Minor AI January 2022

Machine Learning Project

Nassim Bekkali, Rhea Groenenberg, Jeanice Koorndijk

Contents

Milestone 1

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

1.1 INTRODUCTION

1.1.1 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] 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] 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] 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

[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.
[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.
[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.

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]

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. In theory, one might expect that all combinations of diseases may possibly be identified in a leaf. It seems, however, that the Kaggle dataset is limited to particular combinations which results in 12 classes in total.

```
scab                                4826
healthy                             4624
frog_eye_leaf_spot                  3181
rust                                1860
complex                             1602
powdery_mildew                      1184
scab frog_eye_leaf_spot              686
scab frog_eye_leaf_spot complex      200
frog_eye_leaf_spot complex           165
rust frog_eye_leaf_spot              120
rust complex                          97
powdery_mildew complex                87
Name: labels, dtype: int64
```

Figure 1 and figure 2 (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:

[4] from the Kaggle competition description: https://www.kaggle.com/c/plant-pathology-2021-fgvc8

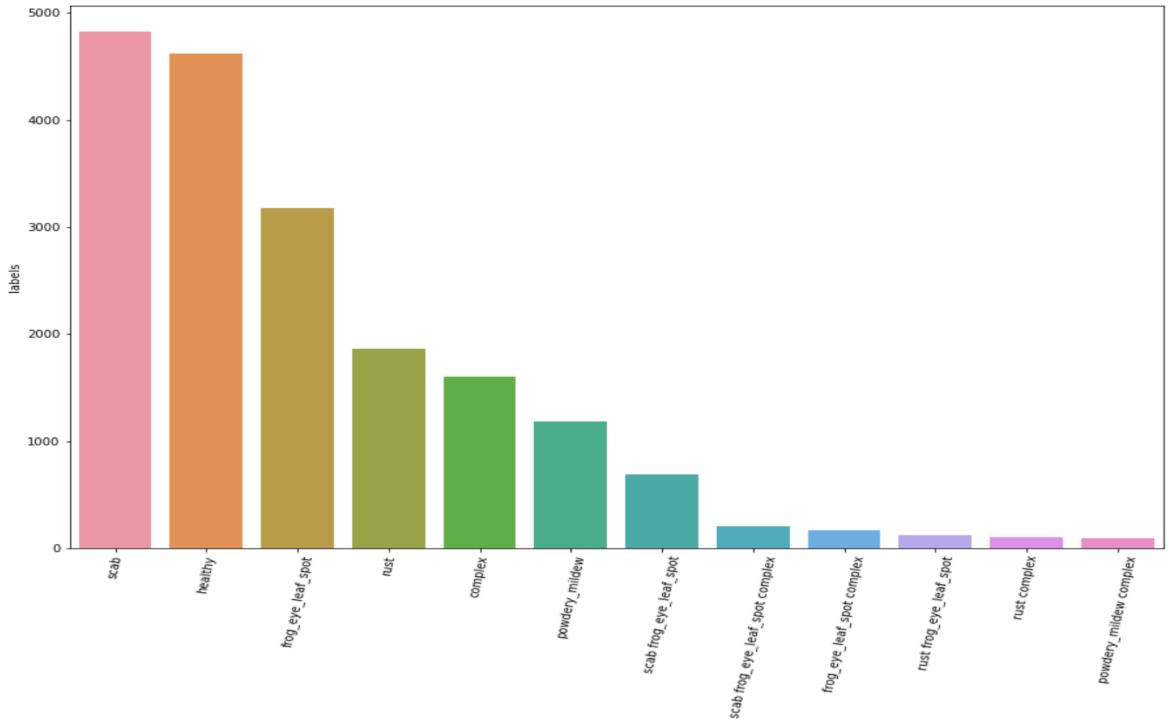


Figure 2 – bar plot visualisation data

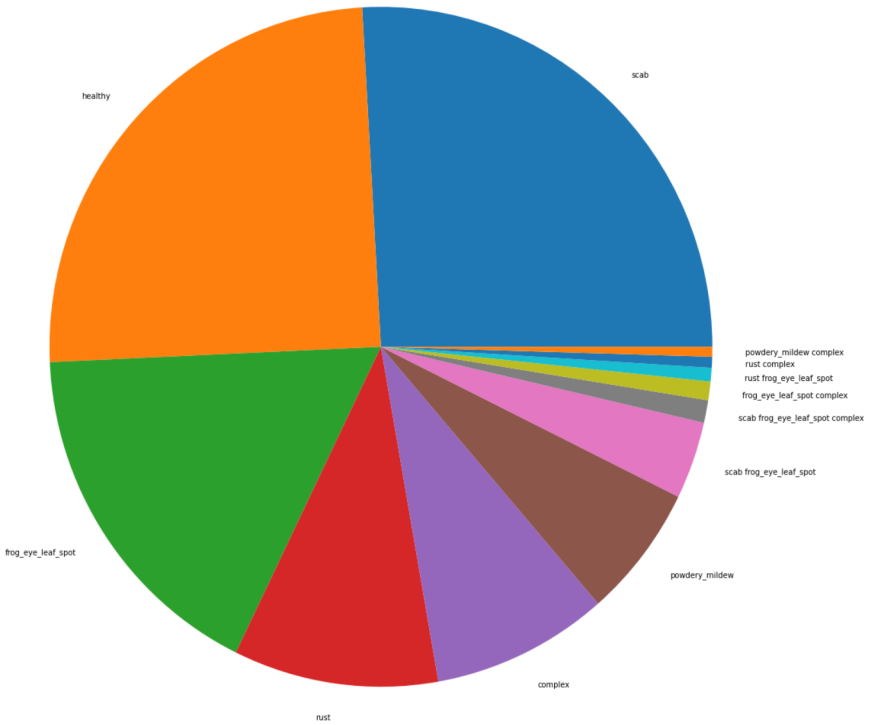


Figure 3 - circle diagram visualisation data

These visualisations indicate the imbalance in the dataset. In order to create a more balanced dataset, we decided to pre-process the dataset by selecting 80 random samples from every class. We selected 80 samples because the class with the least amount of

sample has 87, selecting 80 (< 87) is simply a way in which we ensure that every class has an equal number of samples.

The selection of 80 samples was also, most importantly, made because it would take too much time for the programme to run when going through all data. We ran into some limitations of the software, Google Colab, as the programme was unable to process all the data in the desired manner. Namely, the programme would crash. Our selection of 960 samples out of all 18.632 samples in the dataset, enables a balanced and faster programme.

This is a form of undersampling. We realise that changing the dataset in this way may affect the accuracy in unintended ways. By selecting 960 samples out of all 18.632, we use only 5% of the dataset. This results in a significant loss of data that the ML model could use to train on. This is not a problem for the basic dataset. However, we do intend to adapt the model and use more data in order to optimise the model. at later stages.

Additionally, we also may not need a dataset that is perfectly balanced. 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.

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 in the basic model. Yet, this will be increased in later stages of the project. In this case, a ratio of 70/30 (training data/validation data) will be used, because the data is relatively small (currently 960 but will likely increase to a number closer to 10 000): 100 < n < 10 000.[5]

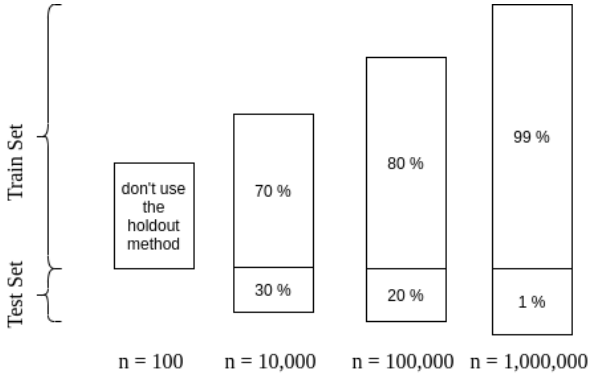


Figure 4 - visualisation recommended distribution of data

The following steps have been taken to load the data.

- Each image is resized to 96 x 96 pixels and the 3 color 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. We want to pick an image size that is not too large to process, but which enables the model to have enough information from the pixels to make meaningful predictions. In our case, we think that 96 x 96 pixels can function as a sweet spot where the size is not too large and not too small. Such model resizing allows for us to build a CNN

[5] https://www.baeldung.com/cs/train-test-datasets-ratio

model with enough layers to be identify complex features, yet not too large for our Google Collab programme to process. If it were to turn out that the resizing does not lead to the desired results, we will change it and opt for a different size.

- 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 Categorical cross-entropy loss function to compute the cross-entropy loss between the labels and predictions. In this case we have a multcassification model, this means that the cross-entropy formula looks like this:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

  - M - number of classes (rust, healthy, scab, etc.)
  - log - the natural log
  - y - binary indicator (0 or 1) if class label c is the correct classification for observation o
  - p - predicted probability observation oo is of class c

  The cross-entropy loss (log loss), measures the performance of a classification model[6]. The output is a probability somewhere between 0 and 1. The loss increases as the predicted probability diverges from the actual label. So in this case if the model predicts a probability of 0.0.12 when the actual label is 1 it would be bad and result in a high loss value. A perfect model would have a cross-entropy loss of 0.

- 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.[7]
- 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.[8] 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.[9]

[6] https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-entropy
[7] https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy
[8] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam
[9] Kingma, Ba, Adam: A Method for Stochastic Optimization, https://arxiv.org/abs/1412.6980

The function goes on to:

- Fit the compiled model using the preprocessed 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.

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.

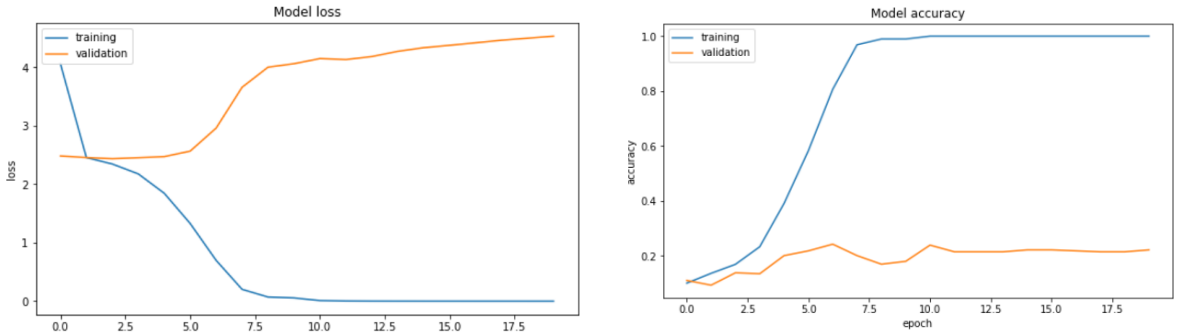


Figure 5 - model loss (left) and model accuracy (right)

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. For now, we have chosen this filter because we have used these numbers in CIFAR and that seemed to work quite well. It is our intention to consciously choose the number of filters and layers during the next milestone in order to optimise the model. 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. We think that the use of a non-negative activation function can be especially efficient at this point because setting all negative inputs to zero means that there is less data to be processed, so that less computations have to be made and the model can be trained more quickly. Since we are still at the beginning stages of the project, we may want to run many different models so a model that is trained quickly can allow for more experiments. We have also learned from the machine learning course (block 2), ReLu prevents the issue of Vanishing Gradient problem which makes the learning process very slow. In later stages of the project, we might change the activation function, and / or 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 optimising 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 256 hidden nodes is applied before a softmax layer with 12 outputs for each of the 12 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 normalised as part of the pre-processing of the data. The data preprocessor ImageDataGenerator is used for this purpose and applied to both the training and validation data.

It is important to centre 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 preprocessing step can also prevent overfitting, since this normalises 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 optimising the algorithm.

1.2.5 Model Summary

Overall, the model looks as follows:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 96, 96, 32)        896

max_pooling2d (MaxPooling2D (None, 48, 48, 32))       0


 conv2d_1 (Conv2D)           (None, 48, 48, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 24, 24, 64)       0
 2D)

 flatten (Flatten)           (None, 36864)             0

 dense (Dense)               (None, 256)               9437440

 dense_1 (Dense)             (None, 12)                3084

=================================================================
Total params: 9,459,916
Trainable params: 9,459,916
Non-trainable params: 0
```

1.3 DISCUSSION AND CONCLUSION

Overall, we have managed to build a simple model that made predictions that did not seem to be random. However, the model seemed to be overfitting. This became apparent by looking at the graphs from the train and evaluate function. The part where training accuracy is higher than the validation accuracy typically indicates overfitting. Additionally, overfitting is also indicated by the fact that the training loss seems to go down whereas the validation loss starts to go up. You'd expect the validation loss to go down over time (each epoch) as the model should be able to more accurately label the validation data. Since this is not the case, the model is overfitting and not generalising well. There are multiple ways to solve this issue such as increasing the sample size and adding dropout. We will apply these and other techniques during the next milestone in order to resolve the problem of overfitting on the following model:

```python
# Define Sequential model
model = models.Sequential()

# create convolutional layer and max pooling layer
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(96, 96, 3)))
model.add(layers.MaxPooling2D((2, 2)))

# create convolutional layer (larger) and max pooling layer
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))

# flatten layers
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))

# apply softmax activation for final layer classification
model.add(layers.Dense(12, activation='softmax'))

# normalize input data: set preprocesing dictionary
preprocess = {'featurewise_center': True, 'featurewise_std_normalization' : True}

# run training and evaluation function
train_and_evaluate(model, x_train, y_train, x_val, y_val, preprocess)
```

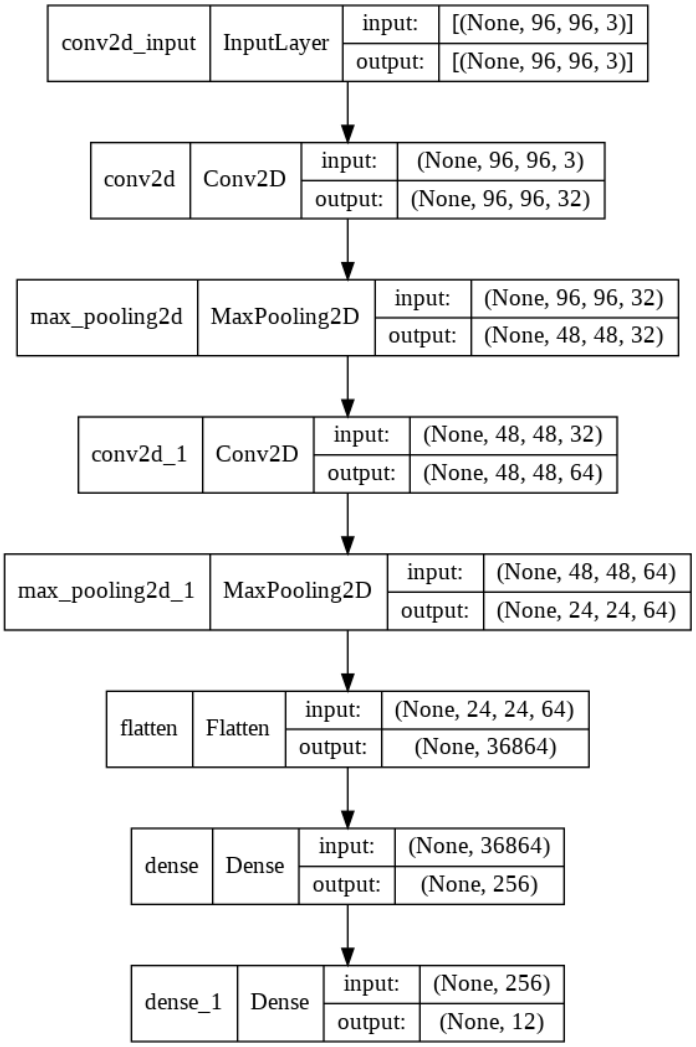


Figure 6 - final network-configuration of Milestone 1

Milestone 2

Deadline: Tuesday January 18th at 17:59

Here you should improve on the model from milestone 1 and at least make reasonable predictions in some of the cases. You can do a more in depth analysis of some part of the data, do a more thorough clean up, expand the size of your model to allow for more complex hypotheses and start to tweak some model parameters, selecting the best parameters on the validation set.

Meeting 1: Monday January 17th

Meeting 2: Wednesday January 19th

2.1 INTRODUCTION

2.1.1 Apple Tree Disease Project improvements

For this milestone, we improve on the basic model created in milestone 1 in two main ways. Firstly, we have set the goal to increase the model accuracy from 22% to at least 40%. This is simply a goal for us to ensure that we work towards the end goal of this project, high accuracy, during this milestone.

Despite the fact that the model made predictions that did not seem to be random, the model seemed to be overfitting. This became apparent by looking at the graphs from the train and evaluate function. Therefore, the second main objective for this part of the project is to resolve the problem of overfitting in order to tweak model parameters and create a deeper neural network in later stages of the project.

In order to meet our goals, we have increased the sample size and re-sampled the dataset in significant ways. We have also added dropout, batch normalisation, and leaky ReLU activations. In the section below, we elaborate on why we have chosen these techniques.

2.2 METHODS

2.2.1 Improved sample size and composed classes

In accordance with the received feedback for Milestone 1, we have chosen to adjust our data in order to improve sample size and the composition of classes.

Since the original data contains both single and combined classes, our previous model considered the combined classes as standalone individual classes resulting in a total of 12 classes. However, since one of the classes would only contain 87 samples total and because we wanted a balanced dataset in which every class had an equal amount of samples, we were limited to a sample-size of a maximum of 1044 samples which would only be 5.5% of the entire dataset. As mentioned in milestone 1, undersampling in this way results in a significant loss of data that could hinder the accuracy of the model. For this milestone, we have increased the data in our dataset in order to increase the accuracy.

We have chosen to restructure the class composition by merging the combined classes along with the main classes, resulting in a total of 6 classes. Combined classes were merged with the main class that was mentioned first in its label. For example scab frog_eye_leaf_spot complex would be merged with scab.

We have chosen this method, because according to the data-information there isn’t a particular order to the labels of the combined classes. This has also been proven by CV-scores computed by others, where, for example, 'scab healthy' and 'healthy scab' both result in the exact same score. This implies that both diseases are equally present in that

particular sample [10]. The CV score can be calculated using the following formula, where tp stands for true-positive and fn for false negative:

$$F1 = 2\frac{p \cdot r}{p + r} \text{ where } p = \frac{tp}{tp + fp}, \quad r = \frac{tp}{tp + fn}$$

This is called the micro-average F1 score, it can be computed by global precision and recalls scores from the sum of fp, fn, tp counts across classes. With all these variables we can use these global precision and recall scores to compute a global F1 score as their harmonic mean. This F1 score is named as the micro-average F1 score.

Although we are still undersampling, this method enables an increased sample size while ensuring that the data is still highly balanced. For the current model we have chosen to select a random sample from each class consisting of 1184 samples, which totals to 7104 samples total and thus 38.2% of the entire data-set. 1184 samples were chosen for each class because 1184 is the number of samples that is present in the smallest dataset. Additionally, a total of 7104 samples still allows for the programme to run rather quickly, which enables us to programme efficiently. We may increase the total sample size during later stages of the project.

2.2.2 Preprocessing and adding batch normalisation

From working with the MNIST Handwritten Image Classification Dataset, we learned that images need to be scaled in order to feed them as input for a neural network. Since we are working with Keras Tensorflow, we can use the ImageDataGenerator in order to scale images as part of preprocessing. We have decided to apply the featurewise_center and the featurewise_std_normalization. In this way, the feature wise input mean is set to 0 over the dataset. We first calculate the mean over the entire dataset and then subtract this mean from each image. So, this results in shifting the mean of the distribution **close to zero.**[11] Additionally, the feature wise inputs are divided by the standard deviation of the dataset. In this way, we divide each image by the standard deviation of the entire dataset. Thus, featurewise centre and std_normalization together known as standardisation tends to make the mean of the data to be 0 and std. deviation of 1.[12] This combination ensures a Gaussian Distribution of the data, which is a way to normalise the data. There are of course other ways to ensure the probability distribution of the data. We have applied the gaussian distribution to our model because we are most familiar with this distribution and they worked well in comparable projects like CIFAR. We may, however, change this aspect of our programme during later stages of the project.

In addition to this type of normalisation, we have applied batch normalisation. This applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. We have opted to use batch normalisation to combat overfitting because we

[10] Understanding the evaluation metric +CV; Python · Plant Pathology 2021 - FGVC8. https://www.kaggle.com/buinyi/understanding-the-evaluation-metric-cv?scriptVersionId=57391136

[11] https://theailearner.com/2019/07/06/data-augmentation-with-keras-imagedatagenerator/

[12] https://theailearner.com/2019/07/06/data-augmentation-with-keras-imagedatagenerator/

have learned that batch normalisation can be quite an effective generalisation technique for CNN's. Batch normalisation also ensures that the model improves at the beginning as the model improves and each epoch can train with an improved model, based on the normalisation. In this way, batch normalisation also contributes to our goal of increasing the accuracy of the model.

Additionally, batch normalisation enables the model to handle internal covariate shift. Batch normalisation makes the weights and parameters in later layers (e.g. layer 5) more sensitive to weights in previous weights which facilitates generalisation. The values in the internal layers basically keep changing. Therefore, if new images have a slightly different distribution from the previous images (e.g. white dogs instead of black dogs), the model will change its parameters according to these new images to still make accurate predictions.

Besides combating overfitting, normalising the input features to mean zero and variance 1, does this for each layer, which speeds up the training process because it reduces the number of training epochs required to train deep networks. That increased efficiency may save us time and allows us to do more testing throughout the project.

We implemented the batch normalisation layer after each MaxPooling layer. This will place the BatchNormalization layers before each of the layers with LeakyReLU activations, except for the input layer (which has already been normalised). We have added LeakyReLU activations because we have opted to add dropout to our programme. This is elaborated on in the next paragraphs.

2.2.3 Adding dropout and LeakyReLu

Dropout changes the network by randomly dropping neurons from the neural network during training in each iteration. We train the same network in different ways. It makes the model less dependent on the contents of specific pixels so that it reduces the chances of overfitting as the model becomes more robust so that it can better generalise.

Research suggests that "dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularisation".[13] "Dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets".[14] For this reason, we have chosen to use dropout to fight overfitting. We also recall other ML techniques such as batch normalisation which can be quite effective. For this reason, we did not rely solely on dropout but combined it with batch normalisation to improve the model.

It has been suggested that a common value is a probability of 0.5 for retaining the output of each node in a hidden layer and a value close to 1.0, such as 0.8, for retaining inputs from

[13] Page 265, Deep Learning, 2016.
[14] Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014. https://jmlr.org/papers/v15/srivastava14a.html

the visible layer.[15] “In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks. For the input units, however, the optimal probability of retention is usually closer to 1 than to 0.5.”[16]

We have chosen to start with simple values described above for the dropout layers, and we might improve upon this at later points in time. We added a Dropout layer before each of the Dense layers, both with a probability of 0.5. We wanted to maintain more of the network activations closer to the network input, so we only added a Dropout layer before the last Conv2D layer, with a lower dropping probability of 0.2.

The use of dropout has led us to also change the activation function to Leaky ReLu instead of ReLu as activation function within our model. This will allow for more nodes to remain activated, bypassing the chance of dead nodes by the small chance of having intermediate derivatives result in 0. Also, this will allow the network to learn much faster, since the activation of nodes would remain more balanced. Since our network isn’t greatly deep, such maintenance seems important. We have thought to combine the LeakyReLu along with the Dropout layers, in order to have a more controlled setting which might help better against overfitting.

After discussing our ideas with Wouter, we suspect that adding dropout may actually not affect the programme in significant ways. Batch normalisation may have a more significant effect on the accuracy. In order to assess this, we ran the programme with and without dropout. The findings can be found in the results section. The following section gives an overview of both models.

2.2.4 Model summary

In summary, the model (with dropout) looks as follows:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 96, 96, 32)        896
```

[15] https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/
[16] Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014. https://jmlr.org/papers/v15/srivastava14a.html

```
max_pooling2d (MaxPooling2D  (None, 48, 48, 32)       0
dropout (Dropout)            (None, 48, 48, 32)       0
conv2d_1 (Conv2D)            (None, 48, 48, 64)       18496
max_pooling2d_1 (MaxPooling  (None, 24, 24, 64)       0
2D)
batch_normalization (BatchN  (None, 24, 24, 64)       256
ormalization)
flatten (Flatten)            (None, 36864)            0
dropout_1 (Dropout)          (None, 36864)            0
dense (Dense)                (None, 256)              9437440
dropout_2 (Dropout)          (None, 256)              0
dense_1 (Dense)              (None, 6)                1542
=================================================================
Total params: 9,458,630
Trainable params: 9,458,502
Non-trainable params: 128
```

