Homework 1 Report - Artificial Neural Networks and Deep Learning

Alberto Rota
Person Code: 10615751
Student Number: 964662
alberto2.rota@mail.polimi.it

Gabriele Santicchi
Person Code: 10579046
Student Number: 969088
gabriele.santicchi@mail.polimi.it

Giuseppe Venezia
Person Code: 10622477
Student Number: 968395
giuseppe.venezia@mail.polimi.it

I. Context

The goal of this competition is to build a classification model that properly predicts the plant species based on pictures of leaves. The available dataset is composed of 17728 *256x256* RGB images grouped into 14 classes with a different number of instances for each class.

II. Preparatory Tasks

A. Data Loading and Augmentation

Either the loading and the augmentation of the data have been performed by the means of the *ImageDataGenerator TensorFlow* class, which allows to both allocate single batches of images on RAM and augment them according to the parameters provided to the class methods. The transformations included in the augmentation phase are rotations (from -90° to +90°), shears (from -20° to +20°), shifts (from -50 to +50 pixels), flips (both left-right and up-down) and zooms (30% closeups). High shift values (nearly 20% of the image size) combined with a reflection filling allows the addition of leaf-like artifacts near the image border, which has been observed improving the generalization capabilities of the neural network.

B. Train-Validation Split

The number of samples in each class is not uniform, therefore a stratified sampling strategy is applied: this way the proportions between class samples are equal in the training (80% of data) and validation (20% of data) datasets. Since the *ImageDataGenerator* doesn't allow for such stratification, folders have been splitted on disk into balanced training and validation subdirectories.

C. Preprocessing

Since the background of the images in the dataset is uniformly black, gaussian noise was added only to the pixels discriminated in terms of low brightness; this step is performed in order to improve the model robustness, as the images to be classified may not have a black background. This can be seen in Figure 1 Nevertheless, the model performance with the addition of noise did not lead to improved accuracy: this strategy has been, therefore, ditched. In the cases where a transfer learning approach was implemented, the corresponding preprocessing function was employed, otherwise a simple normalization was used for handmade architectures. All of these transformations are passed to the *ImageDataGenerator* instances, which applies them after augmentation.

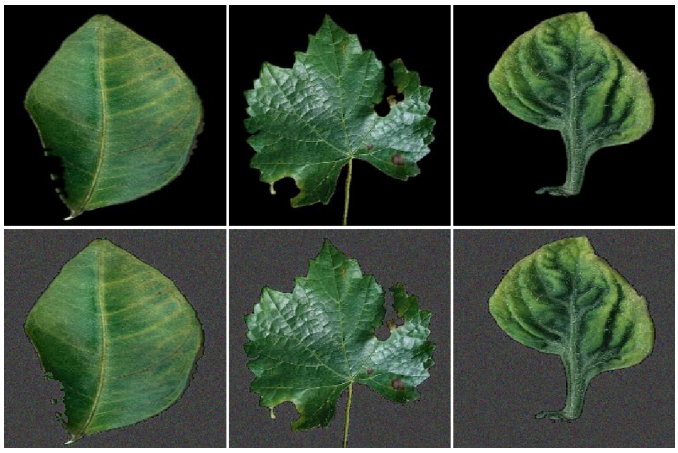

Fig. 1. Addition of Gaussian noise to the images. The noise is added only to the background

III. The Model

A. Feature extractor

The first attempts at building the data-driven feature extractor were focussed on small CNN architectures: here, the kernel size and the number of filters have been properly selected in order to highlight the leaves' features at different scales. More specifically, early layers were assigned small kernels to grasp higher-resolution details, while deeper ones had an increased spatial extent as well as a larger receptive field.

