Challenge 1

Gonzalez-Espitia, Juan Carlos* (10779264)
Petkovic, Nicola† (10821894)
Tonnarelli, Marco‡ (10802888)

2022/2023

CONTENTS

1 INTRODUCTION

The aim of this project is to classify plants belonging to 8 different species, by designing and implementing a Convolutional Neural Network architecture. The provided dataset is made up by 3542 images of 96×96 pixels with RGB colour space: this means that the input shape of the network is $96 \times 96 \times 3$.

In order to properly tackle the problem, the first step we made was to review the literature regarding low-resolution image classification and, in particular, plant images. Plant recognition is still considered an unsolved problem [1], since plants in nature have very similar colours and shapes. Some studies [1, 2] found that the most promising approach is to build networks able to recognise veins patterns on the leaves which, however, in our case is unfeasible since the resolution of the images is so low that these features are not visible. After discarding this option, we found an interesting approach [3] that, although it does not focus on plant recognition, proposes a resolution-aware convolutional neural network for fine-grained recognition with low-resolution images that combines Convolutional Super-Resolution Layers with Convolutional Classification Layers provided by the AlexNet architecture that is, as some studies suggest [2, 3, 4], one of the most promising networks for plant classification problem, together with ResNet and VGG. This study gave us the intuition to resort to transfer learning techniques, by designing a custom-made network on top of some well-known existing architectures such the aforementioned ones, as this study [4] suggests. Our investigation pursued deeper in low-resolution images recognition literature, and we found other interesting studies [5, 6] which validate the idea that one possible solution to the given problem is transfer learning in combination with a custom-made network, which makes use of different layers to achieve a reasonably good performance, in addition to regularisation techniques.

With all the information collected in this preliminary phase, we started to pre-process the dataset and to design the network architecture, trying different approaches.

2 DATA PRE-PROCESSING

After a preliminary inspection of the dataset, we noticed that the 3542 images are not evenly distributed among the 8 plant classes: in particular, most of them have around 500 images each, while classes 1 and 6 have 200 each circa. So, we immediately faced the following problems: data scarcity and data unbalance. These two problems would have negatively affected our model performance, and for this reason we solved them with data augmentation techniques.

First, we addressed the unbalance of data belonging to class 1 and 6 by augmenting the number of samples such that they are around 500. Second, we proceeded to encode the target using one-hot encoding technique and then we split the new balanced dataset in test set (20%) and the remaining 80% in training and validation set (70% and 30%, respectively). Finally, we augmented the training set. In order to do so, we developed a function, that given the data as input, we specify the number of augmentations we want, i.e., ”how much” we want to increase the size of the original training set.

Our data augmentation algorithm is implemented with the help of the ”ImageDataGenerator” library provided by Keras. To achieve an even result, we tuned the parameters of the transformation such that each value is coherent with the other: that is, height and width shift range have the same value (25), horizontal and vertical flip are both active, rotation range has max value of 90 degrees and zoom range varies between .7 and 1.1. Furthermore, fill the shifted images by setting the fill mode to reflect. The reason behind these choices is that, given the low resolution of the images in the dataset, introducing too much distortion to the images could lead to worse performance of the model. For instance, having an image shifted by half the width such that one half is black, damages the training of the model, and thus we fill the other half with the reflection mode.

3 NETWORK DESIGN AND IMPLEMENTATION

In the introductory section of this report, we anticipated how our design process evolved and the choices we made to select the best approach. The first approach was to design a CNN model from scratch: each member of our team built a different model, with different characteristics, in order to explore their capabilities to assess whether or not these models were complex enough to capture the features of the problem. Subsequently, after having demonstrated the inefficiency of these models caused by low

*juancarlos.gonzalez@mail.polimi.it
†nicola.petkovic@mail.polimi.it
‡marco.tonnarelli@mail.polimi.it

complexity, we moved to another setting: transfer learning. We exploited the knowledge acquired by more complex models which were trained on classification problems similar to ours, i.e., image classification.

3.1 Phase 1: Custom-Made Networks

Lee et. al. [1] give a sendible insight on how to properly design from scratch a network, which however was specifically designed to detect venation structure on plan leaves: indeed, the analysis conducted by the research team demonstrated how effective their approach was. Hence, we decided to adapt the network architecture proposed to our specific setting.

