Convolutional Neural Network: Plants Species Classification

Artificial Neural Networks and Deep Learning – A.Y. 2022/2023

Paolo Botta*, Teo Bucci† and Silvia Caresana‡
M.Sc. Mathematical Engineering, Politecnico di Milano - Milan, Italy
*Email: *paolo.botta@mail.polimi.it, †teo.bucci@mail.polimi.it, ‡silvia.caresana@mail.polimi.it*
*Student ID: *10612869, †10621873, ‡10630163*
Codalab Group: "Just3Neurons"

1. Introduction

The given dataset consists of 3542 images belonging to 8 different classes. To build the classifier, many choices about the model had to be made. Since the settings that can be tweaked in the model would generate many more configurations than our computational power could handle, we followed an incremental approach. Specifically, we started with a baseline and slowly made choices from there, seeing whether the performance improved or not, trying to understand why and if the outcome was expected.

The task suggests to use Categorical Cross-Entropy as loss function, and as metric we used Accuracy as requested. In addition, we also considered the F1–score per class to better understand the weaknesses of the model.

The dataset was split according to an 85:15 train-validation ratio and the results of this report are the ones we obtained on the validation set.

2. Data Pipeline

2.1. Data Augmentation

Since the dataset is not very numerous, Data Augmentation was used almost from the beginning in all models, allowing the network to better generalize and have a better performance.

We used **traditional transformations**: Rotating, Zooming, Flipping, Brightness, Shifting, Shear. Initially only the first three were used, but adding all of them improved slightly the performance.

We thought of using also more **advanced transformations** like CutMix or CutOut, but given that the dataset is made of plants more or less homogeneous in each sample, we discarded those options and focused our attention more on the architecture for the time being.

Figure 1. Some examples of augmented images.

2.2. Test-Time Augmentation

Once we had our final model (section 5) we also implemented **Test-Time Augmentation (TTA)** to build more robust predictions and reduce variance. This gave an improvement of 1.52% in accuracy.

2.3. Pre-Processing

According to the documentation, the Features Extractors (section 3.1) that we chose work best with standardized inputs. Therefore, data processing was done according to the `preprocess_input` contained in `tensorflow.keras.applications`.

3. Convolutional Neural Network

We started with a baseline model adopting the classical architecture of modern CNNs. We used 5 blocks of `Conv2D` + `MaxPooling2D` and a `Dense` layer with 512 neurons, and we obtained a starting accuracy of 60.91%.

Then, we turned to some pre-trained models to perform transfer learning. Given the reduced size of the dataset, we were quite sure this would greatly cut training time and increase performance.

3.1. Feature Extraction

To choose our features extractor we compared some famous architectures equipped with a simple fully connected classifier made of 512 neurons, and run the simulation for 30 epochs.

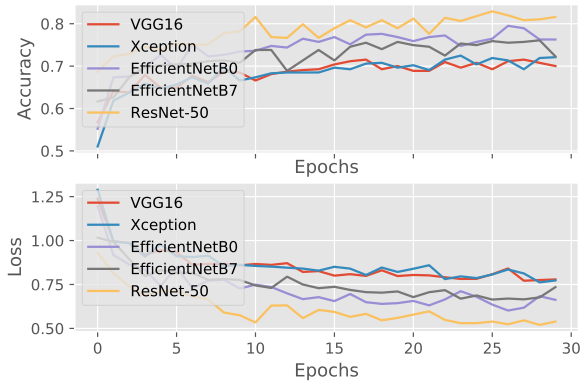

Figure 2. Architectures comparison.

According to the results in Figure 2, the best model was ResNet-50. However, we decided to keep also VGG16 given its fast inference time.

Xception	68.31%
EfficientnetB0	77.23%
EfficientnetB7	73.24%
ResNet-50	**79.51%**
VGG16	68.69%

3.2. Classifier

3.2.1. Global Average Pooling Layer. After the convolutional base we initially had a `Flatten` layer, which we later switched to `GlobalAveragePooling2D` to better summarize the output of the feature extractor. The improvements weren't significant, yet there was a reduction in the number of parameters and in training time.

3.2.2. Dense Layer and Activation. To decide the number of `Dense` layers, the number of neurons and the type of activation we considered different options with the following rule: `tanh` should be used when the number of layers isn't too much to avoid the vanishing gradient problem, while when more layers were used it was better to use `ReLU`.

Moreover we used the Keras tuner with the HyperBand algorithm[1] to tune these hyperparameters.

This testing was performed before fine-tuning (section 4.1).

