Image Classification

Donald Ruci
Mikel Kumria
Joanna Rancew

Problem

The task of the challenge was to classify images of leaves within two classes: either healthy or unhealthy. The images were associated with respective labels. Each image of size 96 x 96 was stored on 3 RGB channels.

Data preprocessing and preparation

At the beginning of our work, we thoroughly inspected the dataset. We detected some imbalance between the representation of ‘healthy’ and ‘unhealthy’ classes and messy data within the dataset. To clean the data we compared the histograms of images and removed all images not pertaining to the 2 classes of interest. To prepare the data for training, we have changed the format of labels to categorical and split data into training, validation and test set (70%, 15%, 15% respectively). To account for the class imbalance, we considered either undersampling the majority class or assigning different weight classes in the following way: class_weights[x] = (total_samples / class_counts[x]), where x = 0 for “healthy” and 1 for “unhealthy”. To use the whole dataset, in the end we decided to implement class_weights to compensate for the imbalance of classes. After initial processing we could proceed to perform the challenge task: *Image classification.*

Custom Convolutional Neural Networks

The first model we trained was a convolutional neural network with 5 layers. When we trained we were getting around 75-80% on our validation set and around 60% on our test set. We submitted it, and we got the same result on codalab. To minimize overfitting, we applied some data augmentation (translation and brightness) to achieve around 68% on codalab. It was better but still wasn’t giving the results that we wanted. We tried to reduce the number of layers since the dataset wasn’t that big and give it a try with 3 layers. Overfitting was slightly reduced but the results were worse than before. Now we decided to move into pretrained models and transfer learning for better results.

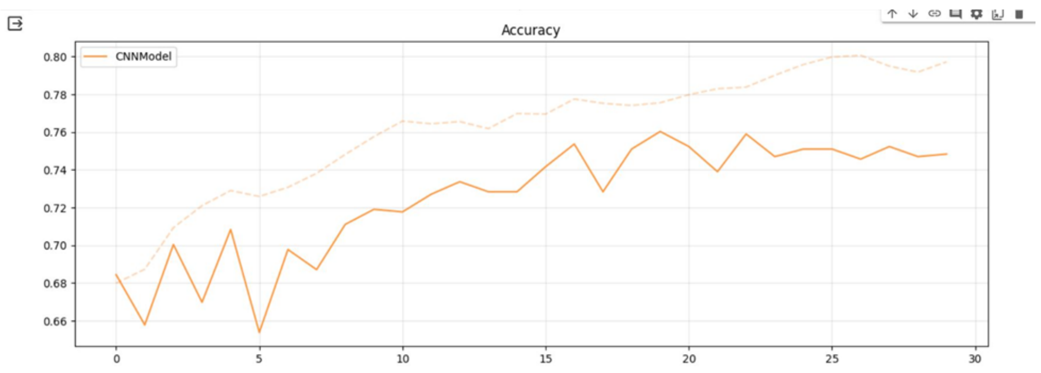

Figure 1. Training custom Convolutional Neural Network

VGG-16

We then transitioned from our initial custom model to transfer learning via the implementation of VGG-16, to which we attached one flattening layer and a dense layer with 256 neurons and a ReLU activation function, and then sigmoid as the output activation function. After various re-runs with different hyperparameters and getting around 80% accuracy on our validation set and 62% accuracy on our test set, we also tried some other architectures, to experiment on the best performance so we could choose which models to keep for the ensemble.

InceptionV3 and InceptionV4

We followed with the implementation of InceptionV3, to which we attached a flattening layer, a fully connected layer with 1024 neurons, ReLU activation function and batch normalization, a dropout layer with dropout rate of 0.2 to 0.5 (of which 0.5 performed the best) and sigmoid for the output layer. We used RMSprop as the optimiser and binary crossentropy, suitable for binary classification. We then also implemented InceptionV4 with several convolutional layers, batch normalization and ReLU, global average pooling followed by a dense layer and dropout, with Adam as the optimiser. The best performance of these models was around 78% accuracy on our validation set and 65% accuracy on our test set and, surprisingly, InceptionV3 performed better than InceptionV4. We then ventured into other models to see if we could find better performing ones.

MobileNetV3 and MobileNetV2

The next pretrained model tested for the class was MobileNet. The base model was implemented with initial weights trained on the ImageNet dataset. On top of the model a dense layer with sigmoid activation was implemented. The input pixels were rescaled between -1 and 1. After training the additional layers, we performed fine tuning of the model. Depending on the number of last layers trained (20-60), the model gave the accuracy on our test set in the range: 84%-86%. After performing hyperparameters tuning (adjusting learning rate, number of layers trained, early stopping patience, batch size) we achieved better accuracy equal to around 86% on our test set and around 78% on Codalab.

EfficientNet

While we were doing our research in models in order to achieve better results, EfficientNet caught our eye. Motivated to find a model that strikes a balance between computational efficiency and performance we thought that EfficientNet might be a good fit for our case. EfficientNet uses a technique called compound coefficient to scale up models in a simple but effective manner (balancing the scale in all three dimensions helps improve model performance). So we began with our experiments.

We trained EfficientNetB0, EfficientNetB1 and EfficientNetB2. We didn’t want to go any further because the other models would be very big and too complex for our case. Firstly, we applied transfer learning to each, and fine tuned models separately. All these models were giving promising results in our validation set (87%-89%) and testset (86%-88%). We submitted EfficientNetB0 in codalab and it gave 78% accuracy so we were more motivated to go on.

Our motivation to enhance performance led us to explore model ensembling as the next step in our research. We got 81% accuracy on codalab by ensembling B0 and B1, and we got 82% trained on the whole dataset with ensembling B0, B1, B2 trained with cross validation.

Additional techniques to increase generalization

To improve the model's ability to classify unseen data and reduce overfitting, we implemented several methods: data augmentation (with several transformations like shift, flip and brightness), a hyperparameter search including dropout (with dropout rates from 0.2 up to 0.5) and early stopping (with patience from 10 up to 50), batch normalization and varying learning rate, increased batch size (starting from a batch of 16 and going up to 256, which improved the generalization of the model). We also used class weights, which reinforced the importance of the less represented class in the training of models. Another thing we tried was training EfficientNetB2 using k-fold Cross Validation, implementing L2 regularization, and ensembling various models. These techniques contributed to an increase in accuracy on our test set and prevented overfitting.

Final results

Combining best models and most useful techniques to increase model generalizations, we created a solution that achieved about 86% on our test sets. In the final stage of training, we trained and tuned the ensemble model on more data (both training and validation sets) to ensure the best possible final results. The two models that performed the best were the EfficientNetB2 trained with 5-fold Cross Validation (76,7% in the Final Phase) and ensemble of EfficientNetB0, EfficientNetB1 and EfficientNetB2 (79,8 % in the Final Phase).