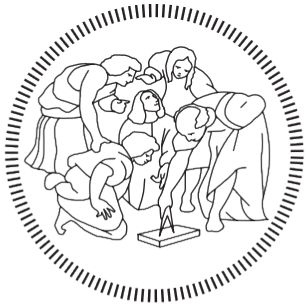
ACCADEMIC YEAR 2022/2023



POLITECNICO
MILANO 1863

POLITECNICO DI MILANO

Artificial Neural Networks and Deep Learning

Challenge 1 - Image Classification

Igor Katselenbogen - 10855055
Leonardo La Rocca - 10930221
Filippo Manzardo - 10864201

Prof. Giacomo Boracchi

Version 1.0
November 25, 2022

1 Introduction

The first challenge of the Artificial Neural Networks and Deep Learning course is about *Image Classification*.
Image Classification is one of the classic problems of Machine Learning and consists in attributing a label from a set of pre-determined labels to an input image. Modern models have surpassed humans in the task, and new and improved architectures are constantly designed. For example, *A ConvNet for the 2020s* [3], presented at CVPR in 2022.
In this challenge, we test a set of Convolutional Neural Network models [1], which have been proven many times to be the most effective approach to tackle this kind of problem. We start from a naive baseline model, adding layers of complexity to achieve better outcomes. Interestingly, the baseline model was written entirely by an AI[1].

2 Problem Definition

The challenge consists in classifying images of plants into eight different categories, each corresponding to a plant species.

Fig. 1: Samples from the dataset

The dataset of the challenge is composed of 3542 labeled images of plants belonging to the eight species. The images are not equally divided among the classes; see Table 1.

Class	Occurrences	Class	Occurrences
Species1	186	Species5	531
Species2	532	Species6	222
Species3	515	Species7	537
Species4	511	Species8	508

Tab. 1: Dataset distribution

[1] https://github.com/features/copilot
[2] https://github.com/filippomanzardo/AN2DL

We've fought this disparity by balancing class weights, for example with the following formula:

$$w_i = \max(n)/n_i$$

with $n = \{n_1, \ldots, n_8\}$ as the occurences vector

3 Approach

To define, train and evaluate Neural Networks, we use Python [4] and Tensorflow [5]. All the developed and tested code is available on GitHub. [2]

To overcome local processing issues, such as long training times and overheating laptops, we've implemented a Cloud infrastructure on the Google Cloud Platform with the help of TensorFlow Cloud, a Python package that takes care of middle and bare-metal details like CI/CD with Cloud Build, containerization, and job running.
For more info, see the related README on the GitHub repository.

We noticed that the dataset provided with the challenge alone is too complex to train models designed from scratch. The result we obtained by training "vanilla" convolutional neural networks were not acceptable, and we observed that the models were underfitting in the training phase. Therefore, we focused on alternative techniques that improve the average performance of the models despite the lack of training data.
Let's now describe which methodologies we used to tackle this challenge.

Train-Validation-Test Splitting

First, to understand how our nets were performing, we chose a $70-20-10$ train-validation-test split. Once we were happy with the results, we trained the model with the best parameters found on the entire dataset for around 10% more epochs. This will be the model to be submitted in Codalabs.

Transfer Learning

Transfer Learning has become a must when it comes to training neural networks. It reduces training times and unlocks higher accuracy. Furthermore, TensorFlow is packed with pre-trained state-of-the-art networks. We used the following models:

ConvNeXt	VGG16	Efficientv2
Xception	EfficientNet	MobileNetv2

Tab. 2: Pre-trained nets

Pre-Processing

Preprocessing is a delicate subject when it comes to using pre-trained architectures as they are trained on data preprocessed in a specific way. Most of the models come with pre-defined preprocessing pipelines that are specific to the net itself. In general, they mean-centered and normalized the data [3].

Data Augmentation

Overfitting was one of the most challenging problems we faced. The small size of the dataset and the complexity of the used networks made it very hard not to overfit models. A solution we implemented to reduce the probability of overfitting is to use data augmentation. Data augmentation consists in applying label-preserving transformations to the training data to increase the size of the dataset. We implemented data augmentation through Keras augmentation Layers like *RandomContrast*, *RandomBrightness* or *RandomCrop* and the *ImageGenerator* class. This enabled us to increase the average accuracy by around $\tilde{8}\%$.
Another technique we implemented was CutMix [6], where patches of training images are cut and pasted together. The labels are also mixed proportionally to the area of the patches. In summary, data augmentation helped our model to generalize.

Fine-Tuning

All the base nets were first frozen to train only the last manually-defined layers. At the end of the training phase, we unlocked around 30% of the layers of the base net and re-trained the model with around 33% of epochs compared to the number of training epochs. In this phase, it is essential to reduce the learning rate to avoid "forgetting" what was learned in the first phase.

Hyper-parameter Tuning

Finally, we had to tune the hyper-parameters of the models to find the best-performing combination of hyper-parameters. We chose two ways to explore hyper-parameters space. The first *naive* approach, which was only possible while training locally and with less-complex nets, consisted in manually adjusting the parameters. The second approach exploited was the HyperTune [2] utility of TensorFlow Cloud, which deployed a smart-distributed search of the defined parameter space, like the *learning rate*.

Ensemble Learning

Ensemble learning is a technique that combines multiple models' predictions into a single one with the hope of making a more accurate prediction. We implemented a multiple classifier system with three models: two ConvNeXt large models and a ConvNeXt base model. We combined their predictions by averaging the softmax activations. We also tested a majority moving combination; however, this prediction combination did not increase accuracy for this challenge. With the average combination of the probabilities, the ensemble model achieved an accuracy increase of nearly 3%.

