1 Introduction

1.1.1 Plant Disease Classification

Early identification of plant diseases is very important as plant diseases affect the growth of their respective species. Advancements in machine learning deep learning, has shown great potential in terms of increased accuracy of classification when compared to conventional machine learning approaches [58]. Traditional classification methods, such as laboratory testing of plant specimens and naked-eye observation have many limitations, for example, being time consuming and subjective. Deep learning methods based on convolutional neural networks (CNNs) solve or partially solve the problems of traditional classification methods [59].

1.1.2 Deep Learning

Deep learning was used in this project in combination with NDVI to monitor plant health. Deep learning is defined as neural networks (NNs) with large numbers of parameters containing one or more hidden layers. These deep NNs include layers in one of the four fundamental network architectures unsupervised pretrained networks, convolutional neural networks (CNNs), recurrent neural networks and recursive neural networks [60]. CNNs are a class of NNs that are known for their dominant performance in various computer vision tasks [61] and were selected to perform the classification task of disease detection in plants for this section of the project.

1.1.3 Convolutional Neural Networks

CNNs are made from a combination of layers such as convolutional layers, pooling layers, and fully connected layers. CNNs are designed to learn image features automatically and adaptively through a backpropagation algorithm [61]. CNNs are analogous to traditional artificial neural networks (ANNs) in that they consist of neurons that self-optimise through learning, the last layer of the CNN will contain a loss function which is used evaluate the performance of the network [62].

1.1.4 Convolutional Layers

Convolutional layers are used to extra features in CNNs [63]. A defined filter(kernel) convolves across an input image starting in the top left corner of the image. The pixel values represented in matrix form in Figure 11 are multiplied with the corresponding kernel values, the products are then summated, and a bias is added. This process is repeated as the kernel convolves across the image producing an output matrix [59].

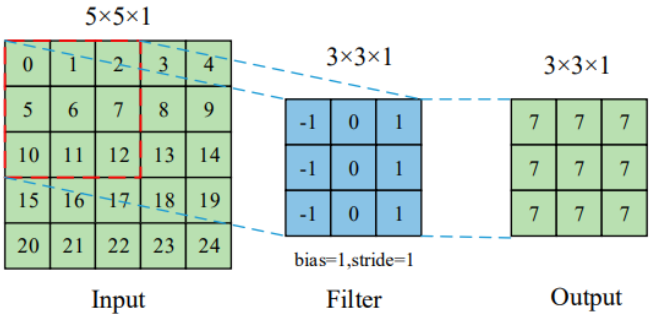


Figure 1 - Convolutional Layer Operations [59]

1.1.5 ResNet-50

The deep residual network ResNet-50 is a CNN that contains 50 layers (48 convolutional layers, one max pooling layer and one average pooling layer). A residual neural network contains "shortcut connections" between layers of the network. In ResNet-50 the "shortcut connections" are created using residual mappings between the network's layers [64]. Implementing residual learning techniques to deep NNs improves model performance as they allow network depth to be increased without compromising the ability to optimise network parameters (which is a common problem caused by vanishing gradients in large DNNs) [64] [65]. ResNet-50 was the CNN architecture chosen for this project.

1.1.6 Transfer Learning

Transfer learning is a technique where the weights of deep learning models trained using large datasets are used to as the starting weights for new models for similar tasks [66]. A common deep transfer learning strategy is fine-tuning, where the neural network is first trained on a large dataset, followed by transferring the first n layers of the trained network to the target network and then transferring the remaining layers with untrained weights and biases (randomly initialised values) [67].

In this project, all layers of ResNet-50 were initialised with the weights achieved from training on the ImageNet dataset. ImageNet is a database of images organised according to the WordNet hierarchy where each node of the hierarchy is depicted by hundreds and thousands of images [68].

1.1.7 Model Training

Training neural networks involves optimising the weights and biases of the network until corresponding input images give the desired output at the final layer (classification layer in this case). Model weights and biases are optimised using a loss function; this quantifies the performance of the model [69]. The loss function used to train the plant disease classification models was categorical cross entropy. The equation for categorical cross entropy is given in equation 33.

$$Loss = -\sum_{i=1}^{M} y_i \log \hat{y}_i \qquad (33)$$

Where M is the scalar value representing the output size (equal to the number of classes), $\hat{y}_i$ is the scalar value of each model output (the scalar value for each class) and y_i is the corresponding target value (output from the softmax activation function on the layer before the classification layer).

The softmax activation function is used to compute a probability distribution from a vector of real numbers, it produces an output which is a range of values between 0 and 1, with the sum of probabilities been equal to 1 [70]. The softmax function is chosen over a sigmoid activation function as this is a multivariate classification task rather than a binary classification task. The equation of the softmax function is given in equation 34.

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{M} e^{z_j}} \qquad (34)$$

Where z_i is the input vector, e^{z_i} is the exponential of the input vector, M is the number of classes in the multivariate classification task, e^{z_j} is the exponential of the output vector.

1.1.8 Datasets

In this project, three datasets were used PlantDoc [71], PlantVillage [72] and A Citrus Fruits and Leaves Dataset for Detection and Classification of Citrus Diseases through Machine Learning [73]. A summary of the original datasets is shown in Table 2. Large, high-quality datasets are important to ensure that trained CNNs produce good results [74].

Table 1 - Summary of Original Datasets [71] [72] [73]

Dataset	No. Images	No. Plant Species	No. Plant Diseases	Contains Healthy Plants	Conditions
PlantVillage	54303	14	26	Yes	Laboratory
PlantDoc	2598	13	17	Yes	Field
Citrus Fruits	759	1	4	Yes	Field

The datasets were augmented, and unwanted plants removed. A summary of the final dataset can be found in the methodology.