In summary, the model (without dropout) looks as follows:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 96, 96, 32)        896
 max_pooling2d_2 (MaxPooling  (None, 48, 48, 32)       0
 2D)
 conv2d_3 (Conv2D)           (None, 48, 48, 64)        18496
 max_pooling2d_3 (MaxPooling  (None, 24, 24, 64)       0
 2D)
 batch_normalization_1 (Batc  (None, 24, 24, 64)       256
 hNormalization)
 flatten_1 (Flatten)         (None, 36864)             0
 dense_2 (Dense)             (None, 256)               9437440
 dense_3 (Dense)             (None, 6)                 1542
=================================================================
Total params: 9,458,630
Trainable params: 9,458,502
Non-trainable params: 128
```

2.3 RESULTS

The changes above resulted in the following model loss and model accuracy for the model with dropout:

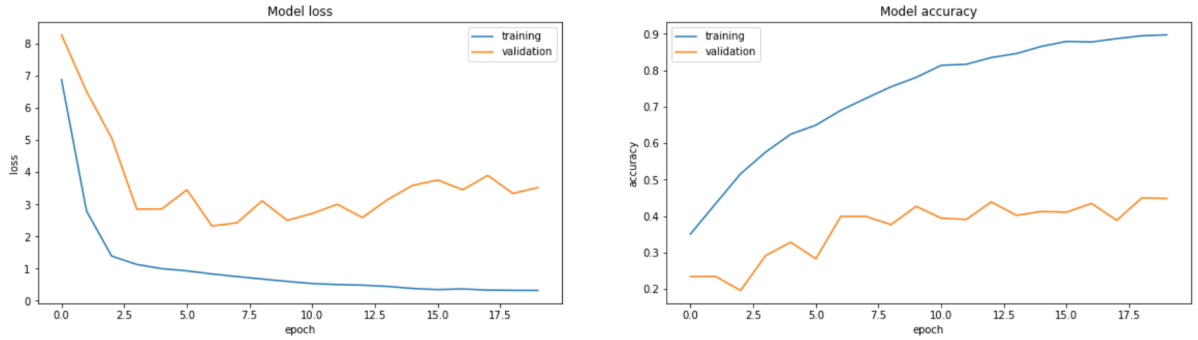


Figure 7 - model loss (left) and model accuracy (right)

The associated confusion matrix of the model looks as follows:

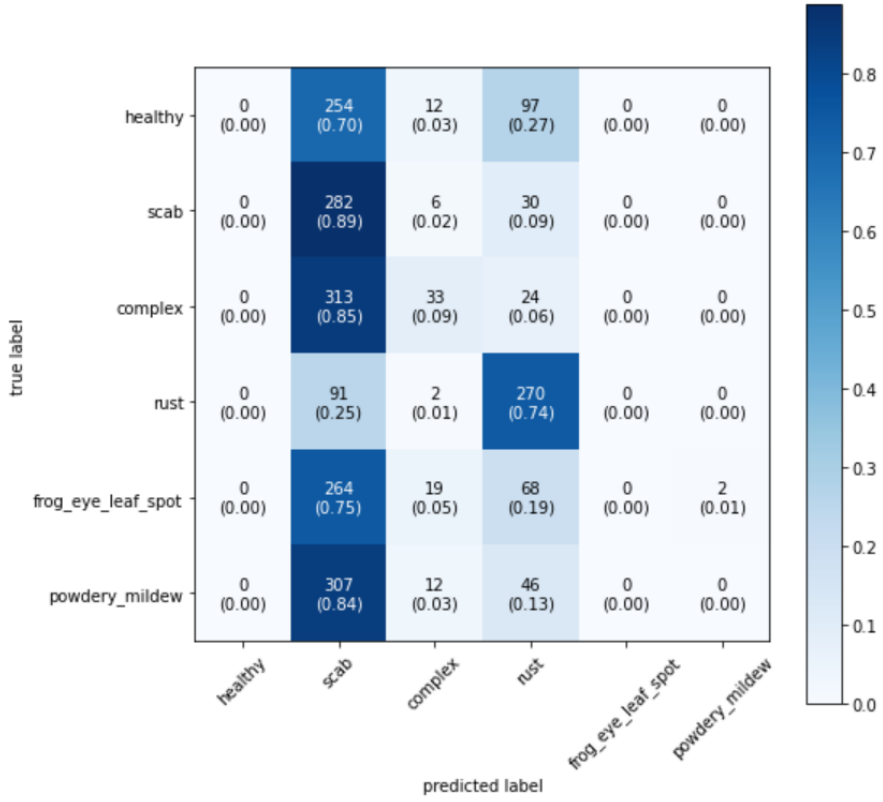
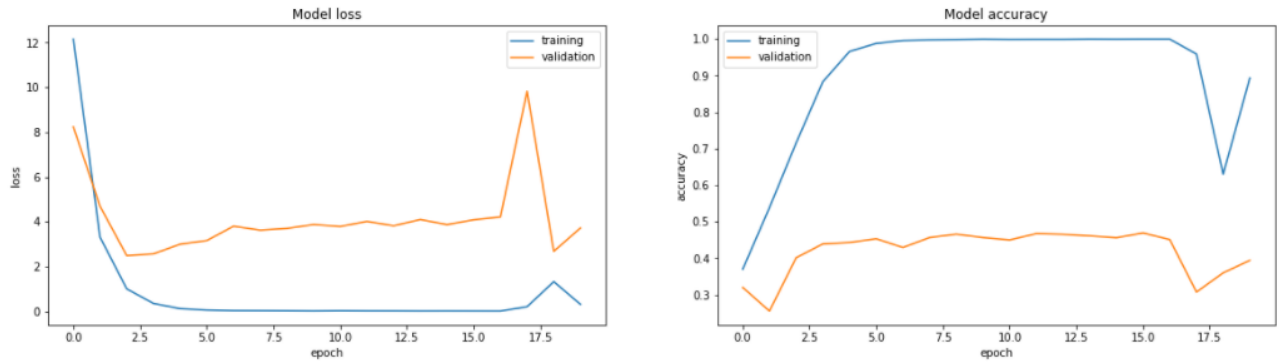


Figure 8 – top to bottom: array of labels, confusion matrix, numerical values confusion matrix

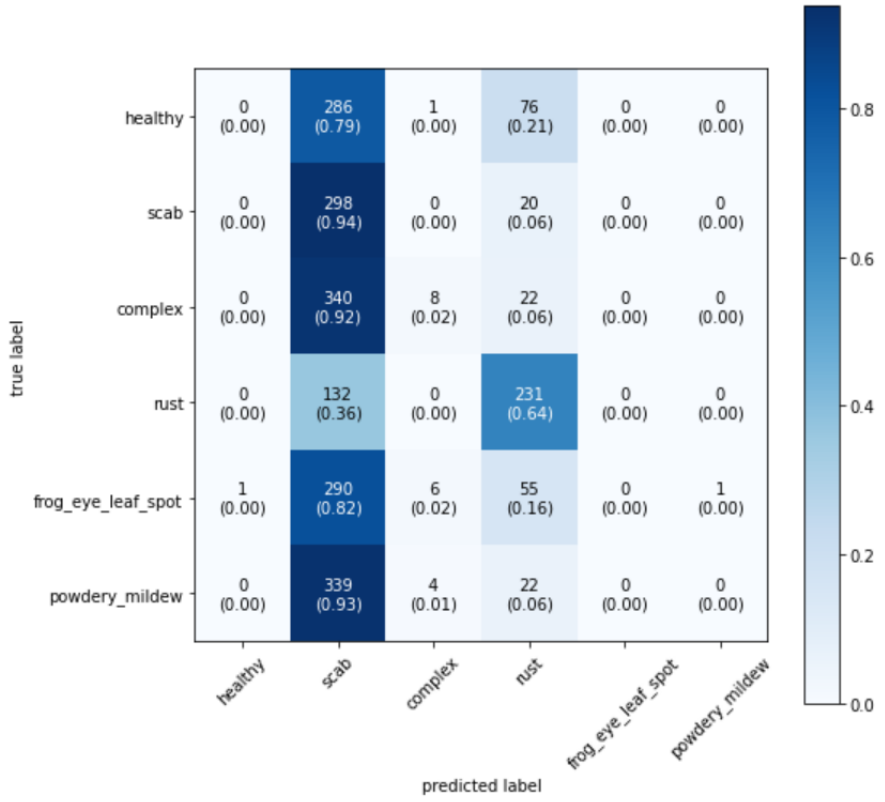
The applied techniques resulted in the following model loss and model accuracy for the model without dropout:




```
67/67 [==============================] - 1s 15ms/step - loss: 3.7198 - accuracy: 0.3940
Validation Accuracy: 0.3939962387084961
```

Figure 9 - model loss (left) and model accuracy (right)

The associated confusion matrix of the model looks as follows:



2.4 DISCUSSION AND CONCLUSION

The model accuracy has increased to 44.8% for the model with dropout. This means that we have met our goal of increasing the accuracy to at least 40%. The model is also no longer overfitting at the beginning of the plot. This is because the model accuracy plot shows that the training accuracy is increasing much more smoothly, Especially at the beginning of the model loss plot.

We suspect that the increased accuracy is the result of the combination of changes that we made, especially the improved sample size and the batch normalisation. Dropout does not seem to have a significant effect on the accuracy, as the model has a 44.5% accuracy. The difference in accuracy of 5% could be due to chance. Since it is still an increase of 0.05, we have decided to keep the dropout and not remove it from the model. We don't think that it can hurt. We might decide differently in later stages of the project.

We do note, however, that the model accuracy on the training data is very high at the end of the plot and the model loss on the training data is very low. Contrastingly, the model loss of the validation data is going up after about 7 epochs and from that point onward the model accuracy on the training data is not really improving much either. We suspect that the model needs to learn more high level features in order to increase performance at this point. This becomes especially apparent when considering the confusion matrix. The confusion matrix indicates that the model has learned to predict rust relatively well. This indicates that the model has truly learned some function. Given the fact that the structure of the model is still quite basic, rust may be the easiest class to predict. However, when the model is not predicting rust, it seems to classify all input mostly as scab.

The plots and matrices cause us to think that the model needs to be adapted to also recognize the other classes. We know that we can enable the network to learn more high level features by making the network deeper. Additionally, we could apply data augmentation in order to artificially increase the size of the data and enable the network to learn interesting features based on augmentation. This may enable it to predict classes other than rust more accurately.

Just to be certain, we ran the numbers by doing some calculations to ensure that the confusion matrices accurately reflect the accuracies of the models. We calculated the ratio of accurate predictions as reflected by the matrices: # correct predictions / # all predictions. We then compared this to the accuracy as described by the model plots. According to our calculations, the accuracy of the matrices do not reflect the accuracies of the plots. For example, our calculations suggest that the accuracy as reflected by the matrix of the model without dropout, is around 0.26. Yet, the model plots suggest that the accuracy lies around 0.39.

The model plots do not seem odd or inaccurate to us. The programming of the model plots is based on our "train and evaluate function" as part of the network code. Contrastingly, we have programmed the matrices separately, which leaves room for some error in the programming. For this reason, we suspect that there is some error in the code of the confusion matrices and we will reconsider this during the next milestone(s). For now, we are cognisant of the inaccuracies in our data and we only take these values with a grain of salt. Since these aspects of our data may likely be incorrect, we will try not to make too many conclusions on the basis of this data, before resolving this issue. As for now, however, we'll carefully stick with the conclusion based on the plots, in which the model's accuracy has

increased to 44.8% due to the dropout-layers. Additionally, we still suspect that it would be beneficial to attempt to improve the model by enabling the network to learn more high level features by making the network deeper along with the use of data augmentation.
Unfortunately, we could not do more experiments with the data because we reached the maximum capacity for the daily storage limitations in Collab. Since we reached our goals, this is not a problem. We remain cognisant of storage limitations for the next milestones in order to ensure that we do all relevant experiments. In the next milestone, we will focus on improving the following network further by implementing these techniques and conducting further research. Overall, the model that we will build upon further during the next milestone looks as follows:

```
# Define Sequential model
model = models.Sequential()

# create convolutional layer and max pooling layer

model.add(layers.Conv2D(32, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same', input_shape=(96, 96, 3)))
model.add(layers.MaxPooling2D((2, 2)))

# create convolutional layer (larger) and max pooling layer
model.add(tf.keras.layers.Dropout(0.5))
model.add(layers.Conv2D(64, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.BatchNormalization())

# flatten layers
model.add(layers.Flatten())
model.add(tf.keras.layers.Dropout(0.2))
model.add(layers.Dense(256, activation=tf.keras.layers.LeakyReLU(alpha=0.5)))

# apply softmax activation for final layer classification
model.add(tf.keras.layers.Dropout(0.2))
model.add(layers.Dense(6, activation='softmax'))

# normalize input data: set preprocesing dictionary
preprocess = {'featurewise_center': True, 'featurewise_std_normalization' : True}

# run training and evaluation function
train_and_evaluate(model, x_train, y_train, x_val, y_val, preprocess)
```

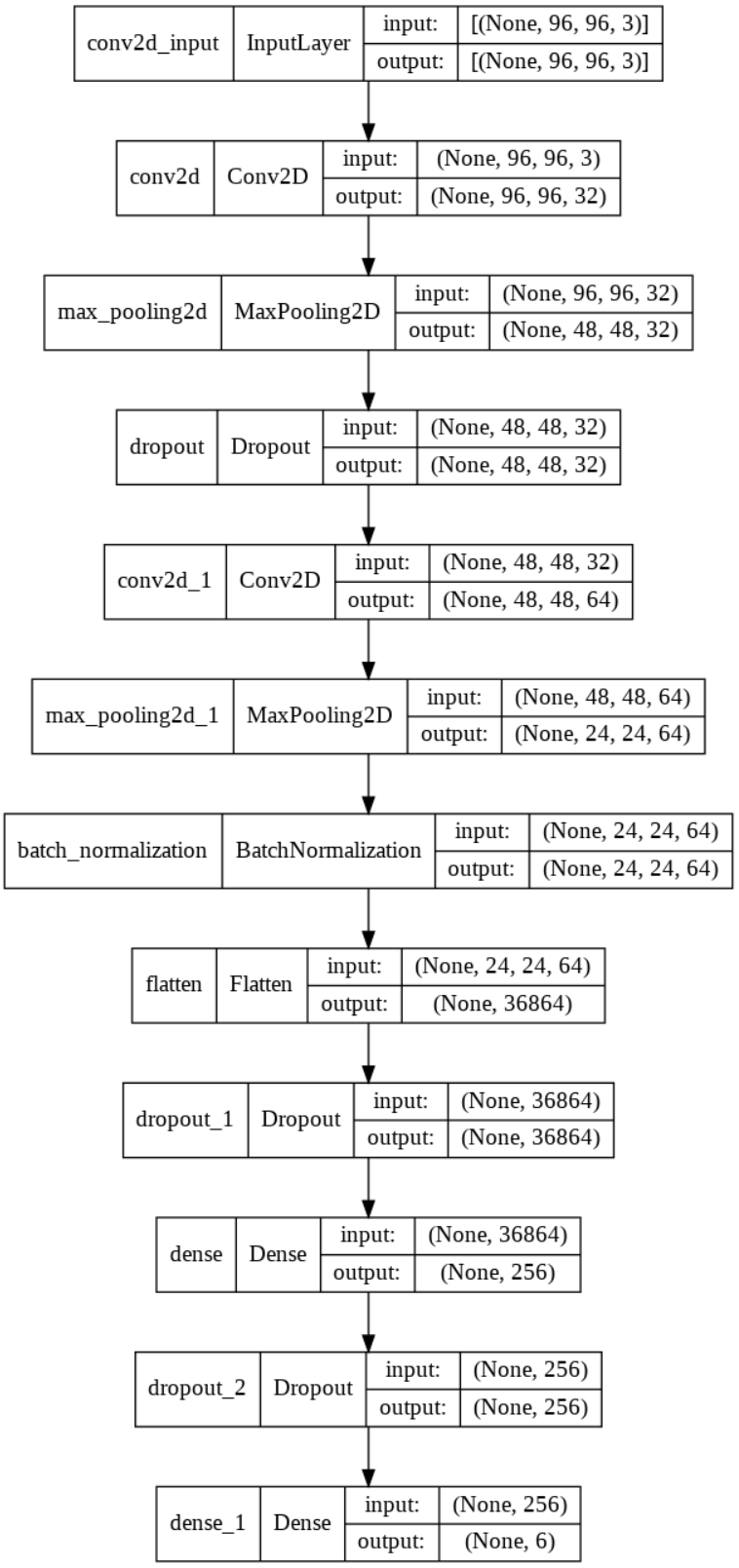


Figure 10 - final network-configuration of Milestone 2

Milestone 3

Deadline: Friday January 21st at 17:59

Assuming you get reasonable results in your milestone 2 model, the changes you try here will be more specific to your project and your results. You could analyse the model results and diagnose problems like under/overfitting, and then change your model appropriately to mitigate that problem. You could try to add additional data using data augmentation or create your own new features based on insight into the problem domain, or analysis you did of the data. The goal, starting from this milestone, will simply be to try and find ways to lower the error on your validation set.

Meeting 1: Thursday January 20th
Meeting 2: Monday January 24th

3.1 INTRODUCTION

3.1.1 Further improvements to the Apple Tree Disease Project

In the previous milestone, we have (partially) addressed the problem of overfitting and increased the average accuracy of the model to 40%. The analysis of our model indicated that, when the model is not predicting rust, it seems to classify all input mostly as scab. In this milestone, we will focus on mitigating this particular problem as well as the continued attention to overfitting.

Our goal for this milestone is to ensure that both plots of the training and validation accuracies increase in tandem, portraying a more smooth curve. Additionally, we want to train the model to accurately recognise the other classes. This means that we will not only focus on average accuracy, but also keep track of the accuracy for specific labels.

We will attempt to reach the goals for this milestone by enabling the network to learn more high level features by making the network deeper. Additionally, we apply data augmentation in order to artificially increase the size of the data and enable the network to learn interesting features based on augmentation. In the section below, we give a more detailed explanation on how and why we implemented these techniques.

3.2 METHODS

Since conducting experiments with data augmentation and changing the network configuration can take a lot of time and memory space (which can be quite limited), we have split the experiments into two main parts. One part of the group focused on finding ways to improve the general accuracy of the model by changing the network configuration. The other part of the group executed experiments with data augmentation, based on the basic model. We then combined the methods that seemed to be the most effective. Both methods are separately discussed first. The results of both methods and the results of the combination of methods are discussed in the result section, 3.3.

3.2.1 Making the Network deeper and changing the network configuration

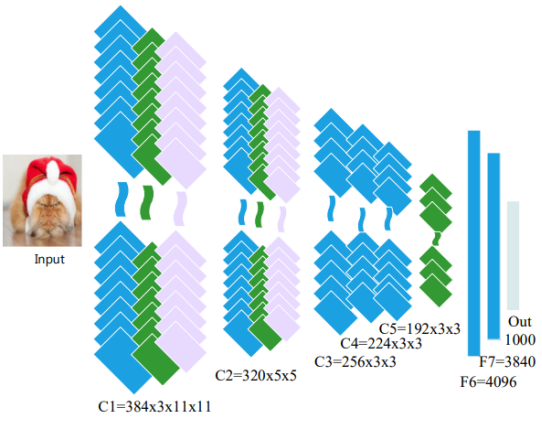
In milestone 2, we explained that we suspected that the model needed to learn more high level features in order to increase performance at this point. Our suspicion was based on the model loss and accuracy plots, as well as the confusion matrix.

One way to enable the model to learn more high level features is to simply deepen the network. This seems like the most intuitive next step for us to take because of the current simplicity of our model. Adding two or three additional layers could change the size of the model by over 50%. From our experience with other machine learning projects, we have learned that this could have significant effects on model performance.

Deepening the network can be achieved by adding extra layers, for example two Conv2D layers. We cannot analytically calculate the number of layers or nodes that are necessary to improve performance in a particular way. For this reason, we have simply tested a different number of layers and nodes and kept track of performance. The results section displays the results for the different configurations of the network.

A study suggests that:

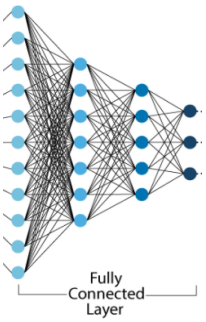
> "Imposing strict pyramidal structure should not only retain and improve accuracy, but also retain/improve accuracy and reduce the number of parameters which results in less [...] memory space on the disk. And this makes CNNs more feasible for applications where there is [a] lack of memory."[17]



Figuur 11 - Pyramidal structure for CNN

Figure 11 gives an example of a pyramid structure for a CNN. We decided that a pyramid structure would be appropriate for our model because of the limited memory space that Google collab provides. Such network configuration could help us to make optimal use of the memory that is available to us. Additionally, we have many input nodes and at most 12 nodes of output in accordance with the classes. For this reason, the network structure would have to decrease the number of nodes in any case, upholding a strict pyramid shape will only work towards this end while contributing to an effective model.

Our experience from a previous machine learning project on the CIFAR database suggests that the use of a diamond-like shape for convolutional neural networks can be quite effective. This is similar to a pyramid structure, although the number of filters are first doubled and later decreased. There was no literature to suggest that one is better than the other, or that both cannot be combined. For this reason, we have decided to take inspiration from a pyramid structure as well as from our previous ML projects. We first doubled the number of filters, and then decreased



[17] https://arxiv.org/pdf/1608.04064.pdf - About Pyramid Structure in Convolutional Neural Networks

them by 50% for each new layer. This ensures a diamond shape that includes the pyramid structure. See for example the following figure (X):[18]

We have decided to only adjust filters in order to assess those effects clearly. This entails that we maintained the same structure as the last convolutional layer. Each layer has a dropout layer (with a dropout rate that is similar to the layer before it) and a MaxPooling layer that comes after it.

A deeper network requires more opportunity to train. For this reason, we have decided to increase the number of epochs to 80 (= current standard in the function train_and_evaluate). This number of epochs seems enough to clearly indicate the accuracy of the model when it is fully trained, without taking too much time to run.

We incrementally changed the configuration of the model, working towards the diamond shape. We evaluated the model at every (big) step, in order to inform us about performance and to maybe guide us towards some other logical changes. Since we are now also interested in improving the performance of the model for particular classes. We rely especially on the confusion matrix to clearly indicate the accuracy of the model for specific classes. As noted before, however, the information from the confusion matrices is taken with a grain of salt until we can be certain that the matrices are correct. A short overview of the results of relevant changes can be found in the results section below.

After finding what seemed like a significantly more effective amount of layers, we were curious about the effects that the max pooling layers had on the accuracy of the model. Additionally, we also experimented with the learning rate in order to assess whether this affected the speed of learning and the accuracy of the model. Some of these results are documented in the results section and discussed briefly in the discussions section.

3.2.2 Data augmentation

We have decided to apply data augmentation because this can benefit our model in ways that complement the regularisation techniques applied beforehand. Since the augmentations can be applied to every class equally, our dataset remains balanced whilst we can significantly increase the amount of training data. We expected data augmentation to improve the model by further preventing overfitting as the model may be able to learn new useful features for particular diseases, based on the additional information provided by the augmentations.

There are many different augmentations that we could apply.[19] In the literature, we have come across geometric transformations, flipping, colour space, cropping, rotation, noise injection, and more.[20] The literature does not seem to suggest that one type of augmentation is superior to another, especially for our task of image classification for apple tree diseases. Notably, a specific study on image classification for apple tree diseases has

[18] source picture: https://www.analyticsvidhya.com/blog/2021/05/20-questions-to-test-your-skills-on-cnn-convolutional-neural-networks/

[19] Tensorflow augmentations: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

[20] Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60 (2019). https://doi.org/10.1186/s40537-019-0197-0.

used basic augmentation types available on Tensorflow to create a model with 98.42% accuracy. The study applied rotation (20), horizontal and vertical flips (True), and width and height shifts (0.1).[21]

We have decided to start experimenting with the application of these augmentations for our model because the model in the study is quite similar to ours. This may entail that our model may also significantly improve the accuracy of our model for particular classes. For example, it utilises a sequential model, max pooling, and dropout. It also has a similar amount of layers as our model. There are some differences between the model, relating to the model structure, dropout rates (0.2 vs. 0.5 and 0.2), activation functions (ReLU vs. Leaky ReLu), and more. We did not change those aspects of our model in order to ensure the authenticity of our own model. Another difference is that the data used in this research was slightly different than ours. From speculation it seems that Kaggle tweaks their website's data yearly by removal or addition of samples; however, we have still chosen to take inspiration from this study, since it provided the beginning of a potential path of experiments leading to the optimality of our network.

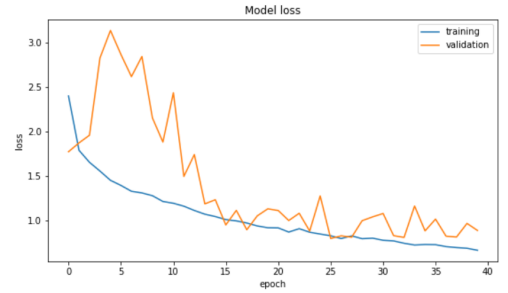
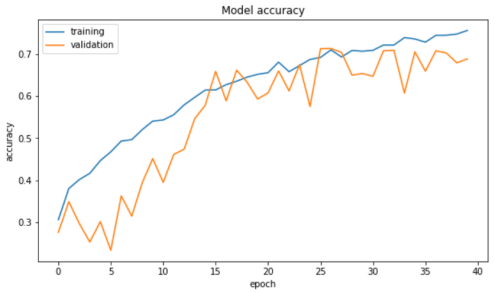
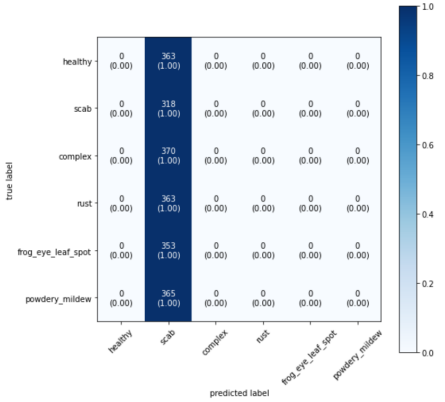
Afterwards, we have also experimented with other augmentations, such as zoom range, brightness_range and channel_shift_range; relevant results are included in the results and discussion section. Here too, we rely on the plots for the model loss and model accuracy, as well as the confusion matrix to evaluate the performance of our model. A short overview of the results of relevant changes can be found in the results section below.

[21] Saraansh Baranwal, Siddhant Khandelwal, Anuja Arora. Deep Learning Convolutional Neural Network for Apple Leaves Disease Detection, Department of CSE and IT, Jaypee Institute of Information Technology, Noida – 201309, India, 263.

3.3 RESULTS

The results section first discusses the results for the deepened network and the data augmentation separately. It then highlights the results of the combined methods. Note that we decided not to include each and every plot and matrix that resulted from different experiments, such as the many different data augmentations. We mostly included plots that we deemed relevant for our discussion and to give the reader an idea of significant results.

3.3.1 important results for the deepened network

Trial ID	model.summary	plot / matrix
1_19122 added 2 layers to the basic model from milestone 2 (1_18122). 1 layer with 128 filters, and batchnorm (128 to double number of filters) 1 layer with 256 layers, no batchnorm (256 to double number of filters)	Model: "sequential_2" _____ Layer (type) Output Shape Param # ===== conv2d_4 (Conv2D) (None, 96, 96, 32) 896 max_pooling2d_4 (MaxPooling (None, 48, 48, 32) 0 2D) dropout_6 (Dropout) (None, 48, 48, 32) 0 conv2d_5 (Conv2D) (None, 48, 48, 64) 18496 max_pooling2d_5 (MaxPooling (None, 24, 24, 64) 0 2D) batch_normalization_2 (Batc (None, 24, 24, 64) 256 hNormalization) dropout_7 (Dropout) (None, 24, 24, 64) 0 conv2d_6 (Conv2D) (None, 24, 24, 128) 73856 max_pooling2d_6 (MaxPooling (None, 12, 12, 128) 0 2D) batch_normalization_3 (Batc (None, 12, 12, 128) 512 hNormalization) dropout_8 (Dropout) (None, 12, 12, 128) 0 conv2d_7 (Conv2D) (None, 12, 12, 256) 295168 max_pooling2d_7 (MaxPooling (None, 6, 6, 256) 0 2D) dropout_9 (Dropout) (None, 6, 6, 256) 0 conv2d_8 (Conv2D) (None, 6, 6, 64) 147520 max_pooling2d_8 (MaxPooling (None, 3, 3, 64) 0 2D) flatten_2 (Flatten) (None, 576) 0 dropout_10 (Dropout) (None, 576) 0 dense_4 (Dense) (None, 256) 147712 dropout_11 (Dropout) (None, 256) 0 dense_5 (Dense) (None, 6) 1542 ===== Total params: 685,958 Trainable params: 685,574 Non-trainable params: 384	  67/67 [==========] - 1s 16ms/step - loss: 0.8880 - accuracy: 0.6881 Validation Accuracy: 0.6880863308906555 

2_19122

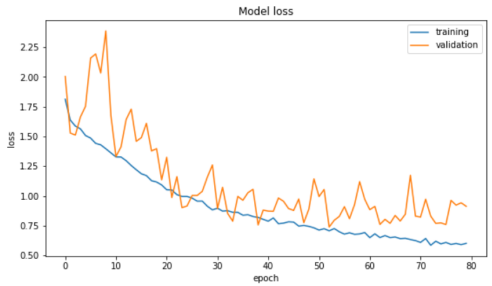
added 3 layers to the basic model from milestone 2 (1_18122).

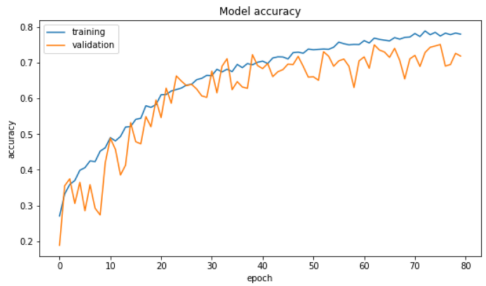
1 layer with 128 filters, and batchnorm (128 to double number of filters)

1 layer with 256 layers, no batchnorm (256 to double number of filters)

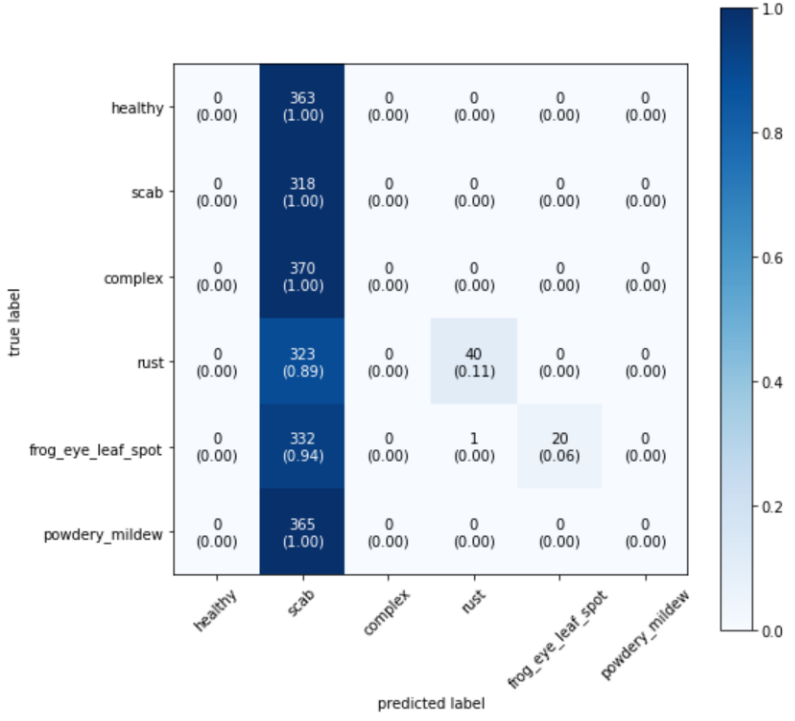
1 layer with 32 filters, no batchnorm

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 96, 96, 32)        896
 max_pooling2d_15 (MaxPoolin  (None, 48, 48, 32)       0
 g2D)
 dropout_19 (Dropout)        (None, 48, 48, 32)        0

 conv2d_16 (Conv2D)          (None, 48, 48, 64)        18496
 max_pooling2d_16 (MaxPoolin  (None, 24, 24, 64)       0
 g2D)
 batch_normalization_6 (Batc  (None, 24, 24, 64)       256
 hNormalization)
 dropout_20 (Dropout)        (None, 24, 24, 64)        0

 conv2d_17 (Conv2D)          (None, 24, 24, 128)       73856
 max_pooling2d_17 (MaxPoolin  (None, 12, 12, 128)      0
 g2D)
 batch_normalization_7 (Batc  (None, 12, 12, 128)      512
 hNormalization)
 dropout_21 (Dropout)        (None, 12, 12, 128)       0

 conv2d_18 (Conv2D)          (None, 12, 12, 256)       295168
 max_pooling2d_18 (MaxPoolin  (None, 6, 6, 256)        0
 g2D)
 dropout_22 (Dropout)        (None, 6, 6, 256)         0

 conv2d_19 (Conv2D)          (None, 6, 6, 64)          147520
 max_pooling2d_19 (MaxPoolin  (None, 3, 3, 64)         0
 g2D)
 dropout_23 (Dropout)        (None, 3, 3, 64)          0

 conv2d_20 (Conv2D)          (None, 3, 3, 32)          18464
 max_pooling2d_20 (MaxPoolin  (None, 1, 1, 32)         0
 g2D)

 flatten_4 (Flatten)         (None, 32)                0
 dropout_24 (Dropout)        (None, 32)                0
 dense_8 (Dense)             (None, 256)               8448
 dropout_25 (Dropout)        (None, 256)               0
 dense_9 (Dense)             (None, 6)                 1542

=================================================================
Total params: 565,158
Trainable params: 564,774
Non-trainable params: 384
```






```
67/67 [==============================] - 1s 18ms/step - loss: 0.9126 - accuracy: 0.7190
Validation Accuracy: 0.7190431356430054
```

