First AN2DL Challenge Report

Francesco Ferrari, Roberto Cialini, Luca Bresciani, Francesco Pastori

INTRODUCTION

The task of the challenge was to classify a dataset containing images of healthy and unhealthy plants. The images format was 96 x 96 pixels in RGB. After a first inspection of the dataset we rapidly noticed how some of the images represented **outliers**, so we proceeded to clean the dataset (saved in clean_dataset.npz[1]) that we then used as a basis for all the following experiments.

FIRST EXPERIMENTS

In order to split the dataset into balanced train, validation and test sets, we used the train_test_split function included in the sklearn.model_selection package. At first we used 70% of the dataset for training, 20% for evaluation and 10% for testing.
We started by experimenting with custom sequential models created with Keras: we created a LeNet Model[2] obtaining 0,77 on validation accuracy and 0,67 on test accuracy when submitted on the hidden test set. So we decided to make some tests with **transfer learning and fine tuning** techniques. In the beginning we used the MobileNetV2 network, with a classification head that consisted in a 2 neurons dense layer. We rapidly started to obtain better results on hidden test set accuracy, around 0,72.

KERAS APPLICATIONS - EfficientNet

After the previous good results we tried other keras applications: the first one that significantly improved our results was the EfficientNet network family. In particular, we experimented with the models that offered the best top-1 and top-5 accuracy[3], such as EfficientNetB7, EfficientNetV2S and EfficientNetV2M. In order to evaluate these different models, we tested them with the same classification head and applied the same training to each of them. We decided to use early stopping on the validation accuracy to reduce the overfitting and we applied fine tuning starting from the 200th layer of the networks (since all of them had the same layer structure). We obtained the best results with the EfficientNetV2M network[4], which let us reach 0,8 accuracy on the hidden test set.

DATASET CLEANING

After removing the outliers from the original dataset, we found that some images had multiple copies. To avoid overfitting the model by submitting the same image multiple times we removed all **duplicates** by calculating a hash for each image and only retaining a copy of each.

AUGMENTATION TECHNIQUES

When we noticed that the dataset was **unbalanced**, we tried different methods to balance it, to try and further improve the classification accuracy:

- Undersampling: we cut the number of images of healthy plants down to the same number of unhealthy plants, by randomly selecting the images to remove. We quickly noticed this did not improve classification accuracy, instead it worsened it significantly;

- Class-weights: we tried giving a higher training weight to the under-represented unhealthy plants, to try and balance the results. This only led to a minimal improvement on the classification accuracy and we ultimately decided to drop this method to focus on the following one;
- Oversampling: we created an augmented dataset by duplicating samples from the unhealthy images, until we had balanced the dataset. This left us with 3060 samples of each class. We further augmented the dataset after some successful experimentation with the keras_cv library: specifically, we first duplicated each image in the dataset, then applied the RandAugment and the RandomCropAndResize preprocessing layers to it. We ended up with an augmented balanced dataset of 12240 samples

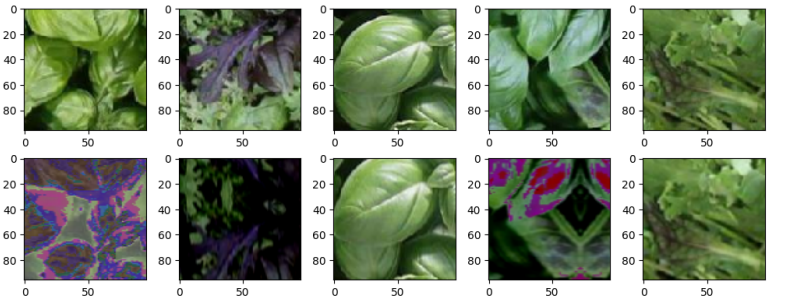


Fig.: Top row: unprocessed samples. Bottom row: samples processed with keras_cv layers.

PREPROCESSING

Since the available dataset contained a small number of samples, we decided to include some **preprocessing layers** in our models. To do this we utilized the preprocessing layers provided by the tensorflow.keras.layers library:

- Random contrast with range from -40% to +40%
- Random brightness with range from -40% to +40%
- Random flip
- Random rotation with range from -30% to +30%
- Random crop with a crop size of (20, 20) pixels
- A final resizing layer with size of (96, 96) pixels

Also we used ‘reflect’ as fill mode to avoid the generation of artifacts in the processed images.

AVOIDING OVERFITTING

In order to avoid overfitting during training, we used an **EarlyStopping** callback function monitoring the validation accuracy, with various patience values ranging from 20 to 100 epochs, along a batch size of 64 and a large number of maximum epochs. In this way we did not limit the training stop to a determined epoch, but we let the EarlyStopping function manage it, to then restore the best weights.
While utilizing the EfficientNetV2M network, we began to experiment with different combinations of dense layers and tried to reduce overfitting by applying ridge regression and dropout. We ended up choosing the dropout technique with a **dropout** rate of 0,5. In particular the best results were obtained using the following layers schema. This network[3] reached a hidden test set accuracy of 0,85.

