Politecnico di Milano

ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING PROJECT

IMAGE CLASSIFICATION PROBLEM

Authors:
Alessandro Polidori
Agnese Straccia
Andrea Alberto Bertinotti

A.A. 2021/2022

Index

1. Introduction

This homework consisted of an image classification problem. We were provided with a dataset composed of 17728 RGB leaves images of size 256x256. The images were grouped into 14 classes, corresponding to different types of plant species. Being a classification problem, given an image, the goal was to predict the correct class label.

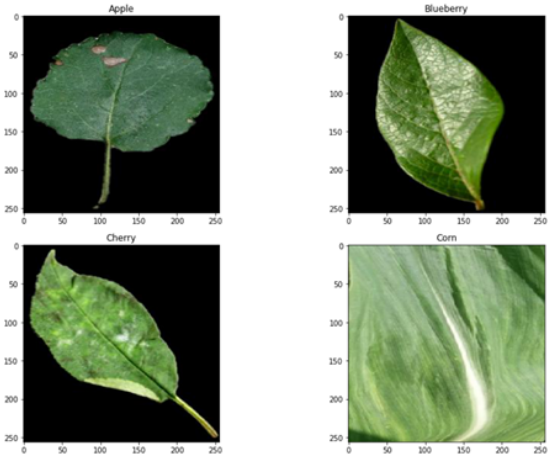

Figure 1: Sample Images

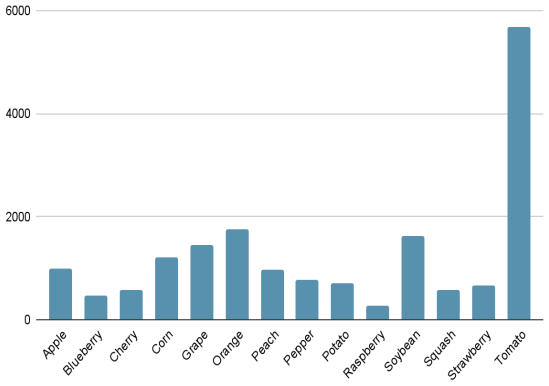

Figure 2: Class Distribution

2. First Model

We decided to start testing a very simple convolutional neural net architecture, inspired by the practice classes. In order to deal with data scarcity, we added data augmentation.

During this first step, we always used the *ReLu* activation function, *CategoricalCrossEntropy* function, *Adam* optimizer with default learning rate (0.001), batch size equal to 128 and validation split of 20%.

To regularize the model we used *Early Stopping*, monitoring validation accuracy (and we kept using that throughout all the homework).

The summary of this first model is the one shown in the figure.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 256, 256, 3)]     0

 conv2d_10 (Conv2D)          (None, 256, 256, 16)      448

 max_pooling2d_10 (MaxPoolin  (None, 128, 128, 16)     0
 g2D)

 conv2d_11 (Conv2D)          (None, 128, 128, 32)      4640

 max_pooling2d_11 (MaxPoolin  (None, 64, 64, 32)       0
 g2D)

 conv2d_12 (Conv2D)          (None, 64, 64, 64)        18496

 max_pooling2d_12 (MaxPoolin  (None, 32, 32, 64)       0
 g2D)

 conv2d_13 (Conv2D)          (None, 32, 32, 128)       73856

 max_pooling2d_13 (MaxPoolin  (None, 16, 16, 128)      0
 g2D)

 conv2d_14 (Conv2D)          (None, 16, 16, 256)       295168

 max_pooling2d_14 (MaxPoolin  (None, 8, 8, 256)        0
 g2D)

