Plant classification challenge

Throughout this challenge we successfully managed to train and test a neural network with the task of classifying the wellness of plants, from leaves' pictures.

Clearly the first thing on our bucket list was gaining a sufficient understanding of the dataset, which was composed of 5200 rgb pictures in the form of (96,96,3) arrays and the corresponding label 'healthy' or 'unhealthy'. At first glance we couldn't notice really distinctive features to clearly classify the images. With this in mind we decided to start our implementation of the task with a very basic model [1], "model zero", that would act as a stepping stone for our assembly line: not surprisingly the performance wasn't too good.

We decided to deeper inspect our data, namely, looking for the presence of outliers and/or unwanted data. At first we applied a usual approach with IsolationForest from sklearn.ensemble, but we didn't find anything too strange, probably because there were so many 'outliers' that weren't considered as such. Then we used the model zero without the last layer to extract features' vectors of all images and on those we looked for the bigger euclidean distances from normal plants and we found around 200 images of popular memes, after removing those we applied again the IsolationForest approach with better results. We found that outliers were considered pictures with soil in them, that were extremely bright or that had leaves with a curly appearance, but since we didn't know if those pictures could be in the test set, we decided to keep them.
Moreover, while inspecting 'near' and 'far' images we found doubles, so we proceeded in removing those as well.
At this point the dataset was composed of 4850 photos, but we noticed an imbalance in the distribution of classes, the healthy images were about 60%, we took note of this and proceeded in seeing how the model would behave with the unbalanced dataset.

Further analysis of data was made about possible augmentation; it was obvious that classes of plants were independent from geometric transformations like flips and rotations of 90°, 180° and 270°. Also other transformations like translation and other rotations were supposed not to modify the classes, but we had to be cautious about this since elements causing the plant to be unhealthy could be found near the edge, a 'wrap' fill mode was considered a safer approach for this reason.

After splitting into training and validation (20%) sets and implementing one hot encoding for the target array, we increased complexity until we started noticing overfitting. This method produced a model of five convolutional layers with max pooling. This architecture was chosen through trial and error regarding: the layer ending the convolution part of the network, activation functions, loss estimators and weight initialization. We inferred that the model was working better with a GAP layer, Relu functions, categorical cross entropy, He Normal initialization weights.

In order to improve our accuracy we chose to implement different augmentations: rotation, translation, flips, zoom, contrast, brightness and sharpness. At first, noticing the difference in colors between some healthy and unhealthy plants and noticing that the misclassified plants were mainly darker healthy plants and faded unhealthy, we thought that the color based augmentations would have helped the model to learn different distinguishing features, more general to the whole of the classes, but the result showed us the opposite: geometric transformations were helping more.

[1] A few convolutional layers with GAP followed by a dense layer with two neurons, one for each class, and a softmax activation function

At this point we tackled the tuning of hyperparameters and augmentation layers, to do this we developed a notebook that simply needed the parameters to run and would shuffle through them training the different models and would output the parameters that resulted in higher accuracies. To fasten this study we applied early stopping, based on validation accuracy, since this was the valuation criterion. From this study we learned that rotation and flip augmentations were always part of the best combinations, sometimes mixed also with a light-medium random translation and medium-high random zoom out. Furthermore, the best hyperparameters for model with this complexity were a dropout of ⅛ or ⅛ and a batch between 100 and 130.

Once we found the best setup for a few different models, which differed by the number of convolutional layers, their kernels and the number of dense layers, we uploaded them, but on the test set we received an accuracy lower by 0.2/0.25.

At this point, before diving into transfer learning, we continued the search for a more complex model through the use of blocks of convolutional layers together with inception and skip layers. This produced an improvement, but we saw that without some ways of lowering overfitting, the training accuracy was growing too fast to have a model that would perform well on the test set.
A few strategies were implemented mainly L2 regularization, batch normalization and learning rate decay. The first two resulted in more stable training w.r.t. overfitting and convergence while learning rate decay seemed to not have a relevant impact.

The improved models with convolutional blocks and batch normalization brought us to a 0.96 training accuracy and 0.92 validation accuracy. Aware of the presence to some extent of overfitting, we uploaded, but nonetheless the results were far worse than what we expected, the very best test accuracy reached was 0.71.

