Milestone 1

Minor AI January 2022

Machine Learning Project

Nassim Bekkali, Rhea Groenenberg, Jeanice Koorndijk

*Deadline: Thursday January 13th at 17:59*

*This is the first model you create for your project. The report introduction should include a description of the problem and data section might include some basic pre-processing. The model itself should be very simple, but show to learning something from the training data, i.e. make a prediction that is (a little) better then randomly selecting an output.*

*Meeting 1: Wednesday January 12th*

*Meeting 2: Friday January 14th*

Afbeelding met tekst, boom, plant

Automatisch gegenereerde beschrijving

**Contents**

[REPORT INTRODUCTION 2](#_heading=h.gjdgxs)

[Apple Tree Disease Project description 2](#_heading=h.30j0zll)

[Data section 2](#_heading=h.1fob9te)

**1. REPORT INTRODUCTION**

**1.1 Apple Tree Disease Project description**

The overall productivity and quality of apple orchards can be negatively affected by foliar (leaf) diseases. Current disease diagnosis based on human scouting is time-consuming and expensive.[[1]](#footnote-1) Computer-vision based models may be able to increase the efficiency with which diseases are detected.

The difficulty of machine learning algorithms to account for variations in symptoms due to aspects like age of infected tissues, genetic variations, and light conditions within trees can be hurdles to efficient and accurate detection of different diseases.[[2]](#footnote-2) This machine learning project sets out to overcome these hurdles by developing a machine learning-based model to accurately classify a given leaf image from the test dataset to a particular disease category, and to identify an individual disease from multiple disease symptoms on a single leaf image.

**1.2 Data section**

**1.2.1 What the dataset looks like**

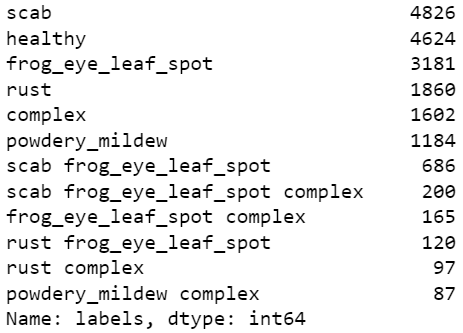
Our project utilises the dataset from the Kaggle competition ‘Plant Pathology 2021 Challenge’. This dataset is based heavily on the dataset from the competition of 2020.[[3]](#footnote-3) The dataset used in this project (2021) contains 18,632 high-quality RGB images of apple foliar diseases, including a large expert-annotated disease dataset. This dataset reflects real field scenarios by representing non-homogeneous backgrounds of leaf images taken at different maturity stages and at different times of day under different focal camera settings.[[4]](#footnote-4)

Overall, the provided dataset has approximately 18,632 training images and 3 test images of apple tree leafs, with 12 possible classes (including both single and combinations of 6 different classes).

The training set metadata consists of the image ID and labels. The labels are the target classes, a space delimited list of all diseases found in the image. An image can have multiple labels, indicating that multiple diseases have been identified for that image. Unhealthy leaves with too many diseases to classify visually will have the complex class and may also have a subset of the diseases identified. Note that the competition had a hidden test set: only three images were provided as testing data as samples while the remaining 5,000 images would be made available to the notebooks of contenders once they had submitted their work.

