POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

# Artificial Neural Networks and Deep Learning 2022 - First Homework

LAUREA MAGISTRALE IN COMPUTER SCIENCE & ENGINEERING AND MATHEMATICAL ENGINEERING

**Authors: Leonardo Gori, Lorenc Leci, Marco Rizzo** **Team: LooneyFine-tunes** **Academic year: 2022-2023**

## 1. Abstract

This report includes the resume of the attempts that have been conducted during the development of a supervised learning-based model for the classification of plant species.

## 2. Dataset

The development task is the image classification of plants which are divided into 8 species. The given dataset contains 3542 images of size $96 \times 96$ in JPG format. As reported in Table 1, the dataset is characterized by imbalance which is represented by Species 1 and 6 containing less than half the samples of the other classes.

| Class | # Samples |
|---|---|
| Species1 | 186 |
| Species2 | 532 |
| Species3 | 515 |
| Species4 | 511 |
| Species5 | 531 |
| Species6 | 222 |
| Species7 | 537 |
| Species8 | 508 |

Table 1: Number of images per class.

In order to test the generalization ability of the later models, we decided to split the dataset in training and validation (80% / 20%).

## 3. Data pipeline

In this section we present the procedures that we have made to preprocess the data and to overcome the unbalanced dataset.

**Data augmentation** In our development, we tested many data augmentation techniques at dataset-level. As a first attempt we compared the application (at dataset level) of oversampling and intra-class cutmix [5] to the minority classes. For both of them, we additionally applied simple transformations such as rotations, horizontal/vertical flips and shifts through Tensorflow APIs. Furthermore, we attempted a different approach by enlarging the original dataset to 1000 samples per class by additionally applying to the aforementioned simple transformations, new filtering tecnhiques such as brightness and contrast changes, blurring, grid and optical distortions through the Albumentation APIs. As a final test case, we used Tensorflow APIs at code-level in order to define a data generator which performed inter-class cutmix [4] with random percentage to the images of the training set. From our personal standpoint, we noticed no improvements, probably since the augmentation was randomly applied to all the classes, thus maintaining the unbalance.
At the end, all these strategies gave us lower accuracy w.r.t. the original dataset with augmentation only at code-level.

In conclusion, the final Data Augmentation Parameters values are the following ones:

| Transformation | Value |
|---|---|
| rescale | 1/255. |
| shear_range | 0.2 |
| rotation_range | 45 |
| height_shift_range | 50 |
| width_shift_range | 50 |
| zoom_range | 0.3 |
| horizontal_flip | True |
| vertical_flip | True |
| fill_mode | reflect |

Table 2: ImageDataGenerator setting values

**Preprocessing** As we will describe in the next section, we analyzed the performances of application of transfer learning on famous models taken from the literature. Depending on the model used, we applied different preprocessing techniques. In the case of EfficientNet we applied image resizing to $224 \times 224$ and performed no image normalization, keeping the input in the range $[0, 255]$, as explained in Keras documen-

tation. Instead in the case of Xception we resized the image firstly to $256 \times 256$ and secondly to $299 \times 299$. Additionally, we applied image normalization both in the range $[0,1]$ and $[-1,1]$. Unexpectedly, as opposed to the Keras documentation, the combination of sizes $256 \times 256$ and normalization in the range $[0,1]$ got us better results. For what concerns the image resizing, our explanation is that since the original size of the images is $96 \times 96$, resizing them to wider sizes means blurring the original information by adding fictitious one, and leading to complexities in training phase. In this sense, the lower is the image resizing, the better is the capacity of the network to learn the correct features.

## 4. Neural Network

In transfer learning, we firstly used the pre-trained ImageNet weights of VGG19 and we changed the top layers in order to solve our classification problem. Therefore, our initial neural network was characterized by the following final layers:

1. Convolutional 2D Layer with 128 neurons, kernel size (3,3) and padding ("same"), with ReLu Activation Function.
2. A BatchNormalization Layer followed by a Maxpooling2D, Dropout of rate 0.2 and a Flatten Layer.
3. A Dense Layer of 256 neurons, kernel initializer of GlorotUniform and ReLU activation function.
4. A BatchNormalization Layer followed by a Dropout Layer of rate equal to 0.2.
5. A final Dense Layer with 8 units, activation function softmax and kernel initializer GlorotUniform.