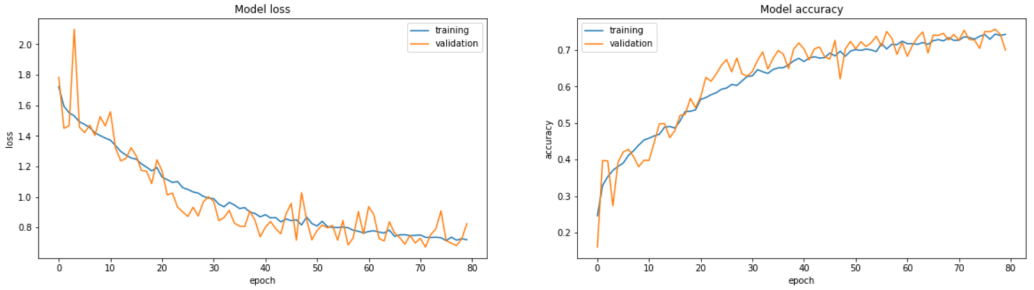


3_19122

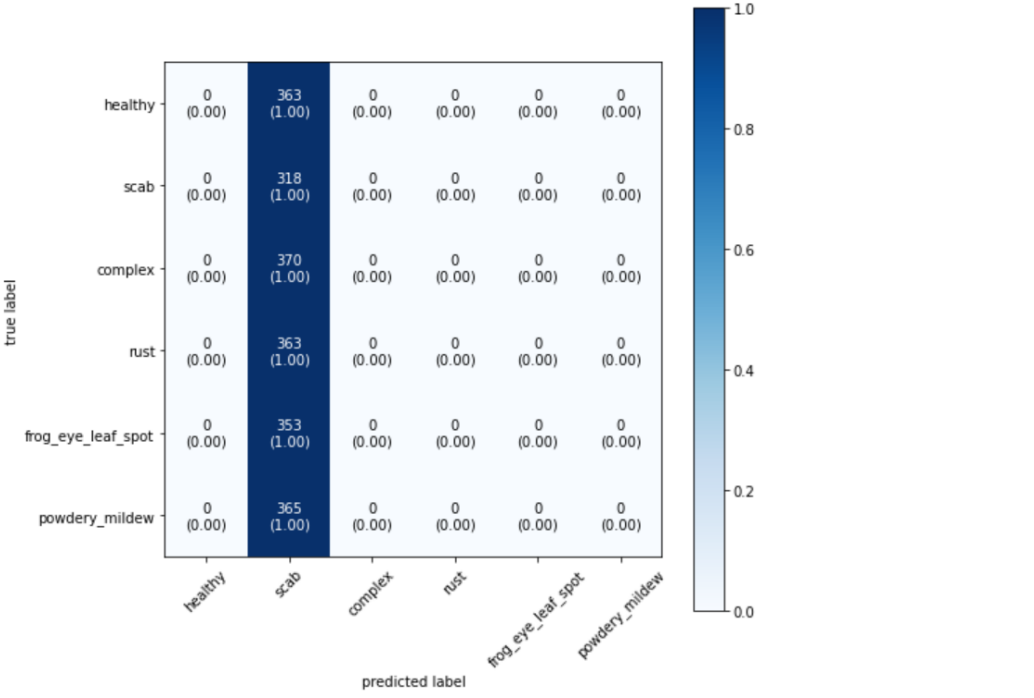
Added 1 layer to the model from before (ID: 2_19122)

1 layer with 8 filters, no max pooling and no batchnorm.

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_41 (Conv2D)          (None, 96, 96, 32)        896
 max_pooling2d_41 (MaxPoolin  (None, 48, 48, 32)       0
 g2D)
 dropout_45 (Dropout)        (None, 48, 48, 32)        0

 conv2d_42 (Conv2D)          (None, 48, 48, 64)        18496
 max_pooling2d_42 (MaxPoolin  (None, 24, 24, 64)       0
 g2D)
 batch_normalization_14 (Bat  (None, 24, 24, 64)       256
 chNormalization)
 dropout_46 (Dropout)        (None, 24, 24, 64)        0

 conv2d_43 (Conv2D)          (None, 24, 24, 128)       73856
 max_pooling2d_43 (MaxPoolin  (None, 12, 12, 128)      0
 g2D)
 batch_normalization_15 (Bat  (None, 12, 12, 128)      512
 chNormalization)
 dropout_47 (Dropout)        (None, 12, 12, 128)       0

 conv2d_44 (Conv2D)          (None, 12, 12, 256)       295168
 max_pooling2d_44 (MaxPoolin  (None, 6, 6, 256)        0
 g2D)
 dropout_48 (Dropout)        (None, 6, 6, 256)         0

 conv2d_45 (Conv2D)          (None, 6, 6, 64)          147520
 max_pooling2d_45 (MaxPoolin  (None, 3, 3, 64)         0
 g2D)
 dropout_49 (Dropout)        (None, 3, 3, 64)          0

 conv2d_46 (Conv2D)          (None, 3, 3, 32)          18464
 max_pooling2d_46 (MaxPoolin  (None, 1, 1, 32)         0
 g2D)
 dropout_50 (Dropout)        (None, 1, 1, 32)          0

 conv2d_47 (Conv2D)          (None, 1, 1, 8)           2312

flatten_6 (Flatten)         (None, 8)                 0
dropout_51 (Dropout)        (None, 8)                 0
dense_12 (Dense)            (None, 256)               2304
dropout_52 (Dropout)        (None, 256)               0
dense_13 (Dense)            (None, 6)                 1542

=================================================================
Total params: 561,326
Trainable params: 560,942
Non-trainable params: 384
```

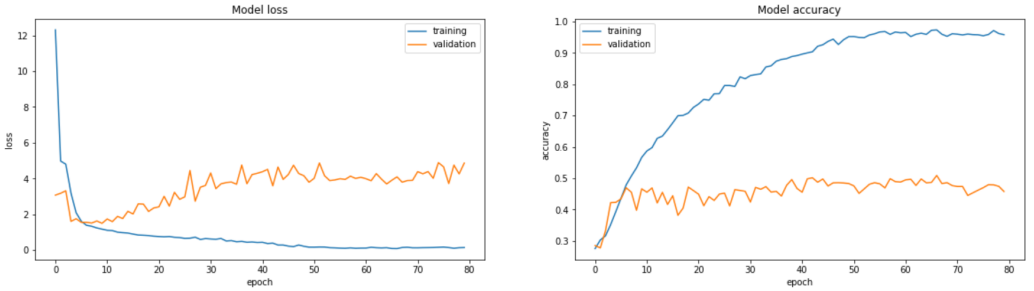



```
67/67 [==============================] - 1s 20ms/step - loss: 0.8249 - accuracy: 0.6998
Validation Accuracy: 0.6998124122619629
```



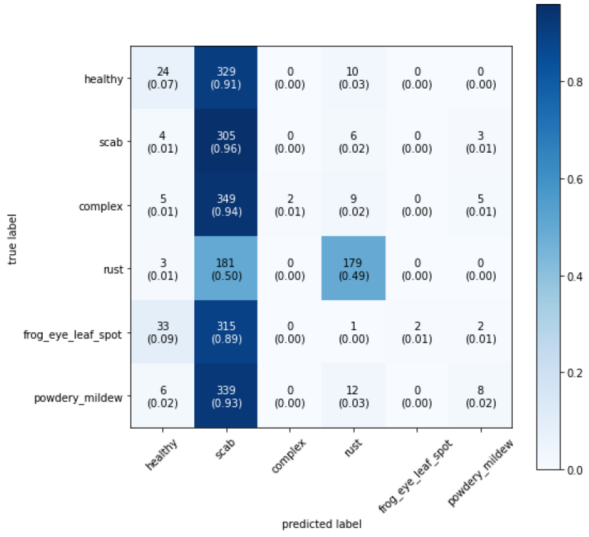
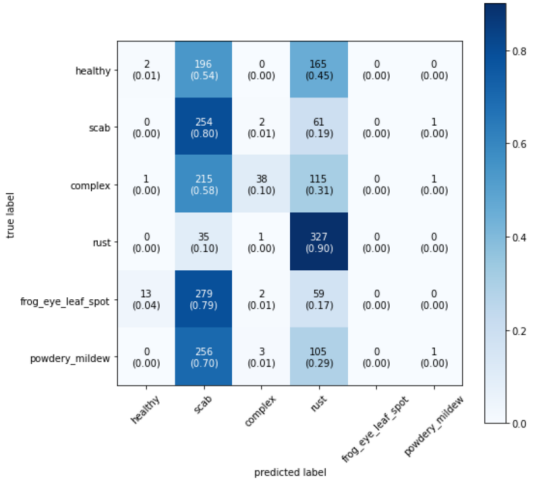
Took the most effective model so far, 2_19122, and removed the MaxPooling layers at the beginning.

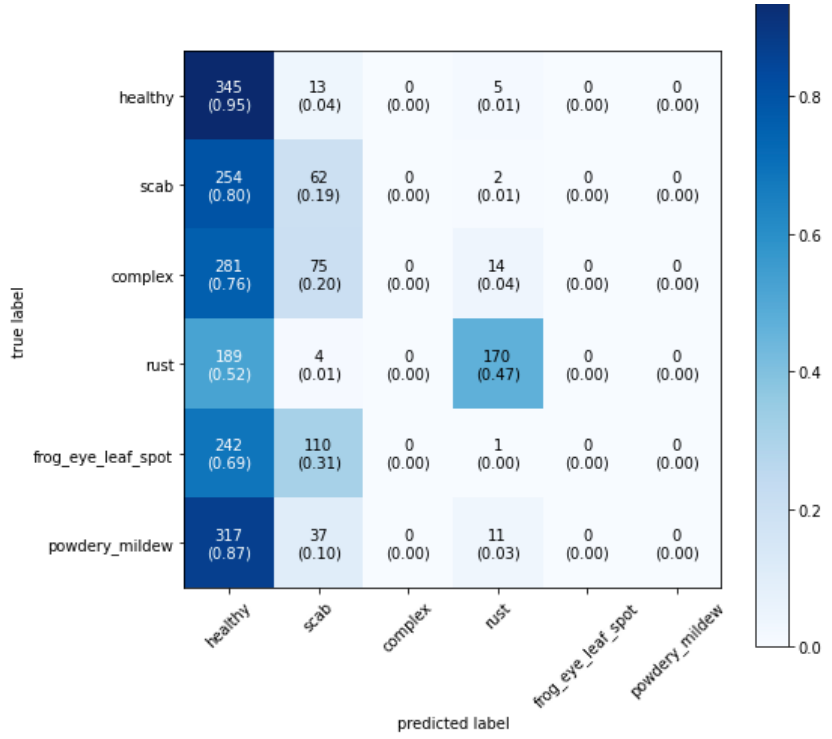
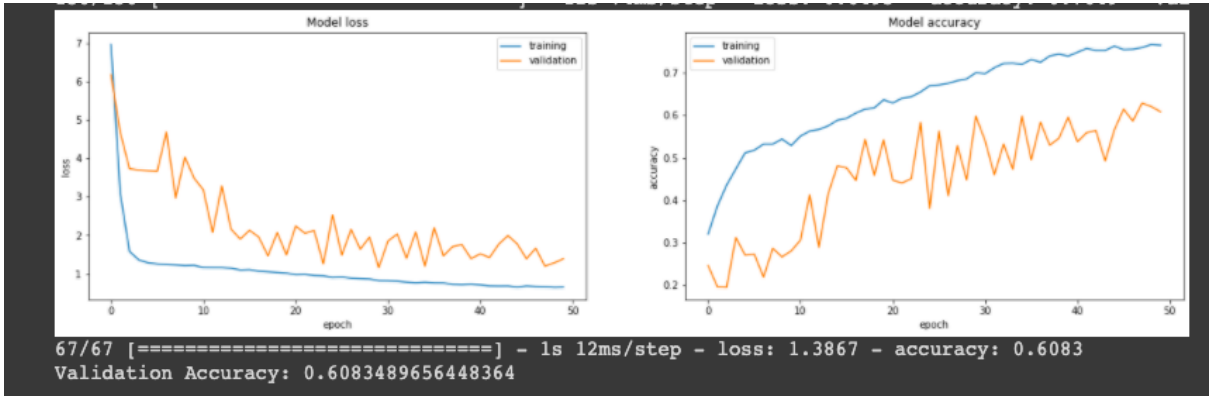
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)             (None, 96, 96, 32)        896
 dropout (Dropout)           (None, 96, 96, 32)        0

conv2d_1 (Conv2D)           (None, 96, 96, 64)        18496
 batch_normalization (BatchN  (None, 96, 96, 64)       256
 ormalization)
 dropout_1 (Dropout)         (None, 96, 96, 64)        0

 conv2d_2 (Conv2D)           (None, 96, 96, 128)       73856
 batch_normalization_1 (Batc  (None, 96, 96, 128)      512
 hNormalization)
 dropout_2 (Dropout)         (None, 96, 96, 128)       0

 conv2d_3 (Conv2D)           (None, 96, 96, 256)       295168
 max_pooling2d (MaxPooling2D  (None, 48, 48, 256)      0
 )
 dropout_3 (Dropout)         (None, 48, 48, 256)       0
 conv2d_4 (Conv2D)           (None, 48, 48, 32)        73760
 max_pooling2d_1 (MaxPooling  (None, 24, 24, 32)       0
 2D)

 flatten (Flatten)           (None, 18432)             0
 dropout_4 (Dropout)         (None, 18432)             0
 dense (Dense)               (None, 256)               4718848
 dropout_5 (Dropout)         (None, 256)               0
 dense_1 (Dense)             (None, 6)                 1542

=================================================================
Total params: 5,183,334
Trainable params: 5,182,950
Non-trainable params: 384
_________________________________________________________________
```

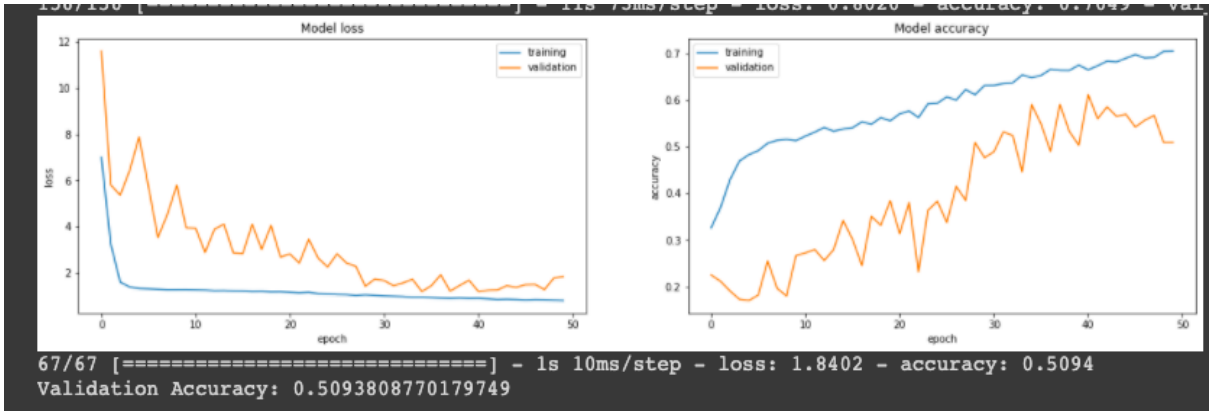
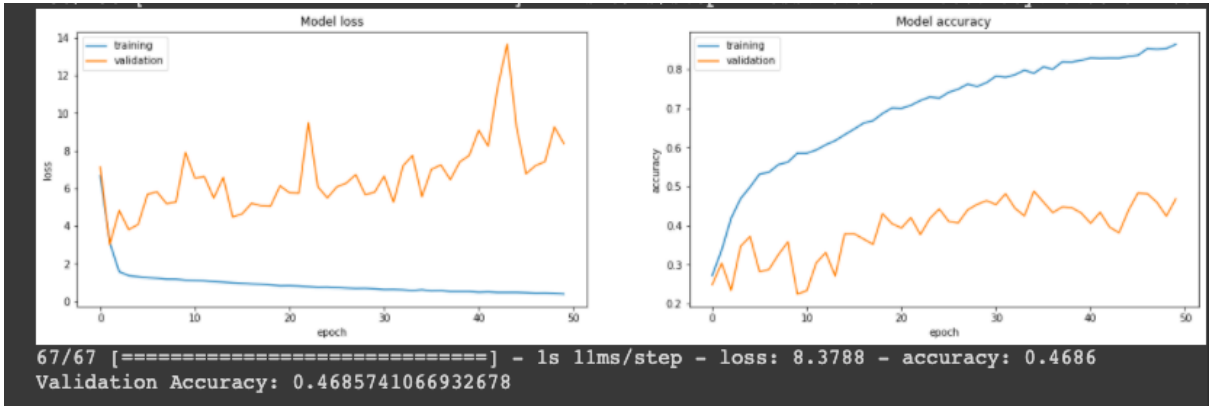


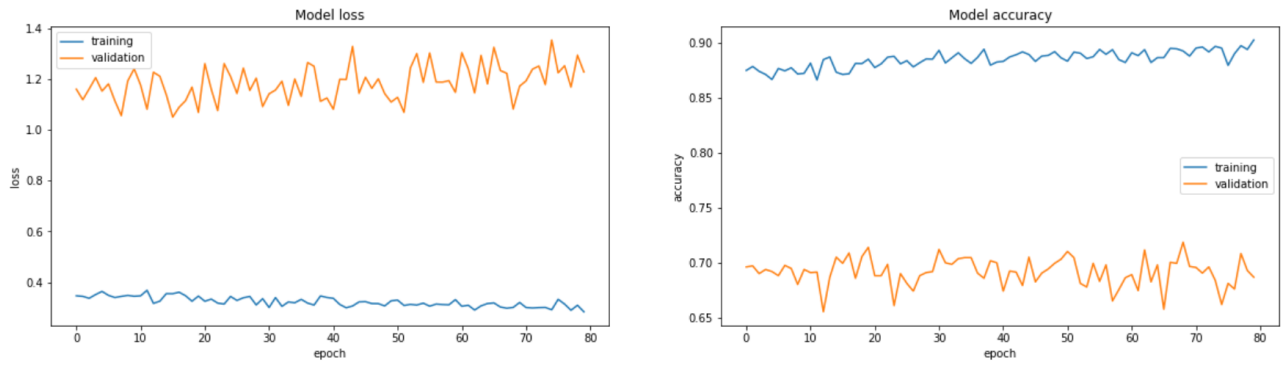

```
67/67 [==============================] - 4s 54ms/step - loss: 4.8655 - accuracy: 0.4578
Validation Accuracy: 0.45778611302375793
```

3.3.2 Important results for data augmentations

Trial ID	augmentation(s)	avg. accuracy	plot / matrix
4_19122	{'horizontal_flip': True}	**0.51**	
5_19122	{'horizontal_flip': True}	**0.54**	

6_19122	{'horizontal_flip': True, 'vertical_flip': True, 'rotation_range': 20, 'width_shift_range': 0.1, 'height_shift_range' : 0.1}	**0.34**	
7_19122	{'zoom_range': 0.3}	**0.60**	

8_19122	{'zoom_range':0.3, 'horizontal_flip': True}	**0.50**	
9_19122	{'zoom_range': 0.1, 'brightness_range': [0.3, 0.5]}	**0.46**	

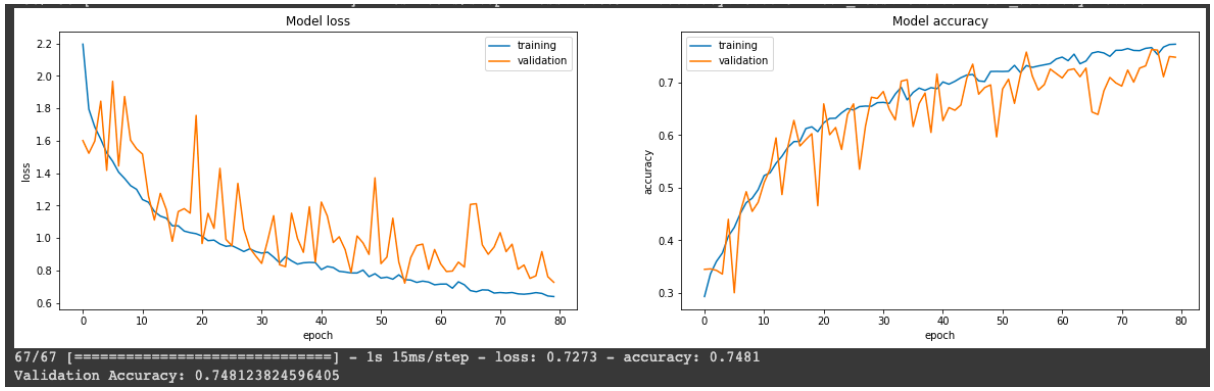
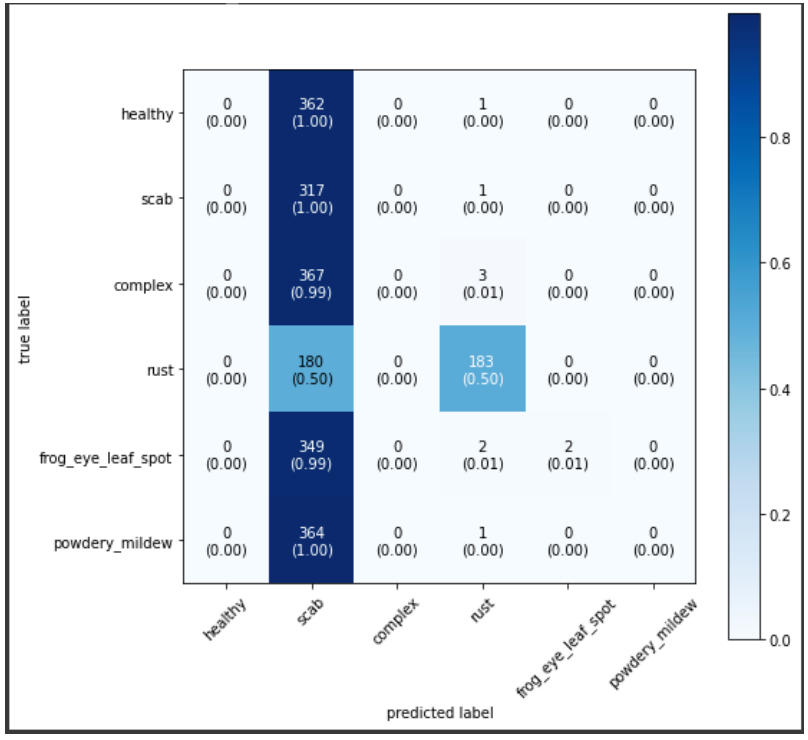
9a_19122	{'zca_whitening': True}		 67/67 [==============================] - 1s 11ms/step - loss: 1.2284 - accuracy: 0.6867 Validation Accuracy: 0.6866791844367981

