ANNDL 2022 - I Challenge

Team Name The Convolutional Trio

Group members Andreotti Vittorio 10824694 Buoninfante Erika 10636425 Pizzo Luca 10614734

1 First steps

1.1 Simple Model

The first thing we did was to look at the dataset and what we noticed is that there were just 3542 samples and that two of the species (1 and 6) were underrepresented.

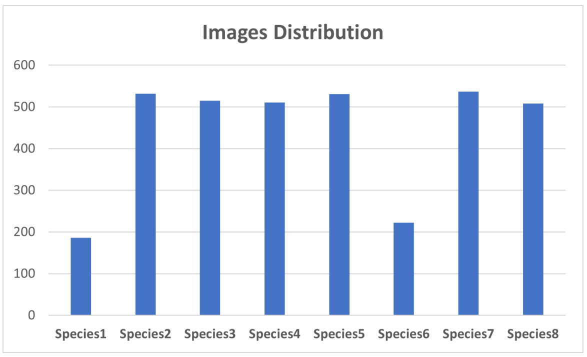

Figure 1: Image distribution before operating on the dataset

With such a small number of samples we decided to select only 10% of each class for the validation set using the parameter `validation_split` of the ImageDataGenerator.
We started by standardizing the images with a rescale of 1/255 to boost the general performance.
The first model's architecture that we built was composed of 4 Convolutional layers (32, 64, 128 and 256 units), each followed by a MaxPooling layer with a downsize of 2. We tried varying the number of convolutional layers and the number of units, but we couldn't improve the model's accuracy, actually it overfitted faster, so we continued with this architecture.
The feature extraction part was followed by a Flattening layer, a single Dense layer with 128 units, a Dropout layer with a rate of 0.3 and a final Output layer with 8 units. With this configuration we reached a validation accuracy of about 50%, not a noteworthy performance especially for the disappointing accuracy on the first species.
At this point we carried on with new configurations and hyperparameter combinations, as well as data augmentation techniques. These actions were performed simultaneously but for clarity they will be covered in separate paragraphs.

1.2 Keras Tuner

Since we were trying to find optimal values of some of the hyperparameters with a manual random research, we thought it would be faster if we could automate this process with an external library. With `keras_tuner` it was possible to try a big variety of combinations of settings and ultimately find the most promising ones. Especially for the optimizer and for the level of regularization to apply (e.g. dropout rate), we were able to try a lot of mixtures of hyperparameters and discard unpromising ones with the HyperBand algorithm.

1.3 Augumentation

1.3.1 Random data augumentation

To increase the size of the dataset, we have exploited different techniques of random data augmentation such as:

- Rotation
- Width/Height shift
- Zoom
- Horizontal/Vertical Flip
- Brightness
- Channel shift

In particular, we initially set the variable `fill_mode` with constant 0, but after several tests, we decided to replace it with the image reflection technique.

1.3.2 Manual naive oversampling

To even the distribution of the species, we manually triplicated and duplicated the samples of the first and sixth species respectively. In this way we expected the network to give more importance to these two classes. We did not consider the method of undersampling since we only had two underrepresented classes.

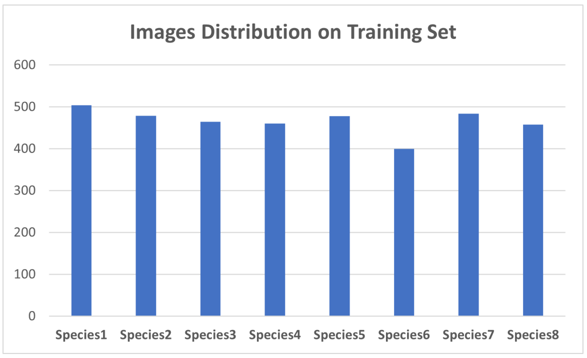

Figure 2: Image distribution on training set after operating on the dataset

Before duplicating the two classes, we manually splitted the dataset in training and validation sets because the resampling should not be applied to the validation set.
Using random data augmentation, together with a more balanced dataset, we obtained slightly better results in validation accuracy.