1.1.9 Image Augmentation

Data augmentation uses image processing techniques to enhance the size and quality of a training datasets such that better deep learning models can be trained. Data augmentation allows the expansion of limited datasets to take advantage of the capabilities of big data [75].

CNNs learn features of images with large numbers of parameters making them susceptible to overfitting to the training dataset and not generalising, this leads to poor performance on unseen test datasets [76]. An example of overfitting is shown in Figure 12. Data augmentation is essential to reduce overfitting in models and in producing more reliable CNNs [77].

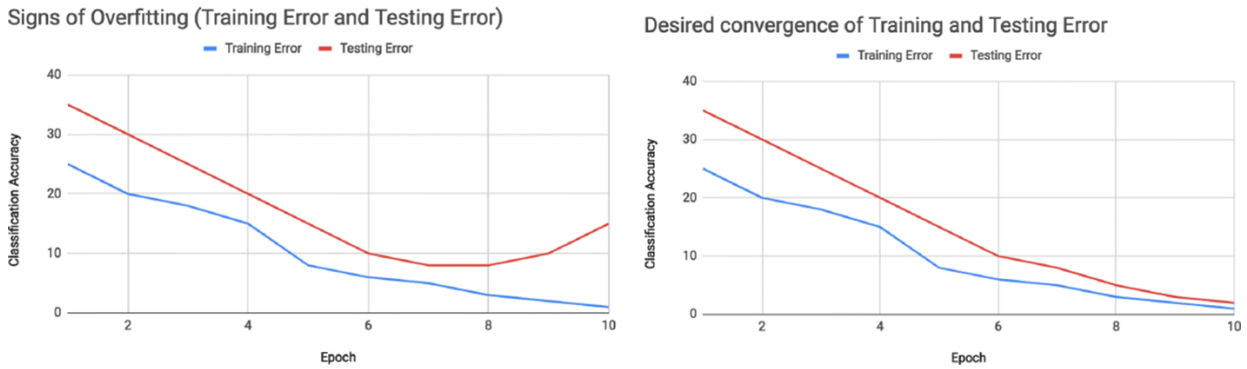


Figure 2 - Graphs showing Signs of Overfitting vs Desired Convergence [75].

The data augmentation techniques used in this project were image flipping, gamma correction, noise injection, principal component analysis (PCA) colour augmentation, rotation, scaling, random brightness, width shifting, height shifting, zooming and reflection.

1.1.10 Plant Disease Classification Subsystem

The plant disease classification subsystem was designed to perform a more particular analysis of plant health in areas of interest highlighted by the NDVI subsystem. The Subsystem main aims were to identify if individual plants were healthy or diseased; to classify the diseases infecting the diseased plants; to classify the plant species distinguishing them from images without any plants in them (background images); to perform real-time disease classification on video frames streamed from a camera located on the plant health monitoring drone.

1.1.11 Disease Classification Model Evaluation

In multivariate classification tasks micro average, macro average and per instance average F1 scores variants are useful in comparing the quality of predictions across systems. As multivariate classification is an extension of binary classification, the metrics of precision and recall are also commonly used [78]. The following metrics were used to evaluate model performance in this project.

Individual Precision and Recall calculated for each class.

$$Precision_m = \frac{TP_m}{TP_m + FP_m} \quad (35)$$

$$Recall_m = \frac{TP_m}{TP_m + FN_m} \quad (36)$$

Where m is the class, TP_M is the number of true positive classifications, FP_M is the number of false positive classifications, FN_M is the number of false negative classifications and m is the generic class.

$$MacroAveragePrecision = \frac{\sum_{m=1}^{M} Precision_m}{M} \quad (37)$$

$$MacroAverageRecall = \frac{\sum_{m=1}^{M} Recall_m}{M} \quad (38)$$

Where M is the total number of classes; Macro-Precision and Macro-Recall are the arithmetic means of the individual precision and recall values calculated for each class.

$$Macro\ F1\ Score = 2 * (\frac{MacroAveragePrecision * MacroAverageRecall}{MacroAveragePrecision\ + MacroAverageRecall}) \quad (39)$$

Macro F1-Score is the harmonic mean of Macro-Precision and Macro-Recall.

$$Accuracy = \ Micro\ Average\ F1\ Score = \frac{\sum_{m=1}^{M} TP_m}{Total\ Classifications} \quad (40)$$

Micro Average F1 Score is equivalent to model accuracy.

The metrics defined in this section were taken from the journal Metrics for multi-class classification: an overview [79].

2 Methodology

2.1 Plant Disease Classification

This section of the report details the methodology behind the plant disease classification subsystem. The Python code to accompany the work in this section can be found in the appendix. A flowchart summarising this subsystem and other subsystems is shown in Figure 48 in section 2.7.

2.1.1 Datasets

The original datasets used to train the CNN were PlantDoc [71], PlantVillage [72] and A Citrus Fruits and Leaves Dataset for Detection and Classification of Citrus Diseases through Machine Learning [73] as summarised in Table 2 of the introduction. A modified version of the PlantVillage dataset [81] was used in place of the original PlantVillage [72] that contained the following augmentations: image flipping, gamma correction, noise injection, PCA colour augmentation, rotation, and scaling. The modified dataset contained 61,486 images instead of the original 54303 images. The original versions of PlantDoc [71] and Citrus Fruits Diseases [73] datasets were used.

The modified PlantVillage [81], PlantDoc [71] and Citrus Fruits Diseases [73] datasets were then organised into a directory tree and crops that did not have both a diseased dataset and healthy dataset were removed. The crop datasets removed were blueberry, raspberry, soybean, and squash this removed approximately 10000 images from the dataset. The final dataset used to train and test the model is summarised in Table 4.

