Artificial Neural Networks and Deep Learning Homework 1

Team: TheLearningLegion4
Camille Buonomo (10978972), Simone Gabrielli (10715031),
Andrea Ricciardi (10931392), Andrea Zanin (10833512)

Dataset

The dataset provided for the challenge contained 5200 RGB images of plants with resolution 96x96 labeled as healthy or unhealthy; the pictures were taken close to the plants and don't show the background.
While manually inspecting the dataset to understand the characteristics of the images, we noticed that there were some outliers: "shrek" and "trololol" images. Since the task is classifying only plants, we removed the outliers from the dataset.
We were left with 5004 images, 62% of which were labeled as healthy and 38% unhealthy; to correct this imbalance between the classes we weighted more the unhealthy images during training.
Then we splitted our dataset in training, validation and test set.

Model improvement

Since we had a vast number of choices (e.g. backbone model, augmentations, …) in choosing the architecture of the model and the training approach, we couldn't simply test all combinations of the various choices; instead we adopted a greedy approach optimizing one choice at a time.
In order to test the various options we fitted and evaluated different models on our train and test datasets, we only submitted in the development phase the best performing models and chose the one with the highest accuracy on the hidden test set as our final model. In this way we reduced the risk of overfitting our dataset.
We used Class Activation Mapping to understand how the network reasons; this allowed us to notice if the network was using non-general information to classify the images (e.g. labeling the image as unhealthy only if the leaf is on the left) and also to make informed decisions about the augmentation techniques to use.

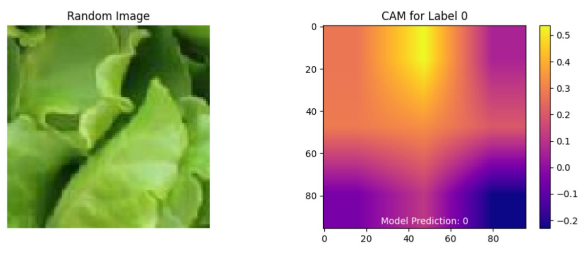
Backbone model choice

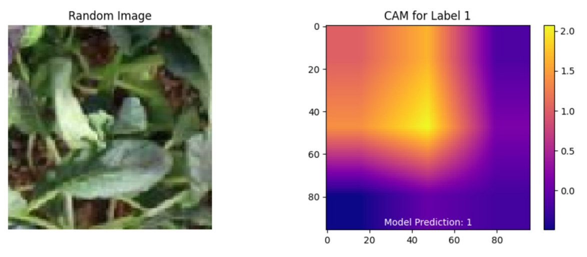
At first we tried fitting a Convolutional Neural Network from scratch, but the results were poor, so we decided to use a well-known model as a backbone model. We tried several models available in keras.applications (Xception, EfficientNetB3, VGG16, InceptionV3, ConvNextXLarge); from our comparison the best one was ConvNextXLarge. The assessment of the backbone model was done using a fully connected part with 256 neurons on the hidden layer with ReLU and an output with 2 neurons using softmax.
We also tried to put 3 models (ConvNextXLarge, VGG16 and InceptionV3) in parallel to obtain a larger feature space; the results were similar to using just ConvNextXLarge, so we decided to use only that model because it had less parameters.

Augmentation

To test the various backbone models we used a simple set of augmentations: random flips, rotations, translations and zooms; we then decided to try some additional augmentations.

Analyzing the ConvNextXLarge model with CAM we understood that the network focussed on the areas with high color difference among the pixels. For this reason we chose to test a random contrast augmentation.





We also tried implementing automated data augmentation as RandomAugment, that randomly selects some standard augmentations. Indeed they are more suitable to train in presence of small datasets and increase generalization. In keras_cv we have also found GridMask, which is based on the deletion of regions of the input image and, as stated in the correspective reference, outperforms the latest RandomAugment, so we have opted for it.
After testing these modern augmentation techniques with our model we didn't see an improvement in performance, so we decided to stick with the simple augmentation and the contrast augmentation for the final model.

Architecture choice

We started by using a shallow fully connected part (a single hidden layer with 256 neurons), then we tried using a deeper one with 512-256-128-64 hidden neurons plus an output of 2 neurons with softmax activation. The latter had 1% higher accuracy on the test set, so we decided to keep it. In order to avoid overfitting we added L2 regularization (l2_lambda=2e-5) and dropout (0.3 probability) between the layers. As optimizer we used AdamW, a stochastic gradient descent based on adaptive estimation of first-order and second-order moments with an added method to decay weights. As for weight initialization we chose He initialization.

