Binary Image Classification: Health state of plants

Artificial Neural Networks and Deep Learning – A.Y. 2023/2024

Roberto Giannini Braghè*, Niccolò Grillo†, Filippo Lipari‡ and Andrea Toccaceli§
M.Sc. Mathematical Engineering, Politecnico di Milano - Milan, Italy
*Email: *roberto.giannini@mail.polimi.it, †niccolo.grillo@mail.polimi.it, ‡filippo.lipari@mail.polimi.it , §andrea.toccaceli@mail.polimi.it*
*Student ID: *10705638, †10900895, ‡10630163, §10913674*
Codalab Group: 'Deep4getting'
GitHub code: https://github.com/Morph1c/Deep4getting1

1. Abstract

The given dataset consisted of 5200 images of leaves belonging to 2 different classes: Healthy and Unhealthy. In this report, we go through the steps and intuitions that allowed us to build the final classifiers. Starting from a baseline, custom-made CNN model we followed an incremental approach, adding more and more complexity to the process, and assessed the results. In particular, we noticed the data augmentation to be the most crucial and delicate part of the project. Transfer learning with fine-tuning and ensemble average of the pre-trained models produced the best results in terms of accuracy. We also implemented methods like: Supervised Contrastive Learning, a customized learning rate scheduler, CNN with attention CBAM layers, and a class-activation map analysis.

1.1. Data preprocessing

We are dealing with unbalanced classes: `#Healthy = 3101, #Unhealthy = 1903`. To tackle this problem we introduced 1198 new images of the class *Unhealthy* obtained by applying some unique transformations to the existing ones. These transformations were: zooming (by a of 0.7), and flipping, so transformations that would not change or create new values of pixels that would otherwise introduce a bias in the models. To be noted that this kind of transformation was avoided in the data augmentation part to make the transformed images as unique as possible. Also, we removed the images of *Shrek* and *Trololo* from the dataset bringing us to a total number of images of *6002* images perfectly balanced. We split the data into train and validation (0.15) and used the submission portal for testing.

1.2. Data Augmentation

Now we switch our focus to image scarcity. First, we tried using Keras class `ImageDataGenerator` for addressing augmentation using classical techniques like brightness, rotation, width and height shift, zoom range, and adding some noise. After training some initial models we inferred that data augmentation was crucial for improving the performance of our model so we decided to switch to a custom procedure, implementing other augmentation techniques: cutout, shearing, jitter, crop, color drop, and mixing all in a stochastic way, employing each one with different probability. This new procedure greatly improved the performance of the models.

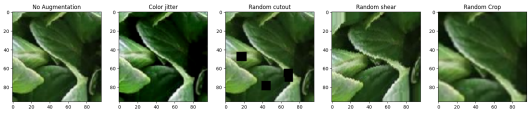


Figure 1. Examples of augmentations used on a sample image

2. Convolutional Neural Network

We decided to proceed in a bottom-up approach starting from a very tiny model. Our starting point

was a vanilla CNN composed of multiple convolutional layers (32, 64, 128, 256, 512 number of filters), GlobalAveragePoolig (GAP) to better grasp the overall feature-extraction process and a Multi-layer-Perceptron network (MLP) made of 1 hidden layer of 128 neurons, dropout layer for preventing overfitting. Despite as a starting point is a quite complex network we got `86%` on validation accuracy but `63%` on phase 1 test accuracy.

2.1. Attention layer

Moreover, we tried to employ an attention layer, with a CBAM structure after each convolution but we didn't appreciate an increase in terms of performance.

3. Pre-trained models - next phase

Given the low amount of data and poor results with a built-in CNN, we shifted towards making use of pre-built architectures taken from `Keras.applications` to finally construct a more robust model.

4. Training techniques

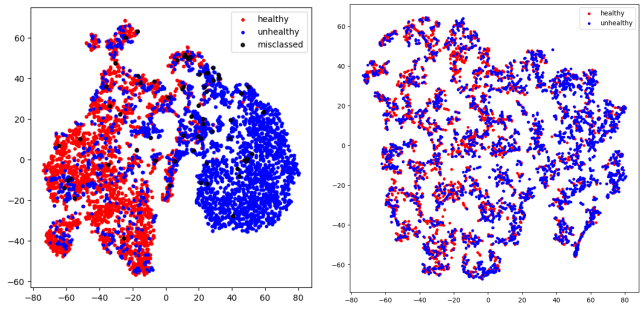
We made use of a custom-made learning rate scheduler and stopping algorithm technique called `LR_ASK` that consists of adding a callback function that enables the user, through an input window, to decide after a fixed number of epochs whether to stop the training or continue iterating and repeat that procedure. Moreover, it changes the learning rate adaptively: after each epoch, if the validation loss is above the lowest validation loss obtained so far then it resets the weights back to those of the one with the lowest loss and decreases the learning rate in an exponential manner. If instead, the validation loss found is the lowest so far the weights and the learning rate are left unchanged. Inside the FCN's we used some regularization techniques like: dropout, batchnormalization, l1 and l2 regularization for the weights.

