POLITECNICO
MILANO 1863

Project Report · Artificial Neural Networks and Deep Learning · Homework 1

Cariati Leonardo [10588999]
Cutrupi Lorenzo [10629494]
Zardi Enrico [10659549]
Group: Pietort

Contents

1 Development phase

1.1 Creating our first CNN

In the first phase of the challenge, we decided to experiment the design of a simple Convolutional Neural Network with a structure inspired by common examples in the literature and seen in the lectures. It started with a sequence of 3 convolutional layers having Relu as activation function, each one followed by a max-pooling layer. A Flattening layer was introduced after the convolution part as input to a Fully connected neural network composed by 2 dense layers and the output, which had Softmax activation function for classification. We proceeded in training the model using $(80 \div 20)$ split ratio for training and validation set, accuracy as a metric and cross-entropy as loss function. With this configuration our first model scored, as expected, 43% in the Codalab leaderboard.

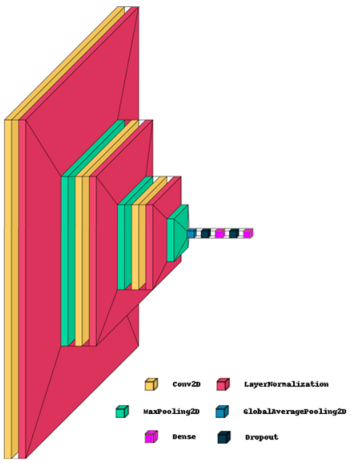
1.2 Architecture improvements

We noticed that our CNN was prone to overfitting and perhaps needed some regularization, so we decided to make some changes in the structure:

- Layer Normalization and Dropout were introduced after each Convolutional Layer and before and after each Dense Layer, respectively. The first one helps in regularizing the convolution part and the second to generalize the model in preventing dependencies between neurons.
- Doubled the number of filters in each Convolutional layer in order to extract more features going from $(16, 32, 64)$ to $(32, 64, 128)$.
- Replaced the Flattening layer with a Global Average Pooling layer, as suggested in the lectures, that reduced drastically the number of parameters of our model, making it faster to train.

Our training method was modified as well: we changed the split ratio to $(70 \div 30)$ to have a more representative validation accuracy and started using Early stopping callback to save our best scoring model. Also a rescaling of 255 in the input images was introduced to regularize the data.

After these modifications, we ended up with the definitive version of our model built from scratch, which improved the overall evaluation metrics, leading to a validation accuracy of around 60%.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 96, 96, 32)	896
layer_normalization (LayerNo	(None, 96, 96, 32)	64
max_pooling2d (MaxPooling2D)	(None, 48, 48, 32)	0
conv2d_1 (Conv2D)	(None, 48, 48, 64)	18496
layer_normalization_1 (Layer	(None, 48, 48, 64)	128
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
conv2d_2 (Conv2D)	(None, 24, 24, 128)	73856
layer_normalization_2 (Layer	(None, 24, 24, 128)	256
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
global_average_pooling2d (Gl	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense (Dense)	(None, 128)	16512
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 8)	1032

Total params: 111,240
Trainable params: 111,240
Non-trainable params: 0

Figure 1: Final version of the CNN built from scratch

Nevertheless the model was still overfitting the training set, so we understood that we needed to focus more on the preprocessing phase, since the dataset was small and the model was not able to generalize itself from it.

1.3 Data Analysis and Augmentation

We noticed that our model scored poorly in the classification of Species 1 and Species 8, usually swapping them, resulting in a low F1 score. Having a look at the dataset we noticed that images belonging to the first and the eighth species were very similar, and there were fewer data belonging to Species 1 and 6, so these two could be the reasons of the problem just described.

```
ImageDataGenerator(rescale=1/255.
                   height_shift_range = 40,
                   width_shift_range = 40,
                   horizontal_flip = True,
                   vertical_flip = True,
                   fill_mode = 'reflect')
```

Figure 2: Augmentation setting

Focusing for the moment on solving overfitting we chose to perform Data Augmentation to generate new data to be fed into the network, and we tested different parameters combinations to find the optimal ones: we found out that using an heavy shift, horizontal and vertical flip improved the accuracy, while using other parameters such as zoom or rotation reduced it, so we decided to use the configuration on the left.