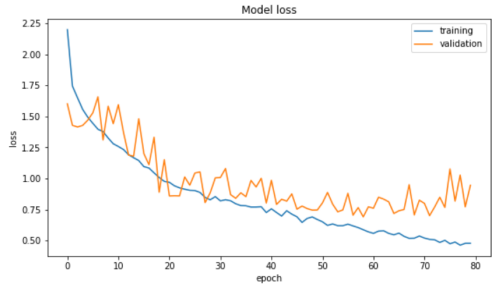
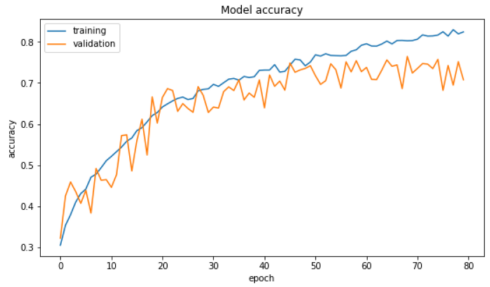
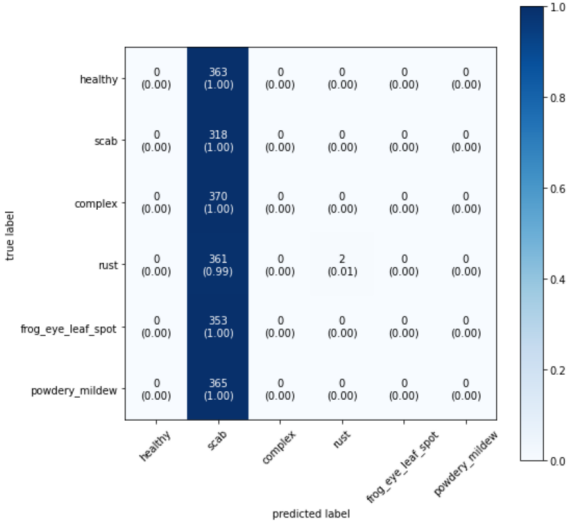
3.3.3 A combination of augmentations and network configuration

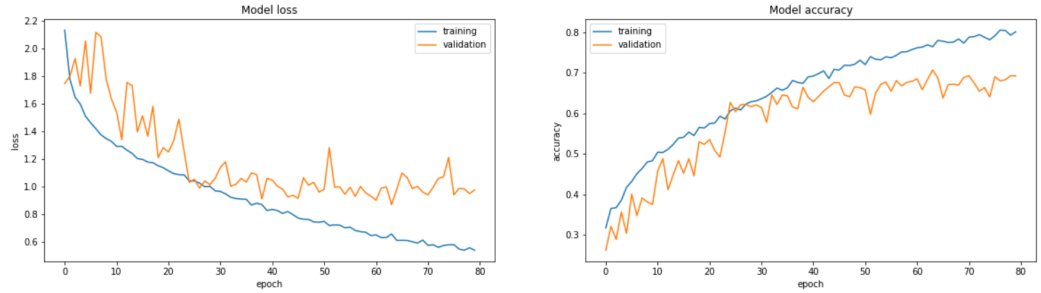
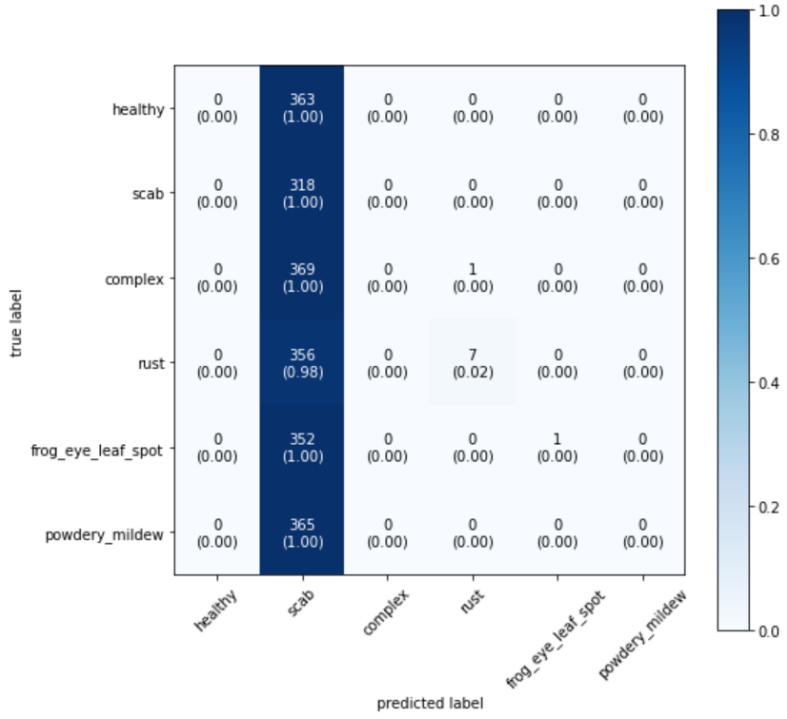
The model with 3 added layers, ID 2_19122, seemed to perform the best because it had the highest accuracy rate, it seemed to overcome overfitting the most. Additionally, the confusion matrix indicated that this model was not completely incapable of identifying two other classes (rust and frog eye leaf spot) whereas the other models were not capable of this at all. With regards to augmentations, we seemed to get the best result with zoom range. We also combined zoom range and horizontal flip because the horizontal flip looked promising. But that combination decreased accuracy a lot. For this reason, we decided to evaluate model ID 2_19122 with zoom range only, and found that this leads to the highest accuracy so far, 0.73. See model 10_19122.

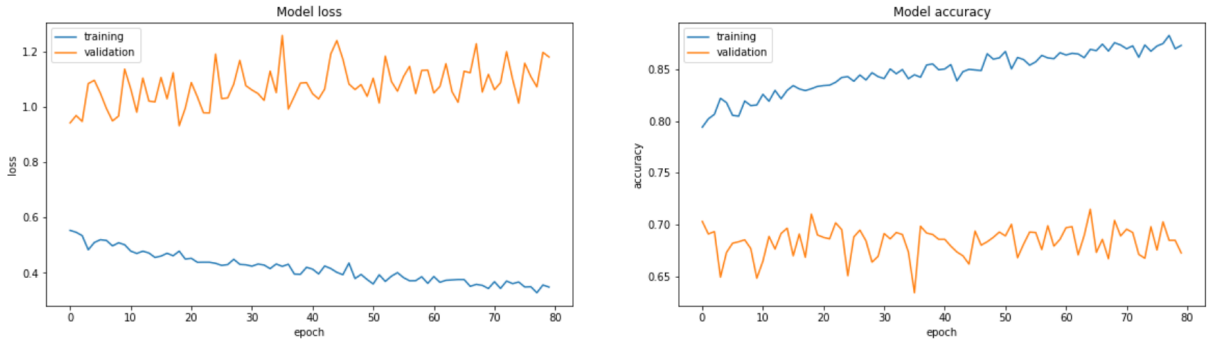
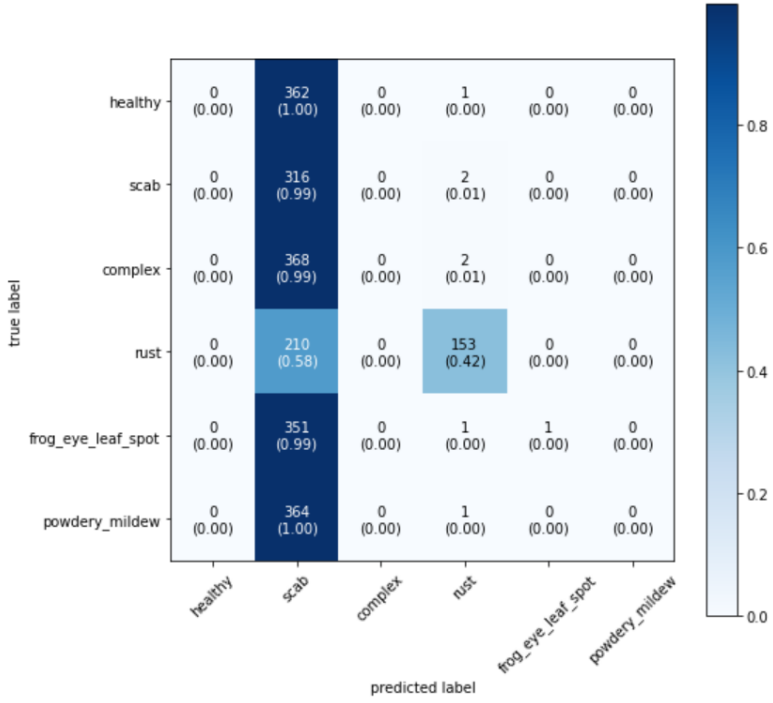
We also noticed that model ID 6_19122 with 4 augmentations ('horizontal_flip': True, 'vertical_flip': True, 'rotation_range': 20, 'width_shift_range': 0.1, 'height_shift_range': 0.1), often seemed to predict healthy leaves and that it had a 50% accuracy rate for scab. For this reason, we decided to evaluate the model (ID 2_19122) with these augmentations as well. See model 11_19122.

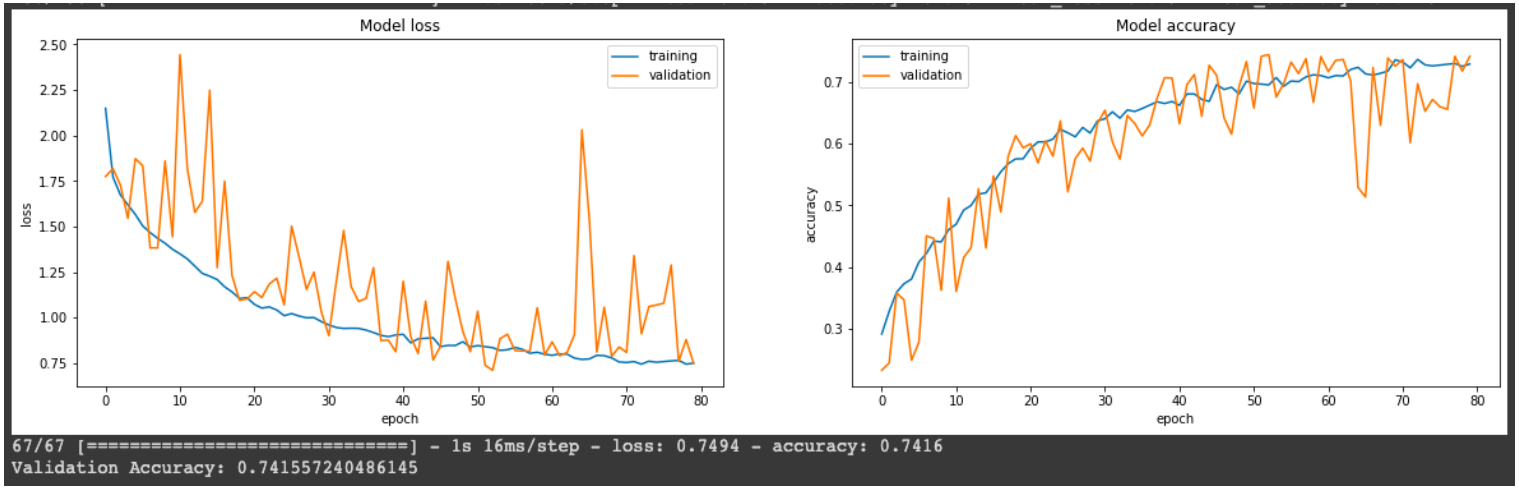
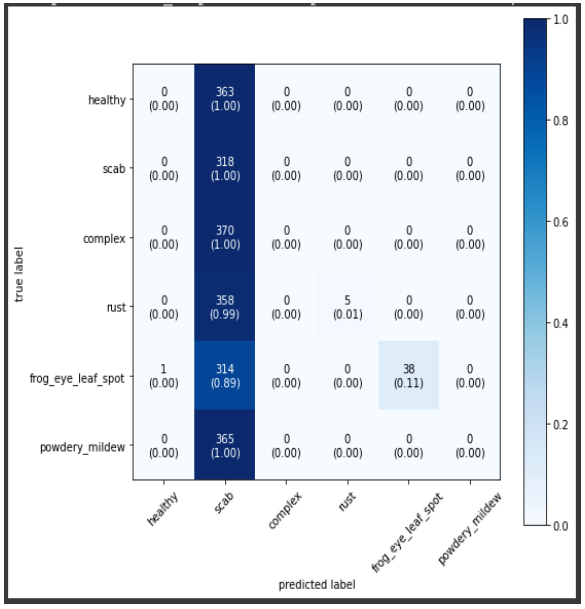
Trial ID	**accuracy**	**plot / matrix**

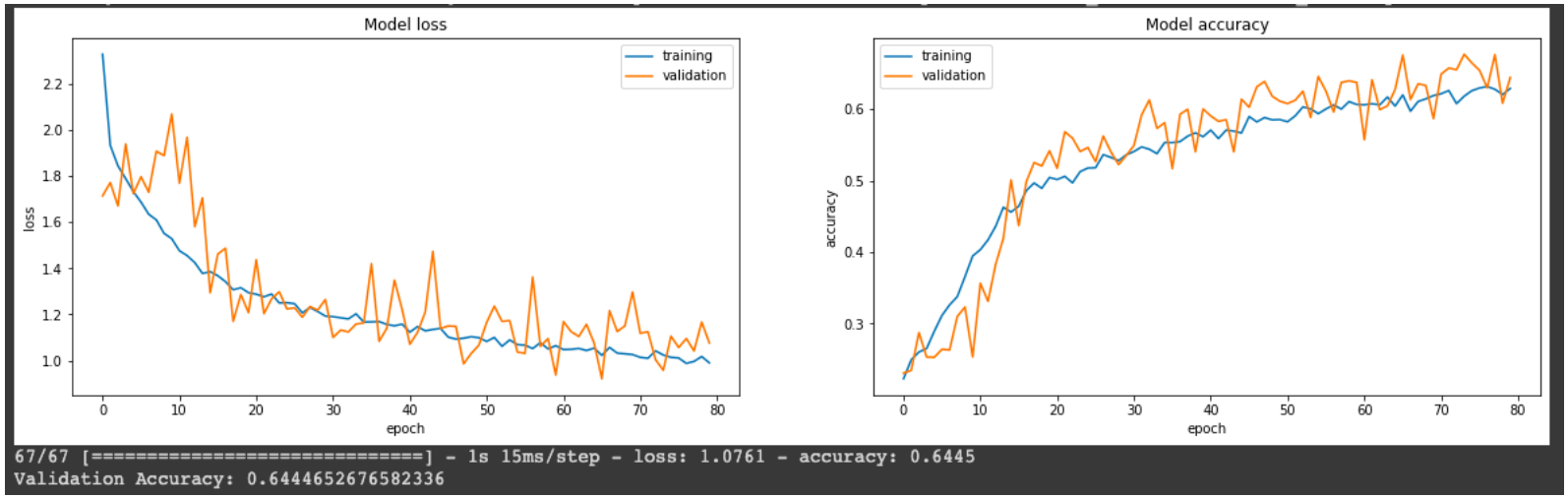
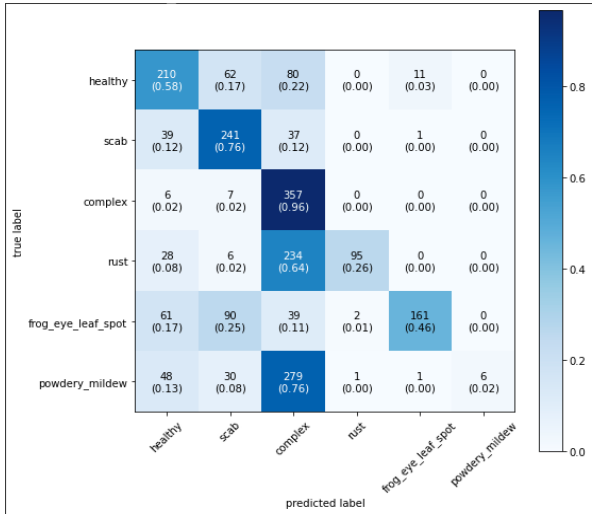
10_19122 combination: model ID 2_19122 and augment = {'zoom_range': 0.3}	**0.748**	 

11_19122 combination: model ID 2_19122 and {'horizontal_flip': True, 'vertical_flip': True, 'rotation_range': 20, 'width_shift_range': 0.1, 'height_shift_range': 0.1}	**0.707**	  67/67 [==============================] - 1s 11ms/step - loss: 0.9457 - accuracy: 0.7078 Validation Accuracy: 0.7077861428260803 

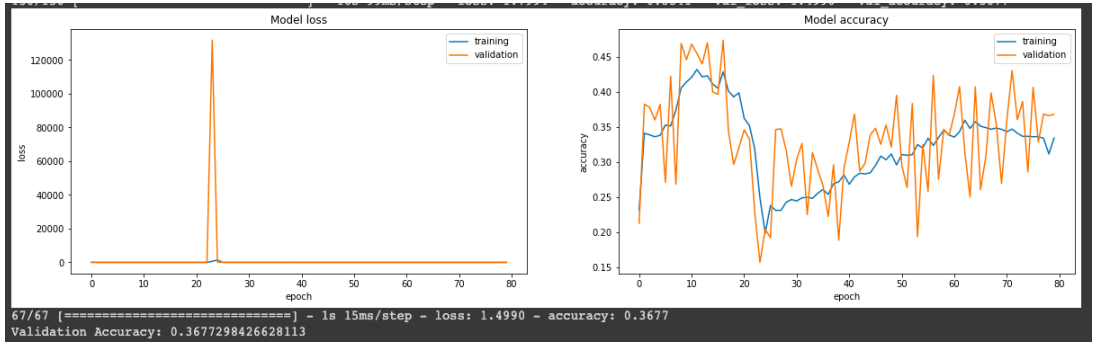
12_19122 combination: model ID 2_19122 and augment = {'horizontal_flip': True}	**0.692**	 67/67 [==============================] - 1s 11ms/step - loss: 0.9740 - accuracy: 0.6923 Validation Accuracy: 0.692307710647583 

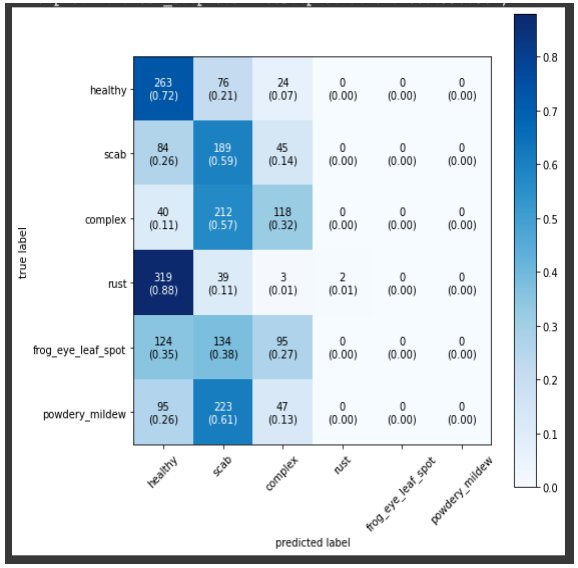
13_19122 combination: model ID 2_19122 and augment = {'horizontal_flip': True, 'zoom_range': 0.1}	**0.673**	 67/67 [==============================] - 1s 11ms/step - loss: 1.1800 - accuracy: 0.6726 Validation Accuracy: 0.6726078987121582 

14_19122 combination: model ID 2_19122 and augment = {'horizontal_flip': True, 'vertical_flip': True, 'rotation_range': 20, 'width_shift_range': 0.1, 'height_shift_range': 0.1, 'zoom_range': 0.3}	**0.742**	 

16_19122 combination: model ID 2_19122 and augment = { 'horizontal_flip': True, 'vertical_flip': True, 'rotation_range': 20, 'width_shift_range': 0.1, 'height_shift_range' : 0.1, 'zoom_range': [0.3,1.0], 'brightness_range': [0.2,1.2], 'channel_shift_rang e' : 0.7}	**0.645**	 67/67 [==============================] - 1s 15ms/step - loss: 1.0761 - accuracy: 0.6445 Validation Accuracy: 0.6444652676582336 

3.3.4 adjusted learning rate

Trial ID	accuracy	plot / matrix
15_19122 combination: model ID 2_19122 and augment = {'horizontal_flip': True, 'vertical_flip': True, 'rotation_range': 20, 'width_shift_range': 0.1, 'height_shift_range': 0.1, 'zoom_range': 0.3} with optimizer = keras.optimizers.Adam(lr = 0.01)	**0.368**	

		 healthy scab complex rust frog_eye_leaf_spot powdery_mildew true label 263 (0.72) 76 (0.21) 24 (0.07) 0 (0.00) 0 (0.00) 0 (0.00) 84 (0.26) 189 (0.59) 45 (0.14) 0 (0.00) 0 (0.00) 0 (0.00) 40 (0.11) 212 (0.57) 118 (0.32) 0 (0.00) 0 (0.00) 0 (0.00) 319 (0.88) 39 (0.11) 3 (0.01) 2 (0.01) 0 (0.00) 0 (0.00) 124 (0.35) 134 (0.38) 95 (0.27) 0 (0.00) 0 (0.00) 0 (0.00) 95 (0.26) 223 (0.61) 47 (0.13) 0 (0.00) 0 (0.00) 0 (0.00) predicted label 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0

3.4 DISCUSSION

3.4.1 Network configuration: layers

The results indicate two important findings relating to the addition of layers. Firstly, the addition of layers, in this case the creation of a diamond shape, seems to greatly increase the accuracy of the model. We have noticed that this step significantly prevents overfitting and in some instances, especially for model ID 2_19122 with 3 added layers, the model validation accuracy can often be higher than the training accuracy. We find that, when it comes to adding layers, more is not necessarily better. This is indicated by the fact that model ID 3_19122 has more layers than model ID 2_19122, but it does not have a higher accuracy and the confusion matrix indicates that ID 3_19122 is not good at predicting classes other than scab. Contrastingly, model ID 2_19122 has a slightly higher accuracy (+ 0.02) and occasionally predicts other classes (although not often). For these reasons, we suspect that model ID 2_19122 has the optimal number of layers for now.

Based on this, we moved on to removing MaxPooling layers in order to consider whether this may increase the accuracy for model ID 2_19122. We have found that removing all the MaxPooling layers, except for the final two, significantly decreases accuracy whilst also slowing down the learning, as more data needs to be processed. Because of this result and because we have a limited amount of experiments that we can run in a day due to Collab memory limitations and time constraints, we decided not to experiment further with the MaxPooling layers, and instead focus on data augmentation. We suspected that data augmentation could help to increase accuracy and the recognition of specific diseases because this was the case with our experience with CIFAR.

3.4.2 Network configuration: data augmentation

The results relating to data augmentation vary widely. Most notably, we have found that certain augmentations increase the accuracy of the basic model, without added layers. Some are also able to work against overfitting. From our experiments, we noticed that the zoom range, horizontal flip, and combination of basic augmentations (such as horizontal flip, vertical flip, etc.) worked to increase the accuracy the most. Notably, some augmentations stimulated the model to classify images as particular diseases more often. This can be seen in the results for image 5_19122 and 6_19122, as the model became more likely to classify images as healthy, and scab and rust respectively. These are interesting findings that we will also take with us as useful information for the next milestones.

Based on these, we decided that the zoom range, horizontal flip, and combination of basic augmentations looked the most promising so we combined these augmentations with the improved model (ID 2_19122). This is discussed in the following section.

3.4.3 Increased layers and data augmentation

We notice that the combination of model ID 2_19122 and the individual augmentations do not seem to affect the accuracy of the model in a significant manner. We think that the slight variations (all , 0.02) could be due to chance. Notably, the augmentation of horizontal flip and the combination of basic augmentations, (11_19122 and 12_19122), seem to increase overfitting somewhat so we would not use these on their own. The combination of augmentations also led to some interesting results as they seemed to decrease accuracy and increase overfitting.

Most notably, the model ID 16_19122, which combined many of the augmentations that we have tried and model ID 2_19122, showed a significant improvement in the model's ability to recognize different classes. This is indicated by the confusion matrix, which now shows much more of a diagonal pattern. We do note that the accuracy of this model is lower than the accuracy of model ID 2_19122 and other

models. This is something that we must remain cognisant of during next milestones, in order to create our best model.

3.4.4 Network configuration: Learning Rate

An additional adjustment implemented is that of the Learning Rate. During the previous milestones we have maintained such to the default for the adam-optimizer, which is 0.001. Adjusting the learning rate should affect the performance of how the network learns, a smaller learning rate would lead to faster performance and vice versa. After adjusting it to 0.01, we've noticed that it didn't run much faster in comparison to previous runs and had also led to a significantly lower accuracy. Accordingly, we decided not to experiment further with the learning rate settled to maintain the learning rate to its default setting. This way, we could spend our time and limited memory space on other techniques that seemed more promising.

3.4.5 The Confusion matrices

The problem that is described in the discussion and conclusion of milestone 2 also occurred in this milestone. Not all the matrices are equal to the accuracy given from the plots, so we tried to re-program this piece of the code. The first solution we tried was the one from Lars, we used the predict function and after that we added to_categorical but this did not work for our model. So then we moved on to the next solution, we used the argmax function for getting a better result. Even this solution did not work well for every matrix, but some matrices were correct after this. We could check this the same way as described in milestone 2 and by printing the y_predict and y_true and checking whether it shows the correct data . We only used the right matrices to come to a conclusion or draw connections.

3.5 CONCLUSION

In conclusion, one of the most promising models in terms of accuracy seems to be model ID: 2_19122 with an accuracy of 0.73. Additionally, the model 14_19122, that includes many augmentation types, has the highest accuracy rate that we have found (0.74) and seems to adequately prevent overfitting. Notably, the model seems to be outperforming on validation data as opposed to training data, which could indicate good generalisation.

There are also some other models that we have found to be significant, as the model seems to be more varied in its classification, as images are not classified solely as one class. This is especially the case for model 16_19122, the model with a combination of ID 2_19122 and augmentations. In contrast to many other of our models, the confusion matrix indicates that the model can also predict other classes instead of just one or two. The accuracy of such models was significantly lower (0.64). We intend to keep this, and other such models in mind during the next milestones in order to further improve the capability of the model to recognise different classes.

