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 size and shape of both leaves and stem, and sometimes even in color but mainly due to the shooting season.
Not every picture was taken in the same way, in many of them the plants are in 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, 200 samples against over 500 for every other class. This led to an under 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 that 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] or $MixUp()$ [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 models seen during lectures: VGG and EffNet. In the specific experiments were performed using VGG19, which resulted in [4] [6]

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].

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, that 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 Transfer Learning technique with the pre-trained model: ConvNeXtLarge. This model is pretty recent; it was invented in the 2020s when the Google's Transformers began to overtake ConvNets as the favored choice for generic computer vision problems. It is a pure ConvNet which borrows some ideas of Transformers. [3] The ConvNeXtLarge model was first pre-trained on the ImageNet-21k dataset and then fine-tuned on the ImageNet-1k dataset, so we used it 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 firs layer is the *Input()* layer, to instantiate a Keras tensor. The second layer is the *DataAugmentation*, it's used apply the transformations mentioned above. The third is the *model_supernet* layer, to use the ConvNeXtLarh

4.4 Test-Time Augmentation

To further improve our model accuracy, we perform a self-ensemble technique called Test Time Augmentation [2].

We apply different augmentations to each test image, classify every augmented image, and 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 one, we tried various sets of augmentations, both random and with fixed modifications. The best results were reached by predicting 6 different images: the original image, the image flipped on the horizontal axis and 4 images, shifted of 10% (9 pixels) of the image dimensions, one for each of the 4 directions.

5 Comparison between models

To conclude, we compare here 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 the last one 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 only for those models reaching a high validation accuracy and using TTA 4.4.

	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 between models

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.