Report of the ANN’s first homework

Alice Brugnoli, Joy Awad, Mattia Sabella

November 2022

1 Introduction

The objective of this challenge is to classify species of plants which are divided into 8 categories according to the species of the plant to which they belong. Being a classification problem, given an image, the goal is to predict the correct class label between species1 up to species8.

2 Dataset preprocessing

The dataset consists of 3542 images of size 96x96 divided into 8 sub-folders, one for each plant species, respectively.

To test the neural network, first thing did was splitting the dataset into two folders: the training set, with 80% of the data, and the validation set, with the remaining 20%. The splitting is done taking random elements. This way we train the neural network with the training set, to accurately classify the validation set images for each species.

Upon analyzing the data, we have learned that it is highly unbalanced: the folders related to species 1 and 6 have about 40% fewer images available than the other species. To overcome the class imbalance problem, we used class weighting. Class weights allows to give different importance to predictions errors on a per class basis. By increasing the weight of a class having a lower number of samples and decreasing the weight of a class having a higher number of samples, the class imbalance problem is mitigated.

Having a limited number of images to train the model, is problematic. To give more robustness to our model, we aimed at increasing the diversity of the training set for a better classification result.

To do so, we implemented many techniques to evaluate the best:

• **Data Augmentation** which applies random transformations (flip, rotate and shift...) within a certain range to pictures. By applying random transformations new versions of the same image are created. Since we have low-definition images and color is critical for plant species recognition, we ascertained that further modifications (e.g., brightness, zoom in and so on) would make learning more difficult so we sticked with transformations for which the classification should be unaltered. The data augmentation was performed with Keras ImageDataGenerator.

• **MixUp Augmentation** that mixes up the features and their corresponding labels. It combines different features with one another (same happens for the labels too) so that a network does not get overconfident about the relationship between the features and their labels, in order to generalize more. The idea is based on combine two images together with a value alpha that gives weights to the two different pictures $(new_x = alpha * x1 + (1 - alpha) * x2)$, the same concept is applied to the labels. Now, we have created a new virtual dataset.

• **CutOut Augmentation** randomly masks out square regions of the input images during training. This helps the model analyze the image in a holistic fashion by considering more of the image context, rather than focusing always in all relevant features, now it is able to see new patterns or minor features to classify the image.

3 From a simpler model to more complex ones

We used a bottom-up approach starting from simpler models progressively increasing the complexity. First, we built a CNN from scratch, having a combination of 11 convolutional layers, 6 pooling, 3 dense layers, using dropout as regularization technique, with a total of 14 million of parameters. This model is able to learn basic pattern but it has big loss in training and validation set and the accuracy is close to 50/60 percentage. So after we reached a minimum performance, the need for more complex models has been highlighted. We checked out pre-trained classification models provided by Keras Applications that are more suitable for the current task.

4 Models

4.1 Transfer Learning

Our approach was to use transfer learning and fine tuning. We investigated a series of pre-trained models such as **VGG16**, **ResNet50v2**, **EfficientNetV7** and **convNextLarge**, for their good pre-trained accuracies. For all of them, we used their respective image size and preprocessing function, replacing their top classification network. We obtained the best results with ConvnextLarge.

4.2 Hyper-parameter tuning

In order to find the best hyperparameters, we automatized the process of grid search. We used **ReLU** as activation function, **He-uniform** as kernel initializer, and **RectifiedAdam** as optimizer since further research showed that are the most suitable for the task. Indeed, RectifiedAdam let us to obtain a more generalizable deep neural network, completing training in fewer epochs. In addition, we used **KerasTuner**, a general-purpose hyperparameter tuning library, to choose the hyperparameters leading to the best accuracy. Using its **BayesianOptimization** search, we retrieved the best model consisting of 3 dense layers having 512 neurons each and using the value of 1e-5 as starting value for RectifiedAdam. We adopted an **Adaptive Method** to the **learning rate** with the **ReduceLROnPlateau** function that reduces the learning rate when a plateau situation occurs. It worked very well learning more and more.

4.3 Regularization

Chosen the best model with respect to the performance and after the Hyper-parameter tuning process we definitively decided to apply **Early stopping** with patient 15, **Lasso** regularization because we noticed that the model, with other kind of regularizations as Dropout, had difficulties to learn in a range of 200 epocs even data without augmentation, so it was underfitting. Dropout was only applied to the model, together with GaussianNoise, to manage MixUp augmentation. We added also new regularization techniques **Weight Decay** and **Batch Normalization**.

4.4 Classification

For the classification task, we focused on two different classifiers:

- Three dense layers composed by 512 units plus a dense output layer with **softmax** activation function. The loss function used is **CategoricalCrossentropy()**, having a multi-class classification task.
- One dense layer composed by 512 units, three dense layers composed by 256 units, one dense layer composed by 128 and a **Quasi-SVM** on the top of the transformed features with a **RandomFourierFeatures** layer that works as linear layer. In this case we used, as loss function, the **hinge** loss, which lead to a better accuracy as it is known to put the emphasis on the boundary points. The idea to choose these particular layout is based on decreasing more the training and the validation loss during the training.

We noticed that SVM outperformed in particular on the loss and also in the accuracy by a small margin.

5 Fine Tuning

After we reached a good accuracy close to **86** percent in both models already presented in the previous section, we decided to improve the perfomance of our strong classifier applying Fine tuning phase. After loading the models trained during the Transfer Learning phase we freezed the first 240 layers to finally retrain it but this time with other augmentation techniques in order to force the model to learn more. Training strong classifiers is not an easy task! It is easy to do underifitting and make the model no more able to recognize patterns, so it is not able to learn anymore. When we perform fine-tuning is critical to do it only after the model has already reached the convergence on our dataset, in the first phase. It's also critical to use a very low learning rate, because we are training more layers, so many new neurons and we already reached the convergence. Here, we only want to readapt the pre-trained weights in an incremental way. That's why we used a learning rate of 1e-6 with a ReduceOnPlateau option. Furthermore, the BatchNormalization layers contained in the pre-trained model need to be kept frozen. In this way, the BatchNormalization layer will run in inference mode, and will not update its mean and variance statistics. In this phase we adopted two different strategies:

- The first model has been trained with MixUp Augmentation with alpha = 0.1 reducing the overlapping of the two images, because if we put alpha = 0.2 the performance drops.
- The second model has been trained with CutOut Augmentation

6 Ensemble

We noticed that the first and the second models reached an accuracy very similar at the end, close to the 0.90 percent on the validation set. To improve a bit the result we decided to ensemble the two models to produce a more accurate final prediction.

7 Training

The training was performed tracking the accuracy and the loss of the model with cross-validation via holdout. By measuring the validation accuracy and loss, we were able to keep track of potential overfitting over the training data and observe the generalization properties. The training was performed mainly on local hardware and on Google Colab.

8 Results

The following tables show a comparison between the evaluation metrics of the best models we used. We notice that the three models performed almost as good. Also we note that species 1 was classified the worst regardless of the model, with its precision, recall and therefore the F1 score being the lowest. A possible interpretation of this result is that Precision and Recall are the two most common metrics that take into account class imbalance. They are also the foundation of the F1 score (see images in the zip file where results are shown, cutout.jpeg, mixup.jpeg and esemble.jpeg).

The final accuracy and the f1 scores, respectively, of our best model (the ensemble) on the actual test set are shown in the table below (see the image of the result in the leaderboard result.jpeg). They are close to the results we obtained on the validation set, therefore we can say that the testing we did on the validation captured the scenario on the test set.