In order to compile the model we used the CategoricalCrossentropy as the loss function of our model, the Adam optimizers with learning rate $1e^{-3}$ and as metrics the "val-accuracy".
We trained the model on 50 epochs using EarlyStopping with patience set to 15 with the aim of maximizing the validation accuracy. This leads us to an accuracy of 0.72 on our validation set and 0.63 in the test set.
Since this was our starting point, we attempted to modify the structure of the network by substituting the base model (VGG19) in order to improve the performance.
Keeping the aforementioned final layers untouched, we tried different architectures provided by Keras Application, such as VGG16, GoogleNet, EfficientNetB0, EfficientNetV2B0, EfficientNetV2L, EfficientNetB7, Inception V3 and Xception. The successful outcome was given by Xception which reached an accuracy of 0.7236 on our validation set and 0.6659 on the test set. Having state that, we focused on improving our neural network based on Xception's architecture.
We attempted to modify our custom Neural Network by adding some Dense Layers, BatchNormalization Layer and Flatten Layer. We also added a Gaussian Noise Layer to mitigate the overfitting. However, all of these attempts failed to give us a better result.
In order to deal with the overfitting, we also tried weighting the loss function based on the number of samples present in the dataset, through the definition of two different weighting vectors $w_i$. They assign individualized penalties to each class, preferentially placing larger weights on minority classes:

1. Through the heuristic defined by [1]:

$$w_i = \frac{N_{max}}{N_i}$$

   where $N_i$ denotes the number of images for the $i$-th class and $N_{max}$ represents the maximum number of images among plant classes.

2. By making use of the "balanced" heuristic implemented trough scikit-learn APIs inside Tensorflow, through a balance vector $w_i$, defined as:

$$w_i = \frac{N}{CN_i}$$

   where $N$ is the total number of samples, $C = 8$ is the number of classes and $N_i$ represents the number of samples for the $i$-th class.

However, the aforementioned approaches were not as successful as we expected them to to be.
We tried to use Support Vector Machine and Quasi-SVM as classifiers but these have not improved the performances of the model.
We attempt to use different Optimizers such as Adam, SGD and RMSdrop. The best Optimizer for our problem was Adam Optimizer with the default learning rate ($1e^{-3}$).
Finally, in order to improve our result we tried to tune the Hyperparameters changing the number of units of the Dense Layer, the rate of the Dropout Layers and the learning rate of the Adam Optimizer. We tested the results based on 15 combinations of such hyperparameters, which gave us worse perfomance than the one of our best model so far.

## 5. Fine Tuning

After having used transfer learning, we focused on fine tuning our model. We tried to unfreeze lay-

ers at different depths and to use different optimizers (Adam, SGD, RMSprop) with several learning rates ($1e^{-3}, 1e^{-4}, 3e^{-4}, 1e^{-5}$). We freezed 23 layers starting from the input and we found out that the best Optimizer for our problem was Adam Optimizer with learning rate set to $1e^{-4}$.

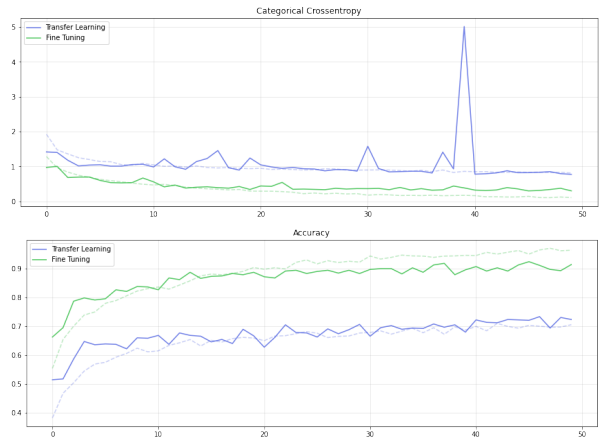


Figure 1: Plots of the Categorical Crossentropy and Accuracy of the TransferLearning (blue line) and FineTuning models (green line).