VGG16	
ReLU, 512	73.81%
ReLU, 256 + 128 (x2) + 64 (x2)	67.36%
tanh, 512	69.45%
tanh, 512 + 256	69.45%
ReLU, 384 + 256 (Keras tuner)	**74.95%**
ResNet-50	
ReLU, 384 (Keras tuner)	**85.39%**
ReLU, 512	**85.39%**

For VGG16 we opted for the tuner configuration.

For ResNet-50, the two configurations performed identically, but we ended up choosing 512 neurons because after fine-tuning the performance was better.

3.2.3. Batch Normalization Layer. We employed the `BatchNormalization` layer in order to improve gradient propagation and reduce internal covariance.

3.2.4. Dropout Layer. `Dropout` was always used after the `Dense` layer for regularization, since on average we saw an increase of 0.76% in accuracy.

4. Training techniques

4.1. Fine-tuning

Initially, transfer learning was performed first by training the classifier with the convolutional base frozen, and then tuning with a lower learning rate with the last layers of the feature extractor unfrozen.

1. Lisha Li et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: (2016). DOI: 10.48550/ARXIV.1603.06560.

However, the performance wasn't really satisfying, so we decided to introduce the idea of **multiple passes**.

We thought that the latent representation of the last frozen layers of the feature extractor wasn't still in line with our dataset, so progressively we unfroze more layers and did more fine-tuning with an even lower learning rate. This gave a great improvement on our validation set as shown in the comparison:

	VGG16	(*ResNet-50*)
No fine-tuning	74.95%	85.39%
Unfreeze last 4 (*32*)	90.13%	89.56%
Unfreeze last 8 (*64*)	91.65%	**91.27%**
Unfreeze last 12 (*96*)	**92.60%**	90.32%

We chose the best fine-tuning configuration for each of the supernets.

4.2. Class imbalance

Classes 1 and 6 were severely under-represented, about half the size of the other classes. So we decided to compute the class weights according to the number of samples: this led to a **weighted loss function** that gave more weight to the less represented classes when performing backpropagation. As expected, the F1–scores for such classes saw an improvement of about 0.10.

5. Model choice

To choose the final model we considered the one built with VGG16, the one with ResNet-50, and a third one: the **ensemble** of the two by taking the simple average. We compared not only accuracy, but also the F1–scores.

		VGG16	ResNet-50	Ensemble
F1–scores	1	0.735	0.723	**0.739**
	2	**0.955**	0.926	**0.955**
	3	0.911	0.913	**0.928**
	4	**0.921**	0.906	0.899
	5	0.931	0.904	**0.933**
	6	0.900	0.862	**0.918**
	7	**0.982**	0.975	**0.982**
	8	**0.911**	0.876	0.903
	Acc.	92.60%	91.27%	**92.98%**

The ensemble showed an improvement in almost all the metrics, therefore that is the model we chose.

6. Conclusion

6.1. Performance

Finally, we added TTA (section 2.2) and the results of the final ensemble both on the train, the validation and the remote test set are as follows:

	Without TTA	With TTA
Train acc.	99.30%	**99.80%**
Val acc.	92.98%	**94.50%**
Test acc.	89.16%	**91.40%**

The final confusion matrix can be seen in Figure 3.

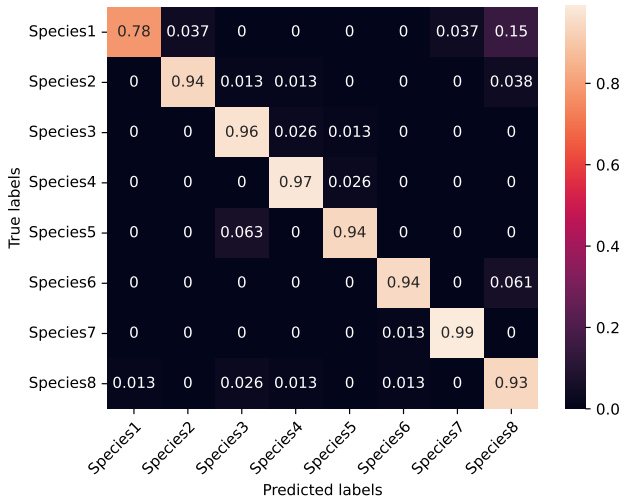

Figure 3. Confusion matrix on the validation set of the final model.

6.2. Further Developments

We acknowledge that our workflow might use some additional procedure, for example an exploitation of more advanced augmentation techniques, in order to help the generalization power of the model. Moreover, we could experiment with more advanced pretrained models, different optimizers and learning rates.

It can be seen that, although we introduced class weights (section 4.2) to tackle class imbalance, the F1-scores of the underrepresented classes are still lower than the ones of the other classes: further procedures may be implemented to handle this issue.

Taking account of these considerations, and given the limited resources at our disposal, we are satisfied with the results.