Other stuff

We used Tensorboard extensively to monitor the progress of our training pipelines. We developed a callback that stores the best model in memory. This allowed us to exploit early stopping at its best. Quick shout out also for the most beatiful and colored local logging. [4]

4 Results

The first coding phase of the project was focused on setting up the model creation, training, and testing environment that would allow us to design and iterate on models as fast as possible.

After that, we took the time to manually classify a subset of images [100#], to get statistics about human performance. In Table 3 the results by *codice persona* completely anonimized.

CP	Accuracy
..55	34.00%
..01	42.00%
..21	40.00%

Tab. 3: Human Performance

This indicates that the dataset is pretty complex, and we could expect low accuracy in the nets.

[3] https://www.tensorflow.org/api_docs/python/tf/keras/applications/xception/preprocess_input
[4] https://github.com/filippomanzardo/AN2DL/blob/main/challenge_1/challenge_1/runtime/log.py

Finally, we evaluated the trained models and picked the best ones to combine in an ensemble model.

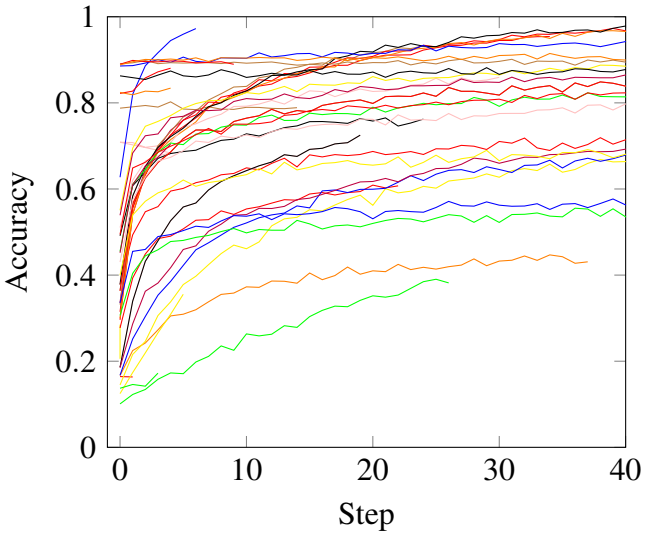


Fig. 2: Training Curves

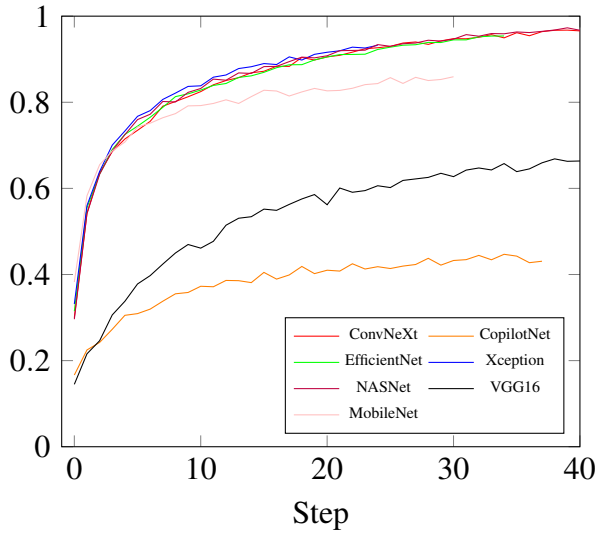


Fig. 3: Final Training Accuracy

In Table 3, the highest tests results achieved on the said $70-20-10$ split.
At the end, the best models were ConvNeXts, either XLarge or Base. These models were trained on Google Cloud for 400 epochs and 100 epochs of fine tuning, which took around 20 hours for the bigger models, and 5 hours for the base model.
On Codalabs we reached top-1 85-86% of accuracy. The final bump to 89% was given by an averaged ensemble of 2 ConvNexTXLarge and 1 ConvNeXtBase.

TL Net	Accuracy
ConvNeXt	86.56%
VGG16	72.87%
EfficientNet	78.34%
Efficientv2	80.21%
Xception	83.90%
MobileNetv2	74.35%

Tab. 4: Test accuracies

5 Conclusion

We realized how challenging it is to train accurate models with little training data available. We tried to fight overfitting in every possible way and acknowledged the importance of preprocessing and data augmentation techniques. We also tested the effectiveness of Transfer Learning with highly positive results. Looking back, we would have focus ourselves in training more ensemble models, which would have given us a competitive advantage. Despite the challenges, it is fascinating how incredibly precise the models can get, even with such a restricted dataset.

Bonus

We also acknowledged how costly training on the Cloud can be, fortunately for us we had free trial credits to spend.

Servizio	Costo
Cloud Build	0,00 €
Cloud Logging	0,00 €
Cloud Machine Learning Engine	155,07 €
Cloud Storage	9,61 €

Fig. 4: Google Cloud Platform costs

References

[1] Chris M Bishop. "Neural networks and their applications". In: *Review of scientific instruments* 65.6 (1994), pp. 1803–1832.

[2] *HyperTune, Tensorflow*. https://cloud.google.com/ai-platform/training/docs/using-hyperparameter-tuning.

[3] Zhuang Liu et al. "A convnet for the 2020s". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 11976–11986.

[4] *Python [3.10]*. https://www.python.org/downloads/release/python-3100/.

[5] *Tensorflow [2.10]*. https://github.com/tensorflow/tensorflow/releases/tag/v2.10.0.

[6] Sangdoo Yun et al. *CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features*. 2019. DOI: 10.48550/ARXIV.1905.04899. URL: https://arxiv.org/abs/1905.04899.