AN2DL - Homework 1

Image Classification

Giorgio Miani, Francesco Micucci, Pietro Pasquini

18 November 2023

1 Introduction

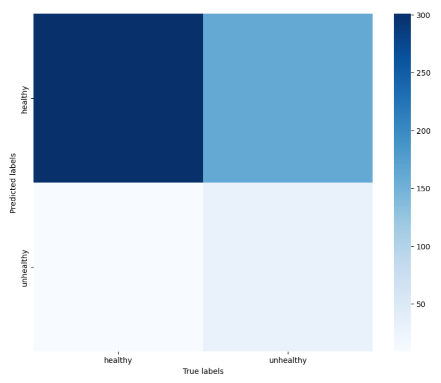
The homework was to train an image classification network in such a way to distinguish whether a plant is healthy or not by using its photo.
To achieve this task we proceeded in the following way: we inspected the dataset, we implemented a baseline model using transfer learning and at the end we tried different optimization techniques.
We looked for the most performing feature extractor to use and we did some tuning on the hyperparameters. We also tried to improve our model by implementing data augmentation first, and K-fold cross-validation and fine tuning then.
Regarding the evaluation of the various models created, we divided the dataset into training and validation using a standard 90-10 split and we used the accuracy on the validation set as a metric for comparing the models. At the end, we used the accuracy on the hidden test set to evaluate the overall performance of the very best models we have obtained.

2 Cleaning Dataset

When we started the project, the first thing we did was to inspect the dataset.
This was a really important step because we immediately noticed that there were outliers inside the dataset (we found 98 images of ‘Shrek’ and 98 images of ‘Trololo’ inside of it).
Luckily, the outliers were all of the same kind so we were able to build a fast outlier removal strategy. Instead of removing the images that were not suitable for our case study one by one, we took the convolutional part of one of the networks provided by `keras` (*MobileNetV2*) and we used it to make a prediction on the 2 images associated with the outliers (more precisely the image 58 which is associated with a Shrek image and the image 701 which is associated with a Trololol image) saving the associated results. Now, whenever the mean square distance between the prediction of one of the dataset images, obtained using the same network as before, and the prediction of one between the Shrek image and the Trololo image was below a certain threshold (0.1 in our case) we added the database image index to a particular list. At the end we create a new dataset adding only the images and labels of the data having indexes not contained in that particular list.
Obviously we checked that the images deleted from the dataset were outliers as well as we checked that no incorrect image was left in the dataset.

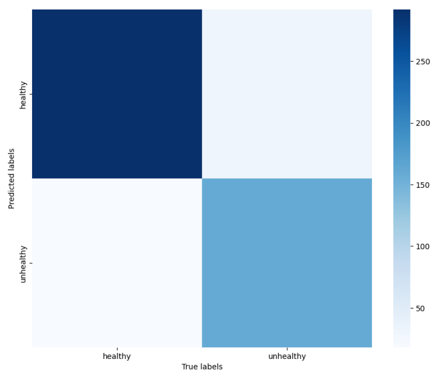
3 Baseline Model

The very first idea was to create a baseline model without any optimization and use the results obtained from this network as a benchmark for future implementations. The choices behind this network were a consequence of what has been done during the laboratories. In fact, we started directly using transfer learning and in particular the model we used as a feature extractor was MobileNetV2. Moreover, we expected the network to overfit the training set and through future modifications we would have liked to limit this phenomenon.
To create such a network we introduced 2 dense layers with 32 neurons each having RELU as activation function as well as a final layer composed of 2 neurons having a softmax. We used the Categorical cross-entropy as loss function and Adam as optimizer.
As expected, the model achieved excellent performance on the training set (because of the overfitting) but failed to exceed 84% accuracy on the validation set.
The first solution that came to mind was to limit overfitting using techniques such as Early Stopping, Learning Rate Scheduling and Dropout. The results obtained with these techniques were slightly better but still unsatisfying.
We also proceeded with a first draft of models using data augmentation techniques however the efforts were fruitless, and the accuracy on the validation set was still disappointing. At this point we decided to inspect our results and we noticed that predictions were heavily skewed towards the healthy side as we can see from the following confusion matrix.



