impact of different parameters' tuning.We found out that even with simple nets the learning process was feasible, therefore, the problem to be addressed was *finding the proper generalization of the model*: when using the proper augmentation set it was more prone to generalize and perform

better in validation; when using mixup there were no significant improvement (this could be due to the fact that convex combinations led to some weird overlapped shapes and thus induced the model to learn noisy information); with small batch size the metrics' behavior was wavy, instead with large batch size we got more smoothness (this could be due to the fact that in the first case the update direction is computed locally while in the second case it is computed globally). As for learning rate, for the time being LR, the problem to address was to find a proper magnitude to cope with problems related to *slow convergence* for small values of LR and *overshooting* for large ones. This issue was addressed by adopting different LR schedulers (*Polynomial, Exponential, Cosine*) and *ReduceOnPlateau* callback. No significant improvements have been recorded while using *kernel initializers* (*He, Glorot and LeCun*) nor with *Dropout*, while *Ridge* was useful for more complex models. For what concerns the training algorithm we chose *AdamW* as it is the state of art optimization algorithm that copes with a wide variety of optimization issues ranging from *non convex optimization*, that is local minima, to *regularization*.
The best performing model developed with this approach scored roughly 70% on validation accuracy and got the same result in testing during the development phase. This outcome, even though not outstanding, was significant because a relatively simple and underfitted net was able to perform better than more complicated ones.
This phase was useful for understanding the impact of each (hyper)parameter in order to properly address more complex architectures. Some additional tests were run on AlexNet-like architecture using CutOut preprocessing.

6. Transfer Learning and Fine Tuning

As the limits of the handmade CNNs became clearer, we decided to gradually move onto transfer learning and fine tuning: as a first step, we performed some benchmark tests, using different backbones for the network, using the same parameters. This first step was done in order to determine the (potentially) most promising networks to be used for the specific problem.
In particular, we tried *EfficientNet* (different versions), *VGG16*, *DenseNet*, *InceptionV3*, *Xception*, *ResNet* and *MobileNet*. From these first tests, we decided to focus mainly on *DenseNet*, *Xception* and *ResNet50_v2*. Later on, *DenseNet* was left aside as deemed too complex for the problem and, as such, seemed prone to overfitting. In these cases, oversampling was applied to get rid of class imbalances.

ResNet50_v2*:** It was the model which achieved the best accuracy on the development test set (80%). We started working on it by performing a simple training phase, freezing the layers of the CNN, while using the original cleaned dataset, augmented through the means of simple keras preprocessing layers (including brightness and contrast). This was later followed by a fine tuning phase. At first, *we tried by unfreezing the first 32 layers* of the net (we also tried other partitions for the convolutional part, but it didn't seem to influence excessively the accuracy on validation) and letting the model train just on the original dataset, which led us to an accuracy of ~80%*** on validation.
We then tried to optimize it by *removing the brightness and contrast filters*, fearing those would introduce too large variability within classes, while *manually expanding the dataset using the keras-cv library* to perform MixUp on the given samples (to be used only for the training of the FC layers). Also, adding some *skip connections* in the dense layers seemed to *stabilize the model during its transfer learning phase*. Following these changes, we achieved ***82%*** accuracy on the validation and 80% on the developing test set.
Trying to optimize this model further still, we tried to apply an internal resizing layer to fit the images to the original input size of the backbone net (this approach didn't seem to improve the performances considerably) while letting all the layers of the convolutional part train during fine tuning. This last change let us achieve a score of ***84%*** on our own validation set and a 72.4% on the test set of the final phase.
A c*osine decay LR scheduler with warmup* was also used, to allow for better adaptation on the augmented dataset but, again, didn't seem to introduce significant changes, as the model would quickly reach convergence or tend to overfit.

Xception (Final Model)*:** This was our most promising work and the one that convinced us the most, getting an accuracy of ~92%*** on our validation set, and showing some nice tendency in loss and accuracy graphs.
We propose two models based on *Xception* backbone, followed by custom fully connected layers.
A first model is characterized by the *same preprocessing used for the ResNet architecture* (with some minor changes due to the use of translations) and *cosine decay scheduler*, which, in this case, greatly helped during the transfer learning phase. As for fine tuning, we started again by unfreezing only some of the last layers of the

backbone, but later realized that by letting all them train, the accuracy on validation greatly improved after this phase (from ***86-87%*** to ***90***%). Another additional ***2%*** was added by rescaling the images inputted in the net. This first model achieved on the final test set an accuracy score of 74.2%.
A second attempt was made by v*arying the translation factors* (from (0.1, 0.1) to (0.3, 0.3)) and *adding brightness* in the augmentation layers with a factor of 0.2, trying to let the model become more robust to translations and different lighting conditions in the input images. An additional change was applied by freezing the last 86 layers of Xception in the fine-tuning phase.
Using this second model, we performed a validation accuracy of ~***87%*** and ***75,6***% on the test set of the final phase, earning rightfully the title of **final model**.

