

University of Southern Queensland
Faculty of Health, Engineering & Sciences

**A Deep Learning Solution for the Detection of Health and
Productivity Metrics in Sandalwood Forest Plantations
Using Drone Imaging**

A dissertation submitted by

I. Humber

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Honours) (Computer Systems)

Submitted: October, 2023

Abstract

The production of Sandalwood Oil is a highly lucrative industry, and one in which Australia is currently the global leader. The processes currently applied to monitor tree health and predict the volume of marketable product at harvest are costly in both time and resources. Deep learning has shown promise for the automatic monitoring of health and volume in silviculture plantations using overhead imagery. However, this has never been done on the individual tree level for the 5 class health score or bole height and diameter at breast height applied by the industry. Nor has it been attempted within Sandalwood plantations. Thus, a two stage, deep learning based system was proposed, informed by the findings of relevant literature. The goal of the system was the of full automation of both general health monitoring and marketable volume prediction on the individual tree level, thus adding value to the imagery already recorded by the industry as part of yearly inventory.

The resulting system was able to detect trees within the plantations at an accuracy comparable to leading algorithms. Moreover, the system was able to classify the health scores used in industry-standard volume estimation calculations with an Average Precision of 0.97 and sample-weighted F_1 score of 0.92, exceeding the performance of other tree health classifiers proposed in the literature.

ENG4111/2 <i>Research Project</i>

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled “Research Project” is to contribute to the overall education within the student’s chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Dean

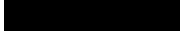
Faculty of Health, Engineering & Sciences

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

I. HUMBER



Acknowledgments

I would like to thank my supervisor, Dr. Tobias Low, for his insightful feedback and input during the formulation of this dissertation and his generous allocation of his valuable time. Moreover, I would like to express my abundant gratitude towards Dr. Precila Gonzales, a distinguished member of the Quintis research team, who went above and beyond expectations to ensure the quality of the dataset used in this study and to assist and advise me on research and forestry practices. Finally, I would like to thank my wife and family for their unwavering support and patience during the completion of this study.

I. HUMBER

Contents

Abstract	i
Thesis Certification	v
Acknowledgments	vii
List of Figures	xv
List of Tables	xvii
Chapter 1 Introduction and Background	1
1.1 Sandalwood Plantations	2
1.2 Tree Health Monitoring in Forest Plantations	2
1.3 Inventory in Forest Plantations	3
1.4 Machine Learning	4
1.4.1 Deep Learning	4
1.4.2 Convolutional Neural Networks (CNN)	6
1.4.3 Image Classification	7
1.4.4 Object Detection	7

1.4.5	Image Instance Segmentation	8
1.4.6	Image Annotation	8
1.4.7	Image Regression	9
1.4.8	Hyperparameters	9
1.4.9	Model Training	10
1.4.10	Transfer Learning	11
1.4.11	Over-fitting	12
1.4.12	Under-fitting	14
1.4.13	Model Evaluation	14
1.5	Research Questions	18
Chapter 2	Literature Review	19
2.1	Sandalwood Tree Volume Prediction Techniques	19
2.2	Innovative General Techniques For Tree Health and Volume Assessment, Detection and Prediction	20
2.2.1	Health Monitoring	20
2.2.2	Yield Prediction	21
2.3	Leading Machine Learning Techniques For Image Analysis	21
2.3.1	Instance Segmentation	21
2.3.2	Image Classification	22
2.3.3	Image Regression	22
2.4	Machine Learning For Individual Tree Health and Volume Prediction . . .	22

2.4.1	Feature Extraction and Shallow Learning	23
2.4.2	Deep Learning	24
2.4.3	Common Design Amongst Health/Volume Prediction Models . . .	25
2.5	Related Techniques in Agriculture	25
2.6	Application of These Techniques in Sandalwood Plantations	26
Chapter 3 Research Methodology		27
3.1	Design Objectives	28
3.2	System Architecture	28
3.3	Data Collection	29
3.4	Data Preprocessing	30
3.5	Data Labelling	32
3.6	Summary of Final Datasets	32
3.7	Training	33
3.7.1	Environment	33
3.7.2	Stage 1: Automatic Individual Tree Detection and Segmentation .	33
3.7.3	Stage 2: Automatic Tree Health and Parameter Prediction	34
3.8	Model Evaluation	36
Chapter 4 System Design		37
4.1	Pilot Studies	37
4.1.1	Matterport Mask RCNN Individual Tree Detection Model	37

4.2	Final System Overview	38
4.2.1	Stage 1: Individual Tree Detection and Segmentation Model	38
4.2.2	Stage 2: Tree Parameter Prediction Model	40
Chapter 5	Results	43
5.1	Stage 1: Individual Tree Detection and Segmentation Model	43
5.2	Stage 2: Individual Tree Parameter Prediction	47
5.2.1	Tree Health Classifier Model	47
5.2.2	Regressor Models	50
5.3	Sources of Error	51
Chapter 6	Discussion	55
6.1	Analysis of Results	55
6.1.1	Stage 1: Individual Tree Species Detector	55
6.1.2	Stage 2a: Tree Health Classifier	56
6.1.3	Stage 2b: Tree Parameter Regressors	58
6.2	Limitations of the Methods	59
6.3	Limitations of the Study	59
6.4	Significance of the Study	60
6.5	Further Work	60
Chapter 7	Conclusion	63
	References	65