One crucial difference between our problem and the one tackled by Ref. [1] resides in the image dimensions, as our are much smaller, and therefore we adapt their network by maintaining the same structure (the simpler one), while reducing the size of the filters and adapting the pooling layers. Overall, the structure of the network is the following one:

- Input layer;
- Convolutional layer, with 96 filters, kernel size (2, 2), strides (1, 1) with ReLU activation function;
- Max Pooling layer, with pool size (2, 2);
- Convolutional layer, with 256 filters, kernel size (1, 1), strides (1, 1) with ReLU activation function;
- Max Pooling layer, with pool size (2, 2);
- Convolutional layer, with 384 filters, kernel size (1, 1), strides (1, 1) with ReLU activation function;
- Convolutional layer, with 384 filters, kernel size (1, 1), strides (1, 1) with ReLU activation function;
- Convolutional layer, with 256 filters, kernel size (1, 1), strides (1, 1) with ReLU activation function;
- Flattening layer;
- Dense layer, with 128 units and ReLU activation function;
- Dropout layer, with 0.5 frequency;
- Output layer, with 8 units and Softmax activation function;

This architecture resulted in poor performance on the validation set, possibly due to the intrinsic differences between both datasets. The authors used the MalayaKew dataset composed of 256×256 images of single leaves on a black background. In our case, the training set contains 96×96 non-uniform images (in terms of content, color, and perspective) of plants in their natural background. Therefore, due to the low resolution and inhomogeneity, we are dealing with a higher level of complexity that the referenced model cannot learn adequately. Indeed, even after augmenting the dataset as discussed in section (4.1), the best result we obtained was around 60% accuracy on the test set.

Since we identified the need for a more complex network capable of learning from noisy low-resolution images, we decided to implement transfer learning on standard CNN architectures on TensorFlow.

3.2 Phase 2: Transfer Learning

Transfer learning is the technique that involves using models (in our case, CNNs) trained on one problem and applying it to a different but related problem. Given the premises in the introductory section, we chose this approach for its flexibility and effectiveness, as the results in the following sections will show. Also, this technique allows us to reduce the training time and the use of computational resources during the learning/inference phases, since much of the parameters are already set, requiring only to train some layers added on top of the pre-trained network. This holds for the preliminary phase of model assessment, since after the selection of the right model (when it converged on the new data), we perform fine tuning phase to adjust the pre-trained weights of the convolutional layers.

For what concerns the choice of the most promising pre-trained models, we implemented proposals from many articles (e.g. [1, 2, 3, 4, 5, 6]) that suggest that the best performing models in low-resolution image classification are ResNet and VGG: in particular, we split our testing by evaluating the performance of ResNet50, ResNet110, VGG16 and VGG19. Overall, this procedure worked nicely and gave us some interesting results, leading to around $65\% - 70\%$ on test set accuracy. However, since the train accuracy was around 60%, we still needed to capture more complexity of the training detaset.

After a review in the literature and in the Tensorflow Applications repository, we implemented ConveNeXt introduced in 2022 by Liu et al [7] that reports an accuracy a top-1 accurace in the ImageNet dataset of around 87%. This model improved considerably the performance, obtaining a precision of around 83% in our test set.

3.3 Phase 3: Fine Tuning

In order to make further improvements, we proceeded to retrain the base model (ResNet and VGG depending on the experiment), by initially unfreezing part of the layers and then all of them, setting a low learning rate. The reason why we decided to first retrain the whole model with just some trainable layers resides in the fact that the most relevant layers should be the first ones (for instance, the first ten) and the latter ones (for instance, the last ten): so, we kept the middle layers non-trainable. Still, the result was slightly better, but not that much we hoped for. In the end, we retrained the whole model, with all the parameters trainable.

Another important remark is that, before retraining the whole model from scratch, we made also some tuning on data augmentation parameters, because initially we introduced so much noise in data that training on the augmented dataset performed worse that the original dataset: the final parameters we chose are the ones described in Section 2.

