POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Artificial Neural Networks and Deep Learning

# Homework 1: Image Classification

**Authors:**
- Domenico Cacace
- Miguel Gonzalez
- Usevalad Milasheuski

**Team name: DMU**

## 1. Introduction

Our strategy for this homework is to start from a very simple model, and gradually analyze the results and make the adequate changes. After reaching a *decent* model made from scratch, we then jump to transfer learning, where we use a pre-trained network as a feature extractor, and train a classifier on top of it; in the end, we compare our best models and draw our conclusions.

## 2. Dataset

The dataset used in this project is a subset of PlantVillage. Our dataset contains 17728 images of various plants, divided into 14 classes; the dataset was explicitly labeled (images of the same class are grouped in the corresponging subfolder, but it was not split in any way. In order to correctly evaluate the model, we split the dataset into a training (80%) and validation (20%) set: to perform the split we first used the `ImageDataGenerator`'s `validation_split` parameter, but in the end we used a *manually* stratified split of the dataset, in order to cope with the different class distributions. We did not create a test set, as it was already provided on the CodaLab platform.
All of the images are 3-channel RGB images, of size 256x256 pixels.

## 3. First steps

As a first step we built some small toy models, to evaluate on a small scale (without the need of waiting hours for the training to finish) which techniques worked best. Our starting point was a simple model, with two convolutional layers (with MaxPooling), two dense layers and no particular technique: this model had a reasonable training accuracy, but it was heavily overfitting the training set. To improve the model we added, step by step, dropout layers, regularization and data augmentation: we observed how the validation accuracy improved with these techniques, but we were aware that the model was too simple to work.
In the following sections we actually tackle the classification problem; this means that we take the experimental results obtained on the toy model and apply them (dropout, augmentation, regularization) on a real one. We also increase the batch size (256), the number of epochs (75-150) and add early stopping. Further details on the parameters used can be found in the model notebook.

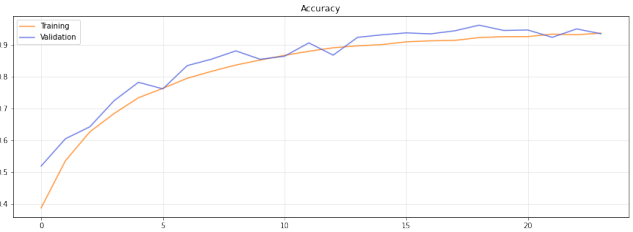
## 4. Our model, from scratch

Our first relevant[1] model was composed by 7 convolutional blocks and 2 dense layers; each convolutional block was composed by:

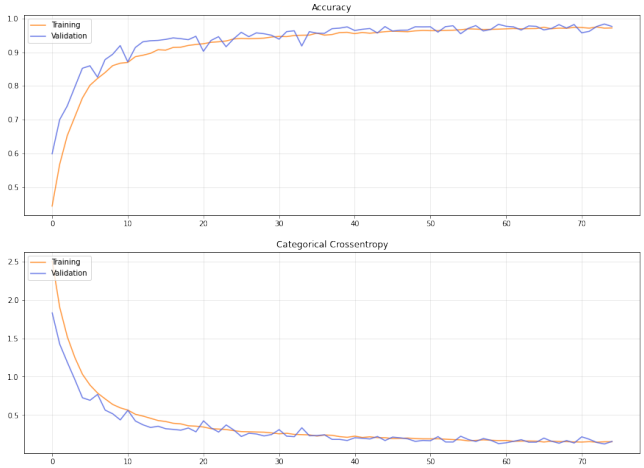
- `Conv2D` layer, $3\times3$ kernel, ReLU activation function
- `BatchNormalization` layer
- `Conv2D` layer, $3\times3$ kernel, ReLU activation function
- `MaxPooling2D` layer, $2\times2$ pooling, $2\times2$ strides

[1] For the sake of brevity we omit models that, in our opinion, do not contribute significantly to the report

We started with 8 filters in the first block, and doubled their number for each subsequent block, up to 512. After the convolutional blocks we `Flatten` the output and add two dense (512, 14) layers. With this model we scored 0.6792 on the test set, which is a decent baseline; the accuracy graph on the training and validation set are the following:




While finding a way to further improve our model we stumbled upon some articles suggesting that, following a similar idea to the one of data augmentation, we could add gaussian noise layers to increase the model robustness. By adding two `GaussianNoise` layers, one after the first convolutional layer and the other before the output layer, we obtained a 0.7302 on the test set, which is a good improvement.