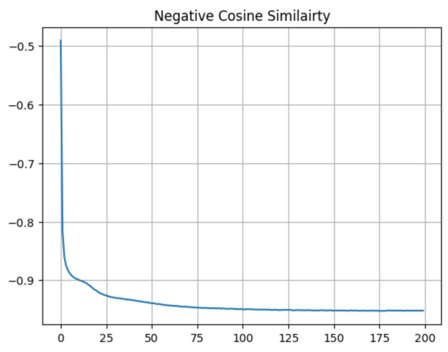
Another architecture we tried was adding 7 new channels to the input images and then using a CNN to fuse them back to 3-channel images. The additional channels were images without the dark parts, the gamma corrected images and the contour of the leaves with the function in the OpenCV library. Those additional channels however didn't improve the model performance, so we discarded them.

SimSiam

Given the relatively limited size of the dataset we explored self-supervised learning approaches to pretrain the backbone network on a pretext task before adding the classification layers and training the model on the actual task.
Since we had at our disposal both positive and negative samples we chose to implement a contrastive approach to self-supervised learning, rather than a non-contrastive one; in particular we have implemented the SimSiam.

To build the SimSiam model we started from our backbone network (ConvNextXLarge), we added two fully connected layers to form the encoder, whose output is fed to a predictor which is another two-layer fully-connected network. The dataset to train the SimSiam model is obtained by augmenting each image in the original dataset two times to obtain two modified copies of each original image. The SimSiam model is trained to maximize the cosine similarity between the features extracted by the encoder applied on one image and the features produced by the encoder plus predictor model applied to the other image.
Since the backbone model is very large we used the imagenet weights as initialization and set only the last N layers as trainable, where N is a hyperparameter we tuned manually.

After the pre-training phase we kept only the backbone model and discarded the other layers; using this pretrained backbone model we then trained a classifier with the approach described in the rest of the report. To assess the quality of the pretraining we plotted the loss, which clearly converges:



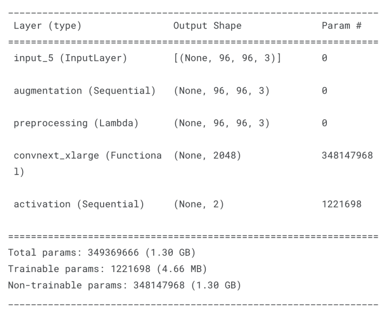
Training

Our first approach to training the network has been transfer learning using the ImageNet weights of ConvNextXLarge; we then tried fine tuning after the transfer learning phase, but the model didn't improve significantly, so for the final model we didn't use fine tuning.

We initially manually tuned the various hyperparameters and then we validated our choices using KerasTuner, an easy-to-use, scalable hyperparameter optimization framework (with RandomSearch and Hyperband).
We tried several batch sizes starting from 16 to 512, selecting the best performing (256).
We used a large number of epochs (1000) with "EarlyStopping" (patience=50,min_delta=0), monitoring the val_acc in max mode. For the learning rate we used "ReduceLROnPlateau" callback (patience=5,factor=0.99), starting from 1e-3 to a minimum of 1e-5.

Final model

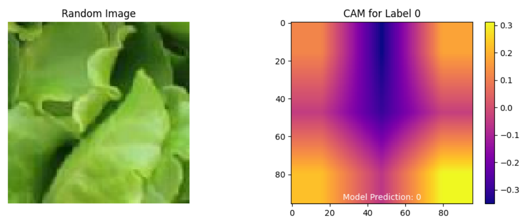
Our final model used ConvNextXLarge as backbone and it had the following structure:

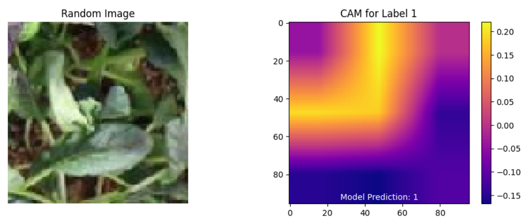


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 96, 96, 3)]       0

 augmentation (Sequential)   (None, 96, 96, 3)         0

 preprocessing (Lambda)      (None, 96, 96, 3)         0

 convnext_xlarge (Functiona  (None, 2048)              348147968
 l)