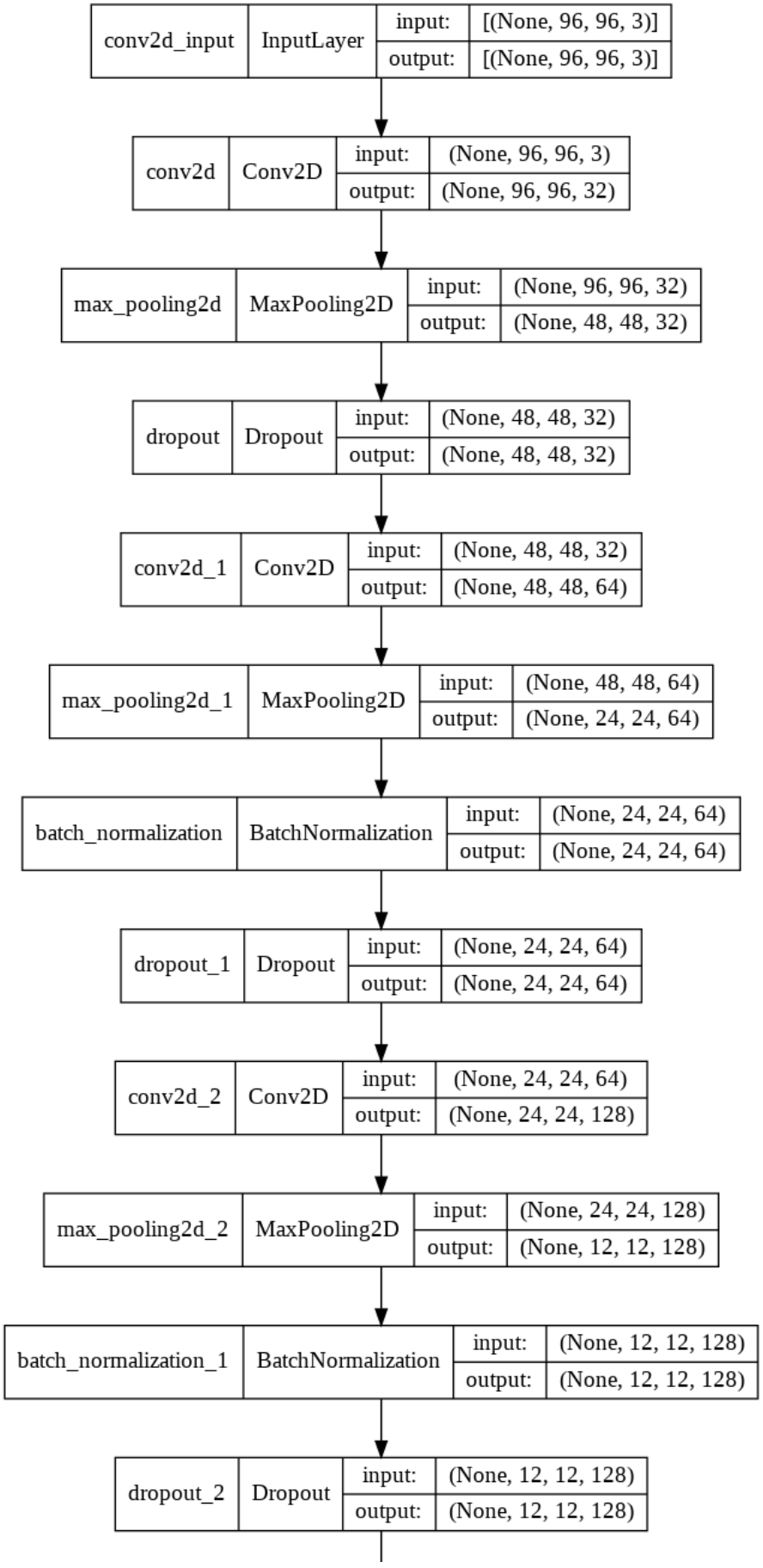
Overall, we have reached our goal for this milestone as both plots of the training and validation accuracies increase in tandem, portraying a more smooth curve. From this perspective, the model seems to be generalising well. However, the model does not perfectly recognise the other classes yet. Although some of our models seemed to improve in this regard, we can still significantly improve our model in this respect.

We intend to work toward this goal during the next milestone, by changing the dropout rate. As mentioned in milestone 2, we think that it would be appropriate to re-consider the values for dropout after adding more layers to the network. After all, these rates were used on a very simple model, in this milestone we improved the model complexity and this could possibly mean that we have to change these dropout rates since the influence of dropout increases as we have more layers. We intend to do some research about dropout and experiment with various rates. The dropout rate is now set to 0.2 in the

flatten layer and 0.5 in the hidden layers because different sources on the internet indicated that these values worked for their ML programmes. However, due to storage limitations in milestone 1 and 2, we have not been able to check whether these values are the most accurate for our model or not. For the next milestone we'll continue with the following model:

```
# ML MODEL ARCHITECTURE
# Define Sequential model
model = models.Sequential()

# create convolutional layer and max pooling layer
model.add(layers.Conv2D(32, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same', input_shape=(96, 96, 3)))
model.add(layers.MaxPooling2D((2, 2)))

# create convolutional layer (larger) and max pooling layer
model.add(tf.keras.layers.Dropout(0.5))
model.add(layers.Conv2D(64, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.BatchNormalization())

# add Conv2D layer with 128 filters and max pooling layer
model.add(tf.keras.layers.Dropout(0.5))
model.add(layers.Conv2D(128, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.BatchNormalization())

# add Conv2D layer with 256 filters and max pooling layer
model.add(tf.keras.layers.Dropout(0.5))
model.add(layers.Conv2D(256, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
```

```
# add Conv2D layer with 32 filters and max pooling layer
model.add(tf.keras.layers.Dropout(0.5))
model.add(layers.Conv2D(32, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))

# flatten layers
model.add(layers.Flatten())
model.add(tf.keras.layers.Dropout(0.2))
model.add(layers.Dense(256, activation=tf.keras.layers.LeakyReLU(alpha=0.5)))

# apply softmax activation for final layer classification
model.add(tf.keras.layers.Dropout(0.2))
model.add(layers.Dense(6, activation='softmax'))

# normalize input data: set preprocesing dictionary
preprocess = {'featurewise_center': True, 'featurewise_std_normalization' : True}

# augment data: set augmentation dictionary
augment = {'horizontal_flip': True,
           'vertical_flip': True,
           'rotation_range': 20,
           'width_shift_range': 0.1,
           'height_shift_range': 0.1,
           'zoom_range': [0.3,1.0],
           'brightness_range': [0.2,1.2],
           'channel_shift_range' : 0.7}

# run training and evaluation function
train_and_evaluate(model, x_train, y_train, x_val, y_val, preprocess, epochs = 80, augment = augment)
```


conv2d_input | InputLayer | input: [(None, 96, 96, 3)] | output: [(None, 96, 96, 3)]
conv2d | Conv2D | input: (None, 96, 96, 3) | output: (None, 96, 96, 32)
max_pooling2d | MaxPooling2D | input: (None, 96, 96, 32) | output: (None, 48, 48, 32)
dropout | Dropout | input: (None, 48, 48, 32) | output: (None, 48, 48, 32)
conv2d_1 | Conv2D | input: (None, 48, 48, 32) | output: (None, 48, 48, 64)
max_pooling2d_1 | MaxPooling2D | input: (None, 48, 48, 64) | output: (None, 24, 24, 64)
batch_normalization | BatchNormalization | input: (None, 24, 24, 64) | output: (None, 24, 24, 64)
dropout_1 | Dropout | input: (None, 24, 24, 64) | output: (None, 24, 24, 64)
conv2d_2 | Conv2D | input: (None, 24, 24, 64) | output: (None, 24, 24, 128)
max_pooling2d_2 | MaxPooling2D | input: (None, 24, 24, 128) | output: (None, 12, 12, 128)
batch_normalization_1 | BatchNormalization | input: (None, 12, 12, 128) | output: (None, 12, 12, 128)
dropout_2 | Dropout | input: (None, 12, 12, 128) | output: (None, 12, 12, 128)

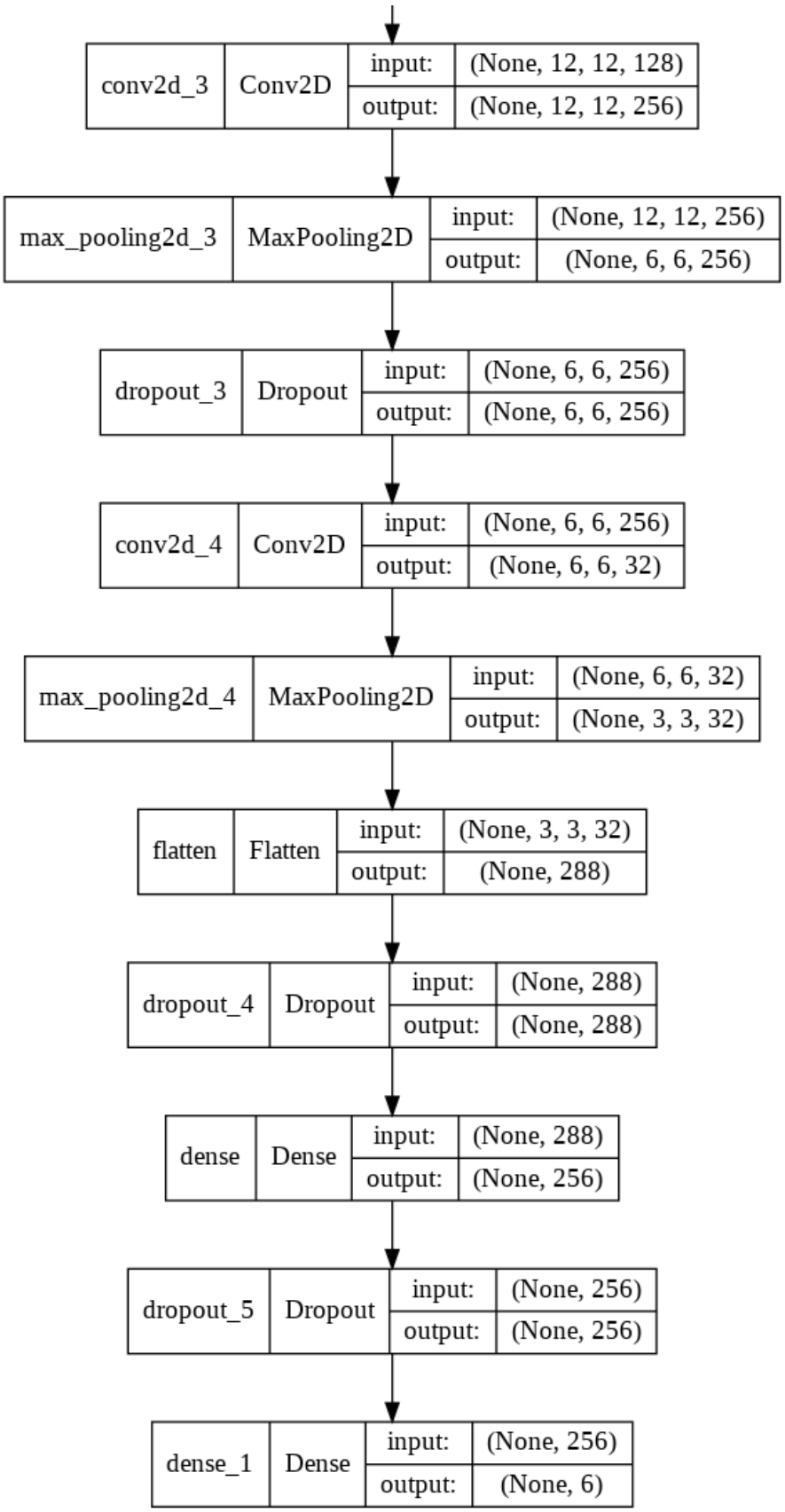


Figure 12 - Final network-configuration of Milestone 3

Milestone 4

Deadline: Wednesday January 26th at 17:59

For this milestone you will continue to fine tune the model, in order to increase its optimality. You could look further into the data augmentation based on the current limitations detected in milestone 3 or the data itself. Other than that, experiment further to confirm that the current configuration is indeed the most pre-eminent version. Furthermore, deepen this fine tuning approach based on the received feedback and think of ways to improve the loss and/or accuracy detected by the analysis of the network-performance. Thus, the goal for this milestone is to conduct the correct experiments in order to improve the network's overall functioning.

Meeting 1: Thursday January 25th
Meeting 2: Monday January 27th

4.1 INTRODUCTION

4.1.1 Increasing the accuracy of the Apple Tree Disease Project

In the previous milestone, we ensured that both plots of the training and validation accuracies simultaneously increased in tandem. Even though the desired smooth curves weren't exactly achieved, the results still seem more promising in comparison to our previous models. The average accuracy has been increased to approximately 65 - 75%; for which the analysis by confusion matrix of the latter implied sole predictions of a single class, while the former showed an outspread result with slight predictions for all classes.

Our goal for this milestone is to efficiently increase the accuracy whilst also improving the results shown by the analysis of the confusion matrix. In order to achieve this, we would figure out the best combination of values for data augmentation along with additional trials of adapting the network configuration.

The changes to our network configuration include an, expectedly, improved approach to determining the amount of layers by comparison to previous models found in literature (such as VGG), looking further into the activation functions of ReLU vs. Leaky ReLU, and most importantly, the increased use of data.

These are quite a number of methods that require us to conduct multiple experiments. Since we are quite limited in time and the number of times we can run Google Collab. For this reason, we have decided to run many of these experiments during milestone 5. This is only something to note in relation to our planning, and it does not affect the rest of the paper. We have simply incorporated all the experiments into this chapter, named milestone 4. In the following section on methods, we explain why we saw added value in applying all these different techniques in order to improve the model.

4.2 METHODS

One of the main limitations we have experienced while conducting the experiments for the previous milestone consisted of time and memory space. In order to by-pass this we have changed up the code to ensure more efficiency. We adapted the code to save the resized images in a separate directory, with the purpose of only resizing when necessary and being able to directly loop through this folder to gather the necessary data. Similar to our previous method we have split the experiments in two parts: data augmentation and network configuration. Eventually, we combined the discoveries of both to retrieve the best possible result for this milestone. The results of both separate and combined parts are further illustrated in the section on our results, 4.3.

4.2.1 From diamond shape to VGG

In the previous milestones, we experimented with building a diamond shape for the network by first increasing the number of filters, in order to decrease them as the network progresses. This has caused the model to have a relatively high number of parameters, which we could choose to lower. After talking to Wouter about the high number of parameters, we realised that changing the network structure may optimise the model. A Kaggle thread on state-of-the-art Deep Learning Models, referred to us by Wouter, suggests that a small 3x3 convolution kernel can be used in all layers in order to deepen the number of network layers and to avoid too many parameters. The method has been used by the Visual Geometry Group (VGG).[22]

The group has released a series of convolutional network models beginning with VGG, which can be applied to face recognition and image classification. There are multiple VGG models, ranging from VGG 16 to VGG 19. VGG16 contains 16 layers and VGG19 contains 19 layers. The overall structure consists of sets of convolutional layers, each set followed by a MaxPool. The difference is that more and more cascaded convolutional layers are included in the five sets of convolutional layers.

The original purpose of VGG's research on the depth of convolutional networks was to understand how the depth of convolutional networks affects the accuracy of large-scale image classification and recognition. In previous milestones, we have determined that deepening our CNN can indeed increase our accuracy. We are curious to know whether an application of VGG, which not only increases the layers but also limits the number of parameters, can increase the accuracy of our model as applied to our dataset. Furthermore, one of our goals in this milestone is to improve the network's ability to learn new features and to recognize certain specific diseases. We suspect that the VGG configuration may work towards this goal as the article notes that VGG increases the ability of CNN to learn features, due to the non-linear transformation.[23]

Given the limited memory available in Google collab, we are not sure how many different models we can experiment with so we wanted to start experimenting with one model, knowing that we might not get to experiment with the other models (at least, during this milestone). Out of the possible VGG networks, we have implemented the VGG-16 network. The reason for this is because we don't think our data requires a model that consists of that many layers. This, because the models that we used in the previous milestone had much fewer layers and these already had accuracies of 70%. So we started with VGG-16 and if time and memory allows, we may also experiment with the different VGG models.

We have implemented the VGG-16 as described in the article. VGG-16 can be illustrated as follows:[24]

---

[22] https://www.kaggle.com/blurredmachine/vggnet-16-architecture-a-complete-guide
[23] https://www.kaggle.com/blurredmachine/vggnet-16-architecture-a-complete-guide
[24] source of image: https://www.kaggle.com/blurredmachine/vggnet-16-architecture-a-complete-guide

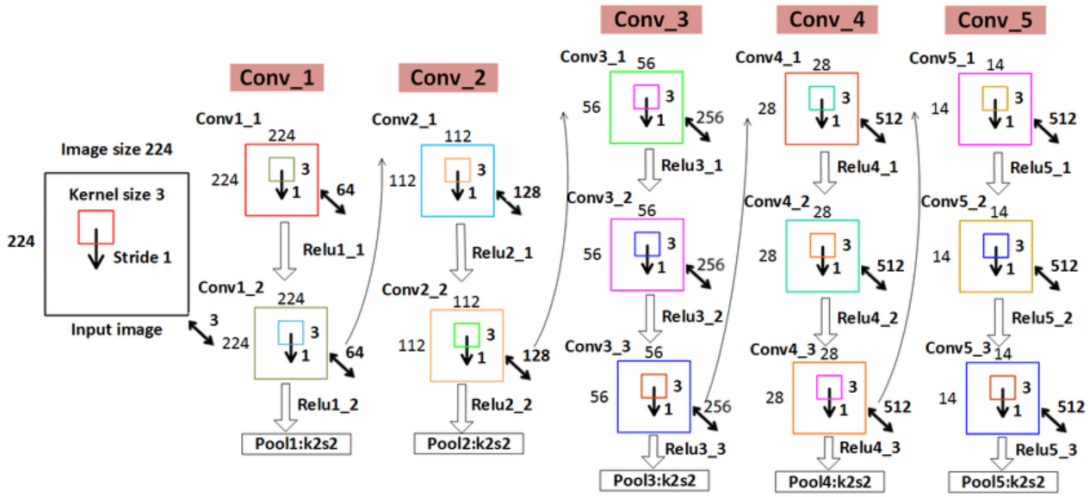


Figure 13 – VGG-16

We have used similar dropout rates (p = 0.5) and the ReLU activation. The significant difference between our experiment and that of the VGGNet paper, however, is that their input images are of size 224 x 224. We suspect that such a size is too large for our programme to process and we may quickly run out of storage space. This will limit our ability to experiment significantly. For this reason, we have decided to stick with the resized images of 96 x 96 pixels, as used in the previous milestones. The results of this network configuration can be found in the results section below.

4.2.2 Leaky ReLU vs ReLU

For all network models found in previous milestones we have maintained the activation function as Leaky ReLU. This was an early decision based on a much simpler network made in milestone 2, in which we encountered the problem of constant overfitting. Our approach then was to tackle this problem by the addition of DropOut layers, which did not deliver any promising results due the speculation of an excess amount of dead nodes being present during training. Subsequently, we immediately thought of replacing ReLU with Leaky Relu, since this decreases the chance of dead nodes resulting from internal computations of derivatives resulting in 0. Other than that, we also expected our network to run much faster due to more balanced nodes-activations.

However maintaining this decision of activation function would make our model seem quite dodgy, since it doesn't correspond with our current network configuration. In order to affirm that Leaky ReLU is indeed the best option, it is necessary to conduct the suitable trials in order to prove this. Furthermore, we noticed that VGG's optimal network only uses ReLU-activations, which made us even more curious about the effect of this. In the results-section below, we compare results from ReLU vs Leaky ReLU.

Hereby we hypothesise that Leaky ReLU will return better accuracy for our network. If this is indeed the case, we will also experiment with the alpha-value of this activation function in order to choose the most optimal version of Leaky ReLU for our network. This too will be illustrated in the result-section.

4.2.3 Further experiments with Data Augmentation

For this milestone, we continue based on the best combination found from the previous milestone. This combination consisted of the following features:

Feature	**Value**
horizontal_flip	True

vertical_flip	True
rotation_range	20
width_shift_range	0.1
height_shift_range	0.1
zoom_range	[0.3, 1.0]
brightness_range	[0.25, 1.25]
channel_shift_range	0.7

During previous experimentation trials, we have noticed that a great difference of increase in accuracy resulted from the augmentation features allowing colour-space-transformations, such as channel_shift_range. As stated in previous research, lighting biases are amongst the most frequently occurring challenges to image recognition problems, making it fairly difficult to intuitively conceptualise .[25] Additionally, the data for our model solely consists of images taken in natural light, implying that colour-space-transformations might be of slightly greater importance than other geometric transformations such as flipping, cropping, and zooming. Thus, for this milestone we will attempt additional colour-space-transforming features such as: zca_whitening, shear_range, and different values for channel_shift_range and brightness_range.[26] Similar to the previous milestone, the relevant outcomes are to be found in the results section with reliance on the plots for the model loss and model accuracy along with the confusion matrix.

4.2.4 Increased sample data and weights

In milestone 2, we mentioned the fact that we might start using more of the dataset, as opposed to the 38% that we were using at the time. During our discussion with Tim, we revisited our use of the dataset. We concluded that we may significantly improve the accuracy of our model if we simply increase the data that it can use to train.

In this respect, we have noted two important aspects of the data used. Firstly, we suspect that the decision to spread all of the combined data into different classes causes the classes to have some samples that are, in actuality, mislabelled. That is, samples that have two or more labels belong to a combined class, but we have decided to simply put them in whatever non-combined class is mentioned first in the label. This basically ‘contaminates’ the classes as there are some samples in there that should not be in there.

Additionally, we have decided to retain the complex class. These are unhealthy leaves with too many diseases to classify visually. We are not sure exactly how this class influences the model's ability to learn features. However, we suspect that this class may be the hardest class for the model to identify because it is a combination of different classes. At the same time, the complex class has relatively few samples, which means that, if we don;t apply techniques to overcome this problem, it can be even harder for the model to learn how to identify that class correctly.

[25] Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60 (2019). https://doi.org/10.1186/s40537-019-0197-0
[26] Tensorflow augmentations: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

For the reasons above, we have decided to remove samples with combined or complex classes, no longer trying to classify leaves with those labels. We have also decided to use more training data by no longer using only 1184 samples of each class, but by simply using all data available.

Such use of the data results in an unbalanced dataset, as some classes have less than 2000 samples whilst others have over 4000. This is problematic because it can make the model biased towards the majority classes. So far we have used undersampling, as we picked 1184 samples from each class to train the model on. We considered using oversampling instead, thereby increasing the amount of samples from minority classes. However, this could involve an increased use of storage and we wanted to minimise this because we have been experiencing significant limitations of storage space in Google Drive and Google Collab.

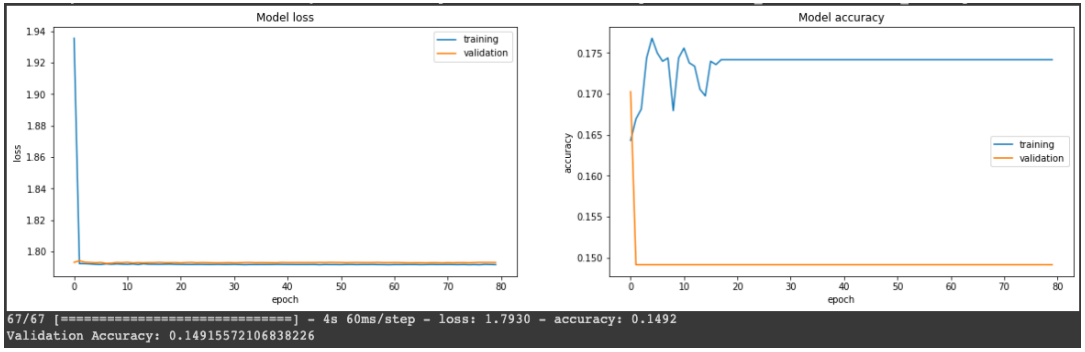
Instead of oversampling, we have decided to use the re-weighting method from the scikit-learn library to estimate class weights for unbalanced dataset with 'balanced' as a parameter.[27] The class weights are given by: n_samples / (n_classes * np.bincount(y)). In this way, the dataset is used in a more biassed way as our loss function is influenced by assigning relatively higher costs to examples from minority classes.

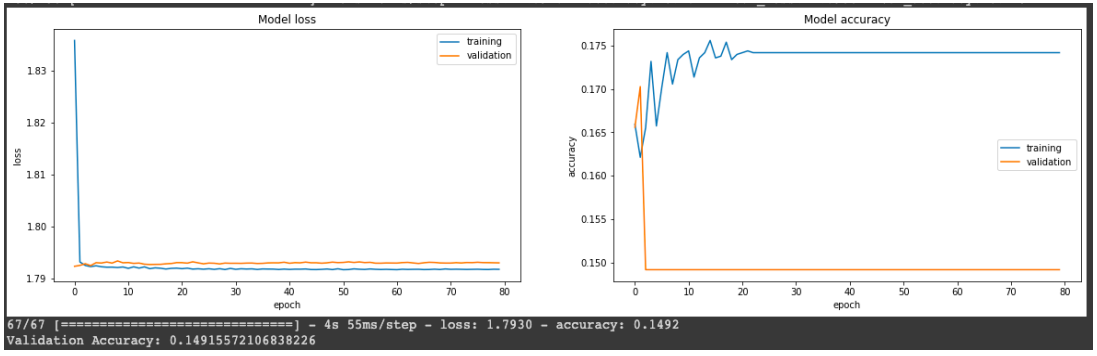
[27] We were inspired by this LinkedIn post:
https://www.linkedin.com/pulse/some-tricks-handling-imbalanced-dataset-image-m-farhan-tandia/

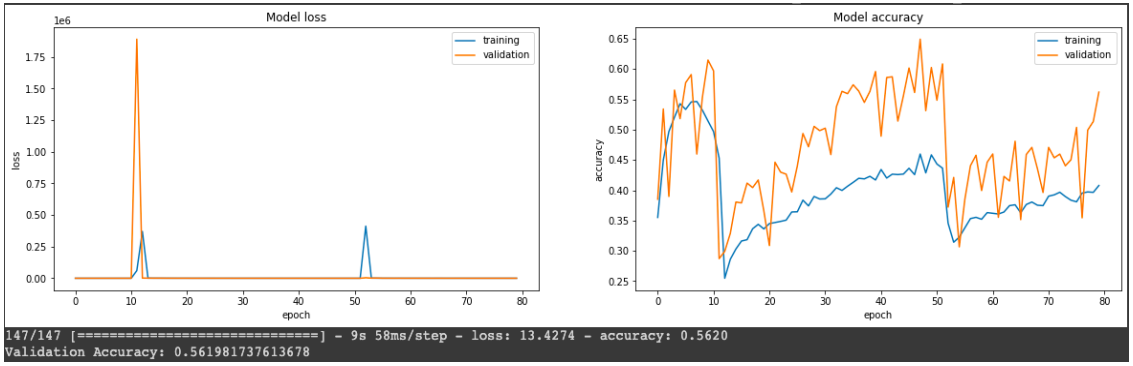
4.3 RESULTS

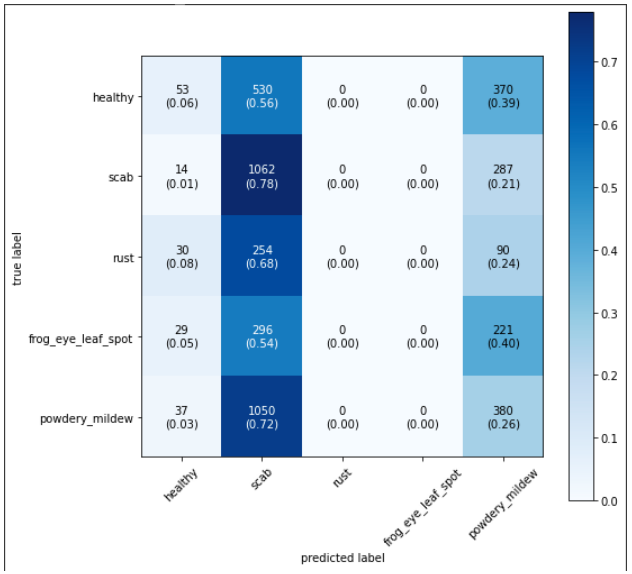
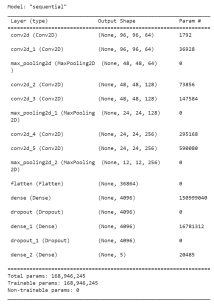
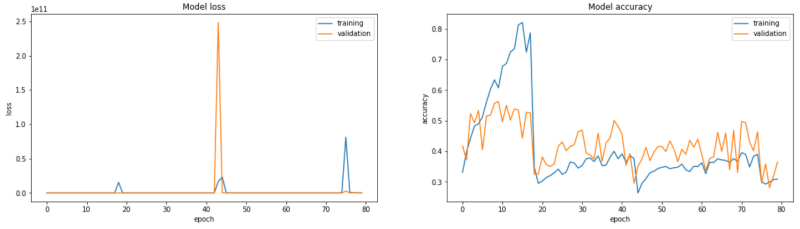
The results section discusses the results for VGG, Leaky ReLU vs ReLU, Increased samples and changed distribution of data, and data augmentation, respectively. Note that these results, along with that of previous milestones can only be reasonably compared, since there has been change to the data distribution as stated in 4.2.4. The VGG and data augmentation results are based on the previous data distribution, while the rest is based on more recent data distribution that uses much more samples and balanced weights. Since these results still contain valuable information, we have chosen to still report on these findings and make a final decision based on all results combined. Our findings are discussed in the discussion section (4.4).

