Image Classification Challenge 2022

1. Introduction

The scope of the project was to design an Artificial Neural Network able to classify plants belonging to eight different species. We were given a dataset of images of these plants labeled with the class they belong to; our goal was to design the network either from scratch or using known pre-trained networks and applying Transfer Learning and/or Fine Tuning.

2. Dataset description

The dataset was composed of 3542 colored images of plants with dimension 96x96 divided into eight folders, one for each species.

2.1. Class balancing

The dataset presented an imbalance of the 8 plant species to classify. In particular the occurrences of species 1 and 6 were very low compared to the others. In other words, there was a bias towards the majority classes present in the target. Our first attempt to balance the dataset was to give different weights to the 8 classes of the training set in order to influence the classification during the training phase. The whole purpose of this method is to penalize the misclassification made by the minority classes by setting the class weights inversely proportional to their respective frequencies. Nevertheless, having obtained weak improvements in our model performance, we decided to perform an upsample by adding random duplicate images to each class until they reach the number of samples of the most frequent class of the dataset, the species 7. Now the dataset is balanced.

2.2. Data split

The split used for the dataset during the training phase was 80%, 10%, 10% for training, validation and test set respectively. This split has been useful also for the hyperparameter tuning of the model and for having a “local“ estimate of our performance in a small test set. After having trained the parameters of our model, we decided to modify the split in training and validation only, with 95% 5% respectively, in order to have more training samples and a more robust accuracy.

2.3. Data pre-processing

Regularization helps us control our model capacity, ensuring that our models are better at making classifications on data points that they were not trained on, which we call the ability to generalize. We have tried 3 types of data regularization techniques:

PATCH SHUFFLE: Divide the image into patches of equal size and shuffle the pixels within each patch creating a sort of blur effect that helps combat overfitting in training.
We have not found satisfactory results with this method probably due to the small size of the images.
CUTMIX: CutMix removes regions of image and replaces the removed regions with a patch from

another image. The added patches further enhance localization ability by requiring the model to identify the object from a partial view.
With this method we have had significant increases in accuracy in the final classification.
DATA AUGMENTATION: Data augmentation is a technique to increase the diversity of our training set by applying random (but realistic) transformations in order to increase the generalization ability of our model.
We tried different types of transformations and we kept those that did not deteriorate the images too much but that allowed us to obtain images significantly different from the original.

3. Model development

3.1. Custom Network

Initially we started building a plain CNN network composed of 6 convolutional layers with 32, 64, 128, 256, 512 and 1024 filters respectively. After the latter, we decided to use a GAP layer instead of a Flatten to reduce the number of parameters of the model and overfitting.

Given this result we decided to use a more advanced architecture, ResNet, to obtain better performance. We built the network with three residual blocks with 32,64,128 filters respectively with dimension 3x3. Finally we used a GAP layer, a Dense layer with 128 neurons and the output layer. With this network we obtained better results with respect to the plain CNN but not enough to use it as our final model.

3.2. Pre-trained networks (Transfer Learning)

We tried many pretrained models to see how they behaved with our dataset, we started from the ones with the best accuracy trained on imagenet. We ended up choosing between EfficientNet and ConvNeXt, in the end the latter performed better in the version ConvNeXtLarge, we then proceeded developing the on top network.

We considered both Flatten and Global Average Pooling as the first layer of our on top network: first we tried to use Flatten layer to create a one dimensional tensor from the last layer of the convolutional network, then we decided to use GAP layer to reduce the number of parameter of the network and to reduce overfitting; the advantages of the GAP were meaningful with a significant reduction in the number of parameters so it became our choice.

For the next part of the network we used some Dense layers with ReLu activation function; we tried different combinations of number of layers and number of neurons per layer using cross validation; we noticed that a large number of layers and neurons increased too much the number of parameters and the complexity of the network.

For this reason we followed the suggestion of the professors; we mixed the feature extractor part of the network with a more robust model, a Quasi SVM. We built the Quasi SVM using a Random Fourier Features Layer where, again, the hyperparameters were chosen using cross validation. This implementation gave us better results in terms of complexity and performance of the network.

The last layer instead has always been a Dense layer with the Softmax activation function and number of neurons equal to the number of classes.

4. Model training

4.1. Fine Tuning

For the training phase of the model we decided to use a Transfer Learning + Fine Tuning approach: we did a first training of the network freezing all the layers of the feature extractor part (preserving the information embedded in it) so that the network on top could learn to classify the images on those features (Transfer Learning). At this point we did a second training, freezing only a portion of the first part of the network and using a lower learning rate, to adapt the pre-trained features to our training data (Fine Tuning).

This approach, given the limited size of our training data not sufficient to train an entire complex network, gave us significant improvements in the accuracy of the network, while making the entire process of training much longer.

4.2. Ensemble

The last part of the training was doing a model ensemble with three pre-trained models with different batch sizes, i.e. 8, 16 and 32. We created an ensemble layer in which we performed a weighted average of the predictions, we used the same weight for all the models because their performances were similar.
We obtained in this way an improvement in the overall accuracy on the test set.

5. Hyperparameter tuning

In this chapter we discuss the choices made for hyperparameter tuning using the framework KerasTuner to execute the hyperparameter search. We must emphasize that our attempts did not bring the desired results, given that the hyperparameter search was an expensive operation and our machines couldn't handle it. Nevertheless, the following describes the phases to perform it. The architectural choices have been:

- the pre-trained networks to use. in particular searching among EfficientNetB0, EfficientNetB7, ConvNeXtBase and ConvNeXtLarge;
- the learning rate used by the Adam optimization method using a range of values going from 1e-5 and 1e-2.

For resource reasons we chose the RandomSearch tuning algorithm, which provides a statistical distribution for each hyperparameter from which values are sampled. As mentioned above, our trials didn't give us meaningful results and our choices have been guided by training individually the models with different pre-trained networks and using the default learning provided by Adam optimizer class of Keras, 1e-3.

6. Conclusions

We were able to apply what we studied and achieved a final accuracy on the test set of 93.03%. Possible improvements could be made with ensembles of structural different networks, with a more in-depth hyperparameter tuning and applying Test Time Augmentation for prediction.