## 6. Final Model

Ultimately our final model has the following structure:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| xception (Functional) | (None, 8, 8, 2048) | 20861480 |
| conv2d_4 (Conv2D) | (None, 8, 8, 128) | 2359424 |
| activation (Activation) | activation_2 (Activation) | 0 |
| batch_normalization_4 (BatchNormalization) | (None, 8, 8, 128) | 512 |
| max_pooling2d (MaxPooling2D) | (None, 4, 4, 128) | 0 |
| dropout (Dropout) | dropout_2 (Dropout) | 0 |
| flatten (Flatten) | (None, 2048) | 0 |
| dense (Dense) | (None, 256) | 524544 |
| activation_1 (Activation) | (None, 256) | 0 |
| batch_normalization_5 (BatchNormalization) | (None, 256) | 1024 |
| dropout_1 (Dropout) (Dropout) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 8) | 2056 |

| |
|---|
| *Total params: 23,749,040* |
| *Trainable params: 23,503,440* |
| *Non-trainable params: 245,600* |

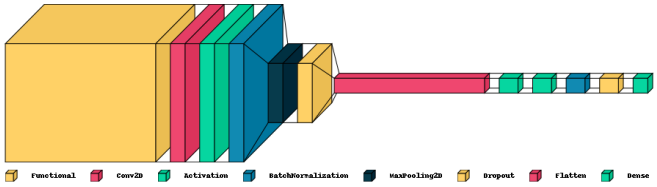


Figure 2: A visual representation of the structure of the model

## 7. Results

Our best model has reached an accuracy of 91.40% on our validation set, of 87.86% in the test set of the first phase of the competition and a final accuracy of 88.51% in the second phase. We can observe the ability of our model to generalize reasonably well considering that the size of the test set of the second phase was 70% of the whole test set w.r.t. the 30% of the phase 1.

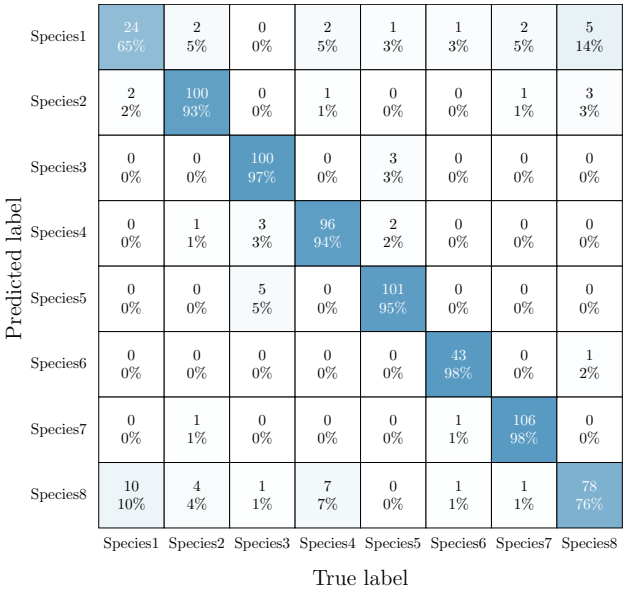


Figure 3: Confusion matrix of the model on the validation set.

## References


[1] Dong Chen, Yuzhen Lu, Zhaojian Li, and Sierra Young. Performance evaluation of deep transfer learning on multi-class identification of common weed species in cotton production systems. *Computers and Electronics in Agriculture*, 198:107091, 2022.

[2] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2016.

[3] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[4] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[5] Caidan Zhao and Yang Lei. Intra-class cutmix for unbalanced data augmentation. In *2021 13th International Conference on Machine Learning and Computing*, ICMLC 2021, page 246–251, New York, NY, USA, 2021. Association for Computing Machinery.