Table 2 - Dataset Summary

Classes	Number of Images	Classes	Number of Images
APPLE_BLACK_ROT	1000	**GRAPE_HEALTHY**	1069
APPLE_CEDAR_RUST	1089	**GRAPE_ISARIOPSIS_LEAF_SPOT**	1076
APPLE_HEALTHY	1736	**PEACH_BACTERIAL_SPOT**	2297

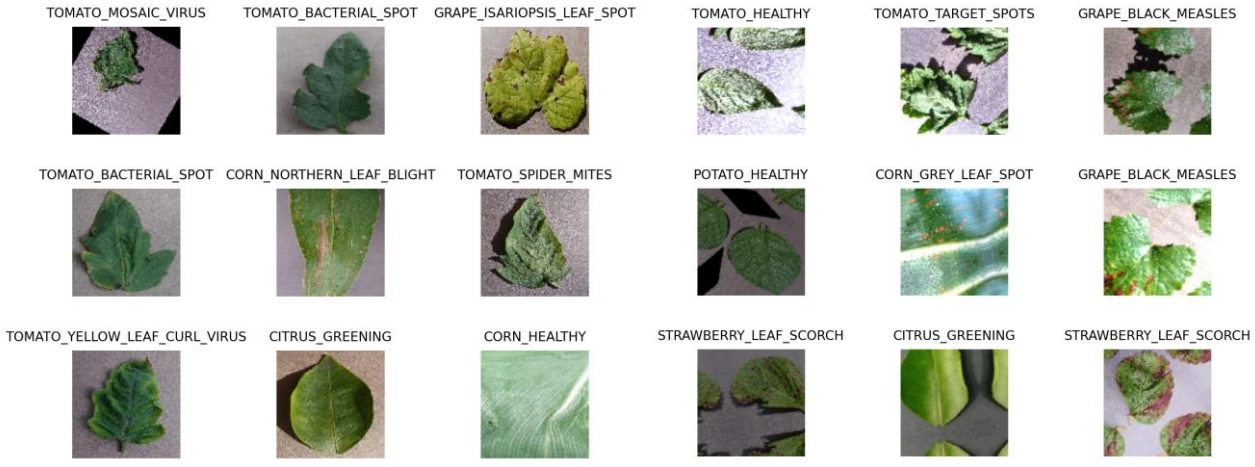
APPLE_SCAB	1087	**PEACH_HEALTHY**	1112
BACKGROUND	1143	**POTATO_EARLY_BLIGHT**	1117
BELLPEPPER_BACTERIAL_SPOT	1071	**POTATO_HEALTHY**	1000
BELLPEPPER_HEALTHY	1539	**POTATO_LATE_BLIGHT**	1008
CHERRY_HEALTHY	1057	**STRAWBERRY_HEALTHY**	1096
CHERRY_POWDERY_MILDEW	1052	**STRAWBERRY_LEAF_SCORCH**	1109
CITRUS_BLACK_SPOT	190	**TOMATO_BACTERIAL_SPOT**	2234
CITRUS_CANKER	241	**TOMATO_EARLY_BLIGHT**	1083
CITRUS_GREENING	5727	**TOMATO_HEALTHY**	1653
CITRUS_HEALTHY	80	**TOMATO_LATE_BLIGHT**	2117
CORN_COMMON_RUST	1308	**TOMATO_LEAF_MOLD**	1091
CORN_GREY_LEAF_SPOT	1068	**TOMATO_MOSAIC_VIRUS**	1054
CORN_HEALTHY	1162	**TOMATO_SEPTORIA_LEAF_SPOT**	1919
CORN_NORTHERN_LEAF_BLIGHT	1192	**TOMATO_SPIDER_MITES**	1676
GRAPE_BLACK_MEASLES	1383	**TOMATO_TARGET_SPOTS**	1404
GRAPE_BLACK_ROT	1244	**TOMATO_YELLOW_LEAF_CURL_VIRUS**	5432

The final dataset contained 54'916 images belonging to 38 classes, 10 species of crop, 27 diseases and 1 background class. The dataset was split into a train, validation, and test sub-datasets with 90% of the data used for training, 5% used for validation and 5% used for testing. The dataset splitting was done using the splitfolders Python module.

2.1.2 Image Augmentation

In addition to the augmentations added to the PlantVillage [72] dataset, augmentations were added to the dataset after a training, validation and testing data split had been performed. These new augmentations were only added to the training dataset and not to the validation and testing datasets. Example from the training and validation datasets are shown in Figure 42.

The additional augmentations added to the training dataset were as follows: rotation, scaling, random brightness, width shifting, height shifting, zooming, reflection and image flipping.