Appendix A Project Specification	71
Appendix B Risk Assessment	75
Appendix C Ethical Clearance	77
Appendix D Data	79
D.1 Introduction to this Appendix	80
D.2 Raw Data	80
D.3 Drone Imaging	85
D.3.1 Kingston Rest 1 Block 47	85
D.3.2 Kingston Rest 1 Block 47 - Digital Surface Model	86
D.3.3 Kingston Rest 3 Block 1	87
D.3.4 Kingston Rest 3 Block 1 - Digital Surface Model	88
Appendix E Source Code	89
E.1 Tree Detection Module	90
E.2 Health Classifier Model	93
E.3 Tree Parameter Regressor Model	98

List of Figures

1.1	Top Left: Sigmoid Function. Top Right: ReLU Function. Bottom: Leaky ReLU	5
1.2	The Convolution Operation (Source: Bechberger 2020)	6
1.3	Accuracy (Left) and Loss (Right) Curves of an Overfitting Network (Source: Rahaman et al. 2020)	12
2.1	Relationship Between Tree Girth and Heartwood Volume in Sandalwood (Source: Brand et al. 2012)	19
3.1	Final Proposed System	28
3.2	Flowchart of the Stage 1 Data Preprocessing Workflow	30
3.3	Orthomosaic of Kingston Rest Plantation 1, Block 47	30
3.4	Orthomosaic of Kingston Rest Plantation 1, Block 47	31
3.5	Orthomosaic of Kingston Rest Plantation 1, Block 47	31
3.6	Examples of Training Data Augmentations	33
3.7	Stage 1 Training Workflow	34
3.8	Stage 2 Training Workflow	35
4.1	Epoch Validation Loss Curves of Model Training Attempts	38

4.2	Flowchart of the Stage 1 Model Workflow	39
4.3	Flowchart of the Stage 2 Workflow	40
5.1	Evaluation Precision Recall Curves for Stage 1.	45
5.2	Confusion Matrix for The Final Model on Test Dataset	46
5.3	Examples of Stage 1 Detections	47
5.4	Confusion Matrices	49
5.5	Precision/Recall Curves	49
5.6	Receiver Operating Characteristic Curves	50
5.7	Regressor Predictions	51

List of Tables

2.1	Windrim et al. Summary of Best Results	23
3.1	Breakdown of Final Datasets	32
3.2	Metrics Used For Model Evaluation	36
5.1	Bounding Box and Segmentation Test Results	43
5.2	Results for test, validation and training datasets	48
5.3	Results for test, validation and training datasets	50

Chapter 1

Introduction and Background

The Global Sandalwood Market Sales Revenue in 2022 was \$820.29 Million USD and is predicted to reach \$1.14 Billion USD in 2027 (Mali 2023). In 2021, Australia occupied approximately 69% of the global market share for sandalwood products (*Global And United States Sandalwood Market Insights, Forecast To 2027* 2021).

Quintis, the world's largest producer of sandalwood and the industry contact for this study, takes inventory of all their plantations (totalling more than 11,200 hectares) every year, measuring tree girth, bole height and health status (a score ranging from 1, denoting a dead tree, to 5, a tree of near perfect general health) and imaging every plot using multi-rotor, rotary-wing drones. The measurements are taken manually by contractors in random samples within each plot, costing the company substantial time and resources. Furthermore, visual health assessment is highly prone to human error, particularly when multiple assessors are involved, as found by Redfern & Boswell (2004).

After reviewing 99 research papers on the topic of the use of UAV imaging to monitor forest health, Ecke et al. (2022) concluded that UAVs are a cost-efficient and robust tool for forest health and volume monitoring and the technology is rapidly advancing. Given that complete UAV imaging is already undertaken yearly by Quintis, there lies a clear opportunity to reduce the time and resources exhausted by the inventory process, without requiring the acquisition of any new hardware or manpower, by leveraging the imagery to enable the automatic detection of the measured parameters.

There is no current research into such a system, though there are studies which have

a degree of applicability to its development. These studies point towards the potential for deep learning to be a well suited candidate for the backbone of the aforementioned system. Furthermore, sandalwood, being semi-parasitic, poses its own unique challenges.

1.1 Sandalwood Plantations

Sandalwood plantations must be non-homogeneous due to the hemiparasitic nature of Sandalwood trees. While Sandalwood trees perform photosynthesis, they also extract water, nitrogen, and other nutrients from the roots of nearby trees. As a result, in commercial Sandalwood plantations trees of one or more other species must be evenly distributed throughout the plantation at a ratio of at least 1:1 (host to Sandalwood), with 3:1 being ideal for new plantations (*FPC: Sandalwood Establishment Guide* 2020, Anon & Chittapur 2021). The plantations used in this study have approximately a 1:1 ratio, although this ratio varies due to tree mortality over time. The plantations are harvested after a minimum of 15 years (though value continues to increase after this point).

1.2 Tree Health Monitoring in Forest Plantations

Constant health monitoring is a key strategy for maintaining productive and healthy forest plantations. It provides critical feedback on the performance of the strategies being implemented by the plantation management personnel. For example, such data allows for the early detection of disease, parasites and any external or environmental threats to the plantation as well as informing on the efficacy of the irrigation strategy being employed.

While there are currently no health detection/monitoring techniques specifically designed for Sandalwood plantations, these plantations typically use traditional health monitoring techniques generally employed by the greater field of forestry and silviculture. For the coarse monitoring of general tree health, particularly for the early detection of diseases, manual aerial sketch mapping (also known as aerial surveying) has historically been the primary method (Muchoney & Haack 1994). However, with the commercialisation of affordable satellite imagery and, more recently, drone imagery, digital images have largely replaced this approach (Dash et al. 2017). Forest plantations also commonly employ drive and/or walk-through surveying to detect general health and disease from

the ground, and then more targeted manual measurement should a health anomaly be detected (Smith et al. 2008). The detection and classification of the general health and disease for individual trees in all these methods is done visually, with the application of professional knowledge and training (Lawson et al. 2008).

