Artificial Neural Networks and Deep Learning - First Challenge 2022

Image Classification

Andrea Chizzola, Raffaele Cicellini, Carlo Cominoli
Politecnico di Milano

November 25, 2022

1 Introduction

Image classification is the task of predicting which class to assign to an input image, given a fixed set of classes. In this challenge, the purpose was to classify plants images and predict their species using Convolutional Neural Networks. To do so, we experimented with different techniques for training the networks and with different models.

2 Data analysis

The dataset provided for training was very small, containing 3542 images divided in 8 directories, one for each species. The images were 96x96 pixels jpgs. As shown in Figure 1, the distribution of the images belonging to the different species was not uniform, resulting in an unbalanced dataset particularly for species 1 and 6.

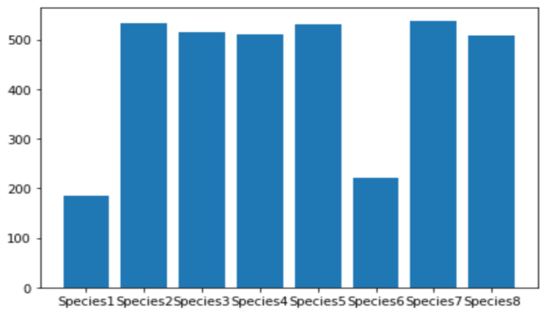

Figure 1: classes distribution

3 Data preparation

3.1 Data splitting

We decided to split the dataset into two partitions: training and validation. To do so, at the beginning we used a ratio of 80/20, but, given the limited number of samples available for training, we moved to a ratio of 90/10.

3.2 Data augmentation

Given the small dimension of the dataset, we could immediately see how overfitting was affecting the training process. So, we decided to perform data augmentation on the training set in order to have more samples. This was achieved using the Keras module ImageDataGenerator(), setting the following parameters:

- rotation_range=45
- height_shift_range=50
- width_shift_range=50
- zoom_range=0.3
- horizontal_flip=True
- fill_mode='reflect'

Moreover, we also rescaled the images to 256x256 in order to facilitate the learning process of the models, since the original dimensions were too small. Bilinear interpolation was used to perform the rescaling.

4 Training

We performed the training procedure measuring the accuracy and the categorical cross-entropy loss both for the training and the validation set. The maximum number of epochs was set to 200, with early stopping (patience equal to 10) and different batch sizes: the best model was trained with a batch size equal to 32. The metric we used to select the best model was the accuracy measured on the validation set, as it approximates the prediction error on unseen samples.

5 Models and techniques

In order to find the best model, we tried two main approaches. At first, we implemented some custom Convolutional Neural Networks trained on the provided dataset. Then, we switched to transfer learning using some known high-performance models (pretrained on 'imagenet'), focusing on different versions of the ones that were presented during the theoretical lectures.

5.1 Custom networks

We started from very simple models, implementing Convolutional Neural Networks with different numbers of convolutional layers (3x3 filters and 'relu' activation function), max pooling layers (2x2) and dense layers (with activation function 'relu' or 'softmax' for the output layer). To help reducing overfitting we then applied different regularization techniques by interleaving all the layers, including the convolutional ones, with batch normalization layers and adding also dropout layers among the fully connected ones. The performance results were however not satisfying (ca. 63% local validation accuracy) and the impact of regularization was not much effective in preventing the training loss to come close to zero. Despite a great margin for further improvements was certainly available, due to limited time and computational power we decided to move on and try different models for transfer learning.

5.2 Transfer learning

The approach we followed for transfer learning was to try different pre-trained models, each with the same fully connected layers added at the end, choose the best one and then apply fine tuning in order to maximize performance.

Originally, the common classifier top included a Flatten layer, followed by two dense layers interleaved by batch normalization and dropout layers, similarly to what we used for the custom models.

