Plant Classification

Alberto Boffi, Francesco Bleggi

ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING

POLITECNICO DI MILANO - NOV. 2022

1 Introduction

Automatic plant classifiers are some of the most useful classification tools, as recognizing plants is for humans a daunting and error-prone task. The goal of this project is to build a performing plant classifier model through Deep Learning Neural Networks, exploiting the functionalities of Keras and TensorFlow.

2 Data

2.1 Description

The available training data includes 3542 RGB images of 96×96 pixels, belonging to 8 different plant species. The dataset is characterize by an imbalance due to the limited availability of images for the first and sixth species compared to all the others.
We split the training data into training and validation sets, on which we applied **hold out** cross-validation to better evaluate the performances of the model. We performed various experiments varying the fraction of data for the training set from 80% to 90%. In the end, we noticed that a split of 85/15 guaranteed the best results, also with regard to the early stopping procedure (discussed later).

Figure 1: Example of Training Images

2.2 Data Augmentation

In order to increase the amount of training data and reduce the risk of overfitting, we pre-processed the data by performing data augmentation. We observed by image inspection that transformations as zoom, rotation or shear were not suitable for the task as they changed the input label. We were instead able to increase the difference between the different classes through flipping (both horizontally and vertically) and 16-bits shifting (both wrt the width and the height).
Data augmentation also helped compensate for the dataset imbalance.

3 Model

3.1 Approach

As a first attempt, we implemented from scratch a Neural Network composed of both convolutional and

dense layers. However, the results on the testing set did not exceed an accuracy of approximately 50%. For this reason, we took an approach based on Transfer Learning, that led to a dramatic growth in the performances. We just noticed some problems in the classification of the plants belonging to the first species, some of which were wrongly classified as species no. 8. By also employing Fine Tuning, the problem was quite satisfactorily solved.
For the Artificial Neural Network, our choice went to **ResNet** over **VGG-16** as the former brought a faster training and better results.

3.2 Convolutional Layers

In the initial design of the Convolutional Neural Network, our main concern was the receptive field. Since the NN had to classify based on the features of the entire image, instead of particular objects inside it, we started by fixing a receptive field of 96x96 pixels using 3 convolutional layers. Despite this, we actually observed that improvements in the results were always associated to an increase in the size of the receptive field.
To exploit the Artificial Neural Network as much as possible and stay within acceptable training times, in the final model we trained up to the 81st of the 174 convolutional layers of the ResNet.

3.3 Dense Layers

In the Feed-Forward Neural Network, the best behavior was reached with two hidden layers. This was an expected result since, in classification tasks, 2 is the minimum number of hidden layers needed to obtain good performances in approximating any nonlinear function. For both layers, we stuck to the classical choice of using **ReLu** as activation function. We also tried **Leaky ReLU** but it didn't cause any improvement. As usual for the multi-classes classification domain, we resorted to the **Softmax** activation function for the output layer, and to the **Categorical Cross-Entropy** as loss function. To minimize the loss function, we employed the **Adam** optimization algorithm over a batch size of 32 samples. We started with a learning rate of 0.0001 and then we halved it, causing an exponential growth in the number of epochs, but also a 5% increase in the accuracy.

3.4 Prevent Overfitting

The first method that we found really helpful to prevent overfitting is **Early Stopping**: by keeping the error on the validation set under control, we stopped the training when we didn't see any improvement after a certain number of subsequent epochs (or patience factor). We agreed that a patience factor of 25 was the right compromise between the amount of computation and the quality of overfitting detection. Moreover, the training/validation split ratio was the best trade off between a good assessment of the model and the loss of data for the training process.
We also applied **Dropout** with a rate factor of 0.3. The decision to adopt this technique was mainly due to its reputation, and our expectations about its usefulness were not disappointed.

4 Results

The effectiveness of the model on the training data is described by the following confusions matrices (in absolute values). The first matrix refers to the model developed with Transfer Learning. We can clearly notice the previously mentioned problem about the classification of the first species. The second matrix shows the difference due to Fine Tuning.

S1	2	7	1	2	2	1	2	11
S2	0	70	0	0	0	2	4	4
S3	0	0	70	4	3	0	1	0
S4	0	3	3	64	3	0	0	4
S5	0	0	6	15	57	0	1	1
S6	0	9	0	0	0	24	0	1
S7	0	9	2	0	1	1	68	0
S8	2	5	3	6	1	0	4	56
	S1	S2	S3	S4	S5	S6	S7	S8

Table 1: Confusion Matrix - TL

S1	15	3	0	3	1	0	1	5
S2	0	70	1	0	0	2	1	6
S3	0	0	73	1	4	0	0	0
S4	1	0	3	67	4	0	0	2
S5	0	0	6	0	73	0	0	1
S6	0	3	0	0	0	31	0	0
S7	0	1	0	0	0	1	79	0
S8	3	5	3	1	1	0	1	63
	S1	S2	S3	S4	S5	S6	S7	S8

Table 2: Confusion Matrix - TL+FT

During the long training process of the final model, we've been disconnected from Google Colab twice. However, by using the checkpoints offered by Tensorflow, we were able to freeze the value of each parameter. The representations of the accuracy in every epoch, produced by TensorBoard each of the three times, is represented below:

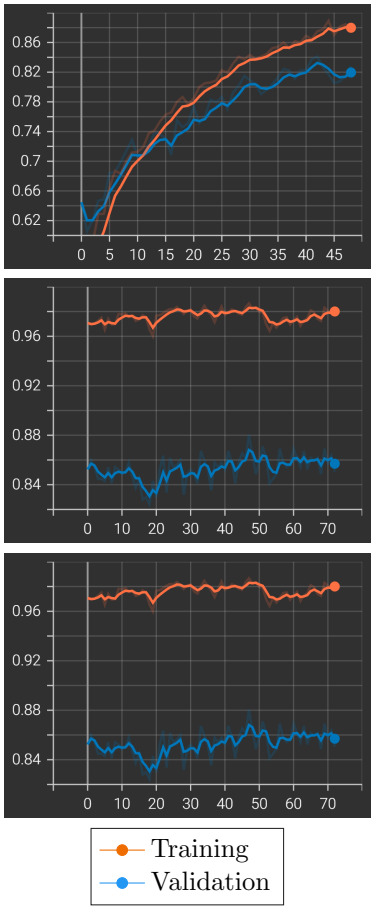

Figure 2: Accuracy per Epoch