Artificial Neural Networks and Deep Learning

Caspar Dietz [caspar]*, Gianmario Careddu [GianMario]†, Jody Roberto Battistini [Jody98]‡

Team: JCGTeam
M.Sc. Computer Science and Engineering, Politecnico di Milano
*Email: *casparvictor.dietz@mail.polimi.it, †gianmario.careddu@mail.polimi.it, ‡jodyroberto.battistini@mail.polimi.it*

***Abstract*—In this report, we lay down the development process of our CNN for the classification of 8 different plants species.**

1. Introduction

The images in the data set have a low resolution of 96×96 pixels and the classes are unbalanced. Across all the experiments we relied on categorical cross entropy as a loss function and monitored the performances using early stopping over the accuracy. Since accuracy could be misleading in classification, especially with unbalanced classes, we have also employed confusion matrices and F1 scores. We decided to split the data in validation (20%) and train (80%), and to use the deliveries as a test. According to this split we were submitting the results only when we were confident about the offline results to avoid over fitting the test set. We have decided to try both building a CNN from scratch and experiment with transfer learning and fine tuning.

2. Data Pipeline

We started by building a CNN from scratch. To deal with unbalanced classes we first used a weighted loss function, under which the data was only ingested without any preparation step. For every experiment the data was shuffled and divided into validation and train sets at ingestion time. Since the results were not satisfactory, we then relied on oversampling. We used this technique to counteract the imbalanced data set and to guarantee maximum representation of each class in the training batches. Oversampling has been performed only in the training set. After this early stage we kept using oversampling for all the other experiments.

2.1. Data Augmentation

Due to the reduced dimension of the data set, compared to the complexity of the underlying architecture, we relied on data augmentation. We generated at run time several new transformed images. Technically this was achieved by means of a transformation layer embedded in the Keras model to benefit of the GPU speedup during augmentation. Thanks to augmentation we achieved on average 5% higher validation accuracy in all type of experiments. Starting from traditional transformations (2.1.1), we also experimented with advanced techniques to further improve the quality of the classification (2.1.2).

2.1.1. Traditional Transformations. Rotating, Zooming, Flipping, Brightness, Shifting. We empirically found that the best ones for our problem were random flipping (RFlip) combined with random rotations (RR), see Fig. 1. However too much extreme augmentation was introducing artefacts on the training images.

2.1.2. Advanced Transformations. CutMix and CutOut. These advance transformations did not increase the overall performance, but indeed, they made the training process slower and not as accurate as without them (around 7% less). Therefore we decided to discard these techniques.

2.2. Pre-Processing

Since for the features extractor we relied on state-of-art architectures, the data pre-

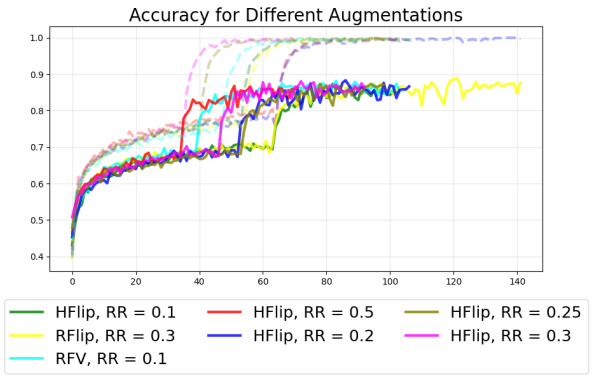

Figure 1. Accuracy for different data augmentations

Figure 2. Augmented images after Coutout and Cutmix

processing function adopted is the same suggested by the corresponding model contained in `tensorflow.keras.applications`.

3. Custom CNN

We started by building a completely custom CNN from scratch. We thereofore tried sixteen base configurations, ranging from 30K to 5M trainable parameters, having the VGG-like architecture. We decided not to include them in the report because of the low quality results. The strategy during this stage was to design a base architecture that reasonably over-fits the train set and then to apply all the regularisation techniques seen in the course (dropout, batch normalisation, L2 regularisation). Under this approach the best models were around 0.65 accuracy on the validation set. We then decided to automate the model, parameter and hyperparameter search, by developing a combination of grid and random search and a function able to generate model of different sizes at run time (*RandomGrid_Htuning* notebook). We also experimented with different activation functions like swish, selu and elu. The best model obtained scored 0.7 accuracy on the validation set, but only 0.65 on the test set. As were over-fitting the validation set with so many experiments, at this point we decided to abandon this strategy and concentrate on transfer learning.

4. Transfer Learning and Fine Tuning

4.1. Features Extractor

We started comparing the most famous architectures presented during the course lectures through transfer learning and GAP + Softmax layers as baseline classifier. We plotted both categorical cross-entropy and accuracy for the validation and training set (dashed line). See Figure 3.