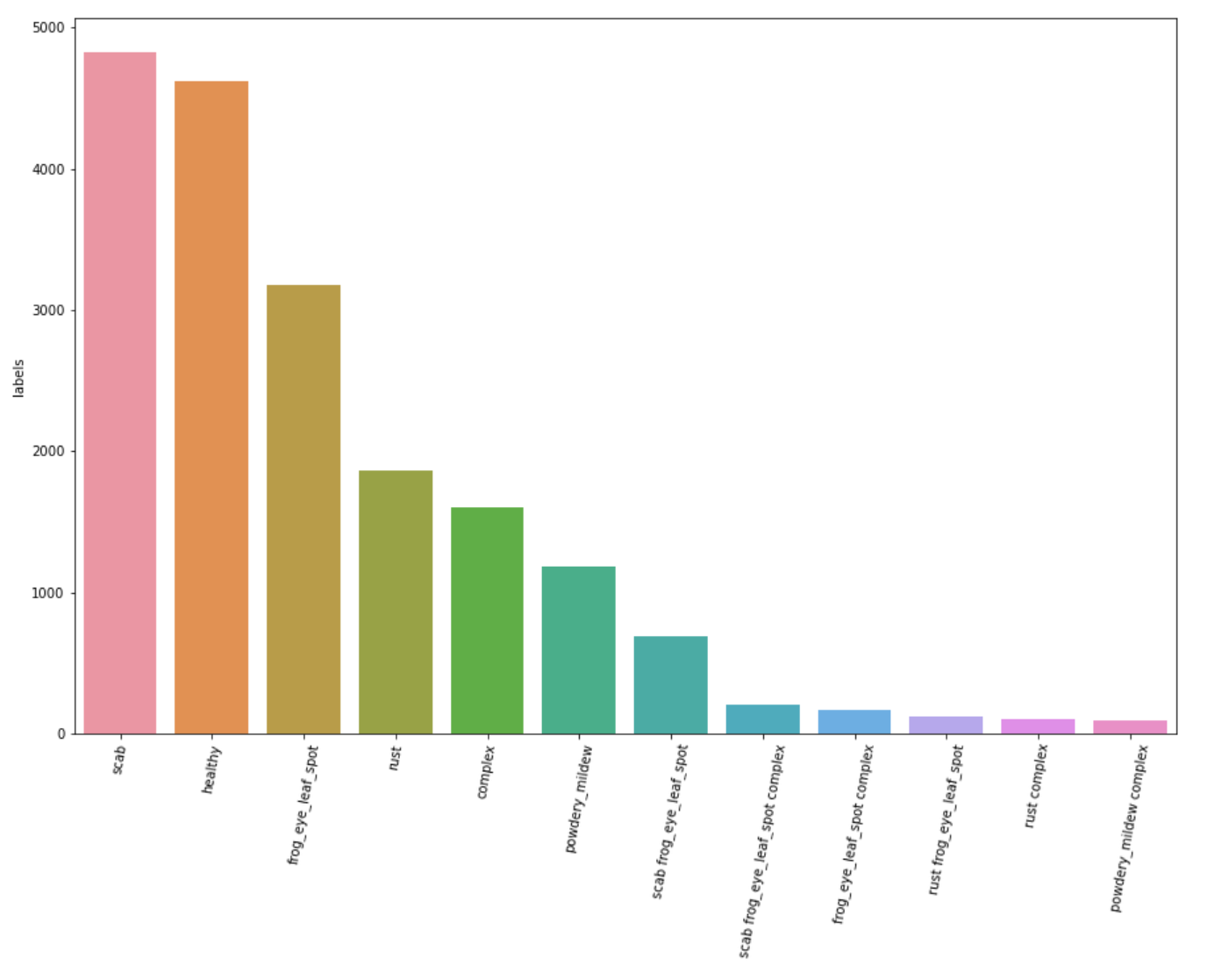
Finally, the dataset is not balanced. We have processed the data in such a way that combinations of labels are considered a specific class in the data. This is elaborated on in the section below.

