Report Challenge 1 Team Drip Mac

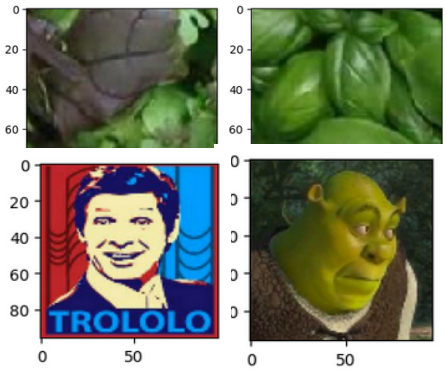
Leonardo Brusini, Marco Scarpelli, Davide Tenedini, Federico Toschi

Problem Description

The problem is a binary classification problem. Each of the images in the dataset depicts a plant that belongs to either the positive class (“unhealthy”) or the negative class (“healthy”). Our objective is to train a model that can correctly predict the class of an image. Formally, we want to learn the function *h* such that *prediction* = *h(img)*, where *img* is an image of a plant. Our model is evaluated based on its accuracy, formally $acc = \Sigma_{1 \le i \le N}(predictions_i == targets_i) / N$, where N is the number of images in the test set.

Dataset Inspection & Augmentation

The dataset consists of 5200 images of plants. Inspecting the dataset we found multiple copies of two different images not representing plants. The cleaned dataset is made of 5004 images.



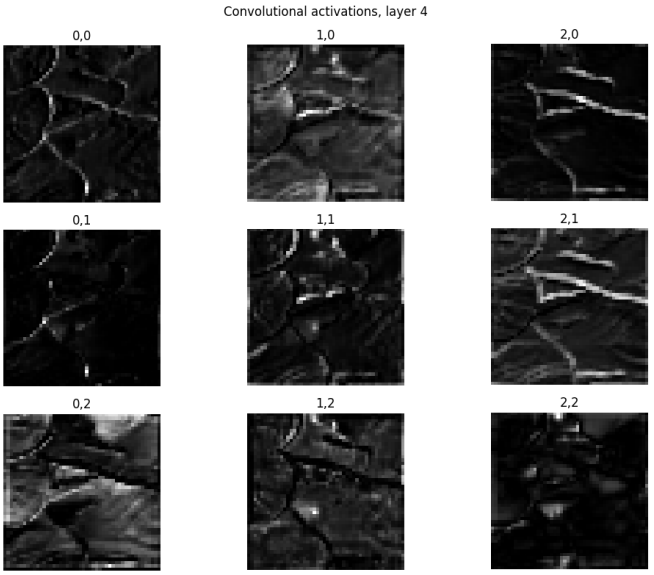
We tried different configurations of data augmentation for training our models, both generating and saving augmented samples in a dataset before training (up to 40*k* samples), and generating samples during training with Tensorflow augmentation layers.
We found that the former was better than the latter because we could randomly select an augmentation at a time instead of having an image run through a series of sequential layers that could severely distort it; in fact, sequential augmentation parameters tied to translation and rotation have to be small enough for the resulting image to be usable, without too much fill-in, and having an image run through both could distort it too much. Furthermore, we determined that brightness and contrast also played a significant role in image discrimination, and we had to be especially careful when dealing with these augmentations; flipping images on the other hand would not significantly impact the majority of the dataset, since most pictures were so zoomed in that it was nearly impossible to tell which angle they were taken from. In addition, we experimented with random perspective augmentations: by running some examples through one of our best-performing models and analyzing the feature distribution, we understood the right parameters to use for the resulting images to be usable. Finally, our last models were trained and fine-tuned with Cutmix [9]/Mixup [10] augmentation.

Custom CNNs & Activations

We strived to implement a good custom model, even though in the end those based on transfer learning turned out to be better.
At first, we experimented with simple, fully convolutional architectures with various configurations of filter sizes and amount of convolutional layers; some also had dense layers in the end. These models suffered a lot from data scarcity, i.e. they tended to overfit a lot even with as low as 20’000 parameters, and this prompted us to resort to data augmentation; some of the first trial models included their own augmentation layers.
Through visualization of each layer’s activation (an example of activations in the next figure), we experimented with different activation functions and filter sizes. We noted that setting the convolution padding as “valid” seemed to work better.

The best result for one of these simpler models was 70%, which was unsatisfactory; we moved on to experimenting with Inception layers and shortcut layers. Unfortunately, even though at this point we had seemingly solved the overfitting problem, it was difficult to figure out the best parameters for such networks; we defined a function that would use a list of layer sizes and automatically build the model.

