# Artificial Neural Networks and Deep Learning - Challenge 1
# Image classification

November 28, 2022

Riccardo Luigi Aielli
Salvatore Buono
Simone Buranti

POLITECNICO
MILANO 1863

## Contents

## 1 Introduction

The goal of this project is to build a Convolutional Neural Network able to classify plants images into 8 classes. In this report we illustrate the process that brought us to build a model that reached 91% accuracy on test set. All our work has been done on Google Colab, because it offers free computational units in order to execute our code. Throughout the challenge, we used a bottom-up approach starting from simpler models progressively increasing the complexity.

## 2 First Approach

### 2.1 Dataset split

We noticed that data were almost equally distributed among the three classes, we selected for the validation set the same percentage of images from each class (stratified sampling). What we wanted to achieve is that the distribution of classes in the validation set reflects the distribution of classes in the test set. We divided entire dataset into training and validation sets with a ratio 85/15. Moreover we decided to not split dataset into test set because of shortage of images but using Codalab platform to evaluate performances.

### 2.2 Base CNN

Firstly, we faced the problem by building a CNN model from scratch, very similar to the one we saw in class, as a baseline model to establish a minimum model performance to which all of our other models can be compared.

### 2.3 Increasing performance with data augmentation

To reduce the scarcity of data we used data augmentation that increase amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It acts as a regularizer and helps reduce overfitting when training the model. In terms of image preprocessing, we applied a normalization of the input thanks to rescale argument of `ImageDataGenerator` class. We obtain these performances:

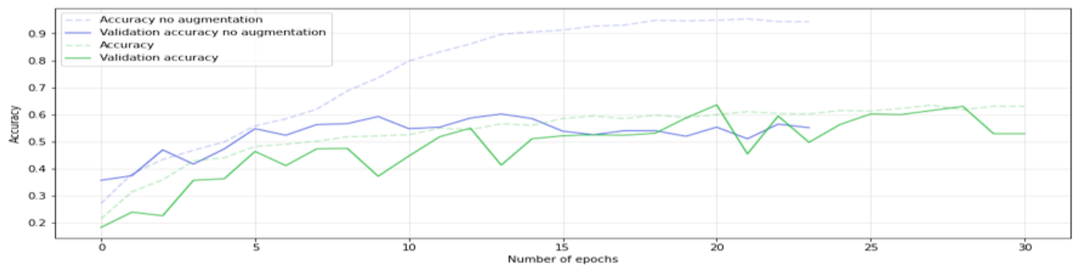

Figure 1: Accuracy of base model with data augmentation

## 3 Transfer Learning and Fine Tuning

Not satisfied with the performance of the handmade model, we decided to use the transfer learning and fine tuning technique. In order to improve accuracy of our model, we decided to duplicate dataset in order to obtain different types of augmentation of the same image

and to insert GlobalAveragePooling layer with the aim of reducing overfitting and enforcing correspondences between feature maps and categories.
We first tried VGG16, DenseNet121, DenseNet201 then InceptionV3, Xception and finally ConvNext Base. Here we present the most relevant models.

## 3.1 VGG16

It is the newtork presented during exercise lessons, since it is one of the first neural network developed, it has poor performance but easy interpretation. These are the performance obtained with Transfer Learning and Fine Tuning freezing the first 14 layers:

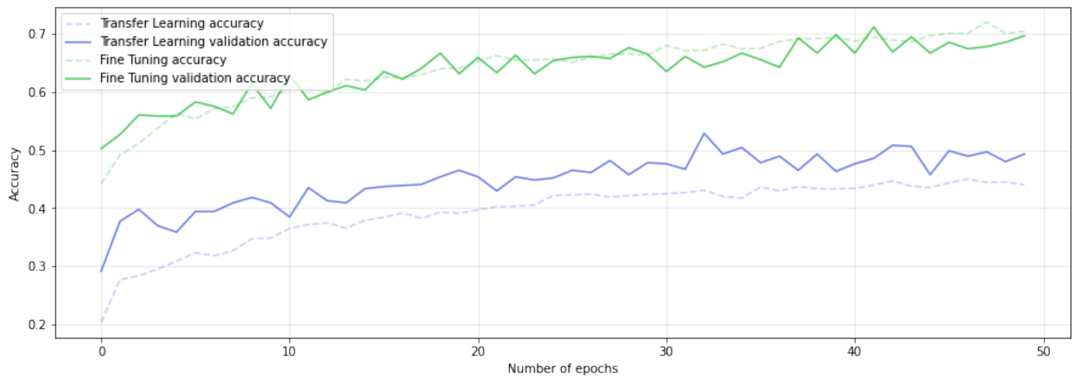

Figure 2: Accuracy of VGG16 model

## 3.2 DenseNet201

Then we tried to use the DenseNet201 model, which greatly improved the performance compared to VGG16, despite this we tried to further improve the results with other techniques and models.

