Bono Federico
Cecere Nicola
Zanutto Davide

Artificial Neural Networks and Deep Learning

Image classification challenge

The first challenge of this year was a classification problem. The goal was to classify various pictures of plants and to determine which class they belonged to. Our team made the decision to start from scratch while creating a Convolutional Neural Network with this goal in mind. After that we made the decision to change our strategy and to adopt a Transfer Learning approach. All our work was done using whether Google Colab or other IDEs such as DataSpell.

Pre-processing Data

Starting from the common parts, we used a simple Python command to automatically divide the dataset into two different ways :

- with the test part : with a distribution between train, validation and test of respectively 80-10-10, this split was used earlier in the process to train and test locally the networks to have a general idea on the quality of the models;
- without the test part : with a distribution between train and val of respectively 80-20, this was used to have a better train phase for the latest versions of the models.

Analysing the dataset it emergerd that the classes were unbalanced, in fact some of them had more images than the other ones. To face this problem we adopted three approaches. The starting method was to duplicate file elements of specie 1 because it was the smallest and with the worst accuracy. The duplication methods introduced some overfitting and results with the custom CNN were slightly better. However, this first approach worked very badly with transfer learning, thus we decided to remove the duplication from the dataset. The second method was similar to the previous one but, instead of simply duplicating the pictures, we decided to flip them. However, these modifications didn't bring an improvement of the accuracy, so we decided to use the simplest dataset: the one we started with. The third method was done with class balancing assigning a weight to each class in order to give less relevance to the classes with a higher number of samples with respect to the ones with a smaller number. Although, we noticed that this last method worked well only with the CNN custom network and not with the transfer learning approach.

Another common part is the data augmentation, useful to generate variation in the data and improve the model's stability. We took inspiration from the lab notebook techniques but changing the single parameters multiple times to know which were the best and also added some other techniques like brightness since we noticed it improved a lot the accuracy and deleted others, like stretch and shear, because, observing the images in the classes, we noticed that this type of modification could modify the features of the plants and thus modify their belonged class.

We used also a learning rate schedule approach in order to have a more efficient training by decreasing the learning rate every few epochs.

Eventually we used a batch normalization technique only on the custom CNN because with the transfer learning approach the results were worse.

Custom CNN

We started by using the model from lab #5 as a base for our custom CNN. This first attempt brings us around 0.3 accuracy on codalab.

At one point, we've also tried to build the network from the ground up, starting with fewer layers (just 3) and fewer parameters (500k vs 2M) and we obtained an impressive (considering the smaller size of the network) 0.6 for accuracy. Unfortunately, while trying to optimize this custom network we found out that simply adding layers or increasing the size of the filters was having a huge (negative) impact on the accuracy, given the small sample size for the input, our network tends to overfit quite quickly.

So, we went back to our first design and tried to optimize more.

We used keras-tuner to tune filter sizes, dropout layers, activation functions, and even the number of hidden layers in the fully connected network. Some results came out, but nothing compared with what we gained by using transfer learning (as expected). Another optimization we tried was adding class weights to overcome the input bias of classes 1 and 5, we used class weight helpers from sklearn but obtained better results with custom weights. We think the reason is behind the conformation of the input images: by simply looking at some of them we observed that some images were less characterizing that others (e.g., brightness, focus, zoom).

To push our network further, we've introduced a learning rate schedule to increase the training efficiency in later epochs, used batch normalization in between the CNN and increased both filters and dense layers units up to 512 filters on the last convolution layer and 600 units in the dense layer.

Our final design for the custom CNN reached 0.7762 on codalab.

Transfer learning

After some time spent all together improving the custom CNN we decided to split: one person would have continued to improve the custom CNN, while the other two would have started thinking of a Transfer Learning approach. As done with the previous model, the very first approach was to check the notebook did during classes and try to add some modifications. We were not quite satisfied with the results, so we started experimenting by our own. We first tried the VGG16 net which gave us decent results, without having to modify the top of the network. After that, we tried Xception: here we performed many experiments by changing the network on top and by trying to set to true many different amounts of the network layers. We decided to change with a different activation function passing from Relu to Leaky-Relu and we had some improvements. At this point we had reached an accuracy of 0.71. To better improve our score, we imported a preprocess function from TensorFlow to be applied on our training, validation and test data. Obviously, we did all the necessary modifications to our code and we inserted the preprocessing of the data also in the "model.py" file. This further improved our accuracy score, jumping to 0.76.

We used fine tuning on every pre-train network we evaluated, and we attempted different combinations of turning the layers that make up the pre-train network on and off.

We were not yet satisfied with the results because we know we could improve our score, so we continued to test new networks to see the differences. We tried EfficentNet and after some tuning on parameters and modification on top network we reached the score of 0.8402 in the final phase, the highest accuracy we managed to reach. After that we also tried DenseNet with Keras tuning but the results didn't improve.

Final Submission

The final model we submitted was done with a Transfer Learning approach: it implemented EfficentNet with Fine Tuning (freezing the first 190 layers) and Adam as an optimizer with dynamic learning rate. The network on top was built with the following layers:

- Applying a flatten layer with a dropout of 0.2;
- Applying a dense layer with a dropout of 0.2 and a size of 128;
- Applying a dense layer with a size of 8.