Since their introduction in the early 1970s, Vegetation Indexes (VIs) have also been used as a tool to visualise the general health and/or productivity of vegetated regions. These VIs generally use mathematical transformations of optical reflectances of various bandwidths to produce a visual representation of certain health indicators (Zeng et al. 2022). The normalised difference vegetation index (NDVI) and the normalised difference water index (NDWI) are popular examples of these VIs and are both used by the management personnel of the sites in this study.

1.3 Inventory in Forest Plantations

Forest plantations, generally spanning thousands of hectares and containing millions of trees, often require inventory assessment to estimate the total volume of marketable wood across the entire plantation. This assessment is typically conducted using statistical sampling, remote sensing (e.g., overhead imagery), or a combination of both (Köhl 2004, Krug & dos Santos 2004). These methods usually provide an approximate estimation of the actual total marketable product. Current tree health is often taken also into consideration during inventory assessment to aid in the approximation of the total amount of marketable product to be expected at harvest. However, the equations used for estimation are prone to inaccuracies stemming from their reliance on manually measured, random samples.

The company managing the plantations used in this study takes drone imaging of all the plantations as a visual reference and uses a set of equations to estimate the total amount of recoverable heartwood (where the Sandalwood Oil is then harvested from) using the diameter of the tree, the height of the first fork in the main stem (called the bole height) and the health status of the trees at 15 years of age. This data is collected using statistical sampling across the plantation, directly measuring random samples of the population to infer the parameters of the entire plantation.

1.4 Machine Learning

The field of machine learning (ML) originated with a paper by McCulloch & Pitts (1943) titled “A logical calculus of the ideas immanent in nervous activity”, the first paper to propose a mathematical model of the interaction of neurons. This paper, closely followed by the work of D.O. Hebb “The Organization of Behavior: A Neuropsychological Theory” (Hebb 1949) arguing the direct connection between neural structure and behaviour, stimulated an explosion of research into the digital replication of neural learning and cognition. Notable research that came shortly after Hebb’s claims include the birth of the Multi-Layer Perceptron (Ivakhnenko 1967) and the Nearest Neighbour classification algorithm (Cover & Hart 1967), culminating in the first Convolutional Neural Network (Fukushima 1980). Other important developments in the following years include “The Strength of Weak Learnability” by Schapire (1990), first proposing the power of voting algorithms and “Random Decision Forests” by Ho (1995), which proposes the now well known algorithm, after which the paper was titled.

Machine Learning has since continued to grow at an exponential rate and the field is drawing substantial attention at the time of writing with the dramatic entry of one of machine learning’s most advanced and powerful offspring, GTP-4, and other competing large language models. The result of this growth is the proliferation of a myriad of Machine Learning algorithms of varying complexity, each best-suited to its own specific suite of tasks.

1.4.1 Deep Learning

The term “Deep Learning” refers to the use of Artificial Neural Networks (ANNs) with many hidden layers to learn a task by repetition, much like the human brain. They do this through the combination of layers of digital neurons — the technology’s fundamental unit of computation — that simulate some functions of biological neurons.

Each neuron can have multiple inputs, each with its own weight factor. Values enter the neuron through these inputs, being multiplied by the weight factors which are unique for each input. A bias factor is applied to the values before they are then added together, and the total is fed through the neuron’s activation function. The activation function can be any function, though its purpose is to output a value that is some function of the

input, while also introducing some nonlinearity to the system. Thus, the sigmoid function (an "S" shaped function with its centre at 0.5 and either end a limit at 0 and 1) is the traditional choice, however, modern models tend to use functions like the rectified linear activation function (ReLU) or some variation of it (such as Leaky ReLU), due to its faster compute speed and optimisation while achieving the same purpose. Examples of these functions can be seen below in Figure 1.1.

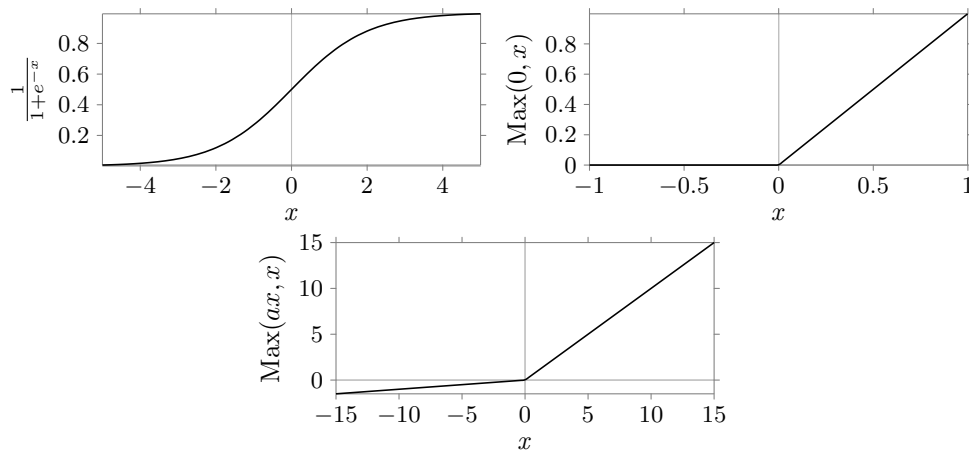


Figure 1.1: Top Left: Sigmoid Function. Top Right: ReLU Function. Bottom: Leaky ReLU

The result of the combination of interconnected layers of these neurons is a system that, with each layer, extracts more and more complex features from the input data and then, focusing on the most useful features, learns to make conclusions based on the input. These conclusions can be anything from the detection of the breed of dog in a given image to a detailed and well-informed response to a given text prompt that rivals the average human in proficiency.

