AN2DL Challenge Report

Matteo Rigamonti, Emanuele Marzorati, Ilenia Marciano, Emanuele Arpino

1. Introduction to the Challenge

The challenge was to build a network able to distinguish healthy from unhealthy leaves belonging to a dataset of 5200 images of size 96*96*3. The adopted approach was to start from a baseline model and incrementally increase the complexity of the model by incorporating more and more established techniques, well-known architectures and number of parameters and by refining the built models.

2. Preprocessing Routines

2.1. Visualization, Data Cleaning and Dataset Splitting

Firstly, in the attempt to feed to the network the cleanest possible dataset, all the images and the labels have been plotted and looked at in order to discover hidden outliers. Two categories of outliers have been found and promptly removed. The data was divided in a 50/25/25 split for training, validation and test set. To avoid having a hold out error based on the specific split we resorted to k-fold cross validation with k=10. To further improve training, class balancing was carried out, however it turned out to be quite irrelevant since the two classes were already sufficiently balanced.

2.2. Hyperparameter Tuning

Regarding the setting of *learning rate, batch size, the number of layers, the number of neurons in each layer, dropout rate* etc. a baseline model was established to see the impact that tuning would have on performance. A range of feasible values was selected, then the training for different combinations was carried out. Eventually the model that performed better determined the final hyperparameters.

To avoid overfitting some measures were adopted. An initial Learning Rate was set which, however, might have been suited only for early stages of training. It is, indeed, too high during the training of later epochs, as it leads to overshooting the minimum of the error function. The solution here is to let it slowly decrease as learning stops to progress with the increase of the number of epochs. Early stopping was included as it avoids overfitting by monitoring performance on the validation set and ending the training when performance plateaus or deteriorates. In addition to this, dropout layers were introduced to further prevent overfitting.

The chosen Loss Function was categorical cross-entropy, that compares the predicted probability distribution to the true binary distribution. Moreover, trying to minimize it corresponds to maximizing the likelihood of the model's prediction to be true. Also, it penalizes more heavily the predictions that are confidently wrong, making it particularly sensitive to the correctness of the predicted probabilities. AdamW was adopted as the optimizer, as it presents the additional feature of L2 norm weight regularization which penalizes large weights in the model, consisting of another measure against overfitting.

2.3. Data Augmentation

Since the dataset wasn't particularly large and knowing that Deep Learning models thrive on large datasets, a common practice is to implement Data Augmentation techniques into the pipeline. Another advantage of using the aforementioned method is its effectiveness against overfitting. Utilizing Keras built-in layers, various geometric and

photometric transformations, including flipping, scaling, cropping, rotation, and adjustments to brightness and contrast, were implemented. A key aspect was that the labels remain preserved by the augmentation techniques.

In an effort to further expand the dataset, an attempt at MixUp augmentation was carried out by hand-crafting the function implementing the transformation to both the images and labels. Plotting the images confirmed that the transformation was correctly performed, however, when the augmented dataset was fed to the network, it experienced slowed down learning. For the project purposes the other kinds of augmentations were sufficient, therefore the decision to leave out MixUp augmentation was made.

With an iterative process, the combination of data augmentation layers and tuned hyperparameters guaranteeing the best accuracy was implemented. For the majority of the final models the chosen values of hyperparameters and data augmentation layers were same as shown in Table 1 and 2.

Hyperparameter	Value
Batch size	32
Epochs	1000
Learning rate	1e-4
Weight decay	5e-4
ES patience	20

Table 1: Tuned Hyperparameters

Data Augmentation Layer	Value
Random Contrast	-
Random Zoom	0.2
Random Rotation	0.2
Random Crop	-
Zero Padding 2x2	-
Horizontal Random Flip	-

Table 2: Chosen Augmentation Techniques

3. Investigated Architectures

The initial approach was to get acquainted with the problem by building a few models by scratch, to also get a hang of the challenges that lower lever coding entails. The first architectures drew inspiration from LeNet, followed by VGG, which proved to be too simple for the intended challenge, in fact the models struggled to learn. It was observed that improving the complexity became necessary as the current ones didn't have enough parameters to learn the most relevant characteristics of the data. The hypothesis is that not having enough layers and filters leads to a small receptive field. This can hinder learning of the model since it means that each neuron is only sensitive to a small local region of the input capturing only low-level features.
Also, by adding a Global Average Pooling (GAP) layer, to introduce regularization and reduce the number of weights, there was a rise in performance. The deeper LeNet model and VGG architectures exhibited enhanced performance, achieving an accuracy of around 0.75.
An attempt to manually introduce skip connection was then made, allowing information to bypass one or more intermediate layers. This is known to help with vanishing gradient issue; this too contributed to a rise in performance. The next step was to try out a self-supervised approach, by employing metric learning through Siamese Neural Networks. The idea was to have a self-supervised pre-training phase using a contrastive loss with positive and negative pairs of images, belonging either to the same class or to two different ones. This way, the model learns to distinguish between similar and dissimilar samples without directly using class labels. After this, the pre-trained Feature Extraction Network (FEN) is used to build a classifier to distinguish the two classes. This approach did not work since pairs of images of the same class could be very different and trying to learn a similarity did not make sense.