**1.2.2 How the data has been processed**

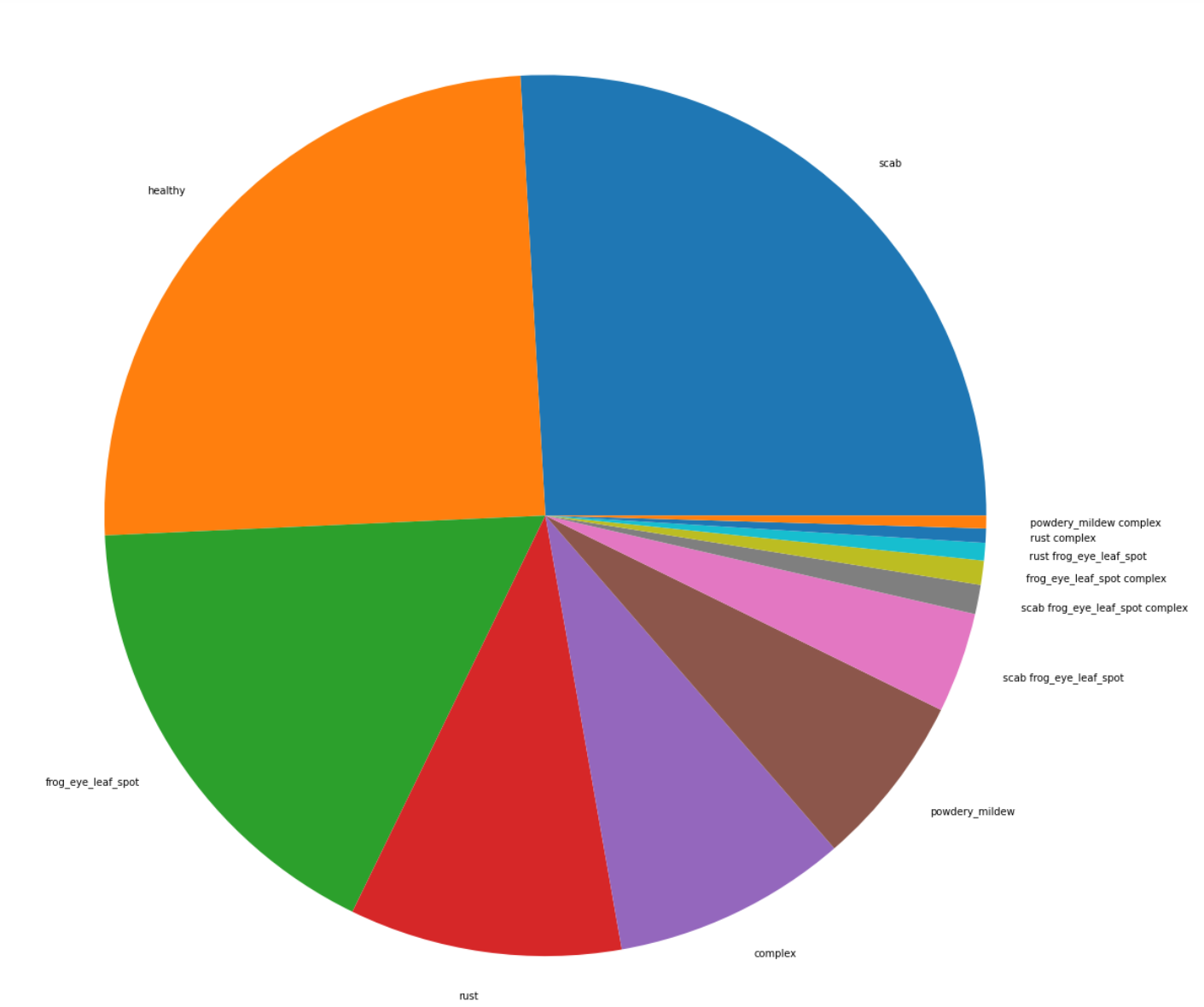
The data has been subjected to some basic pre-processing. Firstly, we have converted the csv-file containing each image ID and corresponding label into a one-hot encoded- matrix, in which each class/ combination of classes were treated as an individual class. This leads up to a total of 12 classes, even though the data only contains 6 unique labels. We analysed the training images using pandas. There are twelve classes, including classes consisting of multiple diseases. 

Figure

Figure 2 and figure 3 (on the next page) indicate that the training data is heavily imbalanced. We applied some visualisation techniques to get a grasp of the imbalance in the dataset:

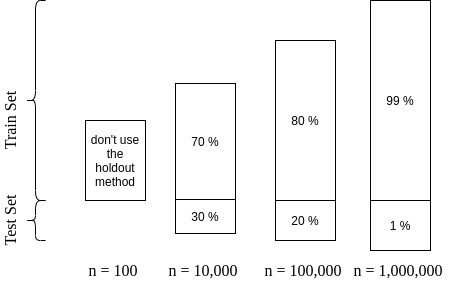


Figure



Figure

These visualisations indicate the imbalance in the dataset. In order to work with a more balanced dataset. Secondly, due to the limitations of the software used, Google Colab, we have chosen to only select 960 samples out of the 18.632. Hereby, we have taken the precaution to make sure the selected samples were equally balanced across all 12 classes by training on 80 samples from each class. This choice was also, most importantly, made because it would take too much time for the programme to run when going through all data. This setup enables a balanced and faster programme.

