Image Classification Challenge

Artificial Neural Networks and Deep Learning @PoliMi

Antonio Ercolani
Francesco Romanò

A.Y. 2021/2022

1 First approach

After a brief data exploration we tried to apply the first model for image classification we saw during the exercise sessions. It was a "simple" model with 5 convolutional layers and 2 dense layers. We did a training with a train-validation split of 0.2 and early stopping, techniques that we kept for the entire project.
After the training we obtained the following results:

- Validation accuracy: 0.910
- Test accuracy: 0.255

Analysing the results we guessed that the images of the test set have to be quite different from the ones we have in the given dataset (different background, leafs positions ecc..), therefore we looked for a model able to better capture the actual features of the leafs, disregarding useless information.

2 Transfer Learning

As said in the previous chapter we were looking for a powerful feature extractor model and the answer was obviously transfer learning. We exploit the pre-trained VGG16 model for the feature extraction layers, attaching a few fully connected layers that we're going to explain in the next paragraph. After the training we get the following results:

- Validation accuracy: 0.970
- Test accuracy: 0.792

We obtained a great improve in the test accuracy and so we knew that our guesses were right, VGG16 did what we expected.

2.1 Dense top classifier and GAP layer

The rule of thumb used to implement the dense classifier at the end of the VGG16 pre-trained model was to start with a simple model and check the behaviours before increase the complexity. While performing Transfer Learning we noticed that the Flattening layer at the end of the VGG16 features extractor was creating a very complex model (around 8 millions of parameters only for the final dense classifier), so we have decided to start with a simpler model.
Global Average Pooling (GAP) layer suited well this issue since it allow us to reduce the model parameters to 135K. A simple model is less prone to overfitting and it is lighter to train, so preferable to a complex if it is sufficient to perform the task without underfitting.
An other reason to choose the GAP layer instead of the Flattening one is that GAP implicitly performs the invariance to shifts, since it performs the average over all the pixels the same image shifted will results in the same, or a very similar, global average. We think that this property can help the model to not focus on the exact position of the leaf to be classified, allowing a better generalization on images not perfectly centered.
After the GAP layer we just continue with a really simple dense classifier with just one hidden layer, a dropout layer to hold off the overfitting and a final output layer to perform the final classification with the usual softmax activation function typically used for multiclass tasks. We used the Categorical Cross-Entropy loss function, typical for multiclass classification. Dropout rate was set initially low (0.3), further on in our experiments we tried to increase it but the results performed worse so we keep on with a 0.3 rate in our models.

This simple model performed well both on our local set and on the challenge test set, as described in previous paragraph 2, and this led us to continue with the tuning of that initial setup.

2.2 Dealing with class imbalance

An immediate consideration about the dataset provided was that it clearly suffers of class imbalance. As showed by the image below the number of samples of different classes are significantly unbalanced:

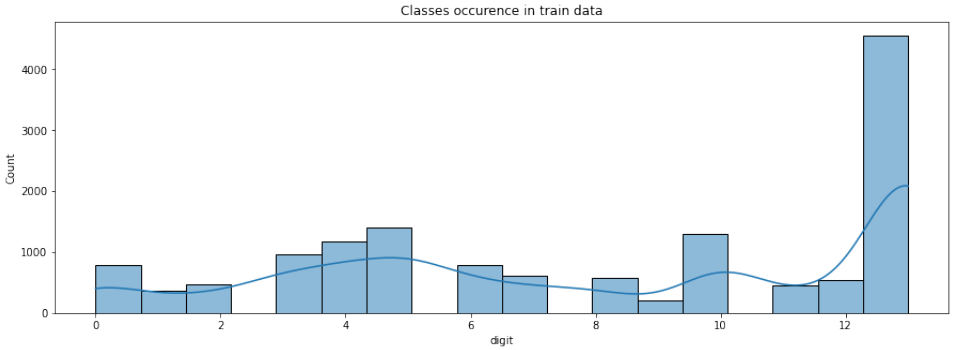


Figure 1: Unbalanced classes distribution

This can lead to bias the model in classifying more easily with labels of classes with more samples and since we could not know how the real population of data was distributed we have decided to counter imbalance applying weights over classes. This is a common strategy to deal with that problem.
To be sure of this intuition we trained the same model two times, one with weights and the other without them. We have noticed that the model with weights on classes had a better accuracy (0.75 without and 0.79 with on test set).
Interesting to notice that instead on validation set the accuracy is better without class weights (0.98 without and 0.97 with), this is due to the fact that our validation set was sampled from the original biased distribution.
Considering these results we decided to continue to use class weights, because probably our dataset was slightly biased towards certain classes.