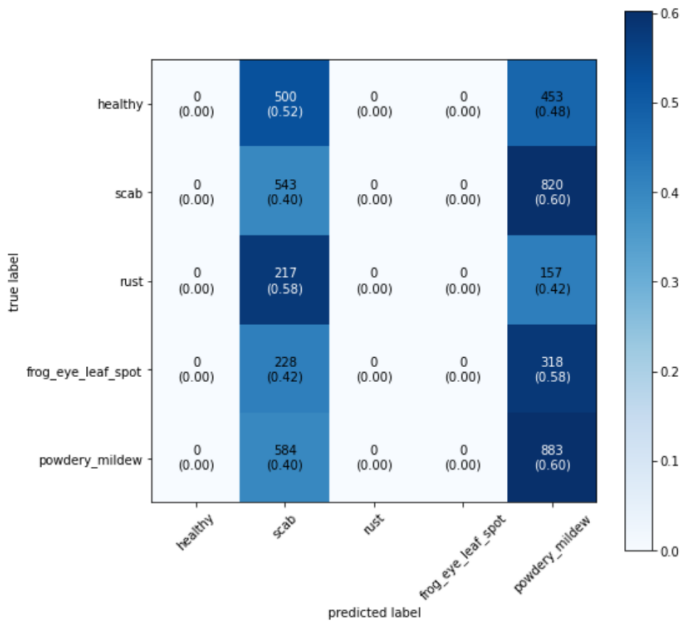
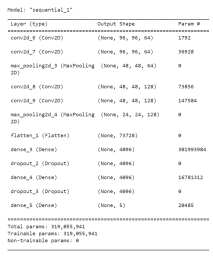
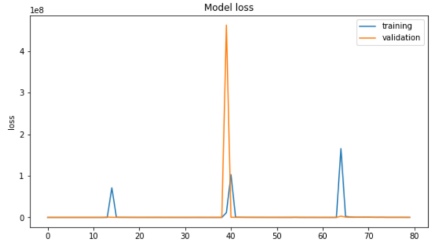
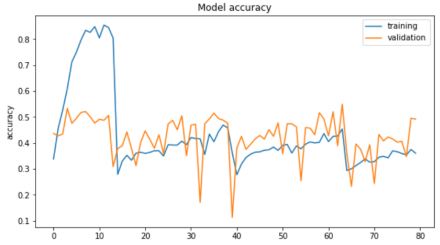
4.3.1 VGG

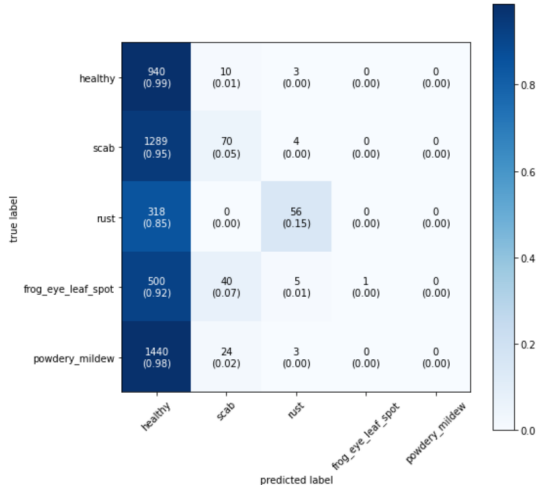
Trial ID	Avg. accuracy	model.summary	plot
1_25122	0.149	Model: "sequential_VGG1" _____ Layer (type) Output Shape Param # ========== conv2d_32 (Conv2D) (None, 96, 96, 64) 1792 conv2d_33 (Conv2D) (None, 96, 96, 64) 36928 max_pooling2d_25 (MaxPoolin g2D) (None, 48, 48, 64) 0 conv2d_34 (Conv2D) (None, 48, 48, 128) 73856 conv2d_35 (Conv2D) (None, 48, 48, 128) 147584 max_pooling2d_26 (MaxPoolin g2D) (None, 24, 24, 128) 0 conv2d_36 (Conv2D) (None, 24, 24, 256) 295168 conv2d_37 (Conv2D) (None, 24, 24, 256) 590080 max_pooling2d_27 (MaxPoolin g2D) (None, 12, 12, 256) 0 conv2d_38 (Conv2D) (None, 12, 12, 512) 1180160 conv2d_39 (Conv2D) (None, 12, 12, 512) 2359808	

		conv2d_40 (Conv2D) (None, 12, 12, 512) 2359808 max_pooling2d_28 (MaxPoolin (None, 6, 6, 512) 0 g2D) conv2d_41 (Conv2D) (None, 6, 6, 512) 2359808 conv2d_42 (Conv2D) (None, 6, 6, 512) 2359808 conv2d_43 (Conv2D) (None, 6, 6, 512) 2359808 max_pooling2d_29 (MaxPoolin (None, 3, 3, 512) 0 g2D) flatten_5 (Flatten) (None, 4608) 0 dense_11 (Dense) (None, 4096) 18878464 dropout_26 (Dropout) (None, 4096) 0 dense_12 (Dense) (None, 4096) 16781312 dropout_27 (Dropout) (None, 4096) 0 dense_13 (Dense) (None, 6) 24582 ========== ========== Total params: 49,808,966 Trainable params: 49,808,966 Non-trainable params: 0	
2_25122	0.149	Model: "sequential_VGG2" _____ _____ Layer (type) Output Shape Param # ========== ========== conv2d_44 (Conv2D) (None, 96, 96, 64) 1792 conv2d_45 (Conv2D) (None, 96, 96, 64) 36928 max_pooling2d_30 (MaxPoolin (None, 48, 48, 64) 0 g2D) conv2d_46 (Conv2D) (None, 48, 48, 128) 73856 conv2d_47 (Conv2D) (None, 48, 48, 128) 147584 max_pooling2d_31 (MaxPoolin (None, 24, 24, 128) 0 g2D)	

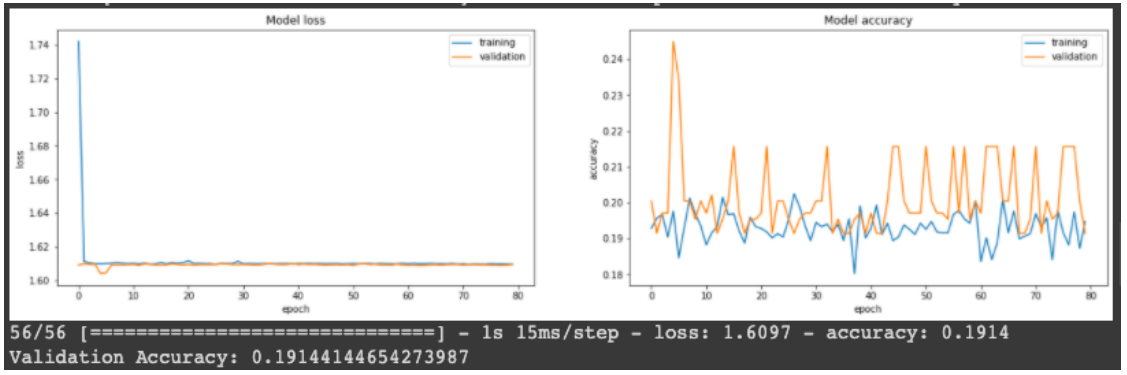
		conv2d_48 (Conv2D) (None, 24, 24, 256) 295168 conv2d_49 (Conv2D) (None, 24, 24, 256) 590080 max_pooling2d_32 (MaxPoolin (None, 12, 12, 256) 0 g2D) conv2d_50 (Conv2D) (None, 12, 12, 512) 1180160 conv2d_51 (Conv2D) (None, 12, 12, 512) 2359808 conv2d_52 (Conv2D) (None, 12, 12, 512) 2359808 max_pooling2d_33 (MaxPoolin (None, 6, 6, 512) 0 g2D) flatten_6 (Flatten) (None, 18432) 0 dense_14 (Dense) (None, 4096) 75501568 dropout_28 (Dropout) (None, 4096) 0 dense_15 (Dense) (None, 4096) 16781312 dropout_29 (Dropout) (None, 4096) 0 dense_16 (Dense) (None, 6) 24582 ========== Total params: 99,352,646 Trainable params: 99,352,646 Non-trainable params: 0	
VGG16 3 layers 'horizontal_flip': True, 'vertical_flip': True, 'rotation_range': 20, 'width_shift_range':	0.562	Model: "sequential_VGG3" _____ Layer (type) Output Shape Param # ========== conv2d (Conv2D) (None, 96, 96, 64) 1792 conv2d_1 (Conv2D) (None, 96, 96, 64) 36928 max_pooling2d (MaxPooling2D (None, 48, 48, 64) 0) conv2d_2 (Conv2D) (None, 48, 48, 128) 73856 conv2d_3 (Conv2D) (None, 48, 48, 128) 147584	

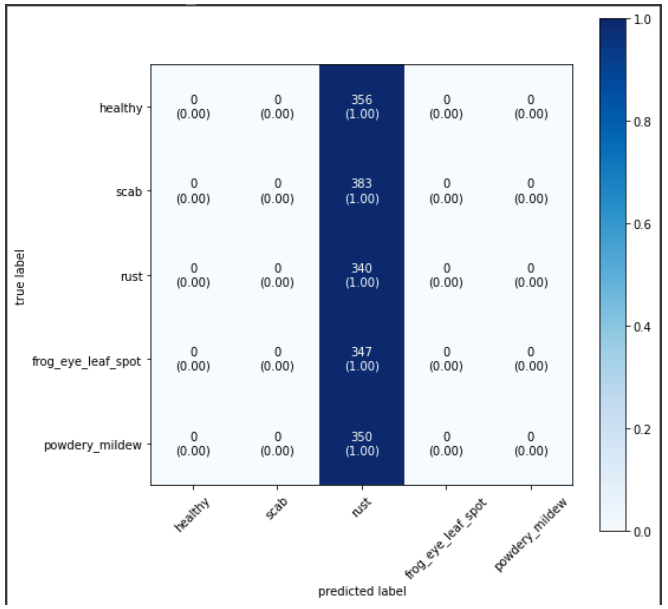
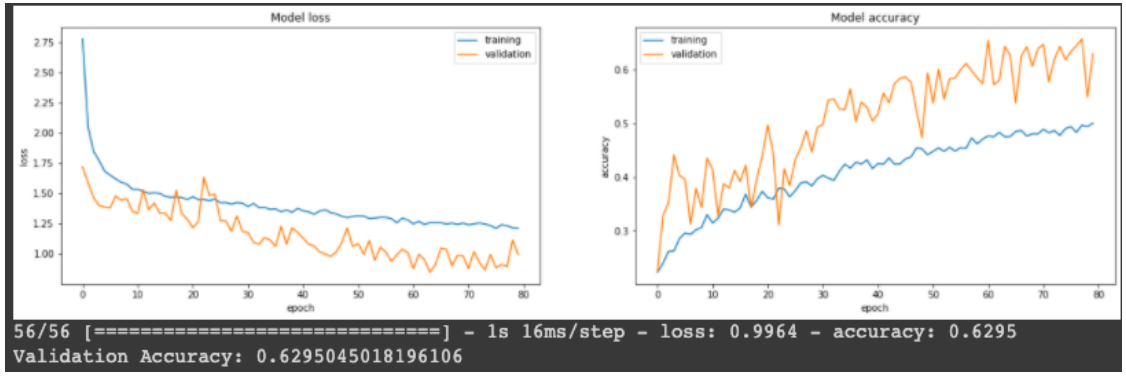
0.1, 'height_shift_range': 0.1, 'zoom_range': 0.3, 'brightness_range': [0,1.0], 'channel_shift_range' : 0.3, 'shear_range' : 0.9		max_pooling2d_1 (MaxPooling 2D) (None, 24, 24, 128) 0 conv2d_4 (Conv2D) (None, 24, 24, 256) 295168 conv2d_5 (Conv2D) (None, 24, 24, 256) 590080 max_pooling2d_2 (MaxPooling 2D) (None, 12, 12, 256) 0 flatten (Flatten) (None, 36864) 0 dense (Dense) (None, 4096) 150999040 dropout (Dropout) (None, 4096) 0 dense_1 (Dense) (None, 4096) 16781312 dropout_1 (Dropout) (None, 4096) 0 dense_2 (Dense) (None, 5) 20485 ========== Total params: 168,946,245 Trainable params: 168,946,245 Non-trainable params: 0	
VGG16 3 layers no augmentations	0.36		 147/147 [==========] - 4s 25ms/step - loss: 39573452.0000 - accuracy: 0.3647 Validation Accuracy: 0.3646608591079712

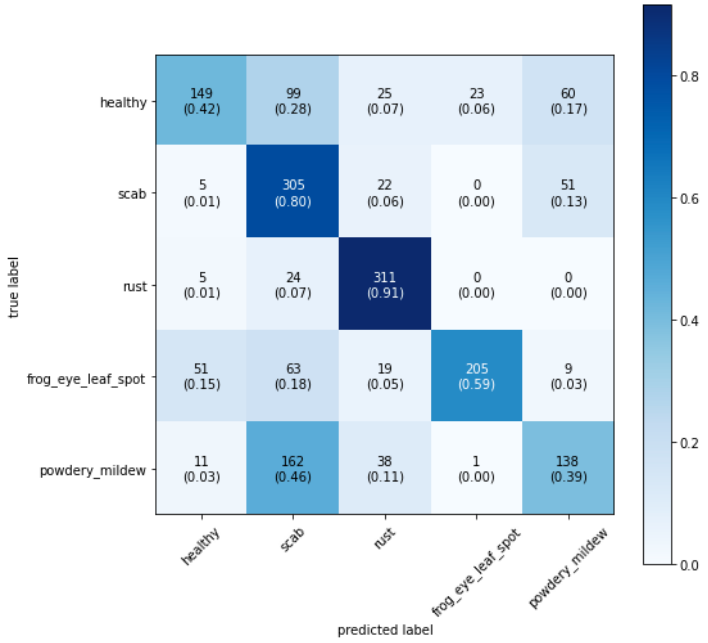
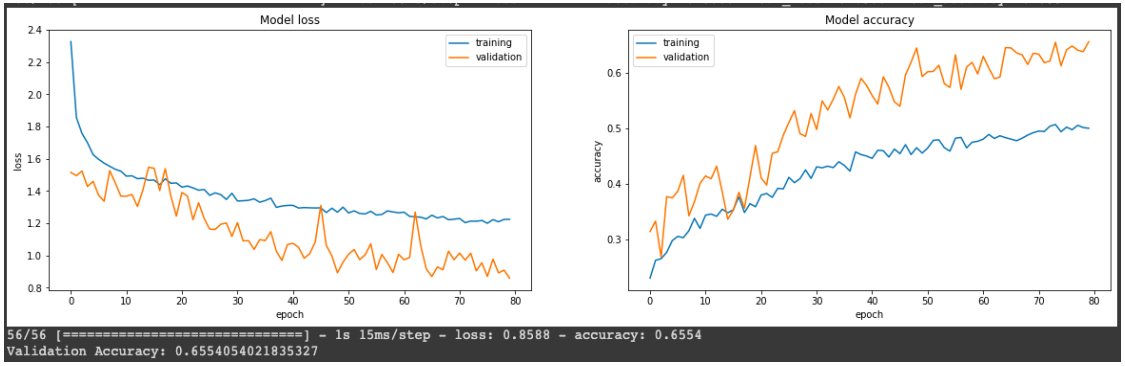
			
VGG16 2 layers no augment ations	0.49		 

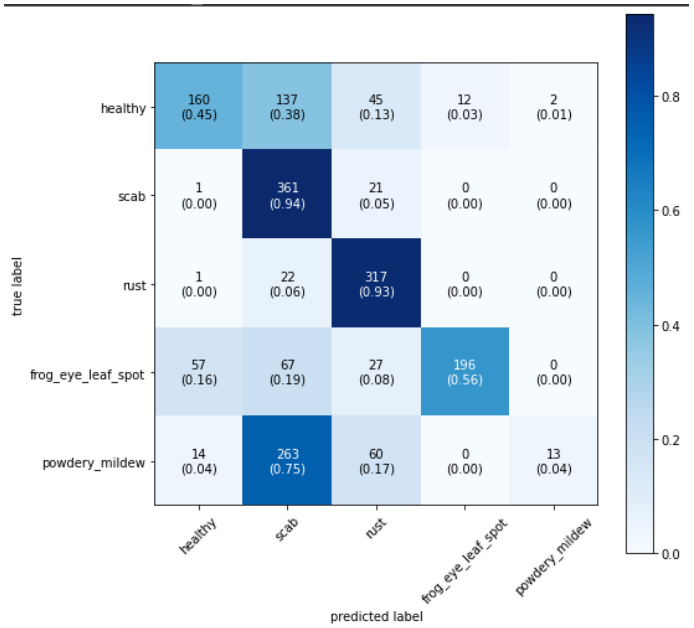
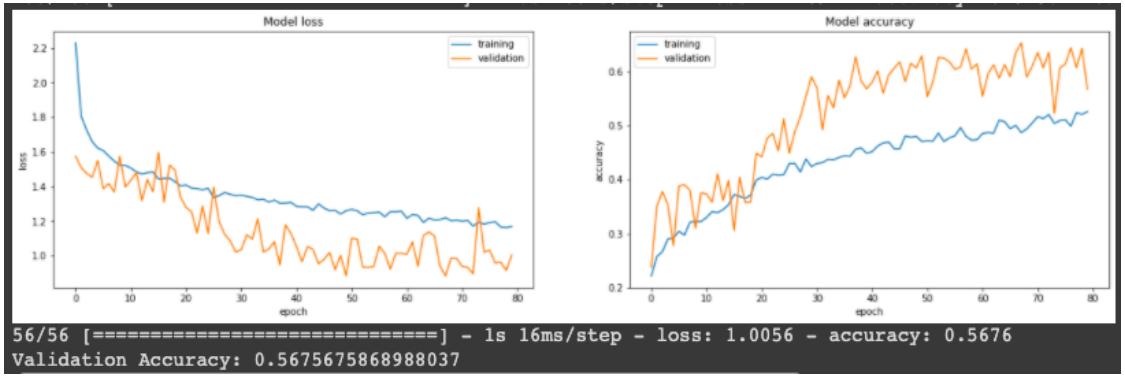
			

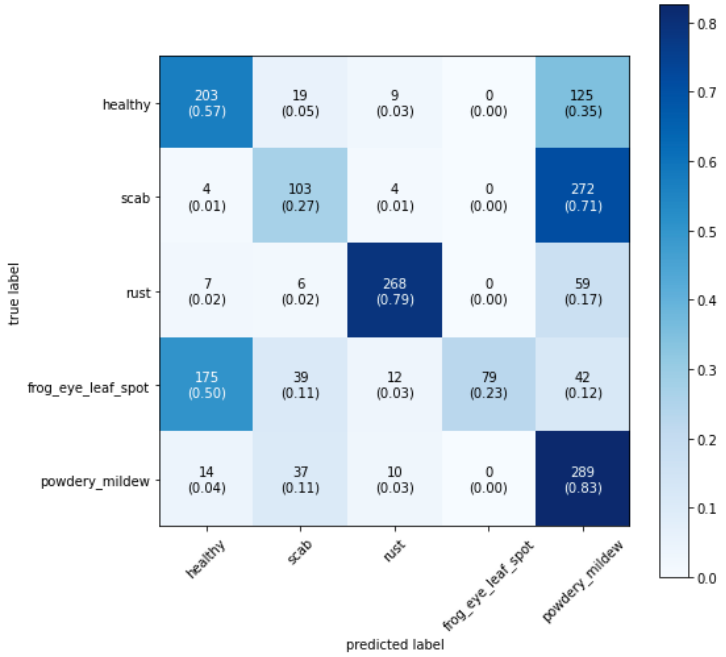
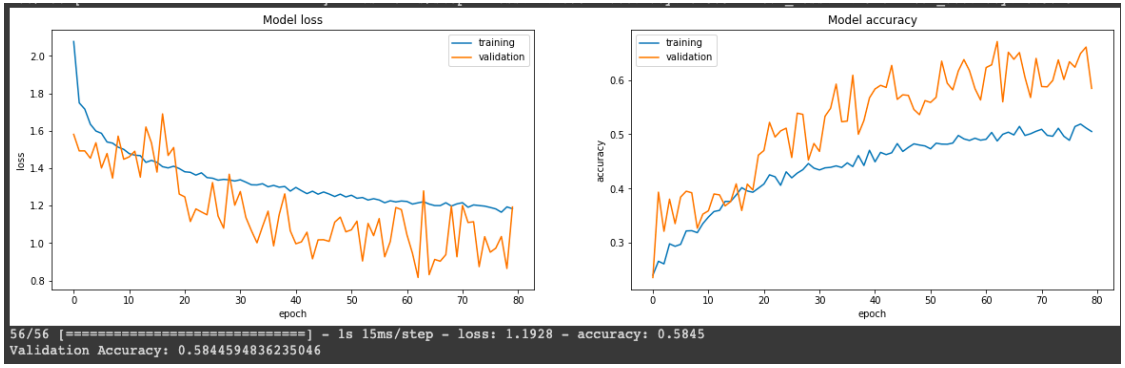
4.3.2 Leaky ReLU vs ReLU

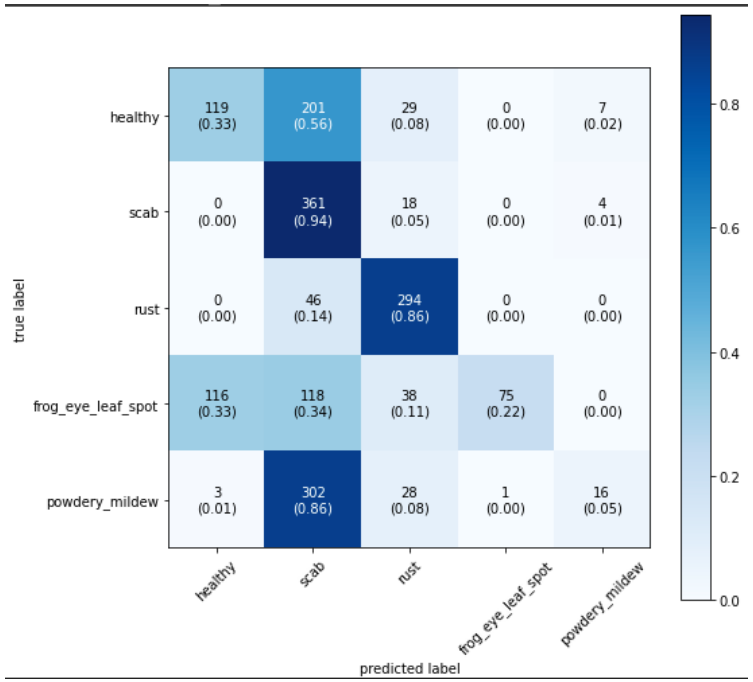
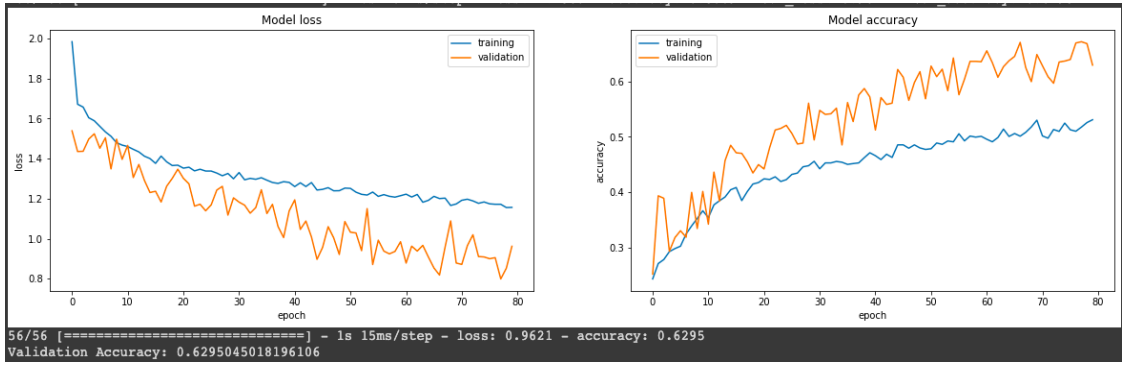
Trial ID	activation / alpha	avg. accuracy	plot / matrix
3_25122	ReLU	0.191	

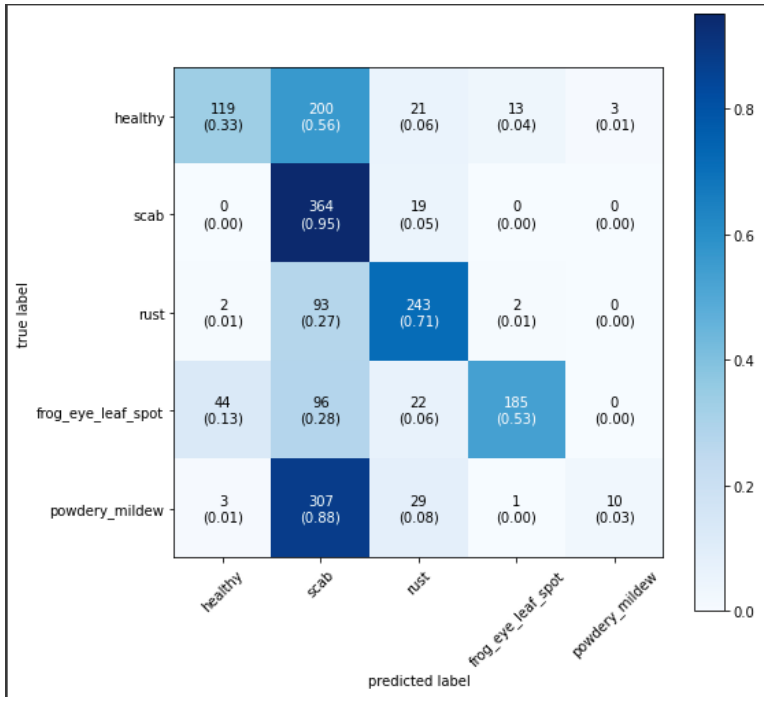
			
4_25122	Leaky ReLU, = 0.7	0.629	

			
5_25122	Leaky ReLU, = 0.6	0.655	

			
6_25122	Leaky ReLU, = 0.5	0.568	

			
7_25122	Leaky ReLU, = 0.4	0.585	

			
8_25122	Leaky ReLU, = 0.3 (default)	0.630	

4.3.3 Increased sample data and weights

Trial ID	augmentation(s)	avg. accuracy	plot / matrix

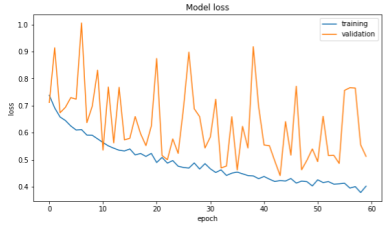
9_25122
full dataset, excl. combined and complex