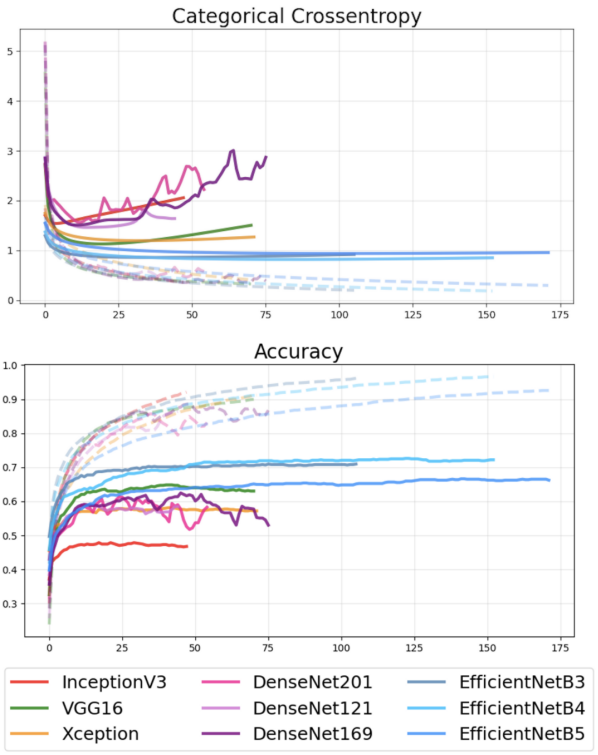

Figure 3. State-of-Art architectures comparison.

As can be seen, the best models were EfficientNetB4 and EfficientNetB3. While the latter converges faster, the former is able to reach a higher peak performance in both loss and accuracy. Thus, we have chosen EfficientNetB4 as feature extractor.

4.1.1. Freezing Layers in the Features Extractor.

Having trained the entire CNN with the EfficientNetB4 layers frozen, we went on to fine tune our model. We started doing so by experimenting with different percentages of the top layer of EfficientNetB4 to be frozen. In Fig. 4 we compare the different freezing amounts. Note that the sudden jump in performance

indicates the moment at which which started fine tuning, i.e. unfreezing parts of the top layers. The best result was achieved by freezing only top 5% layers, and it scored 0.90 on validation set.

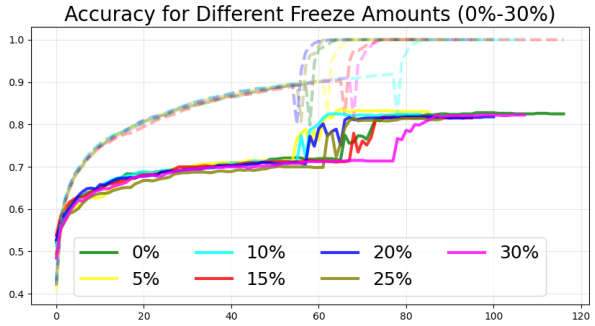

Figure 4. Accuracy for 0%-30% freezing of the top layers

4.2. Custom Classifier

Starting from the previous results for the GAP + Softmax architecture, we now experimented with other, more complex classifiers.

4.2.1. Global Average Pooling Layer. This layer was introduced in order to avoid the usage of the flatten layer, being heavier to manage in terms of complexity and parameters. While GAP is a lightweight layer able to be invariant to intrinsic shift and rotation by structural definition, gaining a more powerful generalisation performance.

4.2.2. Dense Layer. We have experimented with single and double dense layers, with various configurations. The best results were achieved by a single dense layer of 256 neurons. We have observed that increasing the dense parameters even by one order of magnitude was not giving any improvement, probably because the bottleneck in performance was the quality of feature extracted by the convolutional part. See Table 1.

4.2.3. Dropout Layer. We used dropout to counteract over-fitting. It demonstrated to be a powerful technique. In particular we experimented with different values for the dropout in the EfficientNet part and in the dense part. See Table 1. Dropout was preferred to kernel regularisation techniques since it is easier to tune and often better performing.

Table 1. Summary of most important experiments

Val. Acc.	Class. DO	2DConv DO	Dense	LR
0.90	0.2	0.4	256	1e-4
0.90	0.4	0.4	256	1e-4
0.88	0.5	0.5	512	1e-4
0.88	0.4	0.4	1024	1e-4
0.89	0.3	0.2	5k	1e-4
0.89	0.3	0.2	5k, 250	1e-4
0.87	0.3	0.2	10k, 256	1e-5

5. Ensemble

As final step we exploited the concept of Ensemble in order to reduce the overall variance in predictions. We have employed the three best models obtained after all the experiments and we concatenated the Softmax output of each model using the mean function. We decided not to employ a weighted average because the best models had very comparable performances. The ensemble boosted performance by around 1.5% on the validation set, and improved the score by around 2% on the delivery used as test.

6. Final Model

Ensemble of the best model using GAP + softmax and the best two models using a custom classifier. Development phase accuracy: 0.8730. Final phase accuracy: 0.8521

min F1	**max F1**	**Avg F1**	**Valid. acc.**
0.6415	0.9603	0.8910	91.51%

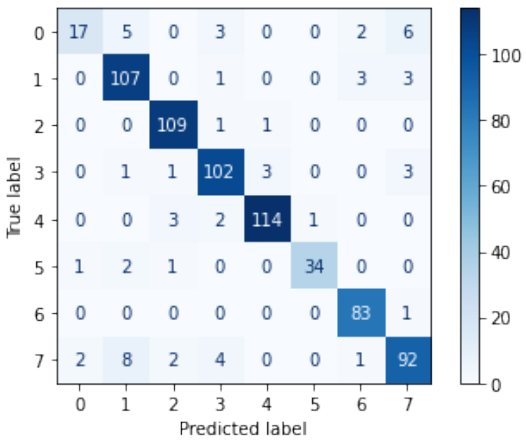

Figure 5. Confusion Matrix on the validation set.