We also spotted pictures in the dataset containing shadows that were covering most of the image, and we thought they could worsen the training phase, so we labelled them as outliers and removed them from the training set, but not from the validation set, to make sure that the validation accuracy was still representative of the effective result, while the training set was free from misleading images.

Figure 3: Some of the outliers found in the dataset

Performing data augmentation with the new training dataset resulted in a serious improvement on the model performances in both the validation accuracy, which increased to 83%, and the submit accuracy, which increased to 80%. To address the Species 1 problem we tried the weight rescale technique which assigns higher weights to classes with less data during the training phase. However it did not work as expected, performing badly overall in each class but not even overfitting the training data, reaching a validation accuracy of maximum 75% in the training phase. Changing the optimizer, the learning rate, increasing the batch size during training and adding batch normalisation layers between convolution did not lead to good results.

Our conclusions were that our model was too simple and, for that reason, it was not able to perform well in this classification task. A change of approach was needed.

2 Transfer Learning & Fine Tuning

We decided to use transfer learning in order to find a well-fitting model for our classification problem. We combined it with fine tuning as it is a common advanced approach in deep learning used to achieve high performances in classification problems and we selected models like VGG16 and ResNet50, popular CNN applications seen during the lectures, to experiment these techniques.

2.1 Application selection

After choosing a specific pretrained CNN, we substituted its feed forward part with dense layers of our choice in order to make that Convolutional Neural Network a 8-classes classifier suited for our task. Running some quick training, we ended up with promising results, slightly better than the CNN we built from scratch.
Therefore we concluded that the newly implemented techniques had a lot of potential for our task, so we checked the list of applications on keras.io/applications and chose the one that, according to the website, achieved the highest Top-1 Accuracy: ConvNextXLarge.
The main issue with this model is its size, around 350 million parameters, so, if we already had problems in training much smaller CNNs, we had to make sure it was worth it and anyway elaborate a strategy to avoid

losing all our work due to a disconnection or memory overflow. Our solution consisted in using a function to save checkpoints of the best model reached so far during the training phase, so that we could split the training in more phases and reduce the number of epochs of each sub-training.

Before training the XLarge version of ConvNext, we tried its smaller brother, ConvNextBase, that required less time to train since it has only 88 million parameters, and it outperformed both VGG16 and ResNet50, achieving a promising 85% in validation accuracy.

2.2 ConvNextXLarge

Having established and confirmed that the selected model was a good choice, we definitely switched to ConvNextXLarge and started training it. We removed its final part to add our feed forward network composed by a Global Average Pooling Layer, one Dense Layer with a Dropout to avoid overfitting, and the Output Layer with Softmax as activation function.

Starting with transfer learning, we only trained the feed forward part using RMSprop optimizer, as we noticed that it reached higher peaks in both training and validation accuracy and it had another benefit: it was faster.

We made some other adjustments that helped up getting better results:

Layer (type)	Output Shape	Param #
ConNextxlarge (functional)	(None, 96 96 32)	348197146
GlobalAveragePooling	(None, 2048)	0
dense (Dense)	(None, 1024)	2098176
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 8)	8200

Total params: 350,254,344
Trainable params: 2,106,376
Non-trainable params: 348,147,968

Figure 4: Final model with ConvNextXLarge as application

- In order to have more samples of plants belonging to species 1, we doubled the images in that folder and rotated the copies by 180°, creating a new dataset used for training the model.
- We followed a training method which consists in using datasets of increasing complexity: we first trained the model with non-augmented images from the clean dataset, then from the same dataset but using augmentation, afterwards with augmented images from the dataset that contains double the images (but rotated) of species 1, and ending with the class weights to balance the importance of different types of plants.

Thanks to these little modifications, we reached an accuracy of 88% in both training and validation, and 86% as submission result, which was great considering we only exploited Transfer Learning.

For the Fine Tuning phase, we tested different combinations of optimizers, layers to be trainable, learning rate and datasets to use ending up with:

- Adam as optimizer, as it worked better with small updates.
- all the layers trainable.
- 1e-4 as learning rate (although we reduced it to 1e-5 for the final couple of epochs).
- dataset with the rotated images of species 1 and class weights.