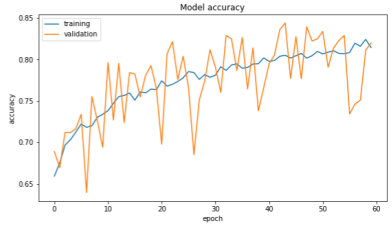
classes balanced with weights

no augmentations

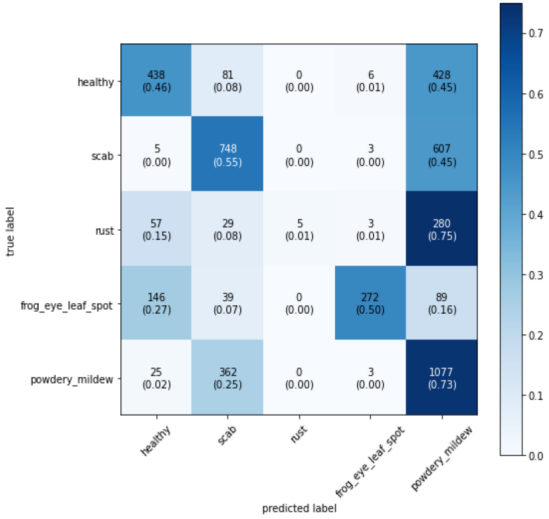
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape
=================================================================
conv2d (Conv2D)             (None, 96, 96, 3
max_pooling2d (MaxPooling2D  (None, 48, 48,
 dropout (Dropout)           (None, 48, 48,

 conv2d_1 (Conv2D)           (None, 48, 48,
 max_pooling2d_1 (MaxPooling  (None, 24, 24,
 batch_normalization (BatchN  (None, 24, 24,
 dropout_1 (Dropout)         (None, 24, 24,

 conv2d_2 (Conv2D)           (None, 24, 24,
 max_pooling2d_2 (MaxPooling  (None, 12, 12,
 batch_normalization_1 (Batc  (None, 12, 12,
 dropout_2 (Dropout)         (None, 12, 12,

 conv2d_3 (Conv2D)           (None, 12, 12,
 max_pooling2d_3 (MaxPooling  (None, 6, 6, 2
 dropout_3 (Dropout)         (None, 6, 6, 25

 conv2d_4 (Conv2D)           (None, 6, 6, 32
 max_pooling2d_4 (MaxPooling  (None, 3, 3, 3
 flatten (Flatten)           (None, 288)
 dropout_4 (Dropout)         (None, 288)
 dense (Dense)               (None, 256)
 dropout_5 (Dropout)         (None, 256)
 dense_1 (Dense)             (None, 5)

=================================================================
Total params: 538,213
Trainable params: 537,829
Non-trainable params: 384
_________________________________________________________________
```

0.81






```
147/147 [==============================] - 2s 15ms/step - loss: 0.5127 - accuracy: 0.8197
Validation Accuracy: 0.8196895718574524
```



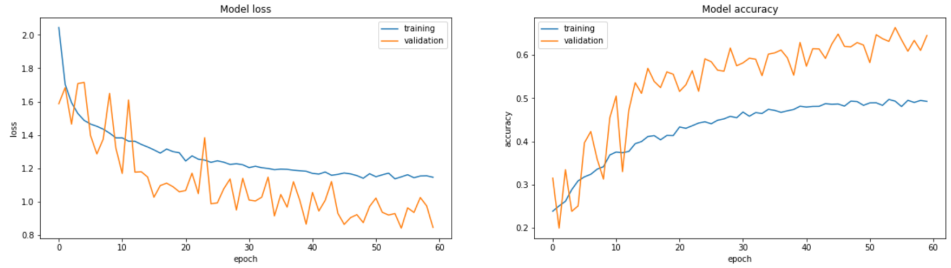
10_25122
full dataset, excl. combined and complex

classes balanced with weights

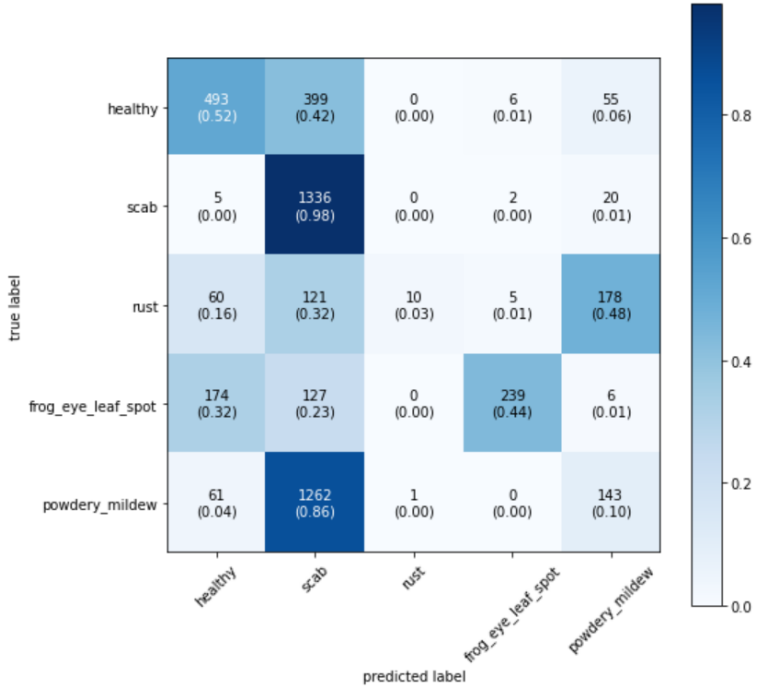
augment = {'horizontal_flip': True,
'vertical_flip': True,
'rotation_range': 20,
'width_shift_range': 0.1,
'height_shift_range': 0.1,
'zoom_range': [0,1.5],
'brightness_range': [0,1.5],
'channel_shift_range' : 0.9,
'shear_range' : 0.5
}

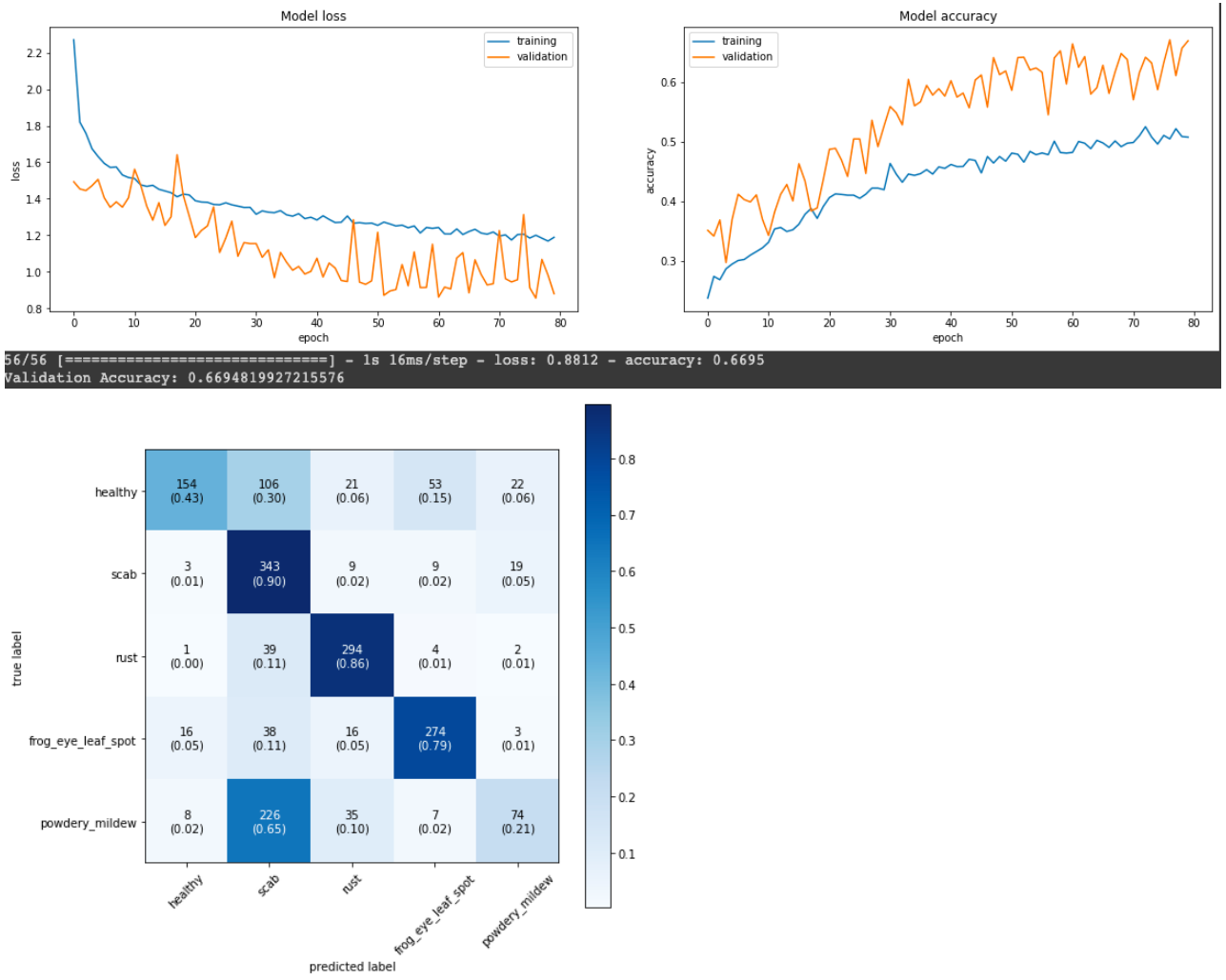
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape
=================================================================
 conv2d (Conv2D)             (None, 96, 96,
 max_pooling2d (MaxPooling2D  (None, 48, 48,
 dropout (Dropout)           (None, 48, 48,

 conv2d_1 (Conv2D)           (None, 48, 48,
 max_pooling2d_1 (MaxPooling  (None, 24, 24,
 batch_normalization (BatchN  (None, 24, 24,
 dropout_1 (Dropout)         (None, 24, 24,

 conv2d_2 (Conv2D)           (None, 24, 24,
 max_pooling2d_2 (MaxPooling  (None, 12, 12,
 batch_normalization_1 (Batc  (None, 12, 12,
 dropout_2 (Dropout)         (None, 12, 12,

 conv2d_3 (Conv2D)           (None, 12, 12,
 dropout_3 (Dropout)         (None, 6, 6, 25

 conv2d_4 (Conv2D)           (None, 6, 6, 32
max_pooling2d_4 (MaxPooling  (None, 3, 3, 32
 flatten (Flatten)           (None, 288)
 dropout_4 (Dropout)         (None, 288)
 dropout_5 (Dropout)         (None, 256)
 dense_1 (Dense)             (None, 5)

=================================================================
Total params: 538,213
Trainable params: 537,829
Non-trainable params: 384
_________________________________________________________________
```

0.67



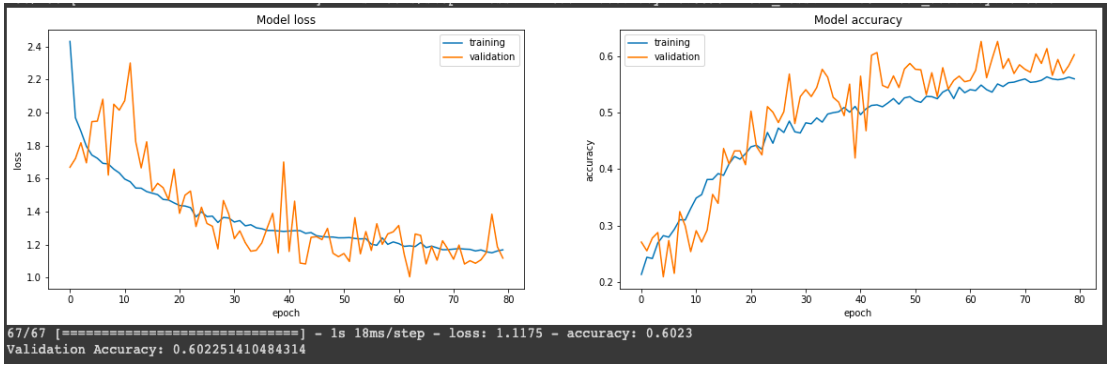

```
147/147 [==============================] - 2s 11ms/step - loss: 0.8461 - accuracy: 0.6451
Validation Accuracy: 0.6451201438903809
```

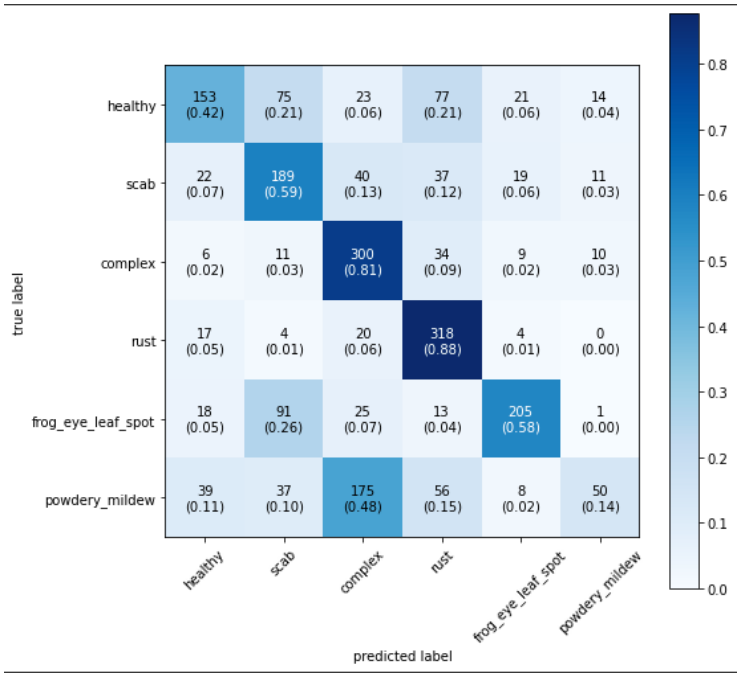


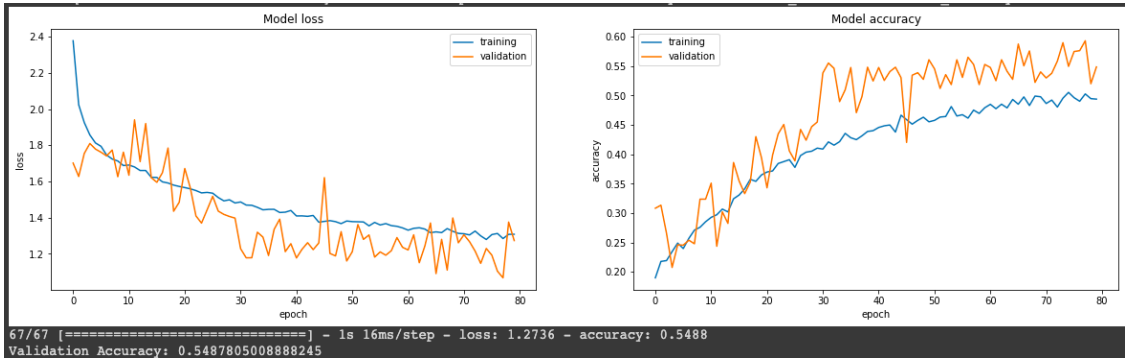
Trial ID	accuracy	plot / matrix
11_25122 - with samples of 1184 - predicting for 5 classes	**0.669**	

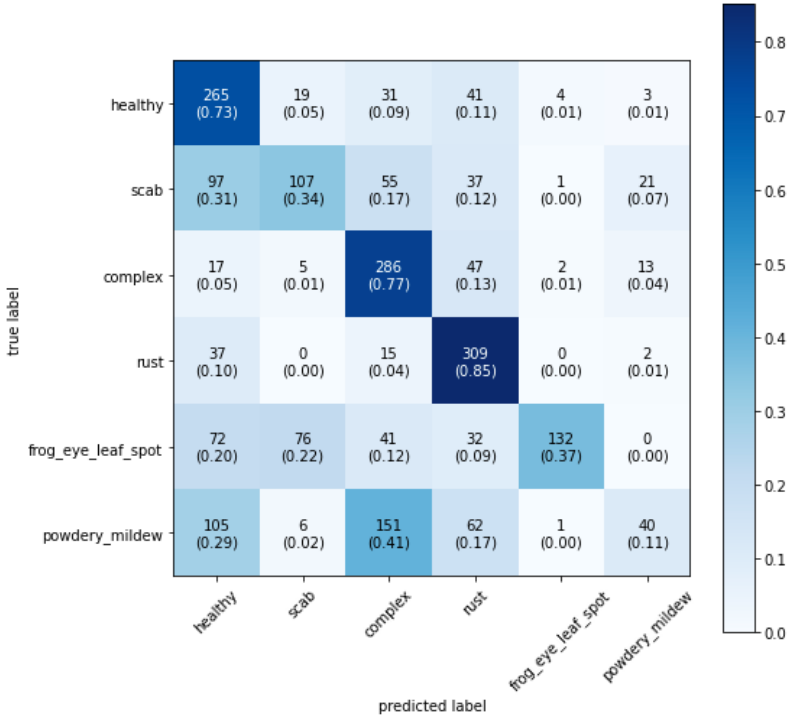
4.3.4 Data augmentation

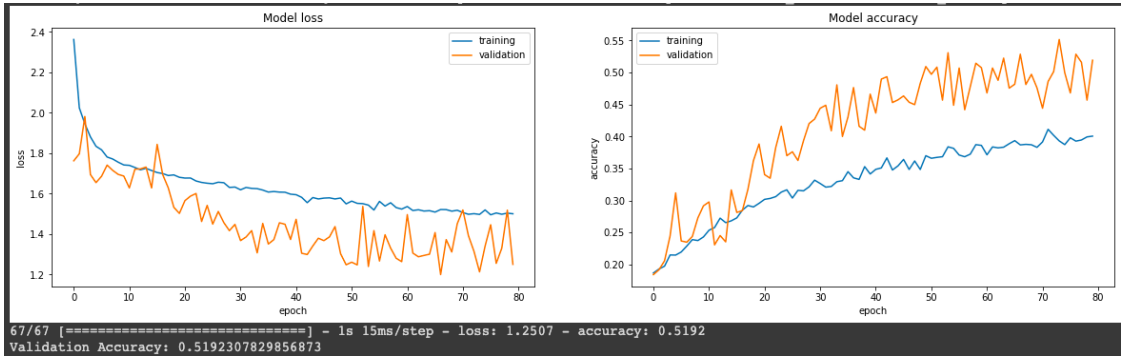
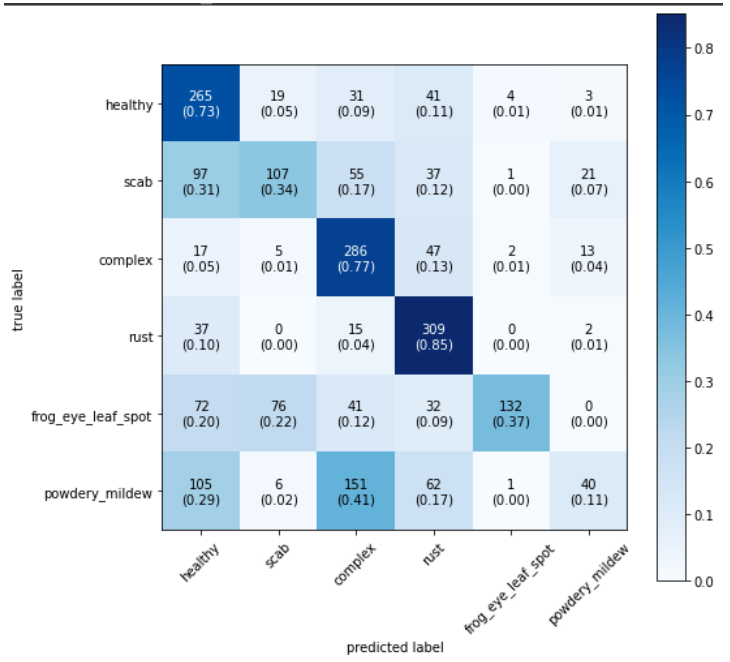
Trial ID	augmentation(s)	avg. accuracy	plot / matrix

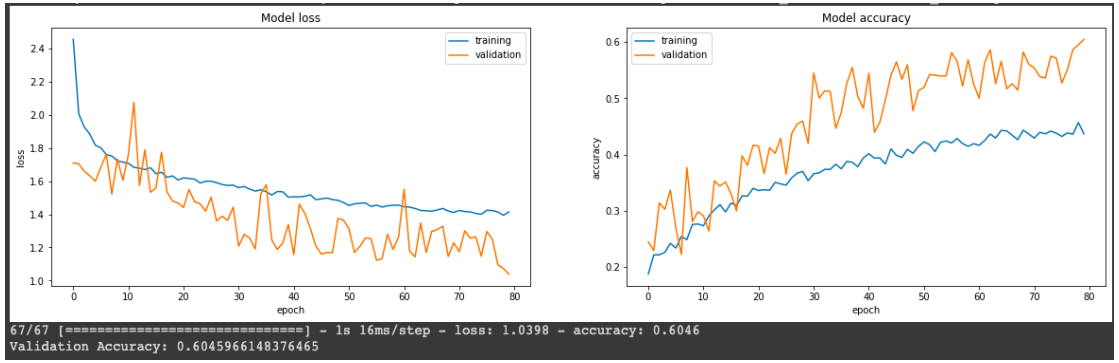
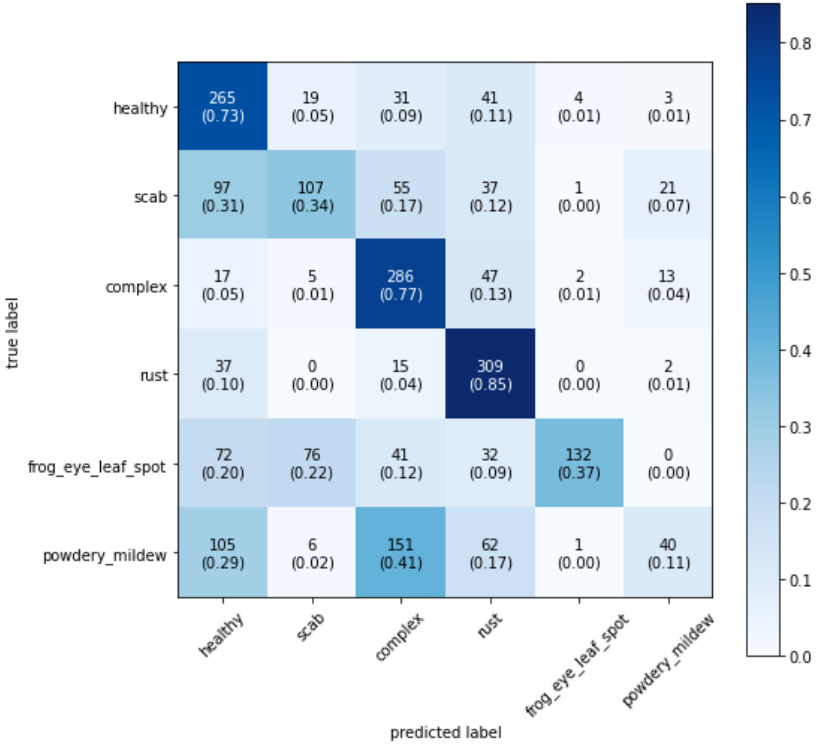
12_25122
 	
Not with all data
	`horizontal_flip': True,`
	`'vertical_flip': True,`
	`'rotation_range': 20,`
	`'width_shift_range': 0.1,`
	`'height_shift_range': 0.1,`
	`'zoom_range': [0.3,1.0],`
	`'brightness_range': [0.2,1.2],`
	`'channel_shift_range' : 0.7,`
	`'shear_range' : 0.3`
		0.603	

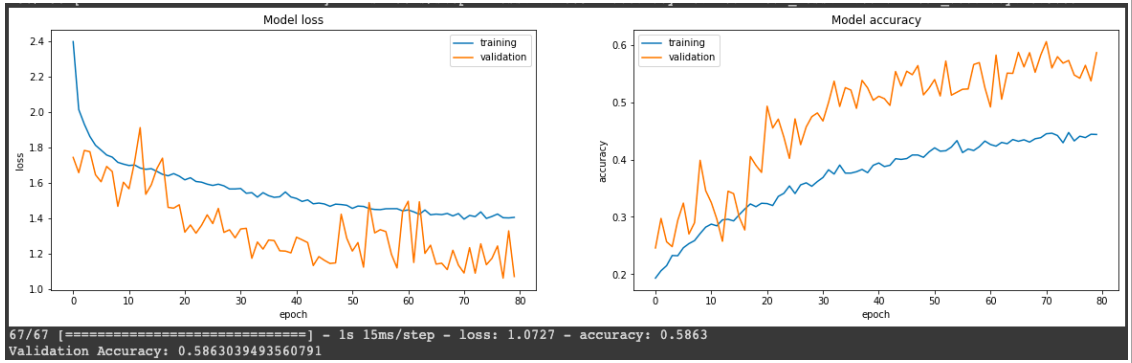
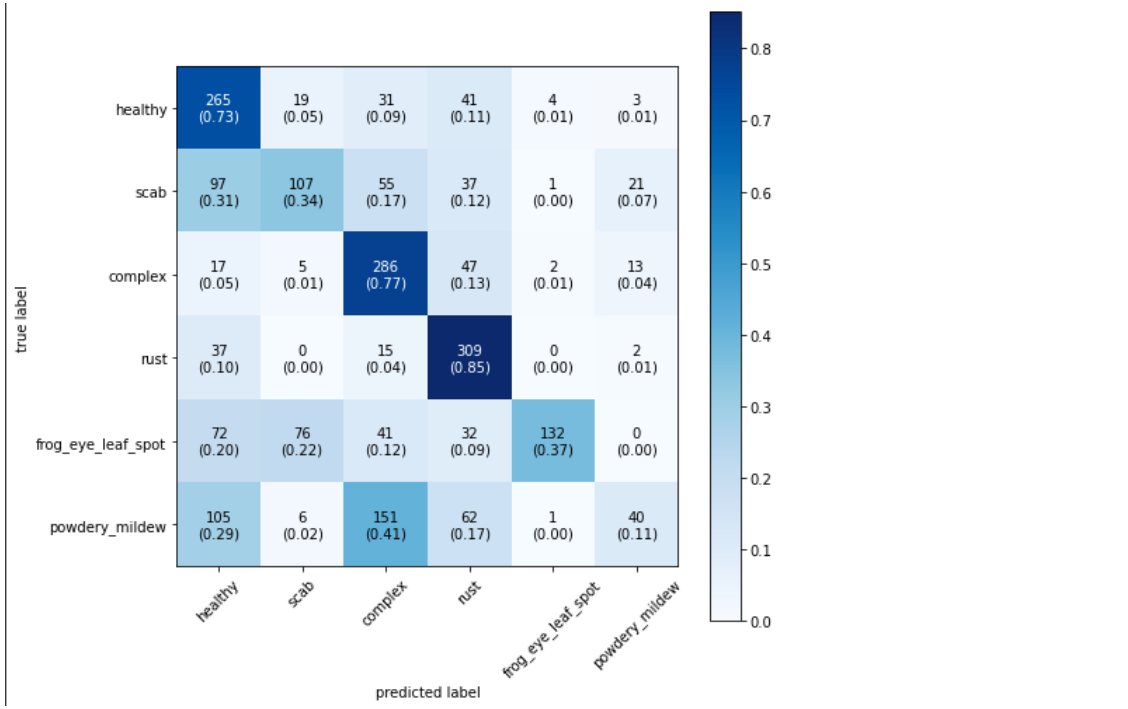
			