3 Data augmentation

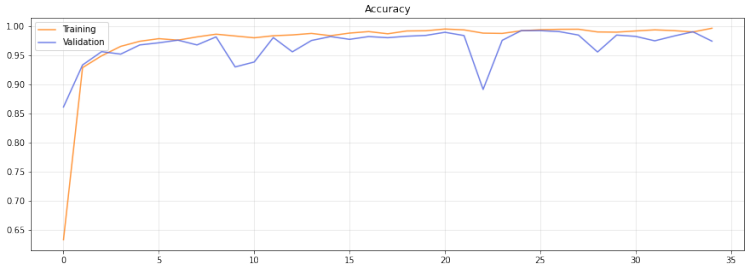
We tried to further improve our model that was already performing quite good with data augmentation. Since we guessed that the images of the test set might be quite different from the dataset ones we thought that the some image transformation could improve the model.
We kept particular attention to the `fill_mode` parameter and after some experiments we decided for a black filling that in our opinion was the one that less "compromises" the leafs.
We also took care of augment only the samples of the training set and not the validation ones.
With this technique we slightly improved the test accuracy (+5%).

4 Fine-Tuning and preprocessing

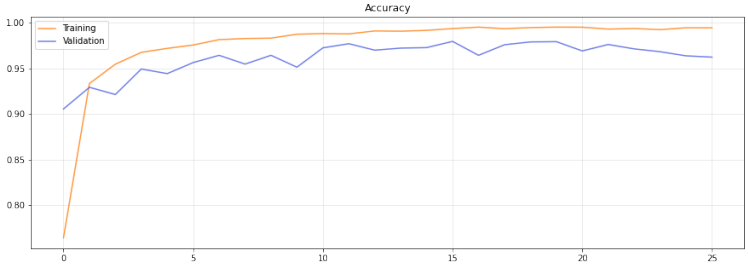
Since the considered dataset was a specific one (it contains only images of leaves) we thought that the VGG16, which is a model trained on huge general images and with much more classes, could be fine tuned for our precise task. We have kept the first layers frozen since they are supposed to be very good general purpose features extractor and we set to trainable only the last 4 layers. In this way the model is supposed to fit better the problem and extract features more relevant for the task of recognizing leaves.
At the beginning we faced a terribly irregular and non-effective training, and we noticed that was probably related to the learning rate (the keras default 0.001 used in previous models). We think that this behaviour is related with the grown complexity of the model since the 4 trainable layers added around 7 millions of parameters. Decreasing the learning rate to 1e-4 was an effective remedy to that and the training became stable and effective. After fine-tuning the model reach a test accuracy of 0.92, an increase of 7% with respect simple transfer learning.
We then try to reduce overfitting and simplify the gradient flow respectively with an increased dropout rate (from 0.3 to 0.5) and a Batch Normalization layer between the hidden layer and the activation layer (hidden - batch norm - activation - dropout). Surprisingly these changes didn't bring to any significant improvement

both on validation and test accuracy, so we removed them from our implementation.
At last we introduced a simple ad-hoc preprocessing for VGG16 which just convert the image from RGB to BGR and zero-center with respect ImageNet dataset (the data originally used to train the VGG16 model). This simple preprocessing phase increase the test accuracy to 0.9377 (more than 1%).

Below we present the training graphs of two models, the final model (a) and the one built using only transfer learning (b). We can see that both model have a high validation accuracy, also if the final is in average better. Anyhow we think that the final model performs much better on test because the fine tuning allows a better generalization (as said before we suppose test set was a little bit different w.r.t our dataset). Moreover we notice a lower divergence between training and validation accuracy in the final model, this thing is usually a good sign that we are not overfitting. In some points the final model presents "strange" jumps, we suppose that this can be due to unlucky data augmentation or to the bigger model complexity.



(a) Training graph of the final model



(b) Training graph of the model built with only transfer learning

5 Summary

Finally the reader can find a table that summarizes the progression of the most significant models that we have trained and the relatives results on the CodaLab test set (developement phase test set).

Model	Test accuracy
Simple CNN	0.255
Simple CNN + class weights	0.232
VGG16 transfer learning + class weights	0.792
VGG16 only transfer learning	0.755
VGG16 transfer learning + data aug + class weights	0.851
VGG16 transfer learning + data aug + higher dropout + class weights	0.801
VGG16 fine tuning + data aug + class weights	0.926
VGG16 fine tuning + data aug + preprocessing + class weights + batch norm	0.907
VGG16 fine tuning + data aug + VGG preprocessing + class weights	0.938

p.s. when we submitted the models in the competition final phase we have noticed that the model without the VGG16 preprocessing performed around 1% better despite previous results on development phase due to the different test set used for evaluation.