A – Validation Dataset without Additional Augmentation

B – Training Dataset with Additional Augmentation

Figure 3 – Comparison of Training and Validation Datasets Shown in 3x3 Grids

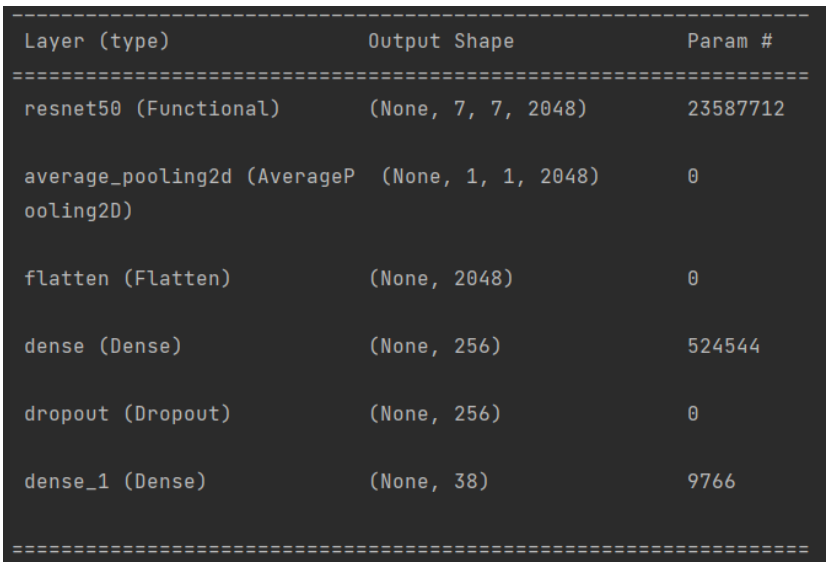
The image augmentation was done using a data generator which generates batches of images in a set batch size (in this case 32), performs the augmentations and returns the augmented images

batch to the model for training. The data generator also resized the images to 224x224 pixels, which was the selected input size to the model (the input size selected was used to reduce training times by the reducing the number of calculations required during training).

Data generators were also used for the validation and testing datasets, these data generators did not augment the images but did resize them to 224x224pixels, this allowed the images to fit the input dimensions of the model.

2.1.3 Model

This model implemented for this projected is summarised in Figure 43. The model builds upon the pretrained ResNet-50 model.



Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23587712
average_pooling2d (AveragePooling2D)	(None, 1, 1, 2048)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 256)	524544
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 38)	9766

Figure 4 - Model Summary

The pretrained ResNet-50 model used parameters optimised from training on the ImageNet dataset. The outputs from the ResNet-50 model were connected to an average pooling layer which calculates an average value for patches of the feature map outputted from ResNet-50 model and uses it to create a down sampled version of the map.

The flatten layer which formats the data from the average pooling layer into a one-dimensional array, this is connected to a fully-connected (dense) layer with 256 nodes and then to dropout layer which drops random connections between nodes.

The dropout layer is connected to a fully-connected layer with the number of nodes equal to the number of classes (the 38 possible predictions). As this is a multivariate classification problem, the softmax activation function is used (described in section 1.10.7) to create a probability distribution for the 38 classes. This returns a one-dimensional array with 38 outputs, the largest output corresponding to the predicted class.

In summary the network contained 24'122'022 parameters, with 15'540'902 trainable parameters and 8'581'120 non-trainable parameters. The difference between the total number of parameters and the number of trainable parameters is caused by making the parameters of the early layers of Resnet-50 untrainable, keeping their original pretrained weights (therefore the first layers of ResNet-50 act as feature extractor for the model and the remaining layers fine tune the output to fit the plant disease dataset).

The code used to implement the model is shown in Figure 44.

```
96  def model_maker():
97      # define input new input layer, load resnet50 model
98      input_t = K.Input(shape=(224, 224, 3))
99      resModel = K.applications.ResNet50(include_top=False, weights="imagenet", input_tensor=input_t)
100
101     # freeze layers in resnet except batch normal layers (not all layers)
102     for i, layer in enumerate(resModel.layers):
103         if "_bn" in layer.name:
104             pass
105         else:
106             layer.trainable = False
107
108     # define optimiser
109     opt = K.optimizers.Adam(learning_rate=LEARNING_RATE, decay=LEARNING_RATE / EPOCHS)
110
111     # define model
112     model = K.models.Sequential()
113     model.add(resModel)
114     model.add(K.layers.AveragePooling2D(pool_size=(7, 7)))
115     model.add(K.layers.Flatten())
116     model.add(K.layers.Dense(256, activation='relu'))
117     model.add(K.layers.Dropout(0.5))
118     model.add(K.layers.Dense(NUM_OUTPUT_CLASSES, activation='softmax'))
119     model.compile(optimizer=opt, loss=K.losses.SparseCategoricalCrossentropy(), metrics="sparse_categorical_accuracy")
120     model.summary()
121     return model
```

Figure 5 - Code used to Implement the Model

The Python code used Tensorflow to load a pretrained ResNet-50 model trained on the ImageNet dataset shown in lines 98-99. In lines 102-106, the layers of the ResNet-50 model were set to be untrainable, with the exception of the batch normalization layers which provide network regularisation and reduce generalisation error. The optimiser is defined in line 109, alongside the learning rate and rate of learning rate decay (these define how quickly the model changes parameters during training). The optimiser selected was the Adam optimiser which is common for computer vision applications. The learning rates selected for the optimiser was 0.01 and the learning rate decay to the initial learning rate divided by the number of epochs (10 epochs).

The layers of the model shown in Figure 43 are defined in lines 112-119. For the average pool layer, the pool size was set to (7, 7) to match the output shape from the final layer of ResNet-50 (line 114). The activation function for the dense layer was set to relu (rectified linear unit), which is a non-linear activation function which stops all connections between layers activating (line 116). The dropout rate selected for the dropout layer was 0.5, meaning half the time the inputs from the layer were set to zero, this is used to prevent overfitting (line 177). The output layer with the softmax activation function is defined in line 118 and the model is compiled with the loss function set as categorical cross entropy in line 119.

