First challenge: Image Classification

Marco Bonalumi
Pasquale Dazzeo
Riccardo Ambrosini Barzaghi

0. Introduction

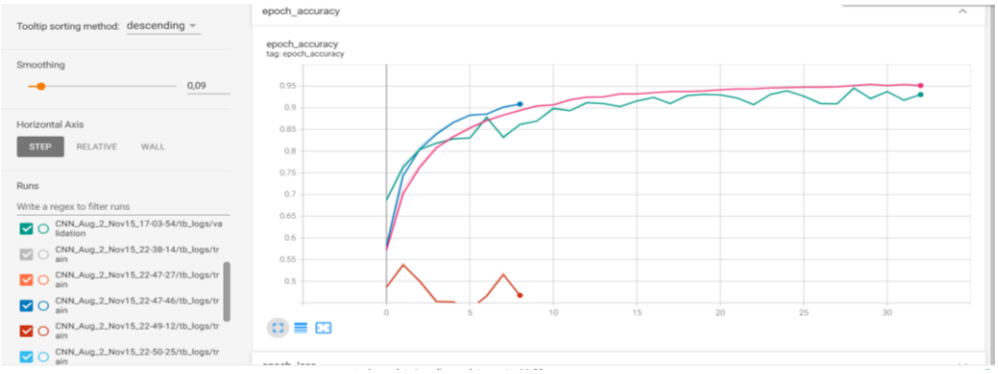
The first challenge required to classify images of leaves, divided in categories according to the species of plant to which they belong; being a classification problem, the goal was to predict, given an image, the correct class label.

1. First steps

We started the challenge by adapting the model seen during the exercise sessions to our problem in order to have a starting point.

Our first submission consisted in a network with 5 *2D Convolutions* and *MaxPooling* in cascade for the feature extraction and *GAP-> Dropout-> Dense-> Dropout-> Dense* for classification*;* we also added data augmentation (flips, zoom and rotation) and we got good result on our validation set; given the little number of images, we noticed that augmentation was a needed procedure, so we always applied it in later models. We substituted the *Flatten* layer with a *Global Average Pooling* because it reduces the number of parameters and makes the classifier invariant to shift transformations.

This model managed to score 65% on the test set, although it had over 90% on the validation accuracy during training. We decided to give a try to more specialized models and see if the submission score would improve.



TensorBoard showing epoch accuracy for the models trained. The submission scoring 65% was the red model.

2. Transfer Learning

In parallel we also tried Transfer Learning on VGG16 and InceptionV3 as to see how some well-established models would perform on our problem.

In this first part, we were only interested in the feature extraction part of those models, the classification layer remained the same as the previously mentioned one.

Both models performed well on the validation set (around 90%) and we decided to try and submit an InceptionV3 model even though it was not fully trained; it got a 74% on the test set.

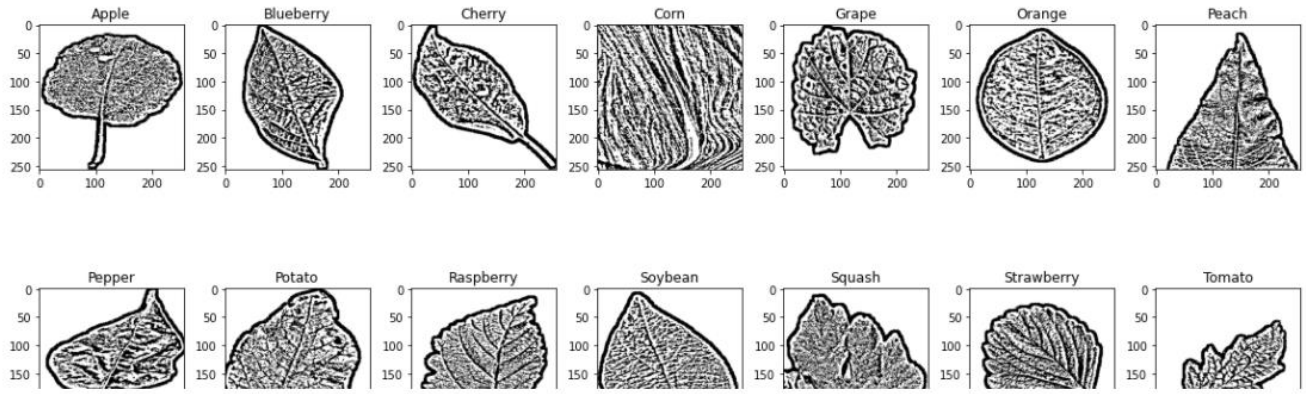
We then tried to fully train it but evidently it overfit given that the complete model managed to score 67% accuracy.