Each “layer” has three convolutions of different sizes (defined by the user) that run in parallel and merge into a concatenate layer, which is then fed into the next three. Note that the convolutions here have to have padding “same” because they would resize the image in ways different from each other. When the function detects that the layer size has changed, it creates a shortcut starting from the beginning of this layer set and adds it to the output of the cascade described before; a *1-D* convolution is added to the shortcut layer to match the output size, and we insert a pooling layer which then feeds the next cascade. In the end, we have a GAP layer. *Open the “Custom model with activation plotting” notebook to see the structure.*



Results were again unsatisfactory, probably because augmentation was not fine-tuned yet, and figuring out how deep and wide the net should be is not a trivial task; in the end, the fourth iteration only managed to score 73% accuracy.

Notable Attempt - Transfer Learning on ConvNextBase

Among all the models we tried Transfer Learning on, ConvNextBase was the best tradeoff between performance and model size: specifically, we attached the base model to a 10-neuron dense layer with Gelu activation function. Because this significantly increased the number of parameters, we contrasted overfitting by using *l2* regularization on the dense layer and following it with a dropout layer.

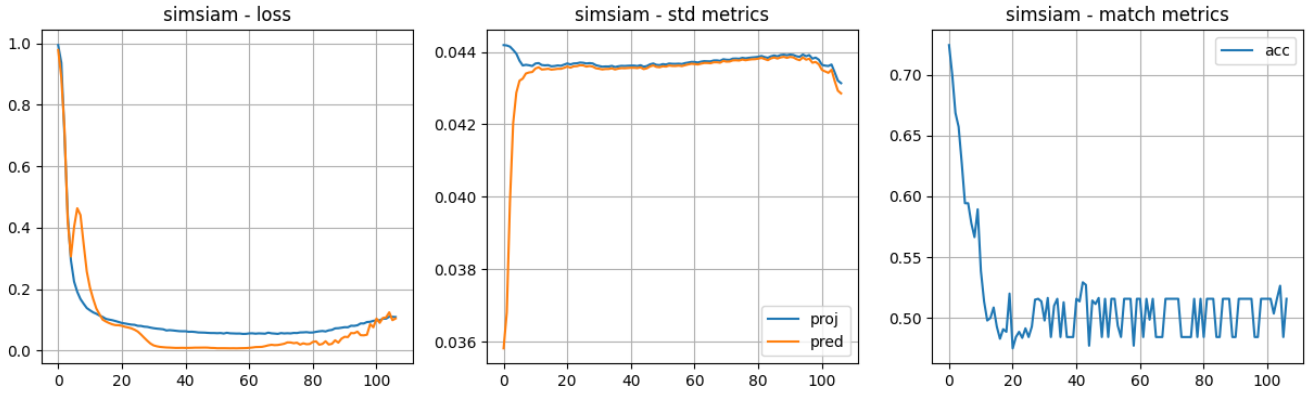
These were then trained with a custom learning rate scheduler with exponential decay that allowed to reduce the learning rate by 10% the further the model was trained. In addition to this, we used two callbacks: EarlyStopping on validation accuracy, and ReduceLROnPlateau on validation loss.

Finally, the model was trained on an unbalanced, augmented dataset consisting of 13*k* samples: to prevent the skew of the distribution from negatively impacting the learning process we also implemented class weighting.

Unsuccessful Attempts - Self-Supervised Learning and Ensemble models

Following suggestions from teaching assistant Lomurno, we tried both self-supervised learning and ensemble models.

Self-supervised learning: following the “hello world” notebook [8] of the Tensorflow Similarity library, we tried SimSiam [1] algorithm for training a feature extractor on a contrastive loss comparing pairs of augmentations of the same image using cosine similarity. Two different sets “query” and “index” are then used for validating the training process. Features are extracted from each query image and then its label is compared to the “closest” image in “index”. Early stopping is applied on the binary accuracy metric over “query” and “index”. We obtained the best results using ResNet50 [4] as a feature extractor, still, the final training showed that the standard training on the model based on ResNet50 would still perform better. We also tried using MobileNet [6], EfficientNet [7], and ConvNeXt [5], but we encountered a common problem in self-supervised learning, representation collapse. Contrastive loss would decrease very fast while the binary accuracy stays around 50% (as shown in the next figure). More work on hyperparameter tuning or choosing different augmentations based on this specific dataset would possibly solve the representation collapse problem.



Ensemble model: we tried building a simple ensemble model by averaging the predictions made on models built using non-trainable feature extractors MobileNet, ConvNeXtLarge [5], EfficientNetB7 [7], and Xception [2], each with one or multiple trainable fully connected layers on top other than the last single unit output layer. The model gave promising results, still, our final model based on making the full ConvNeXt network trainable gave better results. Unfreezing the layers on all networks in the ensemble model would make it less feasible to train in terms of time and memory constraints.
We also tried building an ensemble model using our own custom CNNs; this was based on two twin networks trained on two slightly different augmented datasets (one balanced by removing some elements from the bigger class and then augmenting, and one balanced through augmentation and then augmented more), but since our own CNNs did not perform too well, neither did the ensemble.

