POLITECNICO MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE

# First project report - ANN2DL

**Group:** broccAlly

**Members:** Niccolò Bindi, Matteo Forlivesi, Lorenzo Franzè, Alessandro Lupatini

## 1. Introduction

This report details our process of model development for a plant health classification task, using a dataset of around 5000 images. Our modeling approach involved experimenting with different strategies to identify the most effective one. One slice of the group constructed a convolutional neural network (CNN) from scratch, emphasizing both the feature extraction network (FEN) and the dense layers network. Simultaneously, the remainder of the group employed transfer learning, utilizing a pre-trained model for the FEN. These parallel approaches were developed collaboratively, with ongoing information exchange, shared notes and a final phase working on the same final model that performed best. The report provides a succinct overview of our data preparation, model development, and evaluation processes, shedding light on the challenges and successes encountered throughout the project.

## 2. Data preparation

To harness the provided preprocessed data effectively, two essential tasks were undertaken. Firstly, it was imperative to transform the values corresponding to the data labels. Specifically, the brightness values within the three RGB channels were rescaled from the original range of 0 to 255 to a normalized range of 0 to 1. Secondly, a conversion of the labels was conducted from a categorical format, denoting "healthy" and "unhealthy," to binary encoding. This entailed assigning 0 to represent "healthy" and 1 to signify "unhealthy." Thirdly, the data needed to be partitioned into training, validation, and test sets, utilizing a division ratio of 50% for training, 25% for validation, and 25% for the test set, as demonstrated in the class. To ensure the preservation of label distribution integrity, the "stratify" option was applied during the partitioning process. This safeguarded the consistency of label ratios across the different sets.

### 2.1. Outlier removal

Studying the dataset we found out that are a big amount of outliers (a conspicuous set of “Shrek” images and a set of “Trololol” images), so (since are identical) we use a for loop and compare with an inequality constraints if there are identical to the 2 images (chosen manually from the dataset). The cleaned dataset was saved as “new_public_data.npz”.

### 2.2. Label encoding

After converting the labels “healthy” and “unhealthy” to 0 or 1 respectively we followed two approaches:

- Maintain the encoding 0/1 and have as output shape of the network one single neuron representing the probabilities to belong to the predicted class
- convert the previous encoding to One-Hot encoding, in this case the shape output of the network was 2 neurons.

We noticed that there was no difference in Accuracy in using either one encoding or the other, for the second approach it was needed only at the end to convert the result in 0/1 It’s important to mention that in both the cases the results from the network were probabilities so we had to convert to classes 0/1 usually using threshold 0.5. [2]

## 3. Image Augmentation

We adopted three distinct approaches for image augmentation, employing both online and offline methods. [7]

First, the advantages of the online approach lay in its simplicity of setup and enhanced training performance, despite its increased demand for resources. Implementation involved the straightforward addition of a preprocessing layer after the network input. Our experimentation with various transformations revealed that flipping the image and rotating it up to 90 degrees, along with subtle adjustments to brightness and contrast, were the most effective techniques.

Conversely, we observed that cropping, zooming, and translating did not contribute positively to the network's learning process. These transformations appeared to yield more false positives, possibly due to the exclusion of essential features necessary for recognizing an unhealthy plant from the image. Therefore, we determined that such transformations were not conducive to the network's ability to discern crucial patterns and features in the data.

Second, the offline approach was used sometimes during the data preparation phase and it was useful to increase the available data samples, in particular for the unhealthy class, since the dataset was a bit unbalanced. Both the healthy and unhealthy images were augmented. We paid attention in increasing the samples only on the training set since the not augmented images in validation and test had to be predicted correctly, in this way the network could train on slightly different images but predicting for validation and testing on the correct images. The same considerations for the online method are applicable here.

In the last days we also tried also to try the mixup data augmentation: it consists of generating a new image with mixing two default ones (in our case labeled differently) and summing up pixel by pixel weighted by a lambda factor bounded [0, 1] (which determines also the new label). Unfortunately, it didn't improve the overall performances.

## 4. Models without transfer learning

Our exploration of possible architectures started from simple CNNs like the ones implemented in Le-Net 5 and progressively added layers and features to improve performance.

The model (1), following preprocessing and online image augmentation, consists of a sequence of convolutional blocks repeated four times. Every block consists of two stacked convolutions, one with a unitary kernel used as a bottleneck [3] and another one with a larger kernel succeeding. The number of filters in each block increases by a power of two,in the first block the filters are 8 whilst in the fourth one they are 64.

Max Pooling layers follow each convolutional block, except for the last block, which employs global average pooling across the final 64 filters. Subsequent to the convolutional layers, three tiers of fully connected neurons were incorporated. The architecture culminates with a single output neuron activated by the sigmoid function to align with binary encoding. Leaky ReLU activation functions were applied to all layers. This decision was taken seeing that the training often got stuck at the beginning signaling that probably the gradient was getting reduced by a significant factor because of the depth of the network. To mitigate overfitting, each block includes a batch normalization layer, and L2 regularization (lambda = 1e-5) was employed on all convolutions, serving as a regularization measure to promote generalization and prevent excessive fitting to the training data.The optimizer of choice was Adam [8] that overall showed the best performance compared to SGD and SGD with a scheduled learning rate. Other experiments using the optimizer AdaW showed similar performance to Adam on a similar version of this network that didn't feature any regularizers, probably because regularization was built in the optimizer itself.

The training process employed a substantial batch size of approximately 400 samples [4] and spanned around 300 epochs. To address overfitting, an early stopping procedure was implemented. However, setting a high patience value of around 40 was essential, as the network struggled to initiate learning and tended to stall at initialization values with lower patience settings.