4.1. Transfer Learning and Fine-tuning

Transfer learning combined with Fine-tuning was by far the most effective procedure for creating well-performing models. We equipped the pre-built architectures with a FCN made of a first layer of 256 neurons and second layer of 128 neurons depending on the results. In general, we saw no particular improvements in changing the structure of the FCN.
We tested almost every architecture available in `keras.applications` and we noticed that by far the most effective were: `ConvNextBase` `ConvNextLarge` and `ConvNextSmall`.
For the fine-tuning, we followed a procedural approach that is as follows. First, unfreeze the first 32 layers and train the model, or even less layers depending on the model's size, then decrease the learning rate and increase the data augmentation, do the same for 64, 96, and 128 layers and see where it starts overfitting. When it does overfit the data, go back to the previous step or try to augment even more the training set and go on with the procedure. Indeed throughout the whole of this procedure, data augmentation was revealed to be the key instrument to beat overfitting and increase the model's accuracy.

4.2. Supervised Contrastive Learning

Based on the paper by Chen et al.[1] We implemented a lighter version of SCL than the one proposed in the paper. We first had to built an encoder with a projection head that would predict for every image a vector of 128 units (normalized). For the encoder structure, we were able to get good results only from the EfficientNetV2L and the Resnet152V2 architectures.



(a) (b)

Figure 2. (a) Embeddings generated by EffnetV2L (b) Embeddings generated by ResNet50

To train the encoders with their projector head used heavy data augmentation, a big batch size of 128 (as suggested in the paper), and the *Max-margin-contrastive-loss* as the loss function; this function tries

1. Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *CoRR* abs/2002.05709 (2020). arXiv: 2002.05709. URL: https://arxiv.org/abs/2002.05709.

to minimize the Euclidean distance of the encoded images belonging to the same class and maximize it for the ones belonging to different classes.
With the help of the `TSNE` library, we were able to plot the embeddings in a 2D space of the 6020 images and see how well the backbone architectures could separate the two classes into clusters. As we can see from figure 2. the *EfficientNetV2L* (a) manages to divide the classes into two structured clusters. While *Resnet50* (b) is an example of a backbone that was not able to identify the two classes. Then the next step consisted of tossing away the projector head and attaching (just as in transfer learning) a fully connected network on top of the trained encoder fixing the value of its weights. In this way we could obtain a new classifier.

4.3. Ensemble Learning

As the last step of our procedure, we decided to merge two of the best models trying to reduce the variance of the outputs of our networks in an ensemble way. In particular, Ensemble Learning[2] involves training more than one network, then using each of the trained networks and combining each prediction in some way (weighted or normal average) to output a final score. In particular, we built two ensemble models: 1) *ConvNextbase (Fine-Tuned all layers-3)* + *ConvNextLarge (Transfer-Learning)* and used a merged the two by taking the average of the output neurons 2) ConvNextBase (Fine-Tuned all layers-3), +SupCon (EffNetV2L) and + (fine-tuned all) and merged the three by taking the weighted average (weights = (1,1,1.5)) of the output neurons.

5. Model choice

Throughout the project, we collected the results of all our models in terms of validation accuracy. Despite the high performances on the validation test some of these models didn't perform well on the phase 1 test set like VGG16 which scored `72%`. In phase 2, we submitted the SCL model: SupConv(EffNetV2L). This model obtained very high results in the validation set, with over 95% of accuracy, but only a 75.1% although it seems to have problems with overfitting since its submission (phase 2) resulted in 75.1% accuracy. While the ensemble models were the most accurate ones, in fact Ensemble base+small+SCL got an `85.1%` and ensemble base+large `86.7%`.

Model	val-acc	ph2-acc
ConvNextbase	96.13%	86.20%
ConvNextsmall	93.30%	–%
ConvNextlarge	91.84%	–%
Ensemble base-large	96.24%	86.70%
Ensemble small-base-contrastive	97.61%	85.10%
Contrastive(EffNetV2L)	95.47%	75.10%
VGG16	88.51%	72.00%
ConvNextXlarge	93.87%	82.00%
Ensemble effV2L-convbase	96.24%	–%

5.1. Explainability of the model

To understand what ConvNextBase learned from the training phase we put a dense classifier after its GMP at the end of the net, and take the weights of the winning class. Deducing the transferred net output we found the pixels that were activating more the class.

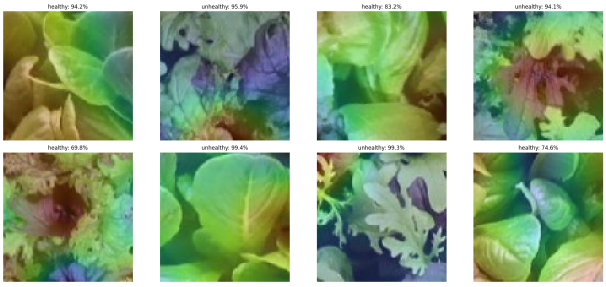


Figure 3. Activation map of convnext base

6. Conclusion

Building a CNN from scratch was not a viable option for tackling this challenge, while transfer learning, and fine-tuning, proved to be more effective. Also, the tweaking and improving of the data augmentation techniques was crucial since it's what allowed us to overcome the problem of overfitting, and even small changes in the functions used proved to have a significant effect on the overall results. Also from the literature we expected great improvements in performances when using contrastive learning,this didn't happen we assume that our pipeline for contrastive learning can be further improved.

2. Mudasir Ahmad Ganaie et al. "Ensemble deep learning: A review". In: *CoRR* abs/2104.02395 (2021). arXiv: 2104.02395. URL: https://arxiv.org/abs/2104.02395.

7. Contributions

Given the complexity of the project we decided to split the activities in order to grasp a fully comprensive view of the work. Niccolo and Roberto were the trainer's masters while Andrea was the github boss that managed all the proceeding of the projects and Filippo was the plots' specialist. At the end every piece of the work of each one was crucial for having the optimal outcome of the results.