As an example of structure added on top of a pre-trained model, we report the layers designed on top of ResNet50, which was the best performing one until we moved to ConvNeXtBase: three dense layers composed by, respectively, 256, 256 and 56 units, in combination of two dropouts layers have both a frequency set to 0.1, one before the three dense layers and one right before the output layer.

4 HYPERPARAMETER TUNING

A very important step in the process of training is related to hyper-parameter tuning, i.e. the selection procedure of the different parameters of the pre-processing, and training. This includes, but is not restricted to, the types of data augmentation (rotations, reflections, flipping, illuminations, etc), the learning rate, the dropout rates, the regularisation strengths, etc. In this section we discuss the approach we used for tuning some of the most insightful hyper-parameters for this particular problem. For each model, we followed the same steps for improving their performance, in order to have, at the end of the process, more homogeneous and comparable results.

4.1 Data augmentations

The first step was, as usual, pre-processing data: also in this case, we followed the algorithm provided in Section 2, by iteratively increasing the number of augmentations: we begun with zero augmentations and then we increased the number to 6, 9 12, 14, 16 and 20.

Between one augmentation and the other, we settled the right number of layers to put on top of the pre-trained model. Our approach was simple but yet effective. We added as many dense layers as we could, in order to over-fit data, and the number of nodes in each dense layer was determined by using powers of 2, in some cases multiplied by the number of classes, 8, as some studies suggest. There is no rule to chose this number, but empirical observations prove the effectiveness of this method.

4.2 Learning Rate

The learning rate plays a essential role in finding the minimum of the loss function. A high learning rate might result in poor convergence of the optimization algorithm, while a small learning rate could, besides being computationally expensive, is more prone to get stuck in a local minimum. To set an appropriate learning rate we proceeded by manual exploration. The default learning rate of 10^{-3} led to periodic oscillations in the accuracy of the training set. These oscillations reflect that, with the mentioned step size, the model is not able to reach easily a minimum of the loss function but instead oscillates around its value. To counter this effect we lower the learning rate and obtained good results setting $learning\_rate = 10^{-5}$. The small amplitude of the value can be justified by the scarcity of training samples and the high complexity of the features to learn. Following the same logic, we used a learning rate of 10^{-6} for fine tuning.

4.3 Dropout and Gaussian Noise

Dropout and Gaussian Noise are two fundamental regularisation techniques [8] used in our models. Dropout acts by randomly setting to 0 the inputs of a layer with an assigned frequency, while Gaussian Noise introduces additive random values with 0 mean to all the input of a layer. So, dropout was fundamental to introduce randomness in the network architecture, while Gaussian Noise could be seen as a form of data augmentation and thus both of them help preventing over-fitting. A combination of both these layers was crucial to refine our final model. Depending on where the Dropout layers are placed, their parameters have higher or lower values: if place before a dense layer with few units (less than 100), frequency was set around 0.1, if place before one layer with more units, frequency was set up to 0.4 (with 1024 units). Instead, since Gaussian Noise applies evenly on all the inputs of a layer, we used 0.1/0.2 for the standard deviation regardless of where we placed it.

4.4 Weights Regularisation

In addition to regularisation layers, we also tested the performance of weight penalties on layers' kernel. In particular, we focused on LASSO regularisation since it exploits the L1 norm, which grants the possibility to set some weights to zero (not possible with the L2 norm used by Ridge regularisation). However, since we applied kernel regularisation on top of already regularised model with Dropout and Gaussian Noise, we discarded it: low regularisation factors (10^{-4}) were ineffective, while higher values performed poorly.

5 FINAL MODEL

Considering the observations made in the previous sections, here we describe the architecture of the best-performing model we created.

The model has been obtained by applying transfer learning to the ConvNeXtBase architecture, adding some layers on top of it: overall, the final architecture has around 97.5 millions trainable parameters. In particular, after the flattening layer, the customised part of the network is structured as follows:

- Dropout layer with frequency parameter set to 0.4;
- Gaussian Noise layer with standard deviation parameter set to 0.2;
- Dense layer with 1024 units and ReLU activation function;
- Dropout layer frequency parameter set to 0.3;
- Gaussian Noise layer with standard deviation parameter set to 0.2;
- Dense layer with 512 units and ReLU activation function;
- Dropout layer frequency parameter set to 0.2;
- Gaussian Noise layer with standard deviation parameter set to 0.2;
- Dense layer with 64 units and ReLU activation function;
- Dropout layer frequency parameter set to 0.1;
- Gaussian Noise layer with standard deviation parameter set to 0.1;
- Output layer with 8 units and Softmax activation function.