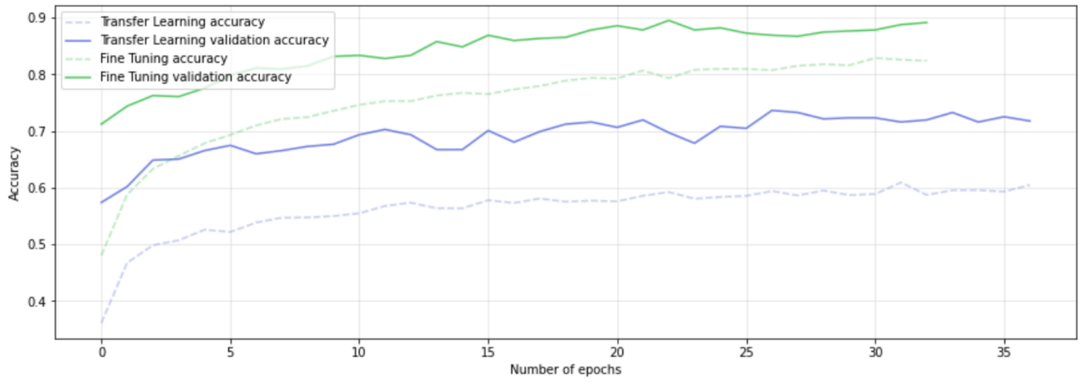

Figure 3: Accuracy of DenseNet201 model

### 3.2.1 K-fold Cross Validation

While proving different settings of hyper-parameters, we also tried k-fold cross validation with k=10 for the purpose of estimating the skill of the model on new data. On the other hand it is very time-consuming and it does not reach desired performance on the test set, so we tried further networks.

### 3.3 ConvNext Base

By taking as a reference the Keras-Applications documentation, we finally tried to implement Transfer Learning and Fine Tuning with this last CNN family, which showed promising statistics. At the beginning, we tried ConvNeXtSmall as it is the lightest one, yet noticing a significant improvement in the accuracy on our dataset. Even better performances were reached when we imported the following, slightly more complex, ConvNeXtBase. Given the prohibitive size and computational resources demanded by ConvNeXLarge, we decided to focus on the previous one.

#### 3.3.1 CutMix

In order to further improve our Data Augmentation pipeline, we decided to adopt Cutmix regularization strategy. In particular, we instantiated two `ImageDataGenerator` linked to the same training folder, and with the same Data Augmentation settings. Then we replaced the previous one with a cutmix version from `cutmix_keras`, fed by the two simple instances.

## 4 Final Results

The model obtained as previously described reached high values of accuracy on our training and validation sets. However, by submitting it on the platform we were still witnessing a performance gap of about 5% of accuracy with respect of our local scores. Thus, we suddenly applied three additional changes. Firstly, to reduce even more the overfitting we might still suffer from, we introduced a `GaussianNoise` layer after the input one in the CNN architecture. This, combined with the Cutmix regularization strategy, allowed us to significantly reduce the drawbacks of having a circumscribed dataset and to generalize as much as possible the images which the model was fed by. Secondly, we added two Dense layers followed by a consecutive Dropout in the final part of the architecture, in order to improve the complexity, thus the effectiveness, of the classification module.

Finally, still to enhance the generality of the model, we turned the patience of the Early Stopping to 7, as we noticed that, due to sparse validation accuracy peaks, slightly better of previous best scores, the training was lengthened.

These ultimate refinements allowed us to reach an overall accuracy of 91.1% on the reference test set.

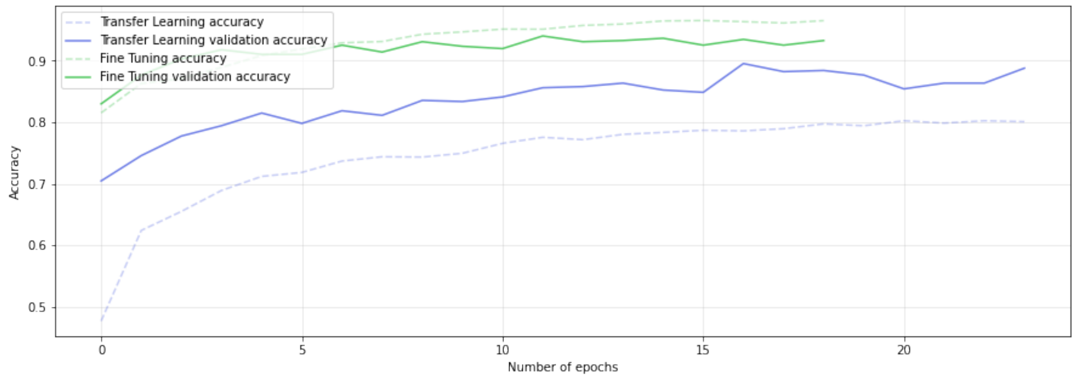

Figure 4: Accuracy of Final model