3. Preprocessing tentative

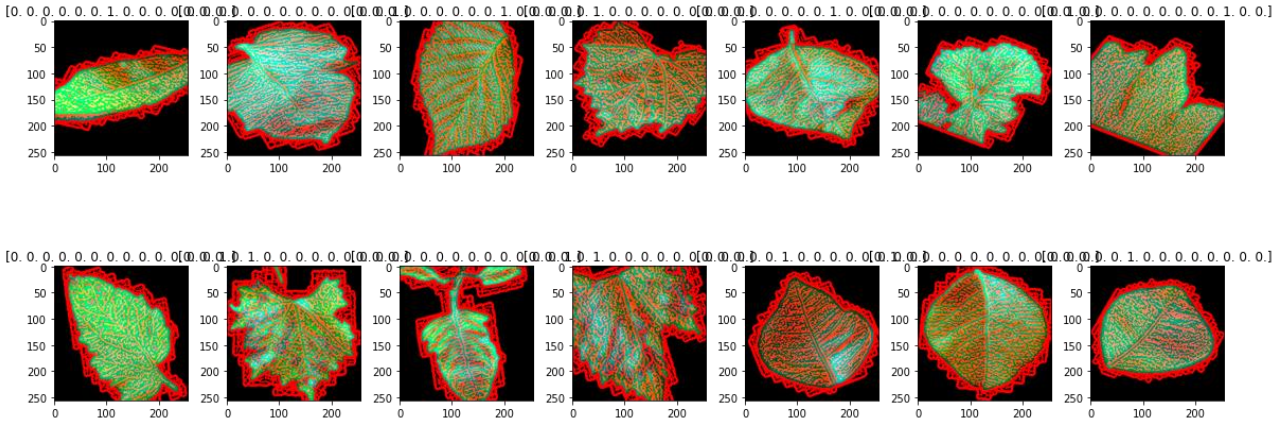
We wanted to preprocess the dataset with the aim of emphasizing the two main characteristics of the leaves: the veins and the shape.

We referred to a paper[1] that suggests a preprocessing of the red channel of the image, the least informative for image classification, and substitute it with a channel on which adaptive thresholding is applied.

We found that by using the *adaptive thresholding* method of OpenCV we could generate images that we considered more informative than the simple red channel.



Unfortunately, we found out only later that *cv2* from *opencv* was not supported, although it was said to be supported in the challenge description, so we tried the *scikit* version of the filters (Hessian filter) and we obtained a considerably worse result (leaf veins are okay but leaf borders do not correctly represent the original image).



Despite our efforts the models generated using this approach did not improve our results (we could not score more than 50% on the test set), so we moved on with the plan of first improving results on models without preprocessing, and only in a second phase try to exploit adaptive thresholding for improving results.