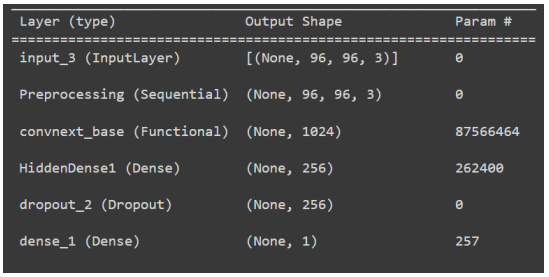
Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 96, 96, 3)]	0
Preprocessing (Sequential)	(None, 96, 96, 3)	0
efficientnetv2-m (Functional)	(None, 1280)	53150388
HiddenDense1 (Dense)	(None, 256)	327936
HiddenActivation1 (Activation)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
HiddenDense2 (Dense)	(None, 256)	65792
HiddenActivation2 (Activation)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense (Dense)	(None, 2)	514

Fig.: Summary of the EfficientNetV2 model

KERAS APPLICATIONS - ConvNeXt

After these initial tests, we tried to use the ConvNeXt networks: ConvNeXtTiny, ConvNeXtSmall, ConvNeXtBase and ConvNeXtLarge. Due to the complexity of the classification task the best results were obtained by using these last two networks that were the ones that had the largest number of parameters .
In particular we noticed that with the ConvNextBase model the performance after the fine tuning phase was better than the ConvNeXtLarge, due to the fact that the ConvNeXtLarge model was overfitting in the fine tuning phase because it was too much complex. We obtained an accuracy of 0.92 on the hidden test set by using the following model[5], in which we also reduced the hidden dense layers from 2 to 1, to avoid overfitting in the final stage of classification.

In particular we decided to use a learning rate of 0,01 for the transfer learning, with a batch size of 64, while for the fine tuning process we opted for a lower learning rate of 0,0001. This choice was made to avoid overfitting during fine tuning, along the use of a ReduceLROnPlateau callback function, monitoring the validation accuracy.



Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 96, 96, 3)]	0
Preprocessing (Sequential)	(None, 96, 96, 3)	0
convnext_base (Functional)	(None, 1024)	87566464
HiddenDense1 (Dense)	(None, 256)	262400
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	257

Fig.: Summary of the ConvNeXtBase model

FINAL SUBMISSION

In the final stage of the competition the accuracy dropped from 0,92 to 0,839, using the ConvNeXtBase model[5]. This result was probably due to the fact that the dataset was still not big enough to generalize on the bigger hidden test set on the second stage of the competition. Under this assumption, it was at this moment that we decided to apply oversampling by duplicating the images in the clean_dataset_no_duplicates.npz dataset, reaching 12k samples[8], and applying on all of them RandAugment and RandomCropAndResize from the keras_cv library. Instead of applying the more complex preprocess used till this point, we only applied rotation and flip, since the other transformations were already applied to the whole dataset. Due to the complexity and the size of this new dataset we decided to use ConvNeXt Large instead of the base version, after the training we submitted the model[6] and obtained 0.871 on accuracy.

IMPROVEMENTS

First idea to improve performance was to move the 'RandAugment' layer directly inside the preprocessing, we tried this solution but the 'RandAugment' layer does not turn off during inference, like the other preprocessing layers, so the results are fallacious.

Second idea was to use ensemble learning. Once the best model[6] was obtained, we decided to apply the ensemble technique to further improve performance. This technique allows us to put several models together and use the average of their predictions as the final result. The models we combined are 3 versions of the best model trained on three different datasets[8], all augmented with keras_cv. Creating new datasets is easy because using randAugment we get a very different sample each time.
The model created by ensemble[7] has a higher probability of predicting correctly because when the prediction of one sub-model is around 0.5 the predictions of the other two sub-models affect the mean and possibly lead the model toward the correct prediction. The outcome of this attempt is still unknown at the time of the writing of this report due to the congestion of the challenge servers and the approaching deadline.

REFERENCES

[1] - clean_dataset.npz in the shared folder (link) - notebook name: “Chall1 DataCleanUp.ipynb”
[1] - clean_dataset_no_duplicates.npz in the shared folder (link) - notebook name: “Chall1 Duplicates.ipynb”
[2] - LeNet Model - notebook name: “Chall1 LeNetModel.ipynb”
[3] - reported on https://keras.io/api/applications/
[4] - EfficientNetV2M Model - notebook name: “Chall1 EffNetV2MDropout 0.85.ipynb”
[5] - ConvNeXtBase Model - notebook name: “Chall1 ConvNextBase 0.92.ipynb”
[6] - ConvNeXtLarge Model - notebook name: “Chall1 RandAugmentDatasetConvL 0.871.ipynb”
[7] - Ensemble Model - notebook name: “Chall1 Ensamble.ipynb”
[8] - augmented_dataset.npz in the shared folder (link) - notebook name: ”Chall1 Augmentation.ipynb”

Shared folder link: https://drive.google.com/drive/folders/15943FDBzUek0tvhIq2myXwmo8shUhCIY

CONTRIBUTIONS

Every member of the group actively participated in every stage of the competition, both in researching new techniques and applying them in practice. In particular:
- Francesco Ferrari made the cleaning of the dataset removing the outliers and the duplicates.
- Luca Bresciani made the first augmented set with a balanced number of healthy and unhealthy images.
- Francesco Pastori improved this code using as augmentation RandAugment and the RandomCropAndResize as described before.
- Roberto Cialini experimented with different keras applications such as EfficientNet and ConvNeXt for transfer learning and fine tuning.