4 Feature Extractor Selection

We started to question the choice of using *MobileNetV2* as a feature extractor, thinking that the features it provides were not able to cover the aspects that identify a plant as healthy or unhealthy. We therefore decided to test all the networks provided by `keras.applications` as feature extractors.
After having trained all the networks using early stopping and Learning Rate Scheduling (to both avoid overfitting and save time), our best results were obtained with *ConvNeXt-Large* which from this moment on became our reference for extracting features. As we can see from the confusion matrix, the predictions are much more balanced now using *ConvNeXtLarge* than when we used *MobileNetV2*.

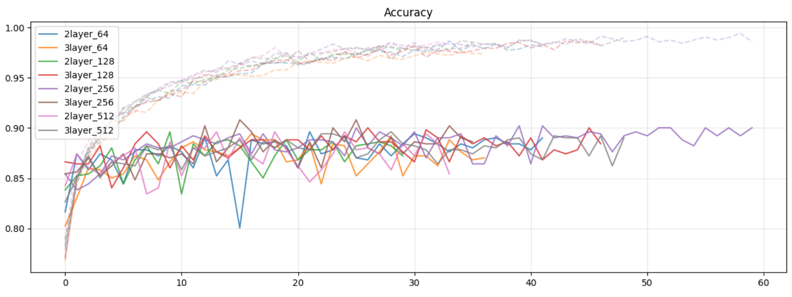


5 HyperParameters Tuning

After having selected the feature extractor we started tuning the hyperparameters: dropout rate, number of layers and number of neurons in the classification layers.

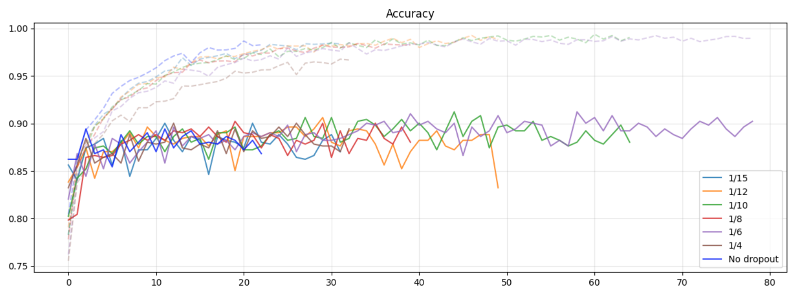
Classifications Layers

The first tuning experiments we performed were on numeber and dimension of the classification layers.
We tried 2 or 3 dense layers, with 64, 128, 256, 512 neurons each. However there was little to no difference between the tests, so we decided to stick with 2 layers of 256 neurons.



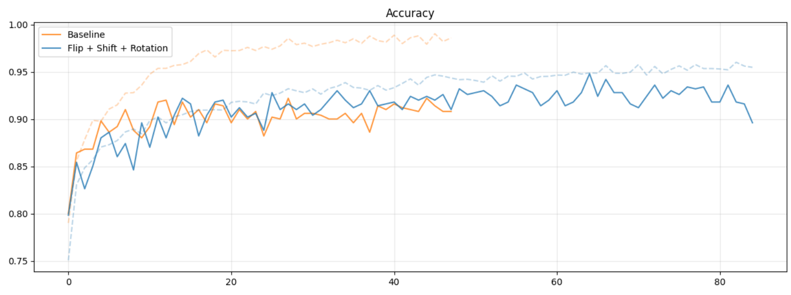
Dropout Rate

Then we tried a similar approach with the dropout rate, testing the values $\frac{1}{15}$, $\frac{1}{12}$, $\frac{1}{10}$, $\frac{1}{8}$, $\frac{1}{6}$ and $\frac{1}{4}$.
Similarly, there was not much difference between the tests, so we decided to stick with $\frac{1}{10}$.

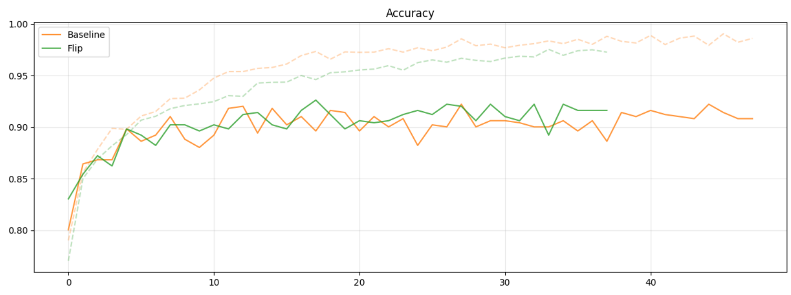


6 Data Augmentation

Next we tried to improve our accuracy on the validation set using augmentation techniques on the dataset. In order to do so we used the `ImageDataGenerator` class from `keras`.
First we created the object of the class, chose different filters with various ranges and then we selected the "reflect" method to fill empty pixels left from the transformations.
Finally, we trained the model using the flow function of `ImageDataGenerator` on the training set.