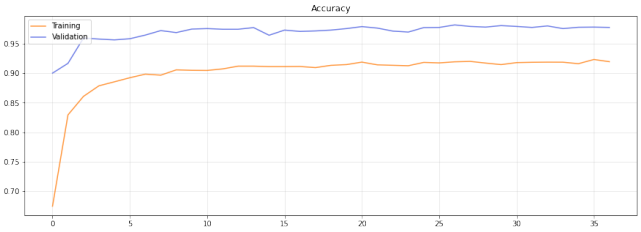



## 5. Transfer learning

We employed VGG16 with ImageNet weights as our feature extractor, and trained a classifier on top of it.
In a first moment we *recycled* a previous MLP model, composed of

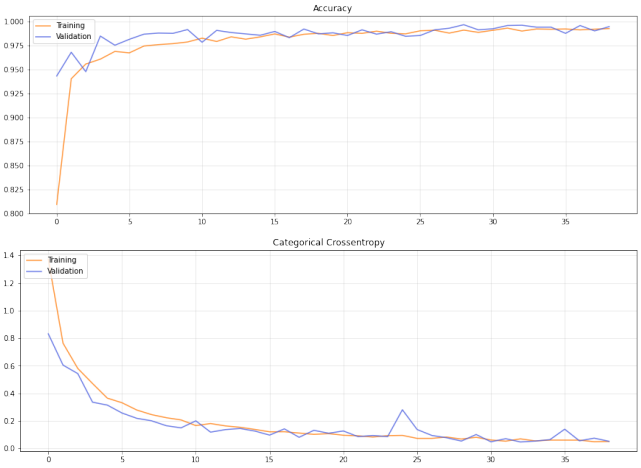
- `Dense` layer, 512 neurons, ReLU activation function, dropout of 0.25
- `Dense` layer, 256 neurons, ReLU activation function, dropout of 0.25
- `Dense` layer (output), 14 neurons, softmax activation function

To connect the feature extractor with the classifier we used a `Flatten` layer, with dropout of 0.5.
This model achieves a validation accuracy of 0.9831, but its test accuracy is way lower (0.49): we ruled as a possible reason for this behavior the scarcity of training data, so we implemented a data augmentation strategy to difersify the training set.
By retraining the model with this tweak we ended up with the following accuracy graph:




As we can see, there is a significative gap between the training and validation accuracy: this can be attributed to the high dropout rates, that cause some neurons to shut down and having the subsequent layers trying to construct predictions based on incomplete data. For this reason we ditched the dropout layers entirely, and observed that the gap tends to shrink.

Finally, to better suit VGG16 to our problem, we disabled the `freeze_layer` parameter for the last convolutional block, in order to fine tune its weights. The outcoming model scores 0.907 on the test set, and is our best model so far.




## 6. Conclusions

In conclusion we found that our best model, in terms of both accuracy and training time, was the last VGG16 model. This however does not mean that transfer learning by itself is the panacea for all problems: as we could clearly see, just straight-up copying an existing model does not necessarily yield the best results, and knowing how to fine-tune it is a crucial step.
On a personal note, we are really happy with the results we achieved: it feels good to put in practice the techniques learned during the theoretical lectures and to see how they can be applied to real problems.

# 7. Appendix A: *from scratch* model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 256, 256, 8)       224
_________________________________________________________________
batch_normalization (BatchNo (None, 256, 256, 8)       1024
_________________________________________________________________
gaussian_noise (GaussianNois (None, 256, 256, 8)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 8)       584
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 8)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 16)      1168
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 128, 16)      512
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 128, 128, 16)      2320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 64, 32)        256
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 64)        18496
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 32, 64)        128
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 128)       64
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 256)         295168
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 512)               2097664
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
```

```
_________________________________________________________________
gaussian_noise_1 (GaussianNo (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 14)                7182
=================================================================
Total params: 2,697,046
Trainable params: 2,696,054
Non-trainable params: 992
```

# 8. Appendix B: VGG16-based model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 8, 8, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 512)               16777728
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_2 (Dense)              (None, 14)                3598
=================================================================
Total params: 31,627,342
Trainable params: 16,912,654
Non-trainable params: 14,714,688
_________________________________________________________________
```