With this hyperparameters configuration, after only 30 epochs we reached an incredible 98% training accuracy and 96% validation accuracy. So we trained 3 different models with these same settings and all of them achieved over 92% in submission accuracy with a high score of 93.2%.

3 Final Phase

We decided to submit all of the 3 models mentioned above, and they maintained more or less the same results of the first phase, all of them gaining around 92% accuracy. One thing to note is that the model which received the highest score in the first phase is the one with the lowest score in the second phase, meaning that they were very similar and had different scores only depending on how much they were prone to overfit the submission test set.

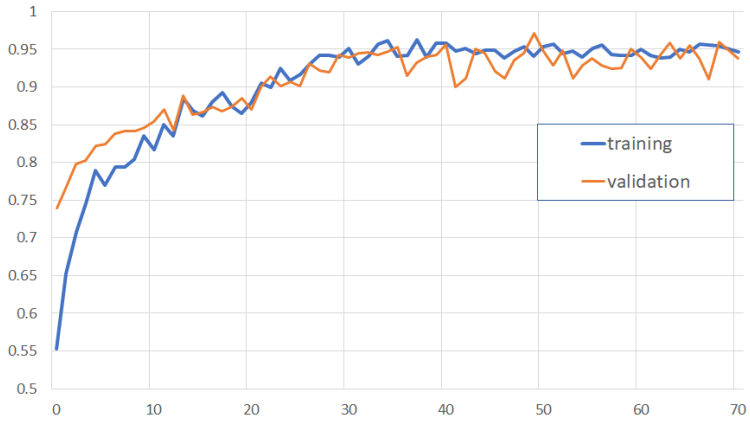


Figure 5: Accuracy of the best model.

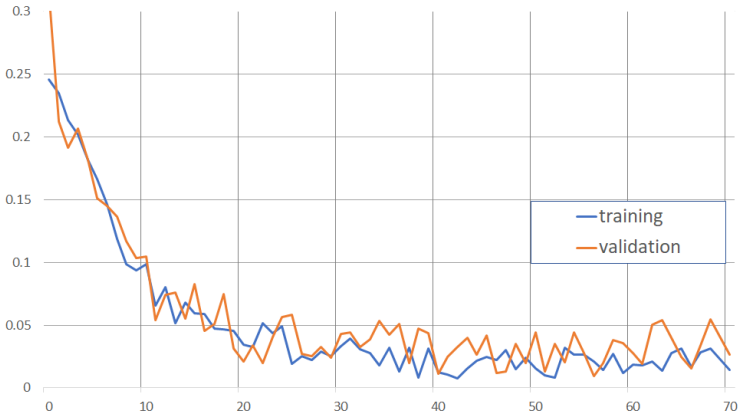


Figure 6: Loss of the best model.

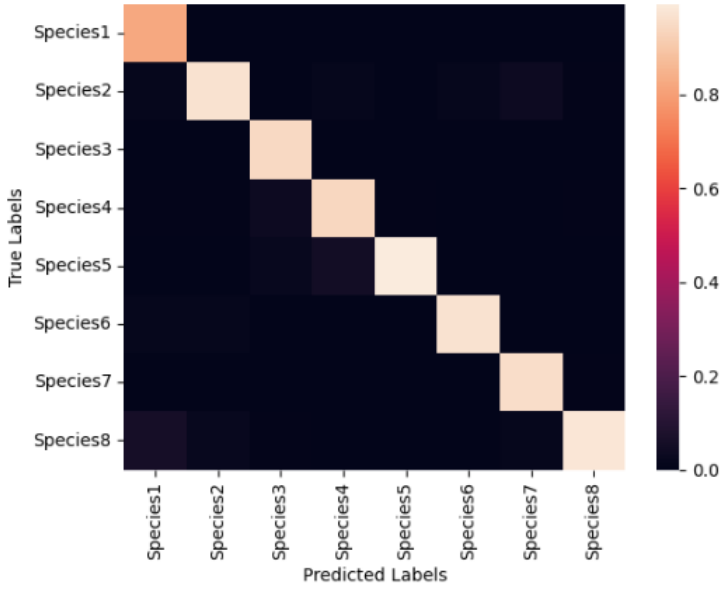


Figure 7: Confusion matrix of the best model