We initially tried 6 different transformations separately: flip, shift, rotation, shear, zoom and brightness. They were all applied to the images provided as input to the best network up to this moment at training time. We used the accuracy on the validation set to evaluate which transformation was meaningful. Using this performance metric we started combining transformations one after the other, adding transformations from the most performant to the least performant one, and evaluating each time the accuracy.
We kept the combination of transformations that have increased the accuracy w.r.t the base case where no transformation was applied.
In particular, the graph above represents the accuracy of the model to which we have provided images augmented with flip, shift and rotation transformations at training time, meanwhile the graph below is the accuracy of the model to which we have provided images augmented just with the flip transformation.



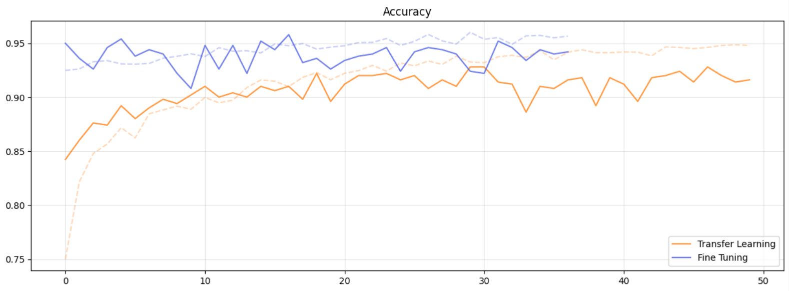
Furthermore we took the model with the best combination of image transformations and we trained it again using different methods to fill the empty pixels produced by shift and rotation transformations: more precisely we tried constant, nearest and wrap techniques. The results were worse than before and the accuracy dropped a little bit. This is probably due to the particular texture of the images from the dataset, which would be disrupted by any significant alteration.

7 K-Fold

To avoid overfitting and to exploit all the data at our disposal, we also implemented the K-Fold cross validation with 10 folds.
For each fold, we used our model to find the epoch with the highest accuracy on the validation set and then we trained the model over the whole dataset for a number of epochs equal to the average of the best epochs found.
Finally we evaluated the model on the hidden test set, but at the end the results were disappointing as the accuracy had slightly decreased compared to the case where the K-fold was not used.

8 Fine Tuning

Finally we implemented the Fine Tuning technique. Since the feature extractor we are working with is pretty big, we decided to apply it just on a small portion of *ConvNeXt-Large*, unfreezing the layers after the 286th. Even though this technique helped to improve the accuracy on the validation set, in the majority of the cases, when we moved to the test set, the accuracy was less than the one achieved with the same model trained just on the dense layer part.



9 Final Models

In the final phase of the competition we submitted 5 different models, all based on our ”standard” model (*ConvNeXt-Large* as feature extractor, 2 dense layers with 256 neurons each, dropout rate of $\frac{1}{10}$, Adam as optimizer and Categorical cross-entropy as loss function, with Early Stopping and Learning Rate Scheduling):

- In the first model, we used data augmentation with flip, translation and rotation. (This was our best performing model in the Development Phase, with an accuracy of 0.94 on the test set)
- The second model is based on the first model’s results, but we tried to improve it by adding K-Fold cross validation with 10 folds.
- The third model was the same as the second one, but with a brightness filter added to the data augmentation. (This choice was based on the fact that the brightness filter, with a very low parameter, didn’t improve the results of the second model, but it didn’t make them worse either.)
- For the fourth model, we tried to improve the third one by adding fine tuning, but the performance was disappointing.
- For the last model, we used the fine tuning technique on the first model.

Contribuitions

Throughout the project we managed to split the work evenly, by often meeting up to work after the lectures or on our off days.
For example, in the casa of data augmentation, each of us tried different combinations of filters and then we compared the results to choose the best one.
In the end we put toghether a complete notebook for each section with all the results.