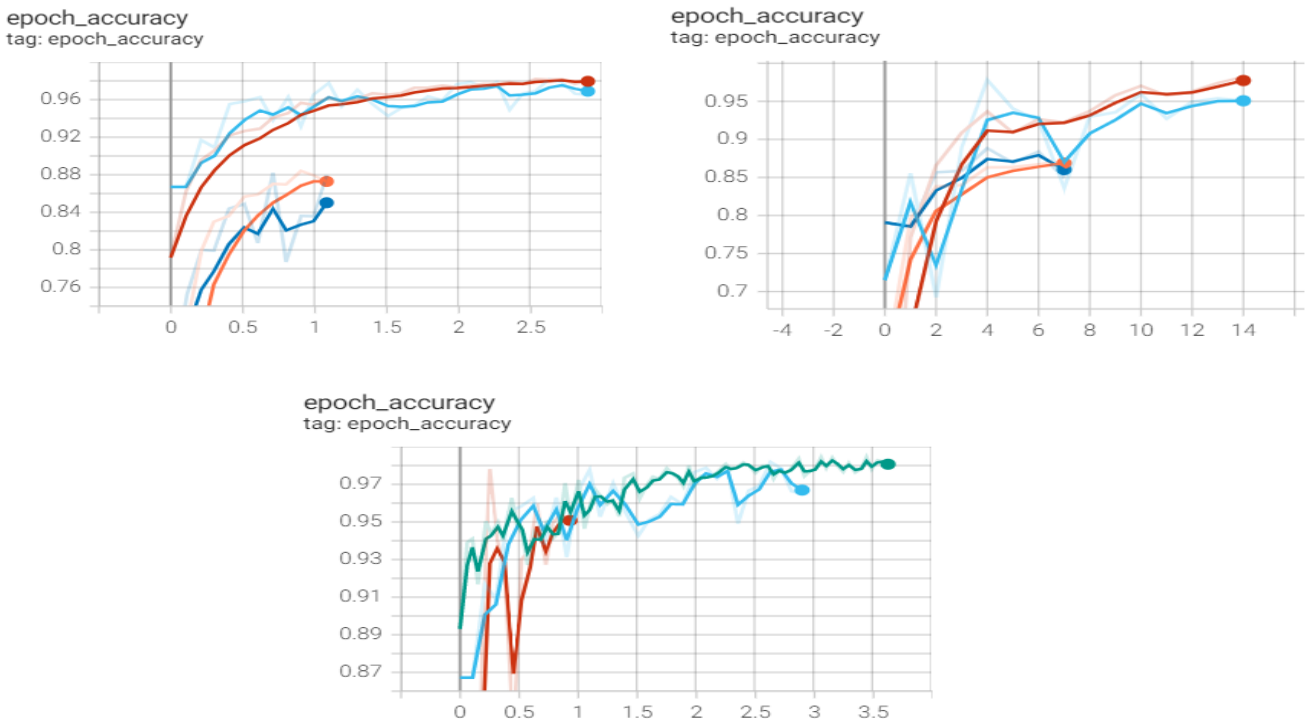
[1] Plant Identification Using New Architecture Convolutional Neural Networks Combine with Replacing The Red of Color Channel Image by Vein Morphology Leaf

4. Fine Tuning

In the next phase we decided it was about time to implement Fine Tuning on the models mentioned before, and also EfficientNet (we compared versions B0, B1, B2, B3 and B4, but ended up using mostly B1 for the similarity of the input shape).

We also added class weights in order to achieve better performance on the least represented classes in the provided dataset, ever since some class had just a few samples.

We found a rule of thumb stating that the classification network added on top of the supernet used for transfer learning should be Dense and each layer should have as units the number of its preceding layer divided by 4, until reaching the dimension of our output. Around this phase we performed between 85% and 90%, finally reaching satisfactory results (more specifically we obtained 89% with VGG and with EfficientNetB1 and 90% with InceptionV3)



Graphs showing InceptionV3 (left) and VGG16 (right) epoch accuracy for training and validation before and after fine tuning (10% increase on both models). Y axis represents accuracy values, X axis represents hours of training.

Below a comparison of the best three models (validation only) for InceptionV3 (blue), VGG16 (red) and EfficientNetB1 (green)

5. Hyperparameters Tuning

During this final phase we got stuck in results very similar but never better than the ones obtained with simpler Fine Tuning.

We also found out the tool *Keras Tuner*, that is able to compare many models after a few epochs and evaluating different hyperparameters values taken from a provided range or list.

With these models we were able to achieve faster convergence in a reduced number of epochs, around 10 or 20, compared to the 50 or 60 epochs needed before. Unfortunately, the overall best score did not improve.

Extra

- We found out a peculiar situation, similar to overfit but inverse: validation score being near 95% while training score never going after the 65%. We found out on stack overflow that the problem was probably a too high dropout causing a loss of important connections in the model during training but not during validation. Reducing the value of the dropout indeed improved the results.
- We found out that BatchNormalization layers does not need to be fine-tuned, as performance would be considerably harmed.
- One of the errors in the EfficientNet model is probably that BatchNormalization's behavior is unstable when batch_size is smaller than 16.
- We experienced exploding gradient, while tuning hyperparameters on Keras Tuning.
- We also tried to implement *fully convolutional layers* for the classification part of VGG and EfficientNetB1, obtaining more or less the same results as the *Dense* counterpart.
- We implemented tf.keras.utils.image_dataset_from_directory instead of ImageDataGenerator.flow_from_directory, and because it is a bit more efficient, we were able to reduce the ETA of a single epoch from 7 minutes and a half to 2/3 minutes.

Conclusion

Our ability and knowledge about the topics addressed during lectures and exercise sessions significantly improved and getting dirty with a similar real-case scenario is a needed experience.