We realise that changing the dataset in this way may affect the accuracy in unintended ways. For example, there may be less samples of a particular class because, in practice, the occurrence of such a class is rare. By artificially adapting our dataset we may prevent the NN from recognizing such correlations. When optimising the programme, we will revisit the sampling method and try to find the most effective method.

Figure

After correctly processing the data, the training and validation data needs to be separated from the data that is used. When determining the distribution, it is assumed that approximately 960 samples will be used. (See figure 4) In this case, a ratio of 70/30 (training data/validation data) will be used, because the data is relatively small: 100 < n < 10 000.[[5]](#footnote-5)

The following steps have been taken to load the data.

* Each image is resized to 96 x 96 pixels and the 3 colour channels are maintained.
  + This gives a new shape to an array without changing its data.
  + We chose to opt for 96 x 96 pixels because this seems to be an (informal) convention when building CNN models.
* Each of the labels is transformed into a one-hot encoding using pandas.

**1.2.3 Model evaluation function**

This function takes a neural network model, training, and validation data, and:

* Compiles the model using a CategoricalCrossentropy loss function to compute the crossentropy loss between the labels and predictions.
* The function has an additional metric of accuracy that calculates how often predictions equal labels. This indicates how well the predictions match the actual labels.[[6]](#footnote-6)
* The function utilises an Adam optimizer, that implements the Adam algorithm. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.[[7]](#footnote-7) The method was chosen because it has been described in the literature as having little memory requirement; being computationally efficient; invariant to diagonal rescaling of gradients and being well suited for problems that are large in terms of data/parameters.[[8]](#footnote-8)

The function goes on to:

* Fit the compiled model using the pre-processed training data for the specified number of epochs.
* Plot the loss and accuracy for the training and validation data for each epoch.
* Print the final validation accuracy.

**1.2.4 Creating a basic CNN model**

*The layers*

For the basic CNN model, our team has chosen the sequential model. A Sequential model seems appropriate for a plain stack of layers such as in our basic model, where each layer has exactly one input tensor and one output tensor. We chose the sequential model because we have already worked with this model in the past, when building a basic CNN model for the CIFAR dataset. The model seemed to work quite well for that basic CNN for image recognition, which is why we have opted to use it in this project as well. Note that we might change the model during later stages of the project, as we attempt to improve the programme.

The basic CNN consists of 2 convolutional layers. The first layer has 32 filters and the second layer has 64 filters. This ensures a diamond shape to facilitate learning an appropriate function. The number of filters may be adjusted when tweaking the algorithm for optimization.

For the convolutional layer, we opted for ReLu activation functions. Although we might use different activation functions in later stages of the project, the ReLu function seems particularly appropriate for the basic CNN because ReLU is a non-negative activation function. In later stages of the project, we might combine this with dropout layers.

After each convolutional layer a max pooling layer is applied. The max pooling reduces the number of nodes. We have added max pooling layers, specifically for 2D spatial data. Max pooling is a common pooling technique for CNN. We expected it to be appropriate for our basic model, as it was for CIFAR. We may change the pooling (for particular layers) when optimizing the algorithm.

We opted for padding as "same", which results in padding with zeros evenly to the left/right or up/down of the input. We have chosen to do this to ensure that the output has the same shape as the input. This worked well in CIFAR and we expect this to work well in this basic model as well. We may choose to change this aspect of the algorithm in later stages of the project.

After all these convolutional layers, a fully connected layer with X hidden nodes is applied before a SoftMax layer with X outputs for each of the X labels.

We applied a SoftMax activation function in the final layer for classification. We chose this because we are already familiar with how this function works from our previous ML projects. It is especially appropriate for the classification of multiple labels.

*Normalizing the input data*

The input data is normalized as part of the pre-processing of the data. The data pre-processor ImageDataGenerator is used for this purpose and applied to both the training and validation data.