At first, we trained the models without class weights, obtaining poor results because of the class imbalance in the dataset. After using class weights, a noticeable performance improvement was reached using ResNet and VGG: after submission on Codalab, ResNet obtained a 70.58% in accuracy, whereas VGG obtained a 60%. Further developments with VGG led us to substantially improve the local validation accuracy, reaching a score of 81.89%. This was achieved by modifying the fully connected layers to 3 dense layers containing 512/256/8 neurons respectively, together with batch normalization and dropout layers among them. Fine tuning was also applied to the last 5 layers of the network, by making them trainable. The training loss obtained during the training procedure was however very close to 0 and, as expected, the accuracy obtained after the submission on Codalab was considerably lower, 73.63%, showing a strong inclination towards overfitting. We then tried ResNet101v2 with custom top layers consisting of a convolution with 256 filters, a max pooling, a global averaging pooling and finally a dense layer with 8 neurons. We trained firstly the network with all the ResNet layers set as non-trainable, using a learning rate of 1e-3 with Adam as optimizer. Then we trained also the last layers of ResNet (from the 338th layer till the last one) and with a smaller learning rate of 1e-4. The idea behind the smaller learning rate is that those trainable layers of ResNet have already been trained, so we want just a little change in order to adapt better the pretrained network to our task. On the contrary the custom layers at the top need to be trained from scratch (since they have random weights) so we use a bigger learning rate. This network obtained an accuracy of 79% on the unseen dataset.

Given the obtained results and the small dimension of the training dataset, we could see how focusing our research on models with a smaller number of parameters could have led to better results, by simplifying the training procedure and reducing overfitting as much as possible. Thus, we selected two different versions of EfficientNet. EfficientNetV2B0 obtained 76.04% local validation accuracy and 72.54% accuracy on Codalab. EfficientNetV2B3 was also tried, given its better performance on the 'imagenet' dataset if compared to the B0 version. The best local result was 77.72% validation accuracy. However, once submitted on Codalab, we obtained 69.9%. Because of this result, we switched back to EfficientNetV2B0.

5.3 Best model

We started fine tuning the model using EfficientNetV2B0 as base model and changing the network top layers by using only dense layers. Changing the dropout to 0.4 we were able to obtain 80.22% local validation accuracy and 78.65% accuracy on Codalab. Other values were also selected, all leading to worse results. By changing the dimensions of the dense layers to 128/64/8 and tuning the dropout, we achieved local validation accuracy in the range between 81.34% and 83.01%, with the best result on Codalab equal to 79.9%.

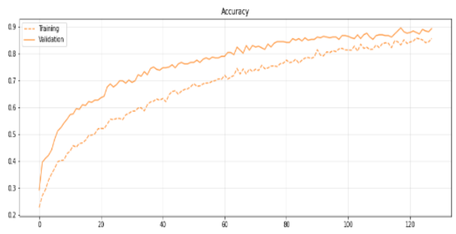

Figure 2: accuracy of the best model

In the end, we opted for three dense layers with 64/32/8 number of units respectively, batch normalization, dropout equal to 0.35, ‘relu’ activation function for the first two layers and ‘softmax’ for the last one and using a learning rate of 1e-5 with Adam as optimizer. The best result was obtained by first freezing the base model (EfficientNet) and train only the network top, and then retrain the whole network. In this way we were able to obtain 84.12% local validation accuracy and 85.4% on Codalab (83.92% in the final phase submission). Analyzing the results, we saw how class weights helped increasing the accuracy for the classes with fewer samples, but it also limited the possible improvements for the other species’ accuracy. So, we tried to train this last model without using class weights, obtaining however slightly worse results: 81.89% local validation accuracy, 81.51% on Codalab. In the end, the improvement that class weights gave us was greater than the limit in the accuracy for classes with more samples.

6 Results

In the following section we report the local and remote accuracy of the best models we used for the image classification task:

Model	**Local accuracy**	**Codalab accuracy**
VGG16	81.89%	73.63%
ResNet	79.0%	77.58%
EfficientNetV2B3	77.72%	69.9%
EfficientNetV2B0 (first version)	76.04%	72.54%
EfficientNetV2B0 (best intermediate version)	80.22%	78.65%
EfficientNetV2B0 (last version)	84.12%	85.4%

7 Conclusions

From the experiments performed for this challenge, we noticed that, due to the small dimension of the dataset, transfer learning models with a limited number of parameters obtained the best performance in terms of accuracy. Probably further tests on fully convolutional networks with an even smaller number of parameters compared to EfficientNetV2B0, could have led us to better results. However, we believe that the performances we obtained with EfficientNetV2B0 were still good.

Tools

- Tensorflow (also gpu version)
- Keras
- Scikit-Learn
- Jetbrains Pycharm
- Google Colab