Challenge 1 - Image Classification

TEAM: NEUROMASTERS

Viola Federico, 10635132
Galetti Filippo, 10625164
Zanotto Luca, 10583439
Selis Riccardo, 10707433

Data-set Analysis and Processing

Anomalies and Outliers

The first significant observation we made was the presence of anomalies within the dataset, represented by a pair of duplicate outlier. We immediately deemed it crucial for our work to eliminate these outliers. Additionally, after conducting further checks, we had to remove duplicates of some photos of the same plants, which were hindering the model's ability to generalize on new data tests. After this initial data cleaning operation, our dataset was considered free from all anomalies and ready for training our model.

Unbalanced Data

A more in-depth analysis of the dataset and its division into classes revealed an inequality in the quantities of images associated with the two labels. The data is indeed unbalanced: there is an unequal distribution between the *"healthy"* and *"unhealthy"* classes. We immediately understood that this could lead to poorer performance of the classification model, especially on the less represented class, as the model could become overly adapted to the predominant class. To overcome this problem, we experimented with various techniques.

- **Down-Sampling and Up-Sampling:** The first technique we implemented to enhance performance was *down-sampling*, which reduced the number of samples in the overrepresented *'healthy'* class. However, we quickly realized the benefits of the opposite approach, *up-sampling*. This involved generating randomly modified duplicates of the *'unhealthy'* images through operations such as horizontal and vertical flipping and image rotations. This not only balanced the number of *'healthy'* and *'unhealthy'* images but also maintained the diversity of the dataset.
- **Mix-Up:** We also tried the *Mix-Up* method on the *'unhealthy'* images to increase their number by creating new *'unhealthy'* images resulting from the combination of two other images from the same category. However, this approach did not lead to an improvement in accuracy.
- **Class-Weighting:** One of the strategies we tried for data preparation for training was to keep the dataset unbalanced among the labels and assign greater weight to the less represented classes. However, this procedure did not prove to be better than the *up-sampling* technique.

Pre-Processing Layer of the Net

Some other operations on Data are necessary before training a Binary Classifier Neural Network:

- **Splitting Data-Set:** Before the training phase, the dataset was divided into three subsets: *training, validation,* and *test*. We ensured a balanced distribution of labels and allocated a larger portion of data to the *training set*. Testing on a separate set is crucial for model inference. However, once it was verified that a particular model performed well on the internal *test-set*, before uploading the network to Codalab, the network was retrained on the dataset integrating the *test-set*, in order to have a larger amount of data on which to train the model.

- **Data Augmentation:** In order to augment the data to train the model, a *Pre-Processing Layer* was inserted in every Architecture. This layer generates randomly modified duplicates of Training-Set images through operations such as flipping, zoom rotation, translation and alteration of brightness or contrast. After testing various combination of these techiques, we chose to implement flipping, zoom and rotation since we found these to be the most effective with our models.
- **Test-Time Augmentation:** We tried this technique but it didn't improve the accuracy of our models.
- **K-Fold Cross Validation:** We tried *k-fold cross validation* to find the number of epochs that best suited the training during transfer learning, then used the entire dataset as the training set. Once submitted, the values worsened, probably due to overfitting.
- **Resizing Images for Differents Architectures:** In some of the models we used, it was necessary to insert an additional pre-processing layer to resize the initial size of the images from $[96, 96, 3]$ to the one required by the architecture. We tried increasing the size of the image by adding costant value paddings or by mirroring the image towards the borders. These kind of techinques were less effective than applying a layer of resizing interpolation to the input layer in the model architecture, so we stick with the latter method.

Experiments

Common Technique Used

- **Preventing Over-Fitting:** To prevent of overfitting, techniques such as *Early Stopping* and *Dropout* have been tested. These techniques have been applied from the beginning in all models and have been refined and adapted according to the architecture and its results. In the more advanced stages of the project, we used the *Reduce Learning Rate On Plateau* technique. This technique involves reducing the *learning rate* when the monitored metric stops improving. This approach can help to optimize the learning process, particularly in situations where progress seems to have stagnated.
- **Transfer Learning and Fine-Tuning:** We decided to rely on Fine Tuning and Transfer Learning techniques, leveraging the features learned from large models such as *ImageNet*. To make the model specific for our binary classification task, we used the *Fine-Tuning* technique, which required numerous attempts to understand how to maximize the yield of progressive unfreezing. This approach allowed us to adapt a pre-trained model to our specific task, despite the constraints given by the size of our dataset.