Notable examples of cutting-edge deep learning projects include:

- GPT-4 — An advanced Generative Pretrained Transformer capable of matching or exceeding average human performance in many tasks such as the Bar Exam, poetry, writing code, language translation, data analysis and more.
- Midjourney — An image generation tool that combines a large language model and diffusion to generate artistic or photorealistic images from a text prompt.
- Tesla Full Self-Drive — A fully automated self-driving car system, using only input from RGB cameras, on its way to being capable of a reliability far in excess of the

average human.

1.4.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNNs) are a form of ANN that utilise the discrete convolution operation to automatically extract and combine features from image data. The discrete convolution operation applied in the convolution layers of CNNs results in a matrix (known as a “Feature Map”) containing the sums of the element-wise product between an $N \times N$ kernel (normally 3×3) and an equally sized sub-matrix from the input as the kernel slides across the entire input matrix — as seen below in Figure 1.2. The edges of the input matrix are padded with zeros to edge clipping.

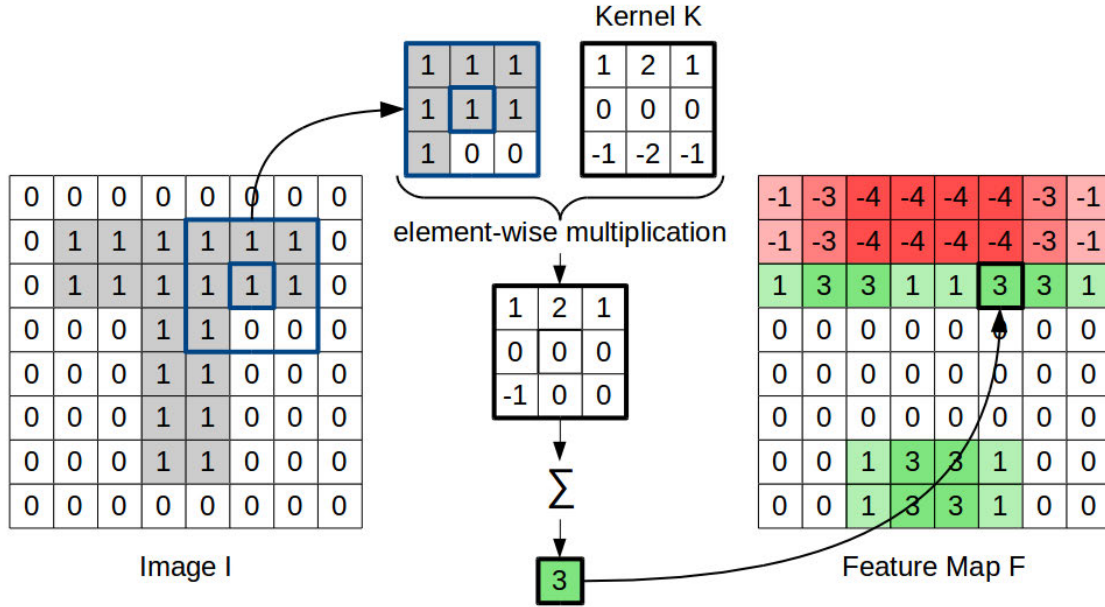


Figure 1.2: The Convolution Operation (Source: Bechberger 2020)

A CNN model learns different kernels, thus enabling it to learn multiple features. These convolution layers are followed by a ReLU activation layer and then, depending on the model architecture, either followed by another convolution and ReLU pair or by a *pooling layer*.

Pooling layers apply a similar operation as the convolution, however instead of a sliding pairwise product and sum, they simply take the maximum value (in the case of the max pooling) or the average value (in the case of the average pooling) within the kernel, which is normally a 2×2 window. The result is a down-scaled version of the feature maps

extracted by the preceding convolution layers.

Finally, the outputs of the last convolutional layers in the network are flattened and passed through one or more fully connected layers to produce the final output.

1.4.3 Image Classification

Image classification in machine learning involves the automatic classification of images, each into a class from within a set of predefined classes. An image is presented to the network and the network outputs a prediction of which class it determines is most likely to describe the main subject of the image. This is usually in the form of a probability distribution over the classes, the highest probability being the most likely class.

For a network with more than two output classes, the classification is made using a *softmax* classification layer. The softmax function is defined as following:

$$\text{softmax}(x) = \frac{e^{x_j}}{\sum_{i=1}^n e^{x_i}}, \quad j = 1:n \quad (1.1)$$

Where n is the number of classes (or number of elements in the vector x). The result is a vector of values, normalised such that the sum of all values is 1. Therefore, the resultant vector can be interpreted as the probabilities of each class given the input image. It should be noted that image classification assumes that there is only one class per image.

Training data for image classification models consists of a set of images, each with a single label.

1.4.4 Object Detection

Object detection is an extension of image classification that adds object localisation to the problem. One common general workflow for these object detection models, based on the RCNN architecture (Girshick et al. 2013), is as follows. Firstly, an algorithm, upon receiving the input image, proposes many regions within the image likely to contain an object (any object at this stage). Then, the regions are passed to an image classification model. The regions with the highest classification confidence are considered valid predictions

and the bounds of the proposed regions become the bounding boxes used to delineate detected objects. The training data for these models are images annotated with bounding boxes and class names describing the object within each individual box.

1.4.5 Image Instance Segmentation

Instance segmentation is a further development onto object detection, wherein (in general) the regions proposed by the aforementioned RoI (Region of Interest) network, are additionally passed to a CNN network specialised in segmenting which pixels belong to the object in question as opposed to those belonging to the background or other objects. The product is in image superimposed by masks for every object within the images, labelled by class. Training data for these models involve images annotated with masks for each training object within the image, labelled by class.

