ANN2DL Homework 1 - Report

Leonardo Breda - MTM - 10695861
Pablo Giaccaglia - CS - 10626428
Alessandro La Conca - CS - 10652084

November 2022

Abstract

In this homework, we were asked to perform an image classification task over a botanical images dataset. Starting from canonical data exploration, we first explored the behavior of different architectures, then we focused on a transfer learning-based approach. Our code can be found on this notebook.

1 Data Loading and Exploration

The given dataset is composed of 3541 RGB images divided into 8 folders, one for each plant species. Images are in `.jpg` format and have the dimension of 96×96 pixels.

The first thing we can notice is that the first and sixth classes have nearly half the data with respect to the other species, respectively 186 and 222 samples against an average of 522 for the other species.

In order to create a folder structure suitable for the Keras data loader, we selected all the image files and then divided them, by stratification, into 3 folders (train, test, and validation), each with 8 sub-folders, one for each plant species.

From this point, we used the function `image_dataset_from_directory` provided by Keras in order to feed the data to the model. This also takes care of correctly importing the labels associated with each sample.

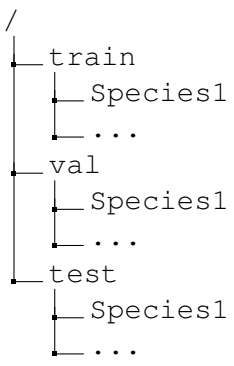

Figure 2: Sample image from species 1

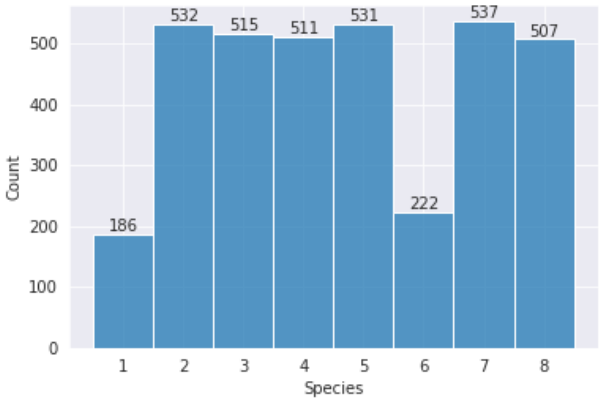

Figure 1: Classes distribution

Before the actual learning part of the model we added Keras' Pre-processing and Data Augmentation layers. We first resized the image with a resolution of 224×224 because we observed better model performance.Then we applied shifts, reflections, rotations, zooms, cut-outs, contrast and brightness augmentations to the training set. These transformations were chosen because they preserve the original image labels. We decided to perform data augmentation due to the scarcity of images and to improve the robustness of the network. For instance, we have noticed the presence of huge shadows in some images, for that, we thought about using cut-out augmentation.

2 First attempts

First CNN: Our first attempt to tackle the problem was a simple CNN from scratch. The CNN is composed of a feature extraction part of three convolutional layers interspersed by three max pooling layers. Then a Flatten, a Dense, a Dropout, and a Softmax Dense layer were placed for the classification task. In this phase no augmentation or resizing was applied to the training data. The training was done using Adam optimizer and Categorical Crossentropy loss. It lasted 32 epochs due to early stopping on validation accuracy. As shown in the plots below, the network reached $\approx 53\%$ testing accuracy, while the training accuracy reached $\approx 90\%$. These are clear signs that the network is quickly overfitting. This CNN contains almost 2.5 million parameters, so it is successfully able to memorize the training set, with scarce success in generalization capability. This is mostly caused by the small training set. Other signs of overfitting are the noisy accuracy and loss plots. The interpretation of this occurrence is that some portion of samples is classified randomly, thus fluctuations are produced since the number of correct random guesses always fluctuates.

Inception-like CNN: The second architecture we implemented, inspired by the Inception network, is mainly composed of 2 "Inception-like" modules, followed by the classification network, constituted by a Global Average Pooling, a Dropout, and a Softmax Dense layer.