2.1.4 Model Training

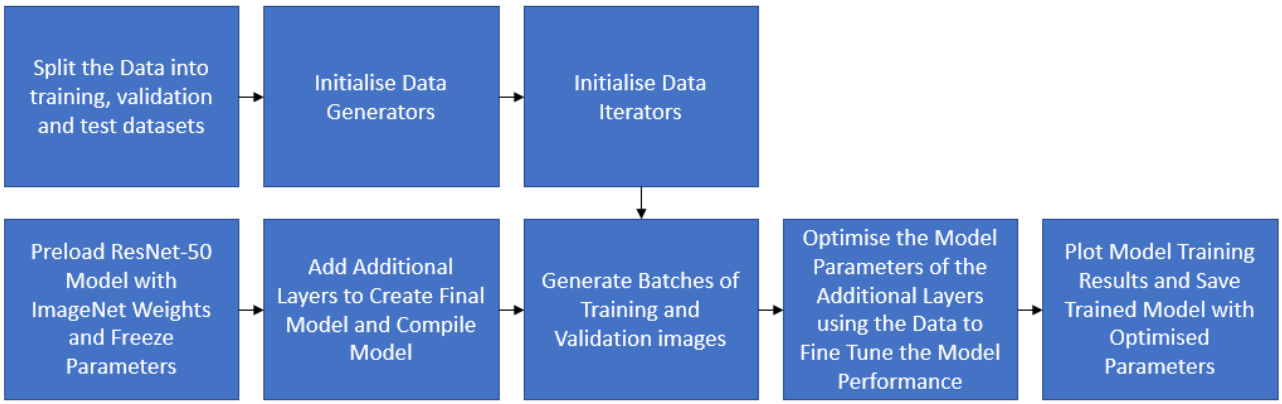


Figure 6 - Flowchart Summarising the Model Training

The model training process is summarised in Figure 45 and explained in this section. Firstly, the dataset was split into a train, validation, and test sub-datasets with 90% of the data used for training, 5% used for validation and 5% used for testing. This provided 49'408 images for training, 2727 images for validation and 2781 images for testing. This split ratio was selected to maximise training and validation data whilst still providing a large enough test set to evaluate model performance. Next the data generators are data iterators are initialised, the code that generates the data batches and augments the images is shown in Figure 46.

```
124  def train_model():
125      # load model
126      model = model_maker()
127      # define train datagen
128      training_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
129          rotation_range=360,
130          fill_mode='reflect',
131          horizontal_flip=True,
132          vertical_flip=True,
133          brightness_range=[0.3, 1.5],
134          width_shift_range=[-20, 20],
135          height_shift_range=[-20, 20],
136          zoom_range=[0.8, 1.5])
137
138      # define val datagen
139      validating_datagen = tf.keras.preprocessing.image.ImageDataGenerator()
140
141      # load train and val datasets
142      train_it = training_datagen.flow_from_directory(DATA_SPLIT_LOCATION + "/train", class_mode='sparse',
143                                                      batch_size=BATCH_SIZE,
144                                                      target_size=IMG_SHAPE)
145      val_it = validating_datagen.flow_from_directory(DATA_SPLIT_LOCATION + "/val", class_mode='sparse',
146                                                      batch_size=BATCH_SIZE,
147                                                      target_size=IMG_SHAPE)
```

Figure 7 - Train and Validation Data Generators

The data generators are initialised in lines 128 (training data generator) and 139 (validation data generator). The augmentations for the training dataset are defined in lines 129-136. The iterators that call the data generators are initialised in lines 142 (training iterator) and 145 (validation iterator). The batch size for the iterators was set as 32, this was limited due to processing power

available for training. The iterators also define the image size (224x224) that all training and validation images are resized to.

The ResNet-50 Model is loaded with the pretrained ImageNet parameters, the parameters of the early layers are frozen so that they remain constant during model training. Additional layers are then added to the model to provide trainable parameters. The training constants are defined when initialising the iterators and compiling the model and are summarised in Table 5.

Table 3 - Summary of Training Parameters

Parameters	Value	Selected/Derived
Initial Learning Rate	0.01	Selected
Number of Epochs	10	Selected
Batch Size	32	Selected
Decay Constant	0.001	Derived
Total Number of Training Images	49408	Selected
Total Number of Validation Images	2727	Selected
Iterations per Epoch	1544	Derived
Validation Iterations per Epoch	85	Derived

The equations used to calculate derived values in the table are defined below.

The learning rate was set to decay with a decay constant of:

$$decay\ constant = \frac{initial\ learning\ rate}{total\ number\ of\ epochs} \tag{41}$$

The learning rate is updated after every epoch step (iteration) using the formula:

$$learning\ rate = initial\ learning\ rate\ * \frac{1}{1+decay\ constant * iterations} \tag{42}$$

The Iterations per epoch was calculated based on the formula:

$$\text{i}terations\ per\ epoch = \frac{total\ number\ of\ training\ images}{batch\ size} \tag{43}$$

The validation Iterations per epoch was calculated based on the formula:

$$validation\ \text{i}terations\ per\ epoch = \frac{total\ number\ of\ validation\ images}{batch\ size} \tag{44}$$

Batches of training and validation images were then generated and used to iteratively optimise the trainable parameters of the network, as there were 10 epochs, the training image dataset was iterated over 10 times in batches of 32 images as defined by the batch size. The model training results were then plotted as graphs and the trained model saved.

2.1.5 Model Evaluation

To evaluate the model performance, the trained model was shown 2781 images from the test dataset, the test images were first resized to 224x224 to fit the model input dimensions. The model predictions and actual labels were compared to calculate the following performance metrics:

precision, recall and F1 score for each class; macro average precision and recall; macro average F1 score and overall model accuracy. These results were used to evaluate the performance of the model for each class and the overall model performance. A confusion matrix summarising the predictions of the model was made but this did not summarise the performance metrics due to large number of classes limiting the resolution.