It is important to center the data around the mean for deep neural networks such as what we build in this project because ReLU activations are only actually non-linear around an input of 0 (where the input switches from 0 to a linear output). This means that, for the ReLU activations to really be effective, we want the average combined inputs to end up around 0 in order to make use of the non-linear aspects of the activation function.

This pre-processing step can also prevent overfitting, since this normalizes the hidden mean and variance, to have some fixed mean and variance. This can prevent overfitting. ReLu functions have values between zero and Z. A standard deviation of 1 would also normalize this in order to prevent outliers so that the average deviations will be more in between this (Sd of 1). This is only a preventative measure against overfitting. We will specifically tackle the problem of overfitting by adding dropout when optimizing the algorithm.

*Validation accuracy*

The validation accuracy is 22%. This indicates that the network has learned a useful function, as a random prediction for the validation set would be expected to result in 8% accuracy.

Additionally, the learning curves indicate that the network has learned a useful function as the loss of the training data decreases and the accuracy seems to increase somewhat.

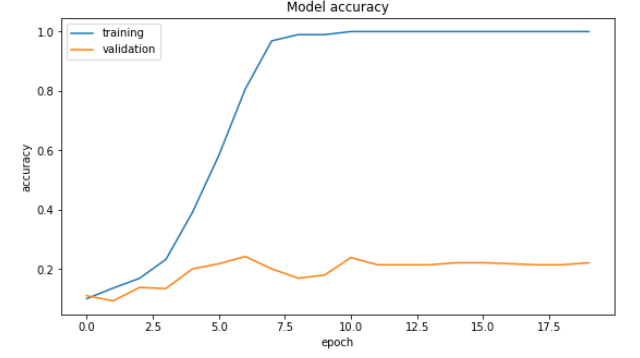
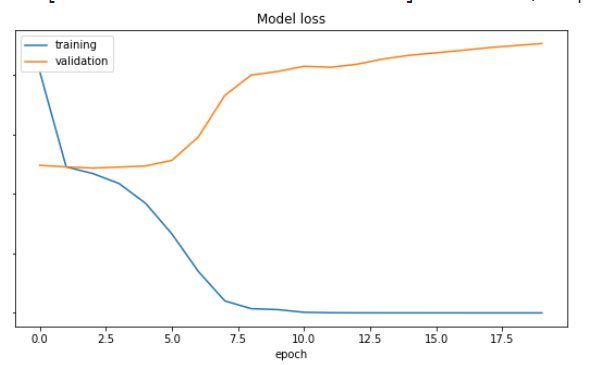


Figure Figure

1. Thapa, R., K. Zhang, N. Snavely, S. Belongie, and A. Khan. 2020. The Plant Pathology Challenge 2020 data set to classify foliar disease of apples. Applications in Plant Sciences 8(9): e11390. [↑](#footnote-ref-1)
2. Thapa, R., K. Zhang, N. Snavely, S. Belongie, and A. Khan. 2020. The Plant Pathology Challenge 2020 data set to classify foliar disease of apples. Applications in Plant Sciences 8(9): e11390. [↑](#footnote-ref-2)
3. Details concerning that dataset were published as a peer-reviewed research article: [Thapa, Ranjita; Zhang, Kai; Snavely, Noah; Belongie, Serge; Khan, Awais. The Plant Pathology Challenge 2020 data set to classify foliar disease of apples. Applications in Plant Sciences, 8 (9), 2020.](https://bsapubs.onlinelibrary.wiley.com/doi/10.1002/aps3.11390) [↑](#footnote-ref-3)
4. from the Kaggle competition description: <https://www.kaggle.com/c/plant-pathology-2021-fgvc8> [↑](#footnote-ref-4)
5. <https://www.baeldung.com/cs/train-test-datasets-ratio> [↑](#footnote-ref-5)
6. <https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy> [↑](#footnote-ref-6)
7. <https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam> [↑](#footnote-ref-7)
8. Kingma, Ba, Adam: A Method for Stochastic Optimization, <https://arxiv.org/abs/1412.6980> [↑](#footnote-ref-8)