Report

The aim is to classify images of plants into healthy and unhealthy categories using CNNs.

DATA EXPLORATION

As a first quick data inspection, I visualized random images from the dataset and immediately noticed the presence of two classes of outliers: Shrek and Trololo. To remove them, I firstly defined a metric based on matrix distance. However, after realizing that the metric had to be able to recognise similar images in presence of slighly different illumination and noise, I ended up relying on the structural similarity metric of the skimage library.
I created a reference image by averaging 50 samples of non-outlier pictures, evaluating the similarity metric of each image in the dataset against this reference. Since Shrek images were more similar to the reference than some non-outlier images, this technique worked just in part. I cleaned the rest of the dataset by finding the remaining outliers using a Shrek image as reference.

BALANCING THE DATA

I proceeded to explore its class distribution and noticed a great imbalance: approximately
63% of the data consisted of healthy subjects, with only 37% unhealthy ones.

After some attempts I found out that the best way was to balance the dataset through flipping and rotating.

SPLITTING DATA BETWEEN TRAIN, VALIDATION AND TEST

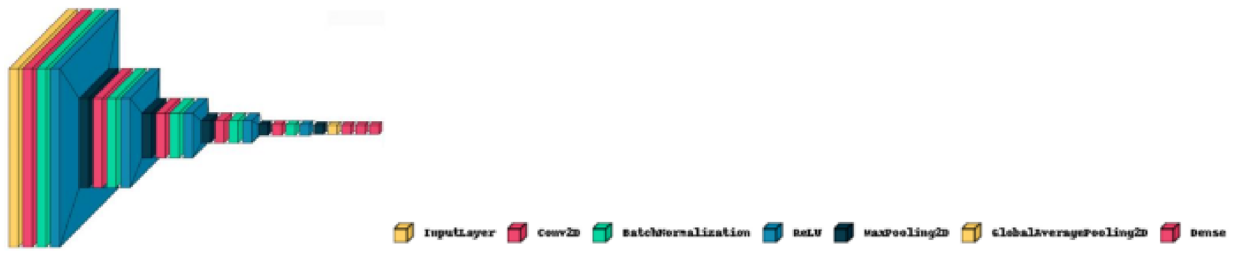
Initially, I chose to divide the balanced data between the Training and Validation sets only because I didn't want to lose any information. Despite trying different percentages, my validation accuracy was significantly higher than the accuracy I achieved on Codalab. Recognising this, I saw the need to construct a balanced Test set myself, aiming to provide a truly unbiased performance estimate. I did this by extracting 100 healthy and unhealthy subjects from the cleaned dataset prior to balancing.

DATA AUGMENTATION

To increase the number of training data, I introduced augmentation techniques like brightness/contrast changes, rotation, shift, and zoom. I did this directly during the training steps of the models.

MODELS

The first approach has been to design a new model from scratch, with the following architecture:

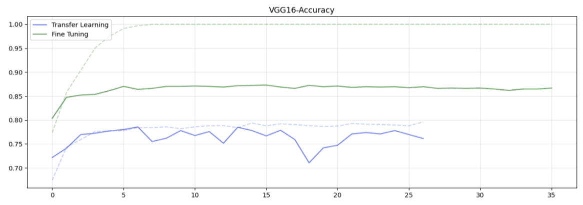


I exploited the simplicity and quick train time of this model by applying Grid Search Hyperparameter Tuning to find the best combination between number of filters, kernel sizes, dense units and learning rate. Despite this experimentation with hyperparameters, the model was incapable of learning meaningfully, as it was not able to overcome 67% of validation accuracy.

To overcome the poor performance of the Feature Extractor, I resorted to transfer learning and fine-tuning. In particular, I employed two distinct architectures: VGG16 and ResNet50.

VGG16

Initially, I adopted a basic transfer learning approach with VGG16, attaching a classification layer directly to its output. I encountered an issue with the vgg16.preprocess_input command, which was resolved by saving preprocessed data into a new variable using vgg16.preprocess_input(np.copy(X_train)). This adjustment bypassed the in-place nature of the function.



Initially, the model stagnated at 79% training accuracy with comparable validation accuracy. To enhance performance, I fine-tuned the convolutional layers of VGG16.

By gradually unfreezing these layers (stopping at 5), I achieved a training accuracy of 100% and a validation accuracy of 88%. Then, to mitigate overfitting, I incorporated L2 regularization (2e-3 lambda) on the fully connected layer. This slowed the convergence of the training accuracy but did not significantly impact the validation accuracy.

RESNET50

Initially, I thought that the ResNet50 architecture, having a multitude of blocks, was better suited for managing the fine-tuning process. Contrary to what I thought, fine tuning the model did not improve its transfer learning performance.

ENSEMBLE

In my quest to find the optimal model, I explored the effectiveness of combining multiple architectures. My initial approach was to employ two VGG16 structures (one of them modified by adding extra layers at the end) along with a MobileNet structure. Noticing that averaging the outputs was not performing well, I concatenated these outputs and introduced additional layers (a dense output layer employing softmax activation). This configuration achieved commendable validation accuracy without succumbing to overfitting.

FINAL MODEL

Final best model (73.7% Accuracy on Codalab).
In my final attempt, I leveraged three previously developed models (two VGG16 architectures and my ensemble) by computing an average of their three outputs. Various combinations were explored to optimize the ensemble. This approach aimed to harness the strengths of each model while mitigating individual weaknesses and contributed to a comprehensive and robust solution.

CONCLUSION

My initial simple model achieved nearly 100% training accuracy with minimal parameters, suggesting that simplicity can be effective for this task. Yet, my top performing models were more complex, likely benefiting from the pre-training that improved their performance on new, unseen data.
I encountered several difficulties in the balancing and pre-processing of the dataset. I suspect that the severe decrease in accuracy my models faced on the Codalab test set is mainly due to these issues. That's probably why the models I built from scratch struggled the most.
This challenge highlighted the criticality of adaptive strategies in NN development. From issues like data imbalances and outliers to iterating through various models, tuning their hyperparameters, and applying fine-tuning techniques, I gained valuable insights into the design of Deep Learning models for image classification. This experience greatly enhanced both my theoretical knowledge and technical skills.