2.1.6 Plant Disease Classification Application

The application for this subsystem was coded in Python. A video stream from the camera on the drone is transmitted to the laptop via a video transmission link. Images from the video stream are inputted into the model and the predictions are shown through the application alongside the video stream from the drone.

The graphical user interface (GUI) for the application was coded using the PySimpleGUI module and the video stream was processed using the opencv module. The GUI is shown in Figure 47.

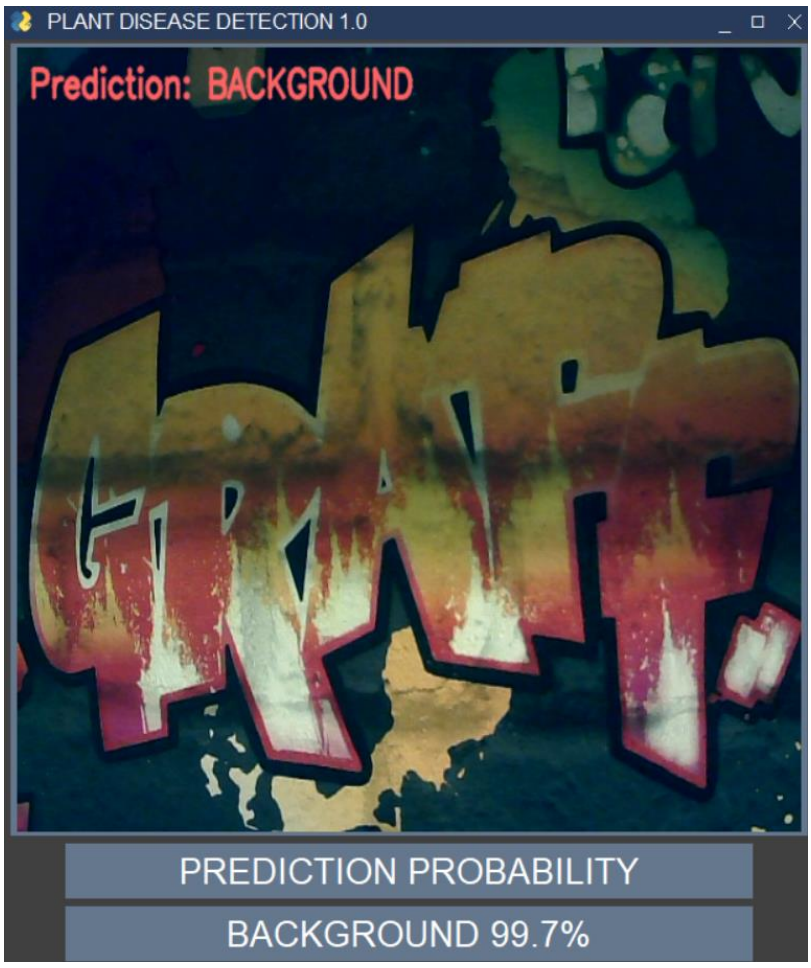


Figure 8 - Application GUI

The GUI was designed to show the video stream with prediction label positioned on top left of the image frame, underneath the video stream, the prediction was shown again with the corresponding probability calculated by the model. The GUI code can be found in the appendix.

2.1.7 Wiring Diagram – Morgan

The wiring diagram of the final system is shown in Figure 49.

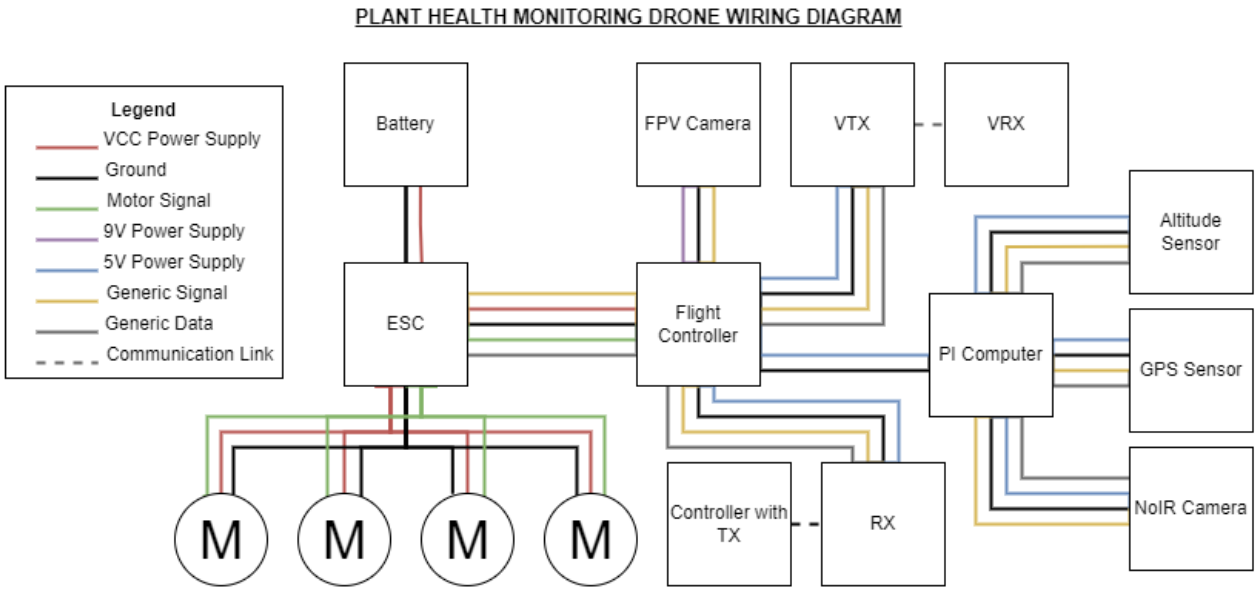
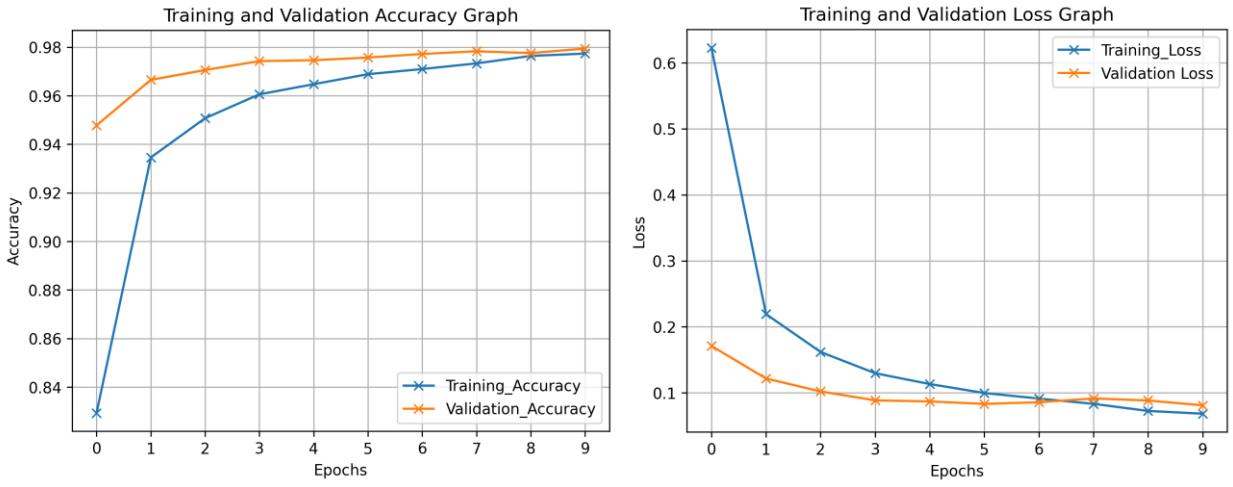