 Flatten (Flatten)           (None, 16384)             0

 dropout_4 (Dropout)         (None, 16384)             0

 Classifier (Dense)          (None, 512)               8389120

 dropout_5 (Dropout)         (None, 512)               0

 Output (Dense)              (None, 14)                7182

=================================================================
Total params: 8,788,910
Trainable params: 8,788,910
Non-trainable params: 0
```

Figure 3: Summary of the first model

Testing it on the CodaLab platform, we obtained a much lower mean accuracy of 29%. The following picture shows the result on the local validation set:

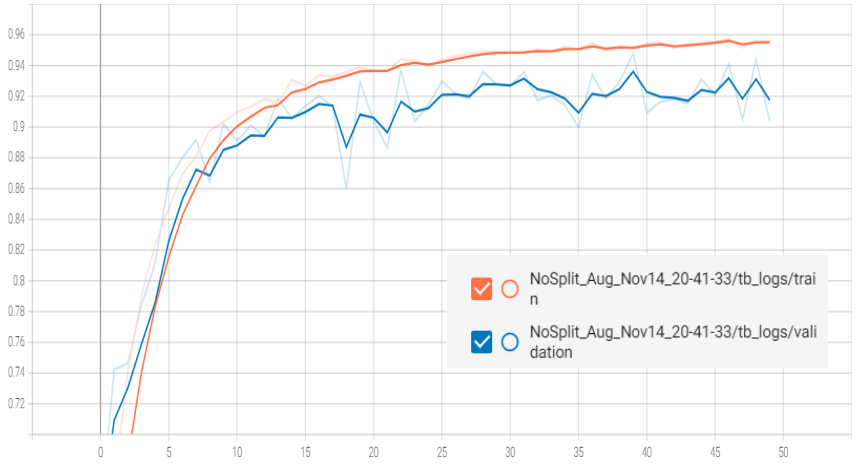

Figure 4: History Plotting of the first model

Analysing our dataset, we observed two main issues:

1. Classes are very unbalanced (as shown in *Figure 2*)
2. Only Codalab test-set presents wild and unhealthy subclasses labels

We tackled class-imbalance by using the *compute_class_weight* function that provides a class-weights vector, based on the target distribution of the training dataset:

```
y_integers = np.argmax(y, axis=1)
class_weights = compute_class_weight('balanced', np.unique(y_integers), y_integers)
d_class_weights = dict(enumerate(class_weights))
```

Figure 5: compute_class_weight function

A dictionary of the weights is then passed to the *fit* function as an argument. Once the classes have been balanced, the model performed worse on the local validation set, but noticeably better on CodaLab (from 29% to 36% mean accuracy).

3. VGG16 Model

At this point we wanted to obtain a model with a higher generalization capability, so we implemented Transfer Learning with VGG16 in order to rely on a strong feature extractor. We immediately noticed better testing performance and, to address the overfitting issue, we reduced parameter count by substituting the *Flatten* layer with the *GlobalAveragePooling* one and we added "l2" regularization inside dense layers. During the Fine Tuning phase, we freezed VGG16 until the 14th layer. These improvements led to a performance of 83%.
Many experiments on the hyper-parameters have been made. Eventually, we increased the *batch_size* from 128 to 256, and changed the *validation_split* from 20% to 15%. On the other hand, we kept unchanged other hyper-parameters like the *Dropout* probability equal to 0.3 and the "l2" regularization coefficient equal to 0.01. Making other tests using preprocessing, rescaling, or a simpler classification network, wouldn't improve the score in any way. We

found out that, freezing the first 12 layers during the fine tuning, we obtained mean accuracy equal to 92,26%, while freezing only the first 11 layers led to a worse score.

4. Final Model

We decided to exploit VGG19. After different tests on the number of layers to be freezed, we found the sweet spot at 18 layers, obtaining a score equal to 93,58%.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 256, 256, 3)]     0
_________________________________________________________________
resizing (Resizing)          (None, 256, 256, 3)       0
_________________________________________________________________
vgg19 (Functional)           (None, 8, 8, 512)         20024384
_________________________________________________________________
GlobalPooling (GlobalAverage (None, 512)               0
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 64)                32832
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 14)                462
=================================================================
Total params: 20,059,758
Trainable params: 7,114,798
Non-trainable params: 12,944,960
_________________________________________________________________
```

Figure 6: Summary of VGG19 model

The best model exploiting VGG19 produced the result shown below on the validation set:

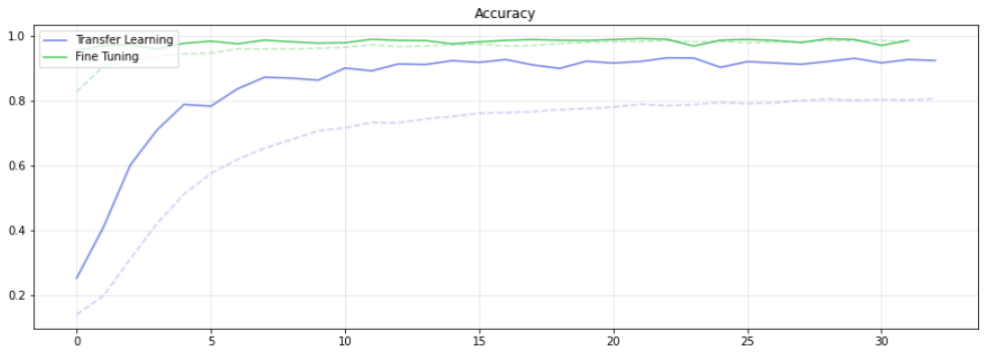

Figure 7: History Plotting of the VGG19 model

With a similar procedure we tried to implement EfficientNetB1, obtaining a slightly worse mean accuracy, equal to 93,3%, but with a much smaller byte size (70.479.143 bytes vs 128.560.979 given by VGG19) and a faster training.