1.4.6 Image Annotation

Image annotation is the process of assigning labels to images (or to objects within the images) and thus creating a dataset with which an image classification, detection or segmentation model can be trained. In the context of this study, the creation of an instance segmentation dataset was required for the training of stage one. This involved the masking of each individual tree within the images and labelling them by species.

Computer Vision Annotation Tool

There are numerous tools and methods to assist in image annotation, both proprietary and freely available to the public. The tool selected for this study was Computer Vision Annotation Tool (CVAT), a free, open-source annotation tool with advanced customisation features. CVAT was primarily selected for its ability to integrate machine learning models to assist in or automate the labelling process.

Segment Anything Model

In the annotation of the dataset for this study, Meta’s Segment Anything Model (SAM) (Kirillov et al. 2023) was utilised to greatly reduce time spent labelling and produce a high quality, instance segmentation dataset quickly from scratch. SAM is a promptable, zero-shot, generalised object segmentation model, capable of segmenting objects it has never seen thanks to it’s ‘understanding’ of what objects are. The model was implanted into CVAT such that trees could be segmented using prompts generated by a mouse click. In many cases, a tree could be accurately segmented using a single click.

1.4.7 Image Regression

Image regression only differs from image classification in that the final layer consists of neurons — one for every output — with linear activation functions rather than softmax. The linear activation function ($y = x$) essentially acts as a sum of the outputs from the previous layer (after weights and biases have been applied). The result is a continuous number that represents the desired, learned metric based on the input image. In this case, training data consists of an image and a corresponding continuous number representing the metric being learned. The model then minimises the error between its output and these numbers until a sufficient accuracy is achieved.

1.4.8 Hyperparameters

The term ‘hyperparameter’ denotes any parameter set before the training of a deep learning model begins that effects the learning and behaviour of a model, such as the learning stability or the degree to which the model generalises. Some of the more detailed effects of these hyperparameters will be discussed in the following sections. The following hyperparameters were adjusted in the models in this study:

1. Learning Rate
2. Batch Size
3. L2 Regularisation Weight Decay

4. Training Steps / Epochs

5. Steps per Epoch

6. Optimiser Function

7. Dropout Rate

8. Model Layers

9. Fully Connected Layer Units

10. Augmentations

1.4.9 Model Training

The goal of training any machine learning algorithm is to minimise the *cost function*. The exact cost function being applied varies depending on the design and application space of a model, however, the purpose of all such functions is this: to summarise to error between a model's predictions and the actual values in a given dataset.

Once the error between a model's predictions and the actual values is known, optimisation of this function can begin. It can be assumed that once the cost function has been minimised such that no further reduction in error is possible, the model is fully trained and has learnt all it is capable of learning with its current architecture and dataset.

The problem faced by machine learning, particularly deep learning, is the processing power required by this optimisation process. Traditionally, optimisation involves using partial derivatives to find the gradient of the function being optimised (in this case, the cost function) with regard to the variables involved. Then these variables are recursively adjusted based on the gradient until the gradient is zero and a minimum has been reached (ideally a global minimum, though this is not always the case). However, in deep learning there is generally tens of millions of neurons, each with their own tunable weight and bias, even in a “small” model; GTP-4 has 170 trillion tunable parameters.

Optimising a system of this size using traditional gradient descent is clearly not viable, thus, deep learning models employ specialised optimiser functions designed for the task.

Stochastic Gradient Descent is one such optimiser function which performs gradient descent on parameters selected randomly with each iteration and is the foundation of most modern optimiser functions. The Adam optimiser (Kingma & Ba 2017) applies a computationally efficient adaptive learning rate to reach a cost minimum faster and more effectively than many other functions and is one of the most popular modern optimiser functions. It was these two optimiser functions that were tested in this study, because, despite Adam’s popularity, SGD has sometimes been found to find lower cost minimums than other optimiser, albeit after a comparatively longer training time.

Another important concept to note is that a second, smaller dataset, called the validation dataset, is used to validate the models training progress every so many training steps to monitor the models fit to the data and cease training if it begins to overfit. This concept is explained further in an upcoming section.

1.4.10 Transfer Learning

When training a deep learning model from scratch, the weights and biases of the model are initialised with random values and then, throughout the training process, are adjusted to progressively more useful values to eventually extract valuable features from the input data. To give a model a head start in identifying and extracting useful features, particularly when the dataset is relatively small, it can be initialised with weights from the same model being trained on another, larger dataset, assuming the architecture for both models is identical. Then, the last few layers, referred to as the classification head, can be exchanged for layers suited to the new model. The model then is trained further on the new dataset to become specialised in the new task. This second stage of training is normally referred to as “Fine-Tuning”.

Depending on the application and the similarity between the original and new datasets, a varying number of layers or sections of the model can be frozen so that their weights and biases remain unchanged during the fine-tuning process. This ensures the model focuses its learning on reclassifying the already high-quality features learned in previous training.

Two pretrained models were used in this study. The first was trained on the Large Vocabulary Instance Segmentation (LVIS) dataset (Gupta et al. 2019); the second on the ImageNet dataset (Russakovsky et al. 2015). Since the classes these pretrained models

were trained for (mostly common objects, people and animals) greatly differed from those required for this study, it was determined (after justifying experimentation) that no layers would be frozen in the fine-tuning of the models.

1.4.11 Over-fitting

The term “Over-fitting”, in the context of deep learning, refers to the phenomena where the model in question becomes so fitted to the training data that it no longer represents the ‘real world’, but instead it has simply memorised the noise patterns of the training data set.

Over Fitting can be easily detected by observing the loss curves of a given model. If the validation loss begins decreasing with the training loss and then reaches a point where, while the training loss continues to decrease, the validation loss begins to increase, the model has begun overfitting. An example of a loss curve from an overfitting model can be seen below in Figure 1.3.