1.3.3 Weight balance

An alternative approach was to use the parameter `class_weight` in `model.fit()` initialized with the attribute `balanced`, which equated the representation of the whole dataset on all classes. This however did not improve the accuracy of the first species, so we abandoned the method.

1.4 Summary of best model built from scratch

After several experiments we built our final model that achieved around 78% accuracy in the validation set.
The main specifications of this model (model_from_scratch.ipynb) are the following:

- **Kernel size**: 3x3. We thought that with an image size of 96x96, a higher kernel size would have been too big to recognize the elementary features of the leaves.
- **Stride**: 1. With the leaves being close to each other, we didn't want the model to downsample the image too much and miss important information. We increased the value to give it a try but with poor results.
- **Padding**: `same`. We wanted to preserve the information on borders of the images.
- **Dropout rate**: 0.2.
- **Number of Dense layers**: 2, the first with 128 units, the second with 256 units.
- **Weights initializer**: Glorot Uniform/He (depending on the layer).
- **Loss function**: Categorical Cross-Entropy.
- **Optimizer**: Adam with 1e-3 learning rate. We tried several optimizers like SGD and RMSprop with various hyperparameter combinations but it turned out that default Adam performed better.
- **Activation function**: ReLu and Sigmoid.

2 Transfer Learning

2.1 Base Model

With such a tiny dataset it was very difficult to create a high performance model from scratch, so we decided to exploit pretrained models belonging to `keras.applications`.
We thought that choosing the model with the most promising performance in transfer learning and then proceeding with fine-tuning would have been a reasonable thing to do. We tested the following pretrained models:

- VGG (16/19)
- ResNet50
- Xception
- Inception (V3/ResNetV2)
- EfficientNet (B3/B4/V2S)
- ConvNeXt (Tiny/Small/Base)

The best performing models were the ones with the most recent pretrained models like ConvNeXt.

2.2 Fine Tuning

In order to improve the ability of our model to recognize features specific to our dataset, it was necessary to finetune the weights of the pretrained models.
We tried three different approaches for the fine tuning:

- Keep some blocks freezed and other trainable
- All trainable with Differential Learning Rates
- All trainable with the same learning rate

2.2.1 Freeze and Unfreeze

Since VGG19 does not have many convolutional layers, we tried a lot of combinations of freezed and unfreezed layers and we found out that freezing 8 of the layers gave us the best results (around 85% of validation accuracy).

2.2.2 Differential Learning Rates

Layers in the bottom learn generic features like edges and shapes while the middle layers learn specific details related to the dataset on which it is trained.
We didn't want to change the learned weights on the initial layers too much because they are already good at what they are supposed to do. Middle layers have knowledge of more complex features specific to the dataset so we wanted to finetune them a little more. We divided the model into multiple layer groups and set different learning rates for each group. Small learning rate for the first, slightly higher for the middle groups and the highest learning rate for the last one.
For deeper models like EfficientNet, Xception, ConvNeXtSmall and ConvNeXtTiny this method gave us the best performance. The last two in particular reached 88.97% and 87.62% accuracy on test set.

2.2.3 Same learning rate

We also tried fine tuning our models with all layers set to trainable and with the same (small) learning rate and we saw that for some models it performed similarly or even better than with the other two approaches. This was the case with the ConvNeXtBase that reached a 89.56% accuracy on test set (best model).

3 Final Model

The best model that we were able to submit had a GlobalAveragePooling layer and a Dense layer following the ConvNeXtBase, both regularized with the dropout.
In particular we noticed that we were obtaining better accuracy with the GAP layer and the sigmoid activation function in the dense layer.
The optimizer that performed best in the fine tuning phase was Adam with a learning rate of `1e-5`, while keeping all layers set to trainable.
The dataset used is the original one augumented with methods seen in 1.3.1 and 1.3.2.

4 Final comments

The results we obtained are fairly good but more experimental tests are required to refine the model even more and reach better accuracy. It's possible to explore new techniques for data augmentation, especially on the first species, the one that ended up being the main penalizing factor in our performance. We tested methods like CutOut and CutMix but we didn't manage to make them work properly.
For even better results, next works could also focus on further inspecting the last ConvNeXtBase model by fixing the configurations together with additional hyperparameter tuning.