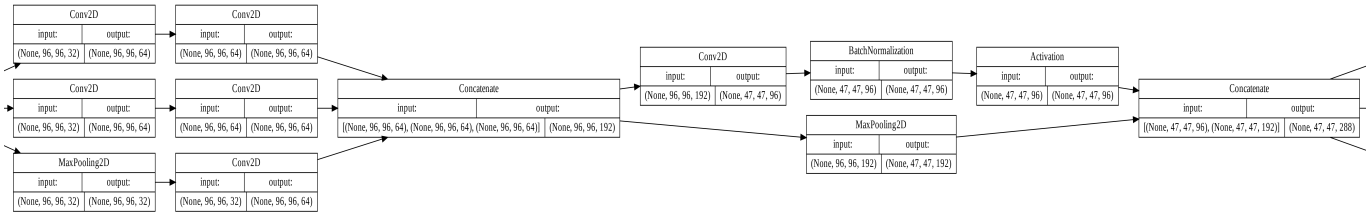

Figure 3: Inception-like module

The factors that improved the performance were several, starting from the deeper structure of the model which allows the training of a bigger amount of parameters. Several convolutions are placed in parallel and the results are concatenated to exploit kernels of different sizes, acting at different resolutions. To lower the computational requirements of this solution, 1x1 convolutions are used to compute filter space dimension reductions before the expensive 3x3 and 5x5 convolutions. Finally, a Global Average Pooling layer was used for both dimensionality reduction and for providing shift invariance to the network, which is something useful for this classification task. This was the first model we implemented leveraging data augmentation but still no resizing was applied. Training was done using Adam optimizer and Categorical Crossentropy loss. It lasted 178 epochs due to early stopping on validation accuracy. The network reached $\approx 90\%$ training accuracy, while the validation accuracy reached $\approx 75\%$.

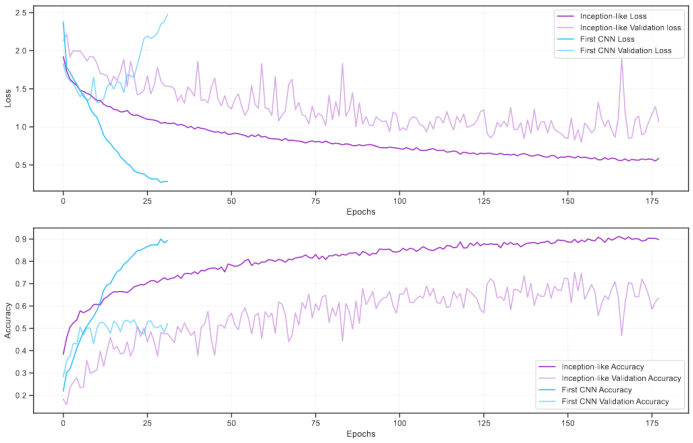

Figure 4: Comparison of Loss and Accuracy between the first model and the inception-like one.

2.1 More into Experiments

In this section, we present some solutions we explored during the development of the final model, that turned out to be sub-optimal.

Imbalanced dataset The given dataset is highly imbalanced with respect to class 1 and class 6. We tried to tackle this problem through 2 approaches. First, we used a weighted Categorical Crossentropy loss to penalize the mistakes in the 2 undersampled classes. Then we changed strategy by employing a Categorical Focal Loss[1], which addresses class imbalance by applying a modulating term to the cross entropy loss in order to focus learning on hard misclassified samples. In both cases we achieved a small increase in accuracy on hard species but a lower overall accuracy. This is probably due to the fact that the difficulty in learning the hard classes' features biases the model weights towards these, worsening the overall performance.

Cutmix With the aim of making the model more robust in predictions we tried to employ Cutmix augmentation, which guides the model to focus on less discriminative parts of the image for having better localization capabilities, resulting in better generalization performance. The approach replaces removed regions of the image with patches from another image. The ground truth labels are also mixed proportionally to the number of pixels of combined images. We observed neither an improvement nor a loss in performance with respect to our base augmentations.

3 Final Model

The final proposed model is based on a transfer learning approach and adds several improvements with respect to the previously shown architectures.