13_25122
 	
Not with all data
	`'horizontal_flip': True,`
	`'vertical_flip': True,`
	`'rotation_range': 30,`
	`'width_shift_range': 0.2,`
	`'height_shift_range': 0.2,`
	`'zoom_range': [0.3,1.0],`
	`'brightness_range': [0.1,1.5],`
	`'channel_shift_range' : 0.7,`
	`'shear_range' : 0.7`
		0.549	

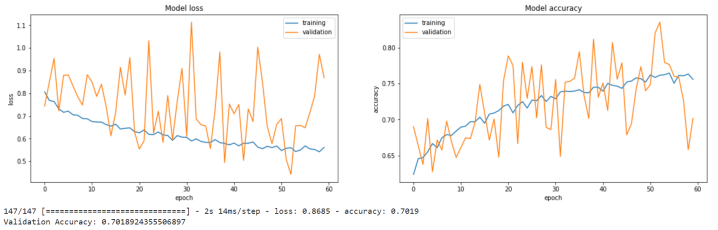
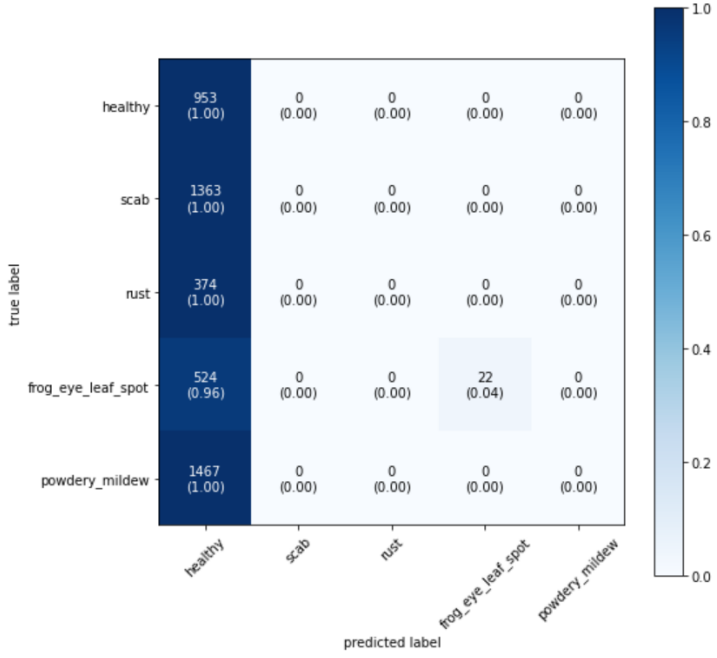
			 healthy scab complex rust frog_eye_leaf_spot powdery_mildew true label predicted label 265 (0.73) 19 (0.05) 31 (0.09) 41 (0.11) 4 (0.01) 3 (0.01) 97 (0.31) 107 (0.34) 55 (0.17) 37 (0.12) 1 (0.00) 21 (0.07) 17 (0.05) 5 (0.01) 286 (0.77) 47 (0.13) 2 (0.01) 13 (0.04) 37 (0.10) 0 (0.00) 15 (0.04) 309 (0.85) 0 (0.00) 2 (0.01) 72 (0.20) 76 (0.22) 41 (0.12) 32 (0.09) 132 (0.37) 0 (0.00) 105 (0.29) 6 (0.02) 151 (0.41) 62 (0.17) 1 (0.00) 40 (0.11) 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0

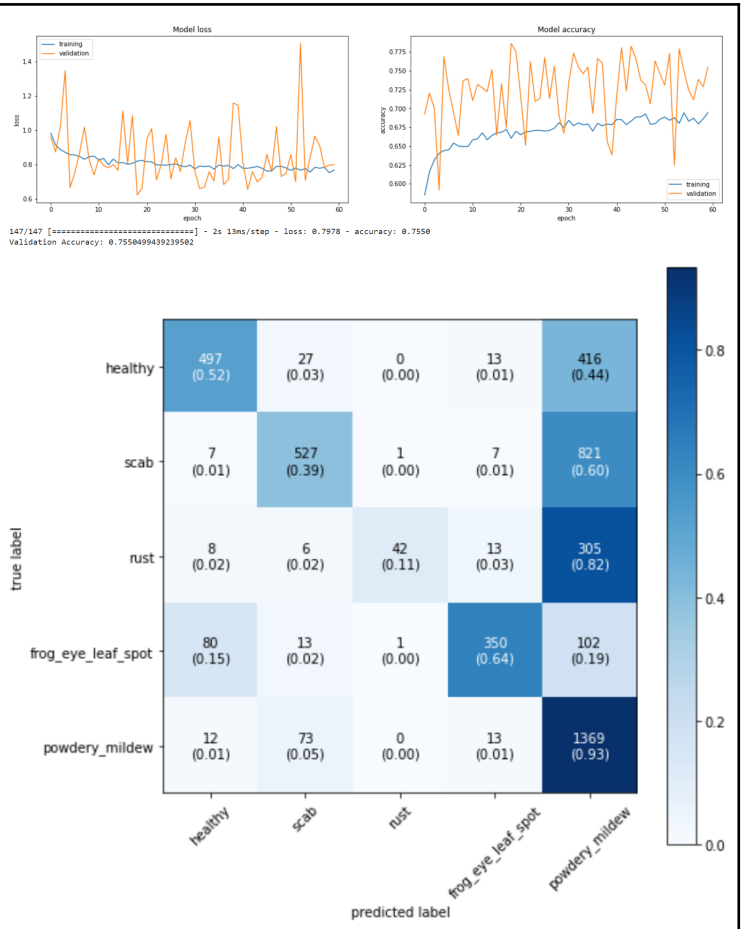
14_25122
 	
Not with all data
	`'horizontal_flip': True,`
	`'vertical_flip': True,`
	`'rotation_range': 20,`
	`'width_shift_range': 0.2,`
	`'height_shift_range': 0.2,`
	`'zoom_range': [0,1.0],`
	`'brightness_range': [0,1.5],`
	`'channel_shift_range' : 0.9,`
	`'shear_range' : 0.3`
		0.519	
			 

15_25122
 	
Not with all data
	`'horizontal_flip': True,`
	`'vertical_flip': True,`
	`'rotation_range': 20,`
	`'width_shift_range': 0.1,`
	`'height_shift_range': 0.1,`
	`'zoom_range': [0,1.5],`
	`'brightness_range': [0,1.5],`
	`'channel_shift_range' : 0.9,`
	`'shear_range' : 0.5`
		0.605	
			 

16_25122
 	
Not with all data
	`'horizontal_flip': True,`
	`'vertical_flip': True,`
	`'rotation_range': 20,`
	`'width_shift_range': 0.1,`
	`'height_shift_range': 0.1,`
	`'zoom_range': [0,1.5],`
	`'brightness_range': [0,1.5],`
	`'channel_shift_range' : 0.9,`
	`'shear_range' : 0.9`
		0.586	
			 

all data

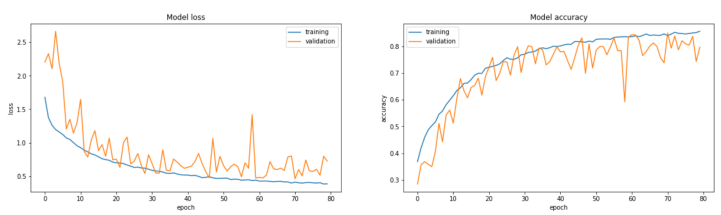
all data augment = {'horizontal_flip': True, 'vertical_flip': True, 'rotation_range': 20, 'width_shift_range': 0.1, 'height_shift_range': 0.1, }	**0.70**	 147/147 [==========] - 2s 14ms/step - loss: 0.8685 - accuracy: 0.7019 Validation Accuracy: 0.7018924355506897 (<Figure size 576x576 with 2 Axes>, <matplotlib.axes._subplots.AxesSubplot at 0x7f0bed94d090>) 

all data augment = {'horizontal_flip': True, 'vertical_flip': True, 'rotation_range': 20, 'width_shift_range': 0.1, 'height_shift_range': 0.1, 'zoom_range': [0.3,1.0], 'brightness_range': [0.2,1.2], 'channel_shift_range' : 0.7, 'shear_range' : 0.3 }	**0.75**	

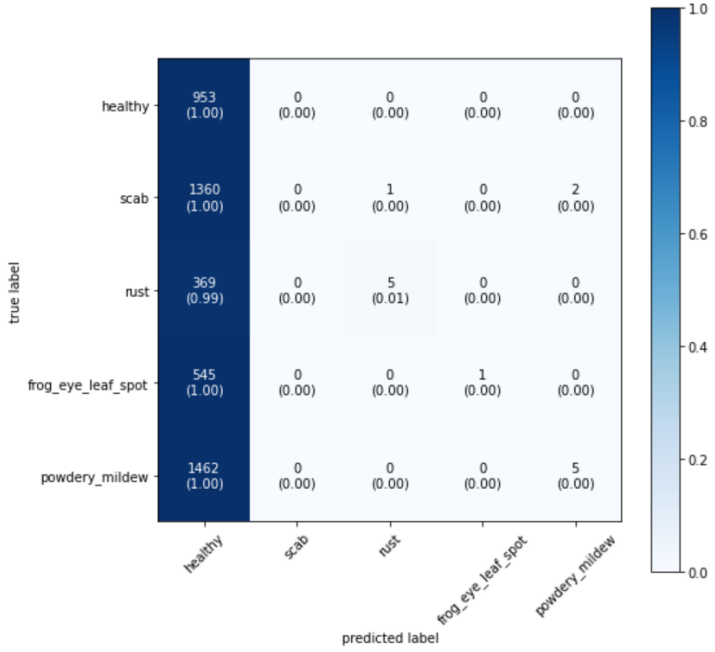
all data

no weights

no augmentations




```
147/147 [==============================] - 2s 13ms/step - loss: 0.7255 - accuracy: 0.7972
Validation Accuracy: 0.7971507310867310
```



4.4 DISCUSSION

4.4.1 The Confusion matrices

We have re-considered the programming of the confusion matrices. Unfortunately, we could not find any flaw in the programming. As we are nearing the end of the project, we have been able to look at the matrices with several TA's. We have reprogrammed the matrices in previous milestones but that did not seem to make a difference. After discussing the issue with Kelly, we suspected that there may be some issue with the way we use Google collab and the matrices may have been using the wrong data. For example, the matrices may have been using data from previous runs. Yet, we ran the programme on different google accounts, different computers and networks, on different days and after many reboots of the system. The problems persisted.

Therefore, we don't think that the issue lies there. Instead, it might be something in the code that we have been overlooking. For now, however, we have analysed the results of our experiments and considered, each time, whether the matrices seemed correct. Oftentimes, the matrices were correct or the data did not seem off by that much. On the other hand, the matrices were off significantly in many cases as well in such a way that we did not want to rely on them. For that reason, we have continued to take the matrices into account only with a grain of salt. That is, assuming that some do not accurately display the workings of the model. We did take some matrices into account for our analysis when they did not seem to be off by that much and/or indicated some interesting aspects of the model.

4.4.2 VGG

The VGG-16 model seems to be clearly overfitting to the training data. At first, we ran the VGG with five MaxPooling layers, each preceded by two convolutional layers. We suspected that the low accuracy may have been due to the fact that we had used the same number of layers as the model in the article whilst our pictures have significantly less pixels (96 x 96 as opposed to 224 x 224). We checked to see whether removing a MaxPooling layer and the two convolutional layers that precede it would improve the functioning of the model. The removal of the layers did not seem to improve the model, as the accuracy remained low and the model was still equally / significantly overfitting. We decided to remove additional layers, and evaluate the results. The removal of each additional layer did improve the accuracy of the model to 49% for the model with two MaxPooling layers. With augmentations, the VGG model improved and showed an accuracy of 56%. The improvements are illustrated by the plots shown in the results section on the VGG-16 models.

Unfortunately, we ran out of GPU so we could not test the accuracy of the model with the removal of another layer. However, we did not think that this is problematic because we did not expect the removal of another layer to aid in the model accuracy because then the model would have only 2 convolutional layers, followed by a single max pooling layer, before the model was flattened. This is quite similar to the basic model that we built in milestone 1, in relation to the amount of layers. As discussed in previous milestones, a simple model like that is probably not capable of learning complex features.

The results of the VGG model indicated to us that the VGG model may not be more appropriate at our image classification task than the model with the diamond shape that we had built so far. Instead of trying more VGG models or continuing to tweak the VGG-16 model, we decided to focus on other techniques to improve the model with the diamond shape.

Throughout our ML courses, we have learned that increasing the training data can significantly improve the model's ability to identify images correctly.We considered that increasing the amount of images available in the training data could lead to higher gains. For this reason, we decided to move back to our original model, the diamond shape, and experiment especially with increased sample data and changes in the distribution of the data.

4.4.3 Increased sample data and weights

The accuracy of the model seems to improve significantly when we use more training data and the weighted classes. Without augmentations, the accuracy of the model shot up to 81%. This indicates that the increased sample data and the use of weights have a positive effect on our model's ability to learn features. Yet, the matrix associated with this model indicates that the model often mistakenly classifies images as powdery mildew. In this respect, the model might be able to improve significantly. Yet, these matrices may not accurately reflect what the model is actually doing. Still, we tried to experiment with data augmentations in order to improve accuracy of the model, especially in relation to powdery mildew. This decreased the accuracy but the model seemed to be able to predict certain classes better as the matrix highlighted more boxes along the diagonal, indicating a higher accuracy for each class. The results of the experiments with data augmentations are discussed in a section below, on data augmentation.

4.4.4 Leaky ReLU vs ReLU

Furthermore, we have also experimented with the activation function to confirm the most optimal as stated in the methods-section. The results illustrate a comparison of the activation function ReLU along with Leaky ReLU at different alpha's. It is shown that ReLU did not return promising results, in which it seems that the network wasn't able to learn anything. The accuracy is very low and the confusion matrix shows the predictions for only a single class. Based on these results, we have opted to try out different alpha values for the Leaky ReLU activation function. The differences between the values are not great, however the value of alpha at 0.6 did show the highest accuracy of 65.5%. Other than that, it's confusion matrix shows a prediction for all classes. The value of alpha at 0.7 showed an accuracy of 62.9%, along with a more outspread prediction illustrated by the confusion matrix. Since the highest value shows a better distribution between classes, we opt that alpha 0.7 is the most optimal parameter for our network.

4.4.5 Data augmentation

We quickly ran out of memory whenever we used all of the data so we could not run the model with many different (combinations of) augmentations. This was to be expected since we are using much more data to train the model on. We picked the (combination of) augmentations that have shown to lead to higher accuracies during our previous milestones.

The changes we made for the data augmentation resulted in slightly different accuracies. But as you can see in the confusion matrices above, these data augmentations may help the model to differentiate between the classes more accurately and predict the correct classes. This is indicated by the diagonal lines that run across these matrices, which was often not the case for the models that we ran without augmentations. Some of these experiments have been executed on the previous distributions of data. They did not show significant improvements in accuracy. However, when we increased the dataset, and used all of the data, excluding combined classes, the accuracies shot up to 70 - 75 %. This indicates that the augmentations can have a positive effect on the accuracy of the model.

4.5 CONCLUSION

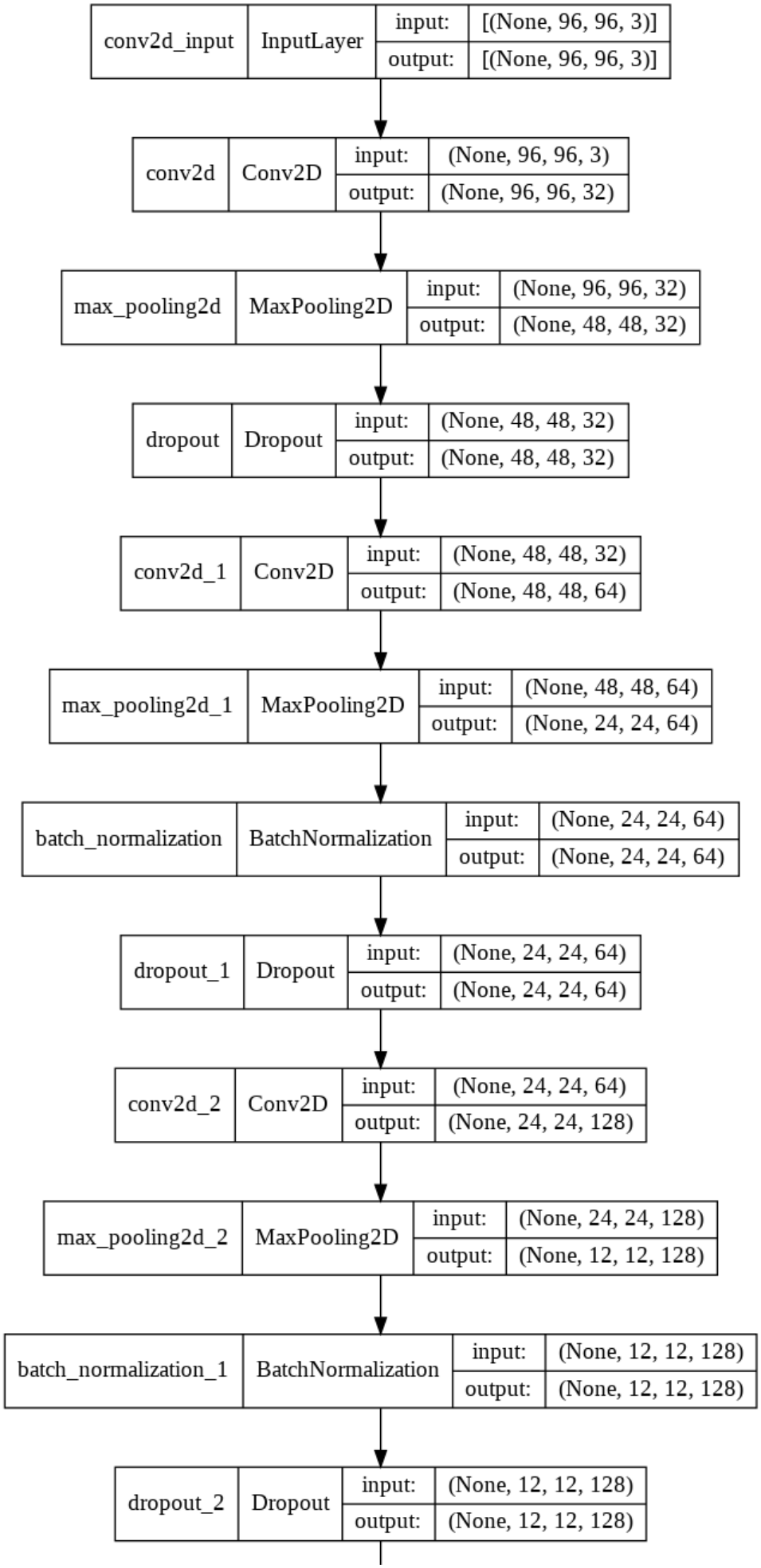
We have applied a variety of techniques to improve our model and we have also had varying results with each of them. It seems that some techniques, like VGG and ReLU, clearly do not improve the model's ability to accurately classify objects. These findings have been very helpful in clarifying what does not work, and pointing us towards techniques that do seem to work. In this respect, we have mostly been taking what has been working so far, data augmentation and improved sampling, in order to further improve the model.

The improved use of our data, which entails that we used much more of it and left out some data that made classification more difficult such as the complex class, significantly contributed to the accuracy of the model. However, this solution was not perfect because the model had difficulty predicting certain classes like rust. We suspected that data augmentation could help to improve the model at this point because, as the results section illustrated, different (combinations of) augmentations seemed to have positive effects on the model's ability to learn features. Yet, the use of data augmentation was unable to significantly improve the model as the accuracy of the model seemed to go down when the augmentations were combined along with increased sample data and weights.

Overall, the model with the highest accuracy (0.81) is the model 9_25122 that trains on the full dataset, excluding combined and complex classes, balanced with weights, applying no augmentations. This seemed quite surprising because the augmentations have shown promising results and have improved previous versions of our model. It may be the case that some other combinations of augmentations, which we were unable to test, may still increase the accuracy of our model. For now, however, the final model looks as follows:

```
# ML MODEL ARCHITECTURE
# Define Sequential model
model = models.Sequential()

# create convolutional layer and max pooling layer
model.add(layers.Conv2D(32, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same', input_shape=(96, 96, 3)))
model.add(layers.MaxPooling2D((2, 2)))

# create convolutional layer (larger) and max pooling layer
model.add(tf.keras.layers.Dropout(0.5))
model.add(layers.Conv2D(64, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.BatchNormalization())

# add Conv2D layer with 128 filters and max pooling layer
model.add(tf.keras.layers.Dropout(0.5))
model.add(layers.Conv2D(128, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.BatchNormalization())

# add Conv2D layer with 256 filters and max pooling layer
model.add(tf.keras.layers.Dropout(0.5))
model.add(layers.Conv2D(256, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))

# add Conv2D layer with 32 filters and max pooling layer
model.add(tf.keras.layers.Dropout(0.5))
model.add(layers.Conv2D(32, (3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.5), padding='same'))
model.add(layers.MaxPooling2D((2, 2)))

# flatten layers
model.add(layers.Flatten())
model.add(tf.keras.layers.Dropout(0.2))
model.add(layers.Dense(256, activation=tf.keras.layers.LeakyReLU(alpha=0.5)))

# apply softmax activation for final layer classification
model.add(tf.keras.layers.Dropout(0.2))
model.add(layers.Dense(5, activation='softmax'))
```

```
# normalize input data: set preprocesing dictionary
preprocess = {'featurewise_center': True, 'featurewise_std_normalization' : True}

# augment data: set augmentation dictionary
augment = {'horizontal_flip': True,
           'vertical_flip': True,
           'rotation_range': 20,
           'width_shift_range': 0.1,
           'height_shift_range': 0.1,
           'zoom_range': [0,1.5],
           'brightness_range': [0,1.5],
           'channel_shift_range' : 0.9,
           'shear_range' : 0.9}

# run training and evaluation function
train_and_evaluate(model, x_train, y_train, x_val, y_val, preprocess, epochs = 80, augment = augment, class_weight = class_weight)
```


conv2d_input
InputLayer
input: [(None, 96, 96, 3)]
output: [(None, 96, 96, 3)]
conv2d
Conv2D
input: (None, 96, 96, 3)
output: (None, 96, 96, 32)
max_pooling2d
MaxPooling2D
input: (None, 96, 96, 32)
output: (None, 48, 48, 32)
dropout
Dropout
input: (None, 48, 48, 32)
output: (None, 48, 48, 32)
conv2d_1
Conv2D
input: (None, 48, 48, 32)
output: (None, 48, 48, 64)
max_pooling2d_1
MaxPooling2D
input: (None, 48, 48, 64)
output: (None, 24, 24, 64)
batch_normalization
BatchNormalization
input: (None, 24, 24, 64)
output: (None, 24, 24, 64)
dropout_1
Dropout
input: (None, 24, 24, 64)
output: (None, 24, 24, 64)
conv2d_2
Conv2D
input: (None, 24, 24, 64)
output: (None, 24, 24, 128)
max_pooling2d_2
MaxPooling2D
input: (None, 24, 24, 128)
output: (None, 12, 12, 128)
batch_normalization_1
BatchNormalization
input: (None, 12, 12, 128)
output: (None, 12, 12, 128)
dropout_2
Dropout
input: (None, 12, 12, 128)
output: (None, 12, 12, 128)

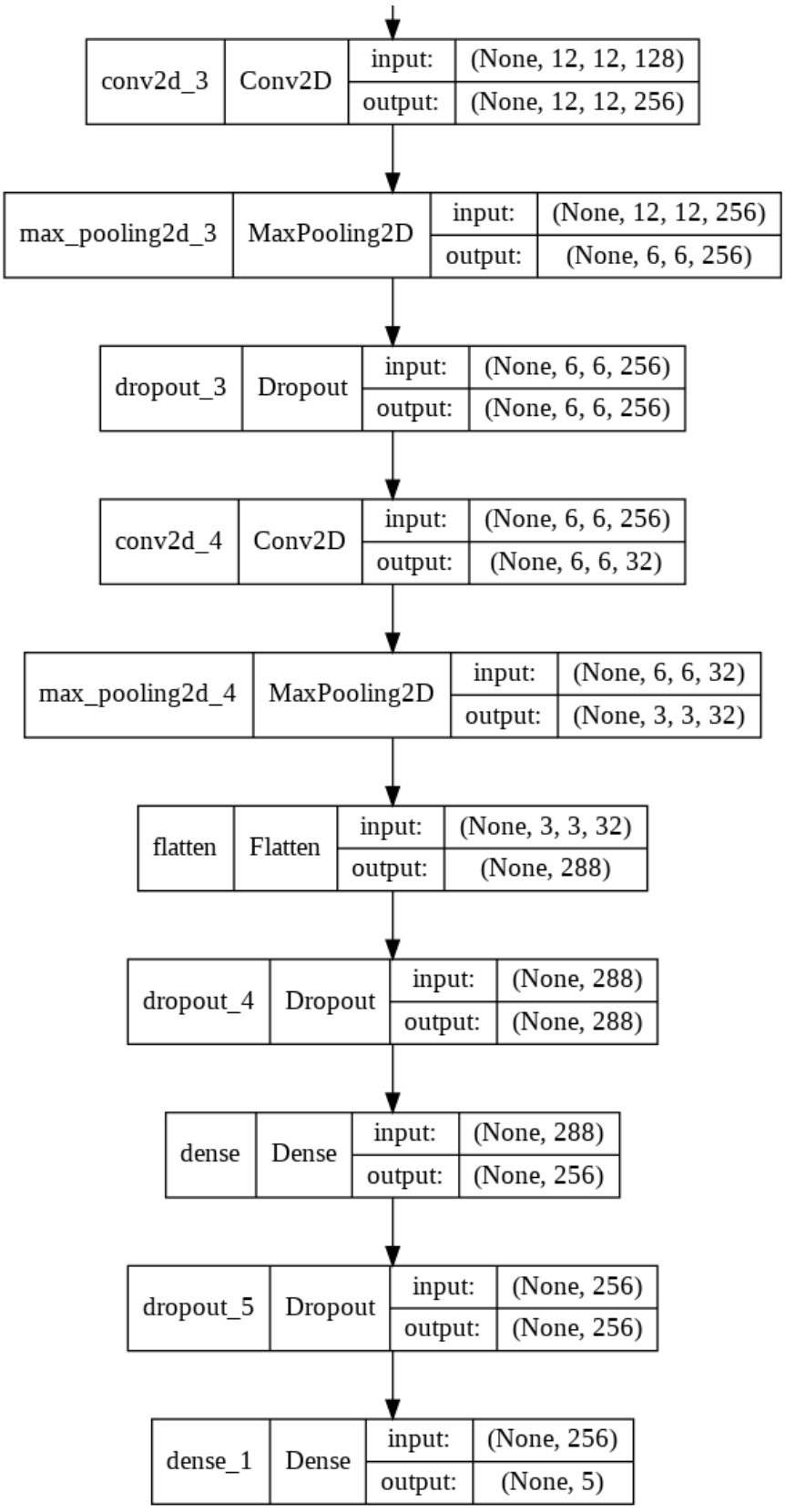


Figure 13 - Final network-configuration of Milestone 4/5