In order to exploit the full power of the network, it was really important to set the right number of units for the first dense layer and so, after some testing, we assessed that the best number was 1024. The following dense layer has been set to half of the units number of the first one, and the final one to 64 in order to avoid too much over-fitting. Nonetheless, these layers alone introduced a high level of over-fitting and for this reason we added, between one dense layer and the other, a dropout layer. Overall, there are four dropout layers which frequency decreases by 0.1 as they get closer to the output layer: this is due to the fact that it is more effective to have a high dropout frequency when there are more parameters. Finally, to achieve better regularisation we added a Gaussian Noise layer after each dropout, with standard deviation fixed to 0.2 (except for the last layer): this combination of dropout and Gaussian noise helps to introduce randomness in the network and contain over-fitting issues.

The highest accuracy results were obtained by initially training the model with 6 augmentations and 10^{-4} learning rate and then fine tuning with 9 augmentations and $5 \cdot 10^{-6}$ learning rate.

6 PERFORMANCE COMPARISON AND MODEL SELECTION

Here, we compare the results obtained by the different architectures we designed. We mainly focus on the ones developed with the transfer learning technique. We compare the performance of the different models trained with and without data augmentation: in Fig. 1 are reported the values of the best models obtained. The metric we used for the comparison is the accuracy on training set when comparing different models. As we can the augmentation procedure has a positive effect on the final accuracy of all the models: indeed, the technique allows to train the model on more complex features. As we discussed before, the best model is ConvNextBase with augmentation.

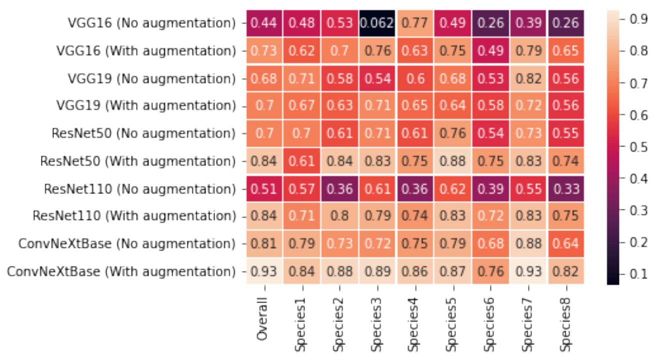

Figure 1. Accuracy scores of the best performing models.

REFERENCES

[1] Sue Han Lee, Chee Seng Chan, Paul Wilkin, and Paolo Remagnino. Deep-plant: Plant identification with convolutional neural networks. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 452–456, 2015.

[2] Guillermo L. Grinblat, Lucas C. Uzal, Mónica G. Larese, and Pablo M. Granitto. Deep learning for plant identification using vein morphological patterns. *Computers and Electronics in Agriculture*, 127:418–424, 2016.

[3] Dingding Cai, Ke Chen, Yanlin Qian, and Joni-Kristian Kämäräinen. Convolutional low-resolution fine-grained classification, 2017.

[4] Mostafa Mehdipour Ghazi, Berrin Yanikoglu, and Erchan Aptoula. Plant identification using deep neural networks via optimization of transfer learning parameters. *Neurocomputing*, 235:228–235, 2017.

[5] David Vint, Matthew Anderson, Yuhao Yang, Christos Ilioudis, Gaetano Di Caterina, and Carmine Clemente. Automatic target recognition for low resolution foliage penetrating sar images using cnns and gans. *Remote Sensing*, 13:596, 02 2021.

[6] Marion Chevalier, Nicolas Thome, Matthieu Cord, Jérôme Fournier, Gilles Henaff, and Elodie Dusch. LOW RESOLUTION CONVOLUTIONAL NEURAL NETWORK FOR AUTOMATIC TARGET RECOGNITION. In *7th International Symposium on Optronics in Defence and Security*, Paris, France, February 2016.

[7] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *CoRR*, abs/2201.03545, 2022.

[8] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.