The choice of the backbone network to use was made by testing some Keras Applications. After trying various combinations of backbones, such as EfficientNet and ResNet, and classification networks, we obtained the best performance using ConvNeXtLarge [2]. We loaded the feature extraction part of the model with the weights learned from the ImageNet dataset and experimented with various numbers of layers to freeze during training.

We found the best value for this hyperparameter to be 88 out of 295 backbone layers. This means that we freezed the first 88 layers of the model. The assumption is that the first layers of the backbone were trained on a rich dataset to perform the extraction of general low-level features, thus the weights are already suitable for our classification task and any further training on our data could lead to worse performance. On the other hand, the remaining part of the backbone has a more specialized role in the feature extraction part, so it is more suited to be a starting point for the search for effective weights to extract the most complex features characterizing our dataset.

The CNN is built by stacking our pre-processing and data-augmentation layers on top of the backbone, which is followed by a Global Average Pooling layer for handling the extracted features. Finally, we built the classification section using a Dropout layer, a ReLU dense layer with 1024 neurons and L^1L^2 regularization, and a SoftMax dense layer with 8 neurons for the classification task.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 96, 96, 3)]	0
resizing (Resizing)	(None, 224, 224, 3)	0
random_cutout (RandomCutout)	(None, 224, 224, 3)	0
random_brightness (RandomBrightness)	(None, 224, 224, 3)	0
random_flip (RandomFlip)	(None, 224, 224, 3)	0
random_translation (RandomTranslation)	(None, 224, 224, 3)	0
random_rotation (RandomRotation)	(None, 224, 224, 3)	0
random_contrast (RandomContrast)	(None, 224, 224, 3)	0
convnext_large (Functional)	(None, 7, 7, 1536)	196230336
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1536)	0
dropout (Dropout)	(None, 1536)	0
dense (Dense)	(None, 1024)	1573888
dense_1 (Dense)	(None, 8)	8200

Total params: 197,812,424
Trainable params: 172,709,384
Non-trainable params: 25,103,040

Figure 5: Final model summary

We employed a Cyclical Learning Rate[3] with custom parameter changes according to the epoch. The network was trained for 56 epochs. The first 15 epochs had a minimum learning rate of 10^{-6} and a maximum learning rate of 10^{-5}. Since we saw a slowdown in the training we tried to rise the learning rate to a minimum of 10^{-5} and a maximum of 10^{-4} for 32 epochs. Finally, since we believed that the model was approaching a minimum, the network was trained for the remaining epochs with a minimum learning rate of 10^{-6} and a maximum learning rate of 10^{-5}. The network reached $\approx 99\%$ training accuracy, while the validation accuracy reached $\approx 94\%$. We also use Test Time Augmentation to improve the accuracy of the model by $\approx 0.4\%$. The model makes predictions on flipped images, images rotated by ±10 degrees and shifted by ±3 degrees. The predictions are then averaged to compute the final prediction of the model.

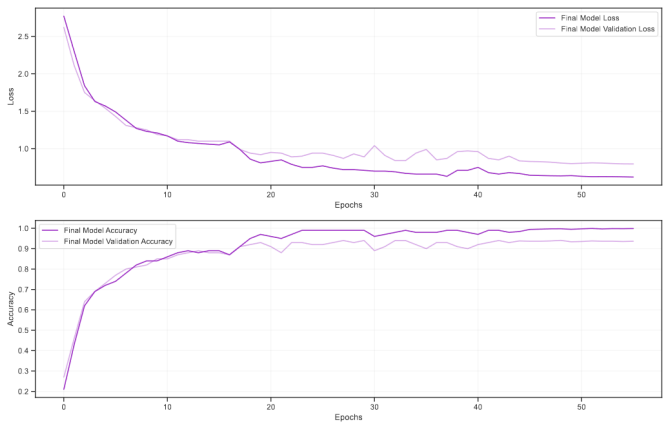

Figure 6: Final Model Loss and Accuracy

References

[1] Tsung-Yi Lin et al. *Focal Loss for Dense Object Detection*.

[2] Zhuang Liu. *A ConvNet for the 2020s*.

[3] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks*.