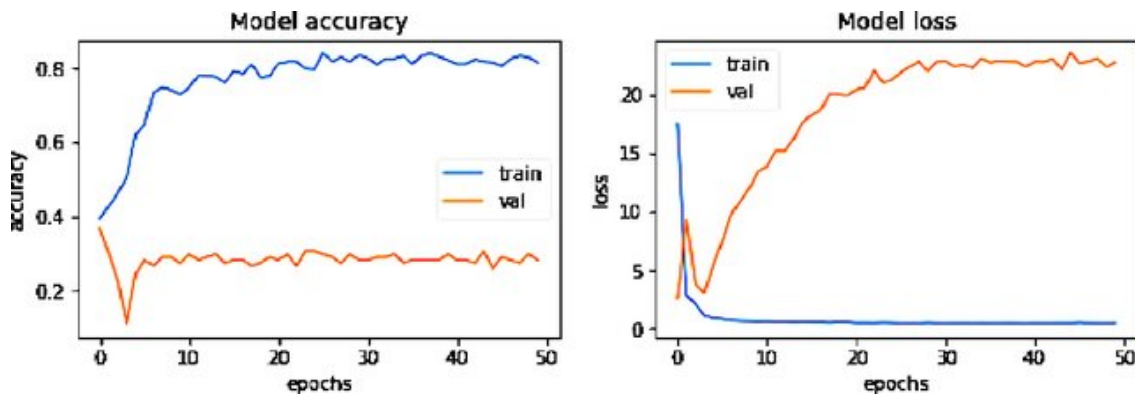


Figure 1.3: Accuracy (Left) and Loss (Right) Curves of an Overfitting Network (Source: Rahaman et al. 2020)

Conversely, a model with a good fit will have a loss curve where both the validation and training losses continue to decrease together until steady state is reached, and no more learning can occur with the given training data.

Mitigation

The process of mitigating over-fitting such that a model will perform as well on unseen data as it does on the training data is termed *generalisation*. The first generalisation technique to be considered in any deep learning model is ensuring the architecture topology matches the complexity of the problem. If a model has far too many layers or classification layer neurons than what is required to extract the important features and make accurate predictions, the model will use the excess units to memorise each solution and overfit the training data.

After the model architecture is appropriate for the task, training-time, active techniques can be applied. The first among these is *Weight Decay Regularisation* (a.k.a. L Regularisation). Overfitting often results in large weights for certain features; this is what memorisation looks like on the neuron level. Weight decay regularisation involves an additional term being added to the cost function which is — in the case of L2 regularisation which was used in this study — the sum of the squares of the weights, multiplied by a settable factor referred to as the “regularisation factor” or the “weight decay”. This term penalises large weights, encouraging the network to make its decisions based on a larger number of features, rather than a select few and results in a more stable, more generalised model.

The second generalisation method implemented in this study is *Dropout*. The addition of dropout layers encourage a network to diversify its decision-making further by randomly deactivating a specified portion of the previous layer’s units each training step. This can be implemented after every dense layer, after larger sections of the network or in the classification layers.

The third generalisation technique used in this study is *Data Augmentation*. To further assist a network in preparing for data outside the training set, the input images can be randomly augmented beforehand or with each training cycle, thus simulating a much larger training dataset and preventing the network from overfitting to particular framing, size or orientation of the objects being classified. For example if a model has only been trained on images of an object in its usual orientation, it likely will not recognise the same object if presented upside-down. The augmentations applied to the input images in this study were as follows:

1. Vertical and Horizontal Flip

2. Rotation

3. Scale

4. Gaussian Blur

5. Multiply

6. Sheer

7. Linear Contrast

8. Scale

9. Additive Gaussian Noise

1.4.12 Under-fitting

In some cases, a model is not sufficiently complex to learn to identify the features required for a certain task. In this case, it is said that the model is *under-fitting*. Under-fitting can also be caused by low quality data, data with too much noise or simply by attempting a task in which there are no identifiable patterns or too many exceptions. This is identifiable when the validation loss of a model reaches a steady state that is well above that of the training loss.

In cases where the model is not complex enough to detect the required patterns, simply increasing the number of layers and/or the number of units in each layer will grant the model the ability to fit the dataset. Otherwise, the quality of the dataset may need to be assessed, or the task deemed unviable for machine learning given the current features or data.

1.4.13 Model Evaluation

After training is completed, a model can then be evaluated to assess its accuracy and investigate the performance of the model in different facets of the presented task. This enables one to make inferences about the model's expected performance on real world

data. This must be done using a new dataset, separate from the training and validation sets, to ensure the data used for evaluation has never been seen by the model before. The new dataset is referred to as the *test set*.

The test set, which is ideally large enough to ensure reasonably complete representation of the spectrum of possible inputs the model may face, is then used to calculate the metrics being used to evaluate the model. The metrics used vary between tasks in an attempt to accurately summarise the model's actual expected ability to perform on real-world data for its given task. The metrics used in this study are as follows:

1. Precision
2. Average Precision
3. Mean Average Precision
4. Recall
5. Precision Recall Curve
6. Receiver Operating Characteristic Curve
7. Area Under Curve
8. Accuracy
9. F_1 Score
10. Sample-weighted F_1 Score
11. R^2

Precision

$$\text{Precision} = \frac{TP}{TP + FP} \quad (1.2)$$

Precision, given in Equation 1.2 above, is a measure of the proportion of correct positive classifications made by a model.

Recall

$$\text{Recall} = \frac{TP}{TP + FN} \quad (1.3)$$

Recall, given in Equation 1.3 above, is a measure of the proportion of actual positive instances a model correctly classified.