Final model

The model that achieved the best result is based on the ConvNeXtLarge feature extraction. On top of it, a 75-unit dense layer, with selu activation function and the final single unit output with sigmoid activation, with a 0.3 dropout rate between them. The training was done on the original dataset, without outliers, split into train and validation (80/20, so 4003 train samples and 1001 validation samples), and augmented using different augmentation layers compiled into the sequential model. The training has been divided into 2 different phases. The first phase is done by only making the dense layers trainable, applying L2 regularization on them with a lambda value of 1e-5, and using adamW as optimizer. The callbacks are early stopping with a patience of 15, starting from epoch 15, and “ReduceLROnPlateau”, which shrinks the learning rate by a factor of 0.8 when the validation loss doesn’t get smaller for 5 epochs. For the second phase of the training, we made the full model trainable, incremented the regularization’s lambda to 1e-3 for the 75-unit layer and 5e-4 for the output layer to make up for possible overfitting, used a smaller starting learning rate (1e-5) and added weight decay with a rate of 5e-5 for the optimizer, and the reduce LR callback factor to 0.7. Both training phases are done with a batch size of 64 and by weighting the classes to make up for the unbalancing of the samples. The best epoch managed to reach 94.11% validation accuracy while reaching 87.1% accuracy on Codalab’s final test set.
On a final note, we tried improving its result by applying test-time augmentation (TTA). The model would give a final prediction on an image by averaging the results of predictions made on the original image and different augmentations of it (we found that the best combination of augmentations were: horizontal flip, 90° rotation, 270° rotation, contrast with a factor of 3 and brightness with a delta of 2). In the end, validation accuracy improved to 94.71% but we didn’t manage to submit the model with TTA on time.

Contributions

Leonardo Brusini (*LeonardoBrusini*):

- Dataset inspection - outlier removal
- SimSiam self-supervised learning
- Ensemble model
- Fine-tuning on frozen ConvNextLarge and training on fully trainable ConvNextLarge
- Test-time augmentation

Marco Scarpelli (*MarcSka*):

- Dataset inspection
- Augmentation trials and fine-tuning of parameters
- Custom CNN and ensemble models
- Training with own GPU on home computer
- Visualizations for fine-tuning

Davide Tenedini (*DavideTenedini*):

- Dataset Inspection - Balancing
- Dataset Augmentation
- Custom CNN models
- Iterative Augmentation
- Transfer learning on MobileNet

Federico Toschi (*ftoschi*):

- Augmentation pipeline
- Feature distribution analysis for selection of good augmentation parameters
- Transfer learning on: MobileNet, InceptionResNet, EfficientNetV2S/EfficientNetV2M, ConvNextTiny, ConvNextSmall, ConvNextBase, ConvNextLarge
- Cutmix/Mixup augmentation
- Fine-Tuning on ConvNextLarge

References

[1] Chen, X., & He, K. (2020). Exploring Simple Siamese Representation Learning. arXiv [Cs.CV]. Retrieved from http://arxiv.org/abs/2011.10566

[2] Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. arXiv [Cs.CV]. Retrieved from http://arxiv.org/abs/1610.02357

[3] Dufour, P. (2020a, August 5). How to correctly use test-time data augmentation to improve predictions. Step Up AI. https://stepup.ai/test_time_data_augmentation/

[4] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity Mappings in Deep Residual Networks. arXiv [Cs.CV]. Retrieved from http://arxiv.org/abs/1603.05027

[5] Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). A ConvNet for the 2020s. arXiv [Cs.CV]. Retrieved from http://arxiv.org/abs/2201.03545

[6] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2019). MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv [Cs.CV]. Retrieved from http://arxiv.org/abs/1801.04381

[7] Tan, M., & Le, Q. V. (2021). EfficientNetV2: Smaller Models and Faster Training. arXiv [Cs.CV]. Retrieved from http://arxiv.org/abs/2104.00298

[8] TensorFlow Similarity Self-Supervised Learning Hello World. GitHub. https://github.com/tensorflow/similarity/blob/master/examples/unsupervised_hello_world.ipynb

[9] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., & Yoo, Y. (2019). CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. arXiv [Cs.CV]. Retrieved from http://arxiv.org/abs/1905.04899

[10] Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2018). mixup: Beyond Empirical Risk Minimization. arXiv [Cs.LG]. Retrieved from http://arxiv.org/abs/1710.09412