Figure 9 – Wiring Diagram of Final System

3 Results

3.1.1 Model Training



A – Accuracy Graph **B – Loss Graph**

Figure 10 - Training and Validation Accuracy and Loss of Model

The graph in Figure 82 (A) shows the variation of training and validation accuracies across the training epochs. The graph shows the training and validation accuracies converging

The graph in Figure 82 (B) shows the variation of training and validation losses across the training epochs. The graph shows the training and validation losses converging.

The final training loss value of the model was 0.0684, the final training accuracy value was 0.9774, the final validation loss value was 0.0811 and the final validation accuracy was 0.9794.

3.1.2 Model Testing

A summary of the performance metrics calculated for the model is shown in Table 11.

Table 4 – Model Performance Metrics

Performance Metric	Value	Standard Deviation
Macro Average Precision	0.97	± 0.05
Macro Average Recall	0.96	± 0.07
Macro Average F1 Score	0.96	± 0.06
Accuracy / Micro Average F1 Score	0.98	NaN

The model accuracy was the highest performance metric followed by macro average precision. The lowest performance metrics were macro average recall and F1 Score. The model performance metrics broken down by class is shown in Table 12.

Table 5 - Model Performance Metrics by Class

Class	Precision	Recall	F1-Score	Test Images
APPLE_BLACK_ROT	1.00	1.00	1.00	50
APPLE_CEDAR_RUST	1.00	0.98	0.99	55
APPLE_HEALTHY	0.99	0.99	0.99	88
APPLE_SCAB	0.98	0.98	0.98	55
BACKGROUND	0.98	1.00	0.99	58
BELLPEPPER_BACTERIAL_SPOT	0.95	0.96	0.95	55
BELLPEPPER_HEALTHY	0.97	0.97	0.97	78
CHERRY_HEALTHY	0.98	0.96	0.97	54
CHERRY_POWDERY_MILDEW	1.00	1.00	1.00	54
CITRUS_BLACK_SPOT	0.86	0.60	0.71	10
CITRUS_CANKER	0.81	1.00	0.90	13
CITRUS_GREENING	0.99	0.99	0.99	287
CITRUS_HEALTHY	0.75	0.75	0.75	4
CORN_COMMON_RUST	1.00	0.98	0.99	66
CORN_GREY_LEAF_SPOT	0.91	0.94	0.92	54
CORN_HEALTHY	1.00	1.00	1.00	59
CORN_NORTHERN_LEAF_BLIGHT	0.92	0.92	0.92	61
GRAPE_BLACK_MEASLES	1.00	1.00	1.00	70
GRAPE_BLACK_ROT	0.98	1.00	0.99	63
GRAPE_HEALTHY	1.00	1.00	1.00	54
GRAPE_ISARIOPSIS_LEAF_SPOT	1.00	1.00	1.00	55
PEACH_BACTERIAL_SPOT	1.00	1.00	1.00	116
PEACH_HEALTHY	0.95	0.96	0.95	57
POTATO_EARLY_BLIGHT	0.98	0.93	0.95	57
POTATO_HEALTHY	1.00	1.00	1.00	50
POTATO_LATE_BLIGHT	1.00	1.00	1.00	51
STRAWBERRY_HEALTHY	0.98	0.98	0.98	56
STRAWBERRY_LEAF_SCORCH	1.00	1.00	1.00	56
TOMATO_BACTERIAL_SPOT	0.96	0.95	0.95	113
TOMATO_EARLY_BLIGHT	0.96	0.95	0.95	55
TOMATO_HEALTHY	0.94	0.96	0.95	84
TOMATO_LATE_BLIGHT	0.95	0.98	0.96	107

TOMATO_LEAF_MOLD	0.93	0.96	0.94	56
TOMATO_MOSAIC_VIRUS	1.00	0.94	0.97	54
TOMATO_SEPTORIA_LEAF_SPOT	0.97	0.97	0.97	97
TOMATO_SPIDER_MITES	1.00	1.00	1.00	85
TOMATO_TARGET_SPOTS	1.00	0.97	0.98	71
TOMATO_YELLOW_LEAF_CURL_VIRUS	0.99	0.99	0.99	273

Table 12 is colour coded with F1 scores below the macro average highlighted in red, equal to the macro averaged highlighted in orange and above the macro average highlighted in green.