The first experiments with shallow CNNs were badly performing, while more complex and deeper architectures presented a huge number of trainable parameters, not suitable for the available computational power: transfer learning is adopted in order to exploit the feature extraction capability of a supernet while keeping the number of operations low. Here, several models have been tested, such as VGG16, ResNetV2, InceptionV3 and Xception. The latter one, in combination with the classifier of our model, achieved better results in terms of validation accuracy and it has been consequently implemented as feature extractor in our final neural network. Since the

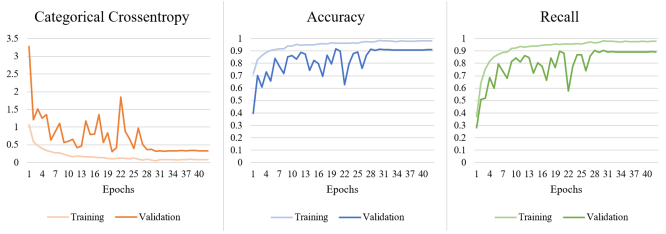

Fig. 2. Loss, accuracy and recall for the training of the best performing model

Xception architecture has roughtly as many parameters as the other mentioned supernets, the performance gains are not due to an increased capacity but rather to a more efficient use of the model parameters.

B. Classifier

Starting from from the latent representation, that can be seen also as a vector of the extracted feature maps, the classifier consists of just one output layer of 14 neurons, the same as the number of classes, with a softmax activation function which takes as input the output of the GAP and returns the posterior probability for each class. Then the prediction for each sample will be the argmax of the output vector, hence the class with the highest score. The tensor of features extracted by Xception is of fixed size (8x8x2048), and it is adapted to the GAP input shape (8x8x14) by two convolutional layers with 18 and 14 filters, respectively. Different combinations of convolutional and fully-connected layers have been tested, but the 18-14 scheme performed best among all of them. Furthermore, suboptimal results have been obtained when adding batch-normalization along with the convolutional layers.

IV. Training

The training process minimizes a categorical cross-entropy loss function by the means of an *Adam* optimizer. In order to compensate for the unbalance in the label distribution a weigth w_k has been assigned to each class, inversely proportional to its relative number of samples

$$w_k = \frac{N_{samples,total}}{N_{classes} \cdot N_{samples,k}} \tag{1}$$

Such weigths are used by the loss function to compensate for the lack of data in specific classes. Since using weigthed labels changes the range of the loss, the optimizer employed in the minimization problems must be unaffected by scaling changes: *Adam* is the optimal algorithm for such cause. The training phase is set to progress for 75 epochs, with an initial learning-rate of $1 \cdot 10^{-3}$; to deal with memory limitations of the hosting VM, the batch size is relatively low at a value of 32. Once the classes are balanced in terms of weights, accuracy represents a suitable metric for evaluation. Graphs displaying the trend of categorical crossentropy, accuracy and recall are reported in Figure 2

A. Callbacks

Multiple routines run at the end of each epoch, in the form of *TensorFlow* callbacks:

- *EarlyStopping* to terminate the training when the valdation accuracy doesn't increase for 10 epochs in a row, in order to prevent overfitting
- *DecreaseLROnPlateau*: A static learning rate scheduler had been implemented initially, but was then substituted by a dynamic "adapt-on-plateau" function, which decimates the learning rate when validation accuracy doesn't increase for 10 epochs in a row
- *ModelCheckpoint* and *TensorBoard* as assistive tools

B. Fine-Tuning

Fine tuning has been performed to increase the overall performance of our final model. Dealing with more than 22 milions parameters in the Xception network, only the 5 final layers (which consist of circa 3 million parameters) has been tuned using the training dataset. After only 15 epochs, using a lower learning rate ($1 \cdot 10^{-5}$) the accuracy increased of 3%.

C. Refining

The last step of model training is a "refinement", in which the network is fitted also on the validation dataset. In this way, all the 17K samples at disposal have been used to get the best model performance.

V. Results

The proposed model is the result of all the considerations stated above:

- *Preprocessing*: Input images are transformed by the preprocessing function provided by the transfer-learning model.
- *Model*: The chosen feature extractor is the Xception model, extended with two convolutional layers of 18 and 14 filters, in order to fit the size of GAP to the number of classes.
- *Training*: The fine-tuning and refining procedures have been applied to the proposed model in order to increase the overall performance.

As a result, since training and validation accuracy are comparable, overfitting isn't an issue that concerns the model and generalization in preserved.