As a separate note, we also tried submitting a supposedly underfit model based on *Xception* backbone with augmented and mixed-up data.The validation accuracy reached a maximum value of 67%, while the development test accuracy proved to be comparatively higher, with a value of 71%. This discrepancy hinted at classical signs of underfitting.
Following the completion of fine-tuning, in which only the last 80 *Xception*'s layers were set trainable, the model achieved a validation accuracy of 68.25%, indicating a mediocre fit on the validation set.
While not excelling in capturing the small details of the training data, this underfit model demonstrated an unexpected accuracy of 72.3% on the hidden test set of the final phase.
This feature led us to highlight the importance of considering the broader implications of model behavior beyond traditional training parameters.

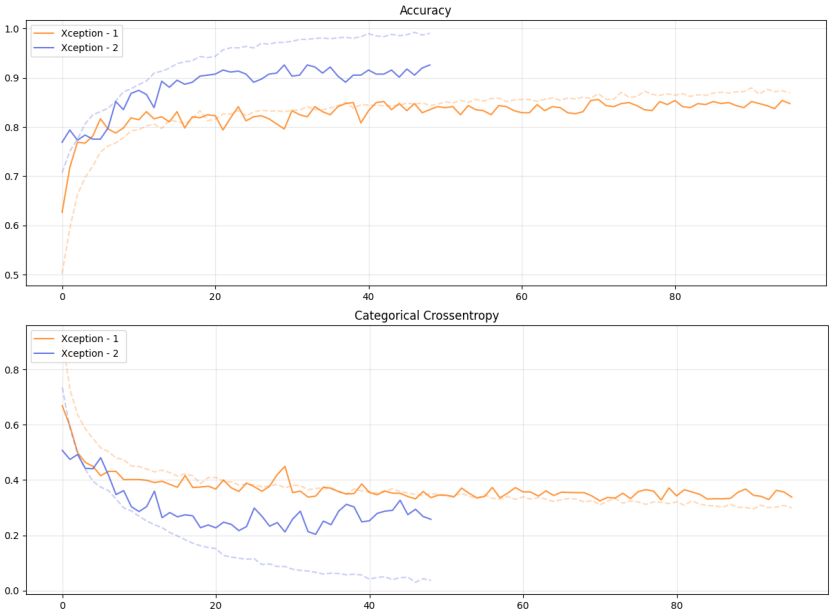

Fig. 1: loss and accuracy graphs of the final model(version 1). Traces of overfitting visible on the loss graph after fine tuning, duly prevented by means of EarlyStopping.

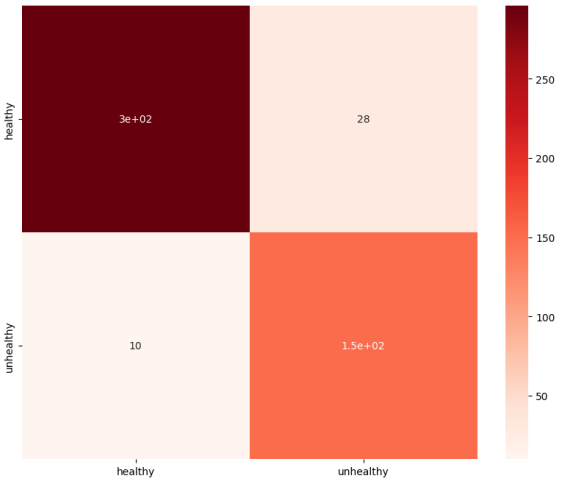

Fig. 2: Confusion Matrix of the final model(version 1) on our validation set

7. Ensemble learning

After having developed the above-mentioned models, we have also tried to merge them using ensemble techniques, combining the predictions of multiple models (2 in this case) to enhance the overall performance. The approach we used is the "majority voting" method that involves combining predictions from different models in the ensemble by assigning a vote to each possible classification outcome. In this way the final vote will be defined by the class that receives the majority of votes.
In our case, this approach increased the accuracy value in our own validation set, as expected, but not on the test set of the first phase.Subsequently, after trying this on our two most promising models at the time (*ResNet* and *Xception*), but getting unsatisfactory results, we decided to stick to simpler architectures.
In our opinion, this behavior could be related to the fact that both the architectures employed in the ensemble had been trained with the same data augmentation filters.

8. *Contributions*

We tried to work in parallel to optimize the job as much as possible, while still trying to exchange relevant information by conducting periodical checks and validations on each other's work. The following were the main topics each member of the team focused on:

E. Broggini: transfer learning and fine tuning of DenseNet and Xception
E. Cabiati: analysis on AlexNet, transfer learning and fine tuning of ResNet and Xception
L.I. Pagliochini: transfer learning and fine tuning on ResNet50_v2, InceptionV3 and Xception.
F. Zhou: hand-made CNNs and benchmarking.