3.1.3 Real Time Classification

The application was able to receive image data from the drone and perform real time classification on the received images and output a prediction with an accompanying confidence of prediction value. The accuracy of the real time classification was not assessed as the drone did not fly.

4 Discussion

4.1.1 Model Training

The training graphs produced and shown in Figure 82 converge over the training epochs with both training and validation accuracy and loss converging. When compared to the graphs in Figure 12 (section 1.10), the training graphs better resemble the graph which does not show signs of overfitting. This demonstrates that the training parameters with the additional dataset augmentations as well as the dropout layer included in the network prevent the overfitting of the model to the data.

4.1.2 Classification of Healthy and Diseased Plants

The macro averaged F1 score of 96% demonstrated that this model can perform classification of healthy and diseased plants, the low standard deviation of 0.06 shows that F1 score is uniform for most classes. The macro averaged precision of 97% showed that the model is good at performing valid classifications of healthy and diseased plants (when it predicts a class it is right 97% of the time), the low standard deviation of 0.05 shows that the model makes valid classifications across most classes. The macro averaged recall of 96% shows that the model is good at identifying true classifications of healthy and diseased plants, the low standard deviation of 0.07 shows that the recall is mostly uniform across classes however less uniformly distributed than the precision.

The micro averaged F1 score of 98% for the model being higher than the macro averaged F1 score of 96% shows that the classes with a smaller amount of test images (and therefore total images) are related to worse model performance than classes with larger amounts of data. This is expected and is demonstrated by classes such as CITRUS_HEALTHY which only had 4 test images and a F1 score of 75% compared to CITRUS_GREENING which had 287 test images and a F1 score of 99%. To improve the macro averaged F1 score more data should be gathered for classes with smaller amounts of data.

A comparison of the performance of related plant disease classification models is shown in Figure 83.

Reference	CNN Network	Dataset	Accuracy	# Classes
Amara et al. (2017)	LeNet architecture	PlantVillage (extended)	92–99%	3
Brahimi et al. (2017)	AlexNet, GoogLeNet	PlantVillage	99%	9
Cruz et al. (2017)	Modified LeNet	Olive tree images (own)	99%	3
DeChant et al. (2017)	Pipeline	Corn images (own)	97%	2
Ferentinos (2018)	Several	PlantVillage (extended)	99%	58[a]
Fuentes et al. (2017)	Several	Tomato images (own)	83%	10
Liu et al. (2018)	AlexNet	Apple images (own)	98%	4
Lu et al. (2017)	AlexNet inspired	Rice images (own)	95%	10
Mohanty et al., 2016	AlexNet, GoogLeNet	PlantVillage	99%	38[b]
Oppenheim and Shani (2017)	VGG	Potato images (own)	96%	5

a
Classes are distributed among 25 plant species.

b
Classes are distributed among 14 plant species.

Figure 11 - Comparison of Studies using Deep Learning for Plant Disease Classification [87]

The 98% model accuracy achieved using transfer learning with the ResNet-50 architecture in this report falls in the range of accuracies achieved using different CNN architectures, this was expected due to the similarities in approach with transfer learning being the main technique used in papers listed in Figure 83.

To improve the performance of the model more high-quality data could be used specifically for classes with low F1 scores. Tuning the training parameters of the model could also be experimented with to evaluate their effect on the macro averaged F1 score of the model as only one model was trained.

4.1.3 Differentiate Plants from a Background Class

The 98% precision value for the BACKGROUND class shows that the model can differentiate plant classes from a background class. The 98% precision value means that only 2% of background predictions were misclassified plants. The 100% recall value shows that all background test images were correctly classified as the BACKGROUND class. This suggests that more data is required for the misclassified plant classes and not for the BACKGROUND class.

4.1.4 Real Time Classification of Plants

The plant disease classification application developed for the subsystem received image data from the drone's front facing camera and outputted a prediction to the GUI, this demonstrates that the subsystem achieved the aim of performing real time classification. Since no in-flight testing of the subsystem was done, there was no method of evaluating the performance of the system and the model when performing real time classification in the drones operating state.

Performing more testing on this part of the subsystem would provide a better evaluation of the overall subsystem performance. This should be the main aim if any further work on the subsystem was to be attempted.

5 Conclusion

5.1 Plant Disease Classification Subsystem

The plant disease subsystem met all subsystem aims. It performed the classification of healthy and diseased plants as evidenced by the macro averaged F1 score of 96% and the micro averaged F1 score (accuracy) of 98%. The subsystem distinguished between a background class and plant classes demonstrated by the 100% recall value and 98% precision value achieved by the BACKGROUND class. The subsystem performed real time classification of plants as shown by the plant disease classification application using the trained model to perform classification images received from the drone's front facing camera. The performance of the subsystem could be improved by acquiring more high-quality data. More experimentation with the model training parameters is required to evaluate the effect of the performance of the model. Finally, the trained model did not show signs of overfitting as shown by the convergence of the accuracy and loss graphs.