Despite the promising appearance of the model, with local test set results indicating accuracy surpassing 85%, the final evaluation yielded a significantly lower accuracy. We hypothesize that the model's depth and complexity may have been excessive given the limited number of images available. Additionally, the training may have continued for too many epochs, potentially leading to overfitting, even with the incorporation of batch normalization and regularizers. For these reasons, during the latter part of the research, our emphasis shifted towards the exploration of transfer learning models.

## 5. Models with transfer learning

While we were trying the "No Transfer Learning" approach, we immediately started to work to find a good Transfer Learning network.
The first thought was to start with smaller networks and slowly scale up to the bigger ones: the main concern was (since our images are much smaller than those of imagenet) to avoid "possible" overfitting or local minima due to the complexity of the most powerful networks.
We started (without fine tuning) to work with the "EfficientNetV2" class, since they have a very high performance on the imagenet without being too large (as for example "ConvNeXt"). [1] The first we tried was V2B0 (the simplest one, with pooling average always) and we attached a dense layer (8 units), a dropout at 0.5 (arbitrarily chosen, since various fonts say opposite things about the probability number [9]) and the output layer. Even with this small setup we had already good performances on our test set (84.42% of accuracy), meanwhile in the submit test set we obtained 78%. [1] After trying to affine it with adding another dense layer (32 units) , we noted a small improvement of 0.4% on our test dataset (almost negligible), so it was discarded. After cleaning the dataset, the first model obtained 87.43% on our test dataset and on the site the 80%.
After we applied the fine tuning (freezing only 15% of the pretrained net) and we reached 88.22%, but on the submission dataset we didn't see more improvements. Subsequently, we decided to apply a batch normalization layer between the 2 dense layers and after the dropout. [5] We improved the performance on the submission dataset at 83% also thanks to the offline image augmentation discussed before.
Since we found that we didn't improve so much, we changed the pretrained net with V2B3 ("EfficentNet") but we didn't improve, instead it seems that the network needed much more data to avoid overfitting and more complex dense layers. In fact, applying the "EfficientNetV2M" (bigger network than to V2B3) we encounter an odd behavior: in the first part of the training the performances were low (78-80%) but with the Fine Tuning we arrived at 90-91% of accuracy on our test set. Unfortunately, on the submission we obtained 82%, even if we had better performances on our side.
In the last days of the development phase, we focused only on another class of pretrained networks: ConvNeXt, in our case we used ConvNeXt Base. To improve the performances, we tried to raise the complexity of the dense layer part and experimented with the batch size value: we succeeded obtaining again 83% on submission test, with 4 dense layer (each one with dropout and batchnorm) and a batch size of 1024 (a very odd situation). We tried also with ConvNeXt Large, but (differently to ConvNext Base) we cannot perform a fine tuning on the majority of the network due to the GPU RAM limit imposed by Google Colab, so we didn't improve the performance so much. [2]

[1] At this stage we didn't clean the dataset.

## 6. Evaluations on Test set

A final evaluation on the test set is performed locally, the results are used to visualize the confusion matrix and the scores: accuracy, precision, recall and F1. At the beginning of the challenge the confusion matrix showed us that the model mispredicted a lot of unhealthy data probably because with the unbalanced dataset the model couldn't learn the unhealthy images well. With data augmentation the results improved. Moreover, as trial, in order to use all the available data, after the model showed a good result we merged the training and validation sets using the test set as new validation data, in order to perform a final training on more data before submitting.

## 7. Conclusions

The ultimate model that emerged as the most effective employed the paradigm of transfer learning. This superiority is attributable to the model's ability to harness the substantial volume of data employed during training in expansive neural networks, thereby mitigating the prevalent issue of overfitting. Notably, the convolutional neural network (CNN) constructed de novo encountered significant challenges related to overfitting, underscoring the efficacy of the transfer learning approach in ameliorating this impediment.The results obtained are decent, with the best models performing with an accuracy of 80% on the test sets. Still, improvements can be made to increase the accuracy even further. The limitations we have in computational capacity played a role in fine tuning big models.

[2] We found that changing the random seed completely changes the submit performance of the Network, even if we apply the data augmentation, so it was difficult to establish what is "better" and "worse" in our decision process. For most of the time we used a seed produced by the clock (so each time different). [6]

## 8. Contributions

Forlivesi Matteo : worked on the CNN architectures and prepared the data ingestion pipeline.
Franzè Lorenzo : worked on the CNN and moved onto the transfer learning approach comparing approaches in labeling the data.
Lupatini Alessandro : worked on transfer learning and fine tuning and removed the outliers.
Bindi Niccolò : worked on transfer learning and fine tuning.

## References

[1] https://keras.io/api/applications/.

[2] https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in python/.

[3] https://machinelearningmastery.com/introduction-to-1x1-convolutions-to-reduce-the-complexity-of-convolutional-neural networks/.

[4] https://medium.com/data-science-365/determining-the-right-batch-size-for-a-neural-network-to-get-better-and-faster-results 7a8662830f15.

[5] https://stackoverflow.com/questions/39691902/ordering-of-batch-normalization-and dropout.

[6] https://towardsdatascience.com/is-a-small-dataset-risky b664b8569a21.

[7] https://www.analyticsvidhya.com/blog/2021/06/offline-data-augmentation-for-multiple images/.

[8] https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in keras/.

[9] https://www.quora.com/What-is-a-good-value-for-the-dropout-rate-in-deep-learning networks.