Architectures

1. **MobileNetV2:** *MobileNetV2* was among the first models we tried since it is very lightweight and efficient. This allowed us to experiment with the dataset thanks to their fast training times. With this model, we tried different types of the aforementioned preprocessing and based on these results, we drew up guidelines to follow that were useful for all the architectures used.
2. **VGG16:** *VGG16* was the other model we used at the beginning since its behaviour was well known from the laboratories. Since the results of our experimentations with this model weren't too impressive we decided to switch to more performing models.
3. **ConvNeXtLarge:** The best results we obtained from this model came from Fine-Tuning the last three stages (around 300 layers) and adding multiple dense and dropout layers to gradually reduce the output to the binary classification task. The model achieved very high accuracy on the training set without overfitting.

4. **ResNet152V2:** Another architecture tested in the preliminary phase is a convolutional neural network model for transfer learning using *ResNet50V2* as the base. In the implementation of the architecture ResNet50V2, we applied Global Average Pooling to reduce dimensionality and a dropout layer was added. A dense layer with 1024 units and Leaky ReLU activation was added for classification, while the last layer is dense with one unit and sigmoid activation. In this case as well, we tested the fine-tuning technique and once again it increased the performance of our model. After that, we moved on to try other architectures from the ResNet class, and *ResNet15V2* was the one that gave us the best results overall.

5. **EfficientNet B0:** *EfficientNet B0* is the smallest and most efficient model in terms of computational resources and we thought it would be best to focus our attention primarily on using this model. Subsequent tests were also conducted with versions *B1, B3,* and *B7,* but these did not lead to significant improvements. The model weights were loaded with those pre-trained on *ImageNet* and the layers of *EfficientNetB0* were set as non-trainable. Since this network gave us good results from the initial tests, we attempted to increase its accuracy in several ways. We refined the *Fine-Tuning* by unfreezing the first two blocks found by inspecting the layers of the efficient net, rather than unfreezing an arbitrary number of layers as we did before. We set a lower *learning rate* value to pass to the Adam optimizer. In addition, we changed the metric monitored for early stopping from *validation accuracy* to *validation loss*, and also implemented the technique of reducing the *learning rate* when the monitored metric stops improving.

Ensembling

The technique of ensembling involves using multiple different models to obtain a single prediction. The basic idea is that by combining the strengths and mitigating the weaknesses of various models, a broader and more robust knowledge base can be formed, which is then used to make more accurate predictions.
We experimented with two different techinques to compute the output coming from the ensembled models. The first procedure consisted in concatenating the results from each models to obtain a single *tensor* of predictions that passes trough a Dense Layer with *sigmoid* activation. The second approach used the *max-voting* method which starting from an odd number of predictions (three in our case), implements a majority mechanism to obtain the classification. Various tests have been carried out with different assembled architectures, gradually refined through the addition of various *Dense-Layers* and *Drop-Out Layers*.
The models combined in our ensembles were our best models: EfficientNet B0, Resnet152V2 and ConvNeXtLarge. Our final model was an ensemble between EfficientNet B0 and ConvNeXtLarge using concatenation on the output prediction.

In the following table are listed the best accuracy values that we obtained for each architecture, one in the development test-set from codalab (1) and the other on final test-set.

Models	MobileNetV2	VGG16	ConvNeXtLarge	ResNet152V2	EfficientNet B0	Best Ensemble
Acc. (1)	0.72	0.69	0.89	0.75	0.88	0.92
Acc. (2)	-	-	0.83	-	0.78	0.84

Contributions

- **Zanotto Luca:** Resnet512v2, efficientNetB0 to B7 transfer learning and fine tuning, callbacks and learning rate, preprocessing, kfold cross validation, Test time augmentation.
- **Viola Federico:** experiments on VGG16, ResNet, Xception, EfficientNet, data processing, transfer learning and fine tuning, data set and mispredictions analysis, ensembling.
- **Galetti Filippo:** MobilenetV2, augmentation techniques sperimentations, convNeXt, fine tuning techniques, report, dataset cleaning and sampling.
- **Selis Riccardo:** Experiments on Data-set with various Models, Mobile-Net, VGG16, fine-tuning, ensembling, report.