Transfer learning was always on the back of our mind since the early stages given the seemingly difficult to interpret dataset. We started implementing the pre-trained models with MobileV2Net and VGG16 which served as a sort of training camp in which we experimented with all sorts of things: numbers of head dense layers, batch size, activation functions, number of fine tuned layers and so on. Since the models were already working well on training and validation sets we decided to upload a few attempts, but we still couldn't surpass 0.7 test accuracy.
We decided to work on the dataset, removing elements from the healthy class. This was done by the application of euclidean distance on features' vectors and removed images most similar to other images, to avoid losing too much information about the class. We decided that lowering the dataset dimension wouldn't be too harmful, since we were using pre-trained models that had fewer weights to learn.
At this point we used the new 'balanced dataset' on a different pretrained model, EfficientNetB2, this finally resulted in a 0.75 test accuracy. With this reassuring information a process of selection for the best EfficentNet started, after training with all the fifteen models between version one and two, EfficientNetV2S revealed itself as the one who seemed to perform best.
We began a day-long process of trial and error tuning through the use of our newly updated code to find the best combinations. In this scope fine tuning was tested more than one time but every time, despite the gradual unfreezing of the layers and the small learning rate, brought to the table minor performance improvements and a lot of overfitting; this could be due to the contrast between a large model and a small dataset and was indeed not considered for the most part as an available strategy.
The final improvement was brought by test time augmentation. Firstly, we decided to split the dataset with k-fold and trained the models on this division, this was done to assure independent results for time augmentation, the results were studied on the means of the different accuracies.

The time test augmentation consisted in predicting results on transformations of images and then, before the computation of accuracies, combining the predictions with each other and normal prediction. Studying the combinations one by one, we found, as expected, that flips and 90°,180°,270° rotation always improved predictions, so these were added equally. Then other transformations as translation, zoom, brightness and contrast were found to have different improvements. Finally, after getting feedback from test accuracies on codalab, we settled that the best test time was the one with basic geometric transformations.

All the preceding information together produced a 0.83 validation accuracy on phase1. But uploading the same model on phase2 produced a mere 0.77, but we noticed an impressive recall value of 0.88, this meant that our solution for class imbalance wasn't enough, furthermore from wrong uploads we noticed that both phases the distribution of the classes on the test set was 0.38/0.62 nearly the opposite of our dataset. So the final stage of our work began.
Cost sensitive training was implemented through the "weighted_categorical_crossentropy", this custom loss function assigned higher weights to the minority class thus penalizing the misclassification of that class more heavily during training and helping the model to pay more attention to the minority class and make better predictions for it. Thanks to this approach we increased the unhealthy accuracy above the healthy accuracy, thus increasing the overall accuracy.

In conclusion we found the best model to be EfficientNetV2S with a classifier made of two dense layers of 128 and 64 both with Relu activation functions, He initialization and L2 regularization and dropout with a rate of ⅛ trained through the custom loss function "weighted_categorical_crossentropy" and with a batch_size of 64.

Extra tries, e did the following sections but poor results:
• mixup augmentation of balanced data:
In order to achieve a better precision, after noticing the difference in number between healthy and unhealthy plants (62% vs 38%), we decided to balance them. First we tried an over-sampling technique which implemented some gaussian noise, but with poor results, so we took the opposite way. In fact, we decided to cut away some of the healthy plants, until we reached the right number. Even if we knew that the dataset would be smaller we gave it a try, placing it side by side with some augmentation, and we obtained good results.
Then, we constructed a mixup augmented dataset, we applied our temporary best model on it but the results were not as desired, so we decided to not have it in the final submissions.
• Uploading a model that put together a 2 different models
• K-fold and training and the full dataset, or k-fold and uploading a mix of the models
• Newest augmentations, taken in keras_cv libraries
• Nearest neighbors
• Binary cross entropy without hot encoding

Plant classification challenge

Contributions:

- Simone Paloschi: data visualization and basic model construction, outlier and double detection, hyperparameters tuning, automation development, transfer learning best model research, test time augmentation
- Claudio Strino: data visualization and basic model construction, augmentation tuning, overfitting modulation, transfer learning best model research, test time augmentation, further augmentation research
- Andrea Rella: data visualization and basic model construction, first transfer learning approach, transfer learning best model research, transfer learning implementation and fine tuning, cost sensitive training