 activation (Sequential)     (None, 2)                 1221698

=================================================================
Total params: 349369666 (1.30 GB)
Trainable params: 1221698 (4.66 MB)
Non-trainable params: 348147968 (1.30 GB)
_________________________________________________________________
```

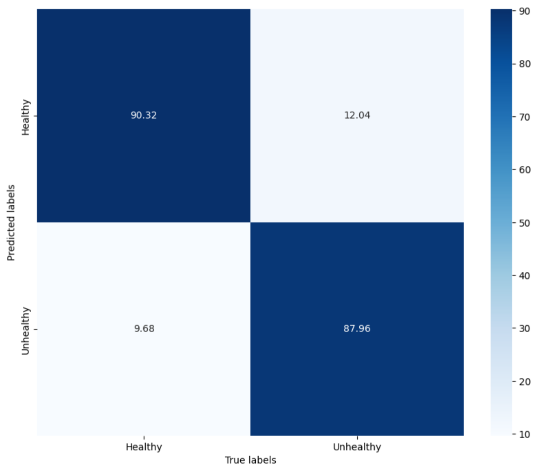
The training pipeline for the model was the following: pretrain the backbone with SimSiam, then freeze the backbone and train the classifier (transfer learning approach).
We trained the model two times: first we used a train-validation split to find the correct number of epochs using early stopping, then we retrained the model rerunning the whole notebook using all the data for training.
In the first training run we got 89% accuracy against our test set, the second training run achieved 84.5% accuracy in the CodaLab test set.

The following are Class Activation Mappings for the final model:





The following is the Confusion Matrix:



Contributions

- Simone Gabrielli (10715031) : He tested the “InceptionV3” network as backbone. He introduced the compute_class_weight from “sklearn” in order to deal with class imbalance and added it in the training. He introduced the “ReduceLearningOnPlateau” as a callback in the training and “AdamW” as an optimizer. He provided the deeper fully connected part of the network. He also tried using KerasTuner in order to tune the hyperparameters of the network, with RandomSearch first and Hyperband then. Finally he explored KerasCv looking for the optimal augmentations and found GridMask,which according to the related paper outperforms AutoAugment.
- Andrea Ricciardi (10931392) : He managed to test the “Xception” model as backbone, understanding that it was not as good as the others. He also managed to try the different batch sizes to tune this hyperparameter in the best way. While exploring the dataset he noticed that there were several outliers to be removed. He even managed to try the data preprocessing with only the function that adds gamma to the images, concluding that it is a bad way to proceed. For this reason he implemented the “compute_CAM” function for the ConvNextXlarge to understand how the network “reasons” to classify the images, hence concluding that any kind of function (for augmentation or preprocessing) that changed the brightness of the image should not be applied. Then he implemented the model that used three nets in parallel (Inception V3, Vgg16 and ConvNextXLarge), concluding that there was no improvement in using this architecture with respect to the ConvNextXLarge alone. In conclusion he provided the inference notebook, which was useful to make inferences about models by taking advantage of the performance indicators (accuracy, F1 score,...) and of the confusion matrix.
- Camille Buonomo (10978972) : He implemented the solution that used a CNN before the backbone model for images with 10 filters and concluded that it was not something worth following. With that, he worked on some preprocessing of the data, like adding contour or removing the dark part of an image to create the new filters. He also worked on various ways to reduce overfitting by exploring noisy methods which he proved to be very efficient. Another way to reduce variance was the reduction of variance by aggregating different models using the mean of each softmax output. But the models were too large to use many of them and the results with 2-3 models were not significant. On an exploring note, he tried to do the classification in the spectral space but it did not work.
- Andrea Zanin (10833512) : He implemented transfer learning and fine tuning of the "EfficientNet" and "ConvNextXLarge" networks. He wrote the script that removed the outlier images from the dataset. He tested several combinations of optimizers (Adam and NAdam) and number of hidden layers and neurons. He analyzed the various self-supervised learning approaches and concluded that SimSiam could be a good fit for our situation, then he implemented it and trained it using several hyperparameter choices. He also implemented a procedure to train the model on the whole dataset, first choosing the number of epochs using a validation set and then retraining on the whole dataset.

References

[1] “OpenCV: OpenCV modules.” n.d. OpenCV Documentation. Accessed November 18, 2023. https://docs.opencv.org/4.x/.

[2] "GridMask Data Augmentation." 2020. Pengguang Chen, Shu Liu, Hengshuang Zhao, Jiaya Jia. arXiv. https://arxiv.org/abs/2001.04086.

[3] "Exploring Simple Siamese Representation Learning" 2020. Xinlei Chen, Kaiming He. arXiv. https://arxiv.org/abs/2011.10566

[4] “AutoAugment: Learning Augmentation Policies from Data” 2019. Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, Quoc V. Le

[5] “RandAugment: Practical automated data augmentation with a reduced search space” 2019. Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, Quoc V. Le