4. Best Performing Architectures

4.1. Transfer Learning

However, the realization that this wasn't the most efficient way to proceed quickly led to the investigation of transfer learning networks. This alternative approach guaranteed much better results in terms of performance, albeit at the cost of model complexity and weight.
MobileNet was experimented with first, not bringing exciting results, followed by *EfficientNet* which allowed to reach the turning point in terms of accuracy that surpassed 80% for the first time. Following this success, the ConvNeXt family was employed which, as expected due to increased complexity, led to, by far, the best results in terms of accuracy, reaching around 90%.
Since these pretrained networks need to be adapted to the task at hand, a very simple fully connected classifier was attached to the feature extraction network. It had a Dense layer of 512 neurons and ReLu as activation function followed by a dropout layer with rate 0.5 and then a final Dense layer of 2 neurons representing the 2 classes and SoftMax as activation function, as it converts raw scores into probability distributions. It is also well matched with Categorical Cross-Entropy to measure the difference between predicted and actual probabilities.
Despite its simplicity it served its purposes and worked well in symbiosis with the FEN without adding unnecessary complexity, as the dense layers are prone to be very computationally heavy.

4.2. Ensemble Architecture

Given the high variance of the proposed models that leads to different predictions for each image, it can be interesting not to choose a single best model but to choose a handful of models letting them all have their own prediction, then vote and decide the final label accordingly. To improve even further, each prediction can be weighted either by the model overall accuracy or by penalizing unsure predictions. This can be done by assigning a weight to each prediciton with the formu

$$w = \ (\mathrm{P}[0] - P[1])^2$$

Where P[0] and P[1] are respectively the probabilities of the image belonging to class 0 or class 1. The score is computed for each prediction and if the model is significantly convinced of the class of a sample e.g. P[0] = 0.9and $P[1] = 0.1$ its vote the score will be higher, with respect to that of an unsure model e.g. $P[0] = 0.55$ and $P[1] = 0.45$
Eventually the votes of every model are averaged together to get the final class prediction.
The proposed **ensemble** is composed of models obtained through transfer learning starting in particular with the established networks: *efficientnetv2, convnet_tiny, convnet_small, convnet_base, convnet_large, convnet_Xlarge*.
By using this approach, the highest accuracy on the test set of was achieved as it is shown in Figure 1. This model has a 0.84 accuracy on the final test set.

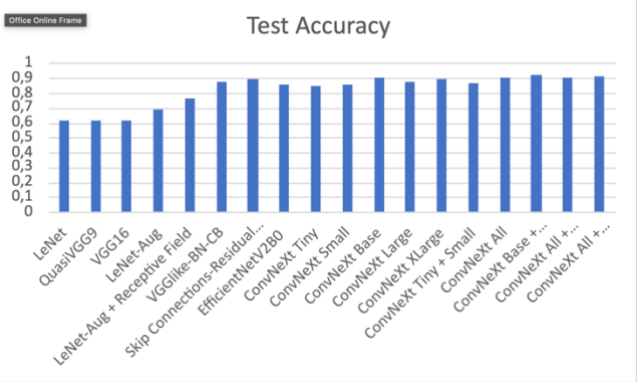

Figure 1: plot of the accuracy of every model.

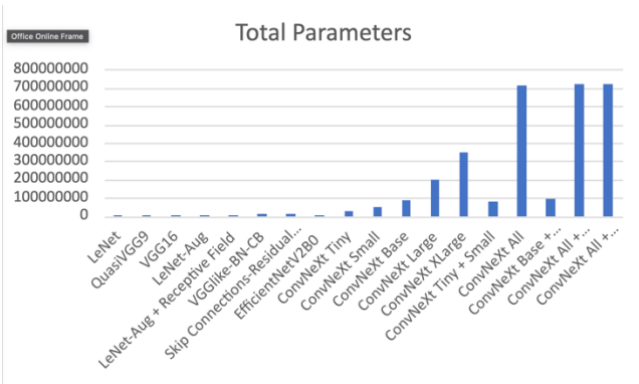

Figure 2: plot of the number of parameters of every model

Contributions

- Matteo Rigamonti: Transfer Learning, Siamese NN, Ensemble Models
- Ilenia Marciano: Transfer Learning and Augmentation
- Emanuele Marzorati: Transfer Learning and Ensemble Models
- Emanuele Arpino: Built from scratch architectures, Report

We all helped one another with the problems that came up during the challenge and contributed to some extent to the topics of other members.