Precision Recall Curve

The Precision Recall (PR) curve is a plot of Precision as a function of Recall across a range of classification thresholds (the levels at which a class’s prediction confidence output is considered a positive prediction).

Average Precision

The Average Precision (AP) metric is the average value of Precision across the full range of corresponding Recall values (as plotted in a Precision Recall curve) or, more simply, the integral of the PR curve. While the AP should, therefore, always be equivalent to the Area Under the PR curve (AUPRC metric), sometimes they differ slightly because of differences in the way they are computed in a discrete context. The Mean Average Precision (*mAP*) refers to the mean of the APs of each class in a multi-class classifier.

However, the COCO object detection challenge standard defines AP as the average of the AP values of ten Intersection over Union — or IoU — (the proportion of overlap between a models predicted boundary box or segmentation and that of ground truth) thresholds between 0.5 and 0.95 (0.5:0.05:0.95). Thus, in this study, when referring to the AP values of an object detection model, “*AP*⁵⁰” is used to refer to the AP values calculated at IoU 0.5 and “AP” when referring to the COCO AP.

Receiver Operating Characteristic Curve

The Receiver Operating Characteristic (ROC) curve is a plot of True Positive Rate as a function of False Positive Rate across a range of classification thresholds.

Area Under Curve

The Area Under Curve (AUC) is commonly used for both the Precision Recall (PR) curve and the Receiver Operating Characteristic curve as a way to summarise the curves in a single figure. The AUC is simply the integral of a given curve.

Accuracy

The accuracy metric is simply the proportion of matching values between a model's output and ground truth.

F_1 Score and sample-weighted F_1 Score

$$F_1 \text{ Score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (1.4)$$

The F_1 score is the harmonic average of the precision and recall scores. It will increase as precision and recall do, but will penalise a difference between the two.

The *sample-weighted* F_1 Score is appropriate where there is class imbalance within the dataset and is simply the weighted average of the F_1 scores for each class of a multi-class model. The weights are based on the relative representation of each class within the dataset, as defined in Equation 1.5 below.

$$\text{Sample-weighted } F_1 \text{ Score} = \sum_{i=1}^N w_i \times F_1 \text{ Score}_i \quad (1.5)$$

where

$$w_i = \frac{\text{No. of samples in class } i}{\text{Total No. of samples}} \quad (1.6)$$

R^2 - The Coefficient of Determination

In the context of this study, the R^2 or “coefficient of determination” is the same as the metric of the same name commonly used in statistics. The only exception being that, due

to the way it's calculated by the TensorFlow package, a negative result is possible. Such a result simply means the fit is arbitrarily worse than random.

1.5 Research Questions

The questions posed in the development of this study are as follows:

1. How could modern machine learning technologies aid the Australian forestry industry?
2. How could the existing yearly imaging be utilised to provide more value?
3. Could the yearly inventory process be digitally automated, removing the need for manual measurement?

Chapter 2

Literature Review

2.1 Sandalwood Tree Volume Prediction Techniques

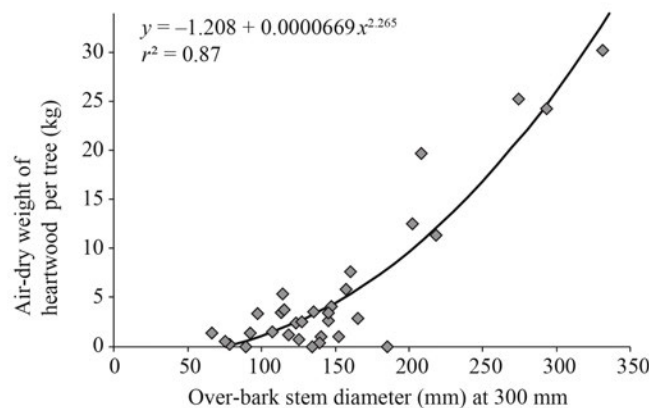


Figure 2.1: Relationship Between Tree Girth and Heartwood Volume in Sandalwood (Source: Brand et al. 2012)

Sandalwood heartwood volume has a strong, positive exponential relationship with tree girth (Das 2021), as seen in Figure 2.1 above. This is the foundation of all existing sandalwood oil yield prediction techniques found in the literature.

Quintis, the company with which this study was completed, applies this fact to estimate tree yield. They also operate under the assumption that economically viable wood generally only extends to bole height, given the cost of processing. Thus, they use an equation that includes bole height (the height of the first fork), the diameter of the main

stem and health status (a 1-5 rating of a tree’s overall health) to estimate the volume of heartwood, and therefore, essential oil within a tree. It is worth noting that the health rating is only used such that a rating of one (a dead tree) results in no oil and a rating of two reduces the effectual diameter by a set factor. All other health ratings, however, have no effect on the estimation, in that they do not reduce the effectual diameter.

2.2 Innovative General Techniques For Tree Health and Volume Assessment, Detection and Prediction

2.2.1 Health Monitoring

Visual assessment of tree health is susceptible to bias, human error and inconsistency, and although attempts have been made to mitigate these errors (Redfern & Boswell 2004), this method will always be hindered by its decentralisation.

LiDAR and SAR (Synthetic Aperture Radar) have both been applied for health assessment via biomass measurement (Tanase et al. 2014). LiDAR shows promising accuracy and high resolution, though bring with it a high operational cost. SAR, however, still achieved an r value of 0.5 (compared to LiDAR’s 0.9) correlating with biomass and has a much lower cost and high availability due to the technology’s ability to “see through” cloud-cover, as opposed to traditional satellite imagery. SAR is not, however, capable of individual tree level resolution and its usage is limited to the general monitoring of larger areas.

