Image Classification Challenge 2022

TEAM: ALPOLIZZATI

Andrea Aiello, 10863133 **Dario Cavalli, 10820532**
Claudio Moriello, 10869147 **Paolo Basso, 10783951**

1 Dataset Specification

The dataset provided for the homework consists of 3542 total images, grouped into folders, of different plants belonging to one among 8 classes. The different species differ heavily from each other, having different sizes and shapes of both leaves and stems, and sometimes even in colour but mainly due to the shooting season.
Not every picture was taken in the same way, in many of them the plants are in the foreground making it easier to distinguish species to the naked eye, others instead were taken from a further distance and in some cases plants are covered by shades.
From a technical point of view, the first challenge is represented by an unbalance between the different classes with Species 1 and 5 severely outnumbered by the others, with 200 samples against over 500 for every other class. This led to an under the classification of said species with F1 scores way below expectations.
A simple solution was to use different weights for each class [1], easily obtained thanks to the $sklearn\ module$ which returns a dictionary used later during training: $\{0 : 2.38, 1 : 0.83, 2 : 0.86, 3 : 0.87, 4 : 0.83, 5 : 2.0, 6 : 0.82, 7 : 0.87\}$

2 Data Augmentation

An essential step, especially with such a relatively small dataset, is data augmentation which was implemented in two ways:

- **Data Augmentation Layer**: A specific set of augmentations (Flip, Rotation, Zoom) was inserted as a layer in the model, just after the input one. In this way every batch during training was subject to random transformations, further reducing overfitting and in order to avoid enlarging the training dataset beforehand.
- **CutMix & MixUp**: Despite the use of the layer above, results were not outstanding on the validation set while the model was overfitting on the training set. This led to the addition of a further, and stronger image augmentation, that was performed on the entire dataset. Thanks to the $keras\_cv\ module$ we implemented $CutMix()$ [7] and $MixUp()$ [8] on the entire dataset, randomly transforming each sample with one or the other strategy.

3 Experiments

In this section, we want to list some experiments that we did but ended up not using in the final model.

3.1 CNN Model

The first approach to the challenge was to design a tailor-made CNN model as seen during lectures. Starting with only two convolutional layers we obtained underwhelming results on the classification

task, then obtained slight improvements at first with hyperparameter tuning and later with the addition of more convolutional layers to the network.
These experiments were run during the first days of the competition, without a real comparison to the other models developed by other teams. Once we noticed a substantial gap in terms of accuracy we decided to move on to different techniques such as using a pre-trained supernet with $weight\ initialization$ and $fine\ tuning$.

3.2 VGG & EfficientNetV2

The approach to supernets started with two increasingly bigger models seen during lectures: VGG19[4] and EffNet[6]. The former one brought a slight increase in accuracy so it was quickly replaced with the latter which was on par with other teams' performance, until the decision to use ConvNeXt.

3.3 Models Ensemble

Another strategy that we tried, but we ended up not using in our final models, is one of ensembling models. In particular, we tried the "Integrated Stacking Model" technique where multiple networks are run in parallel and their results concatenated for another fully connected classification network [5]. We tried ensembling different famous architectures, favouring smaller models.

This technique was very fruitful for most of the models we tried but not for the final, most performant, one so we ended up not using it in our final submission.

4 Final model

4.1 Hyperparameters

- *VALIDATION_SPLIT*: we split the dataset with a ratio of 0.125, in order to have 3100 images for training and 442 images for validation.
- *BATCH_SIZE*: we used mini-batch gradient descent with a batch size of 64 samples.
- *MAX_EPOCHS*: we used a maximum of 200 epochs both for the first and the second training, but due to early stopping this maximum was never reached.
- *EARLY_STOPPING_PATIENCE*: we used a patience of 19 epochs in the first training and one of 23 epochs in the second. We noticed that the greater the second patience, the better the performance, being careful not to overdo it to avoid overfitting.
- *OPTIMIZER*: we used Adam Optimization because it is faster, requires less memory and achieves good performances.
- *LEARNING_RATE*: in the first training we started from the default value of the Adam optimizer, which is 1e-3 (0.001). In the second training, we used a much lower learning rate, that is 5.2e-5 (0.000052), because we want our model to adapt itself to the new dataset in small steps to avoid losing the features it has already learned.
- *LOSS*: considering that it is a multi-class classification problem, we used Categorical Cross Entropy in order to have a probability over all the classes for each image.
- *UNFREEZE*: we unfreeze 670 layers, so we unfreeze the whole model, with the exception of batch normalization layers.

4.2 ConvNeXtLarge

To achieve our best results we used the Transfer Learning technique with the pre-trained model: ConvNeXtLarge. It is a ConvNet which borrows some ideas from Transformers. [3] We used this model by initializing it with the imagenet weights.

4.3 Model Layers

We built a Sequential model by passing a list of layers to the Sequential constructor. The first layer is the *Input* layer, to instantiate a Keras tensor. The second layer is the *DataAugmentation*, which is used to apply the transformations mentioned above. The third is the *model_supernet* layer, to use the ConvNeXtLarge pre-trained model. The *Flatten* layer is used to flatten the input, but without affecting the batch size. A first *Dropout* layer with a rate of 0.39 is used to reduce overfitting. Then, we used two *Dense* layers with 1024 and 512 units respectively; in both layers we used a ”relu” activation function, and a ”He uniform” variance scaling initializer. Furthermore, in order to limit overfitting, we constrain the model through *L1L2* regularization. Another *Dropout* layer to further reduce overfitting. Finally, the last *Dense* layer with 8 units (that is the number of classes), a ”softmax” activation function and a ”Glorot uniform” initializer.

4.4 Model training

Even though we used the weights of the imagenet dataset only as initialization and we retrained the whole model we still followed a two-step process because it yielded better results: first we trained only the classification head, keeping the supernet frozen, and then we retrained the whole model unfreezing all the layers (with the exception of batch normalization layers).

4.5 Test-Time Augmentation

To further improve our model accuracy, we perform a self-ensemble technique called Test Time Augmentation [2]. This technique involves creating multiple augmented copies of each image in the test set, having the model make a prediction for each, and then returning an ensemble of those predictions. We define the final prediction by aggregating the posterior vectors, keeping the class with the biggest sum of the prediction.

In order to get the optimal performance, we tried various sets of augmentations, and the best results were reached by using 6 images: the original image, the image flipped on the horizontal axis and 4 images, shifted of 10% (9 pixels) of the image dimensions, for each of the 4 directions.

5 Comparison between models

To conclude, we compare the results obtained by the best models, for each technique used. The validation accuracy is based on the split of the dataset in train and validation set, using as validation a percentage of the dataset varying from 10% to 20%. The accuracy on the training set reached 99% in almost any case, apart from ConvNeXtLarge, in which it stabilized at 90%. The test was made using TTA (4.5) and only for those models reaching a high validation accuracy.

Model	Validation Accuracy (%)	Test Accuracy (%)
ConvNeXtLarge	95.93	95.48
Custom CNN	70.58	\
VGG19	75.28	\
EfficientNetV2	91.56	88.41
Ensemble	95.48	93.85

Table 1: Validation and Test accuracy comparison

References

[1] *Classification on imbalanced data*. https://www.tensorflow.org/tutorials/structured_data/imbalanced_data. Accessed: 2022-11-28.

[2] *How to Use Test-Time Augmentation to Make Better Predictions*. https://machinelearningmastery.com/how-to-use-test-time-augmentation-to-improve-model-performance-for-image-classification/. Accessed: 2022-11-28.

[3] Zhuang Liu et al. *A ConvNet for the 2020s*. 2022. DOI: 10.48550/ARXIV.2201.03545. URL: https://arxiv.org/abs/2201.03545.

[4] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[5] *Stacking Ensemble Machine Learning With Python*. https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/. Accessed: 2022-11-28.

[6] Mingxing Tan and Quoc V. Le. "EfficientNetV2: Smaller Models and Faster Training". In: (2021). DOI: 10.48550/ARXIV.2104.00298. URL: https://arxiv.org/abs/2104.00298.

[7] Sangdoo Yun et al. *CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features*. 2019. DOI: 10.48550/ARXIV.1905.04899. URL: https://arxiv.org/abs/1905.04899.

[8] Hongyi Zhang et al. *mixup: Beyond Empirical Risk Minimization*. 2017. DOI: 10.48550/ARXIV.1710.09412. URL: https://arxiv.org/abs/1710.09412.