Vegetation indexes — most prominently the Normalised Difference Vegetation Index (NDVI) — calculated from satellite imagery have also been used for monitoring water stress, biomass and other health related anomalies (Wang et al. 2010). While achieving reasonable accuracy (R^2 0.765 for water stress and > 0.5 for biomass), again, satellite imagery is still not capable of individual tree resolution and, additionally, this form of imaging is susceptible to occlusion by cloud-cover, which can further reduce its already limited temporal resolution.

Finally, UAV imagery has been used, particularly more recently, to monitor and detect health related anomalies in forests and forest plantations. After reviewing 99 papers, written between 2012 and 2021, on the use of UAVs from various forms of health monitoring,

Ecke et al. (2022) concluded that UAVs are a cost-efficient and robust tool for this application and the technology is rapidly advancing, further increasing its applicability.

2.2.2 Yield Prediction

A study by Pertille et al. (2023) produced a regression model capable of predicting the yield of a pine forest plantation in Rio Negro, Paraná, Brazil with an adjusted R^2 of 0.51. The linear model used selected vegetation indices calculated from satellite imagery as inputs, and they were able to find no significant differences between the performance of their model and the current, statistics-based inventory methods. This study alludes to the potential for aerial imagery to be used to create yield prediction techniques that surpass traditional inventory methods.

Popescu et al. (2003) proposed a method using LiDAR measured crown diameter in combination with a linear regression model to predict tree volume of assorted species in a forest in southeast America, achieving an R^2 of 0.83. While impressive, the application of LiDAR was deemed outside the scope of this study as one of the reasons drone imaging was selected as a data source is its low cost and high availability. Furthermore, it would mean adding tasks to the inventory process, rather than reducing the workload.

2.3 Leading Machine Learning Techniques For Image Analysis

2.3.1 Instance Segmentation

For the purpose of instance segmentation, Mask RCNN (He et al. 2018) was consistently found to be the leader in terms of accuracy and popularity amongst the literature. The only other architecture that stands as a competitor for Mask RCNN in terms of accuracy is the image segmentation model, originally developed for the analysis of medical imaging, called “U-Net” (Ronneberger et al. 2015). However, the U-Net architecture only outputs binary masks, thus can only be used for semantic segmentation (segmentation without delineating between individual instances, only class). Moreover, several studies (Widyaningrum et al. 2022, Durkee et al. 2021) have compared the performance of the two architectures for semantic segmentation performance and the Mask RCNN had superior

performance. The only exception was that in a few, specific use-cases where “attention” to very fine detail is vital, it was found that U-Net could perform marginally better due to the skip connections that transfer low-level features from the architecture’s encoder to the decoder.

2.3.2 Image Classification

Convolutional Neural Networks (CNNs) are the most used form of deep learning for image classification in all fields by a significant margin (Aslam & N 2019, Li 2022). Regarding specific CNN architectures, there are a myriad of options presented in the literature. Ignoring architectures that are not publicly available or are highly task specific, variations of ResNet (Bello et al. 2021), DenseNet (which is an extension of the concepts underlying ResNet) (Huang et al. 2018) and InceptionV3 (Szegedy et al. 2015) appear to be the generally top performing architectures across the literature and in image classification competitions such as ILSRVC (Russakovsky et al. 2015).

2.3.3 Image Regression

The body of literature around image regression is far smaller compared to that of image classification. However, two studies, applying CNN-based image regression to calorie content prediction from food images proved the concept to be achievable (Myers et al. 2015, Ege & Yanai 2018). Moreover, the Ege & Yanai (2018) model achieved an R^2 score of 0.817. The studies suggest that an existing CNN classification architecture can be utilised for this task, simply replacing the classification layers for a regression output layer.

2.4 Machine Learning For Individual Tree Health and Volume Prediction

The emergence of machine learning techniques facilitated the automatic detection of complicated relationships within data, enabling accurate predictions of parameters and classifications that may surpass human capabilities. Therefore, machine learning holds

2.4 Machine Learning For Individual Tree Health and Volume Prediction 23

promise in addressing the challenge of more accurately estimating the total marketable product and tree health within entire plantations.

After reviewing the existing body of papers addressing similar issues, using machine learning and overhead imagery for tree health and/or volume related predictions, two primary approaches emerged. The first approach seen amongst the papers was the extraction of features from the images, followed by a shallow machine learning model that uses the features as input rather than the images. The second approach was to use deep learning models—usually a form of Convolutional Neural Network (CNN)—to analyse the images of the trees to make predictions.

2.4.1 Feature Extraction and Shallow Learning

While shallow learning was certainly the less prevalent of the two main approaches, it is important to mention because, like with any machine learning problem, if the same outcome can be achieved using shallow learning, then shallow learning is a better solution as it generally requires far less resources and computational power, both in training and in post-training application.

Windrim et al. (2020) tested four shallow learning models—the most accurate being a Support Vector Machine (SVM)—for their affinity at detecting trees, classifying healthy & unhealthy trees and classifying healthy trees & those attacked by the Sirex woodwasp in a Pine plantation. This study notably achieved 100% accuracy in classifying healthy and unhealthy trees using only RGB image data. They also found that for both tree detection and Sirex attack detection, the addition of Near Infra-Red (NIR) image data significantly increased accuracy and the addition of pointcloud and Digital Terrain Model (DTM) data slightly increased accuracy further. A summary of the best results in this study are seen below in Table 2.1.

Application	Error		Features Used
	Commission	Omission	
Tree Detection	2.5%	3%	RGB, NIR, pointcloud
Health Classification	0%	0%	RGB
Sirex Attack Classification	24.1%	10.9%	RGB, NIR

Table 2.1: Windrim et al. Summary of Best Results

Guerra-Hernández et al. (2021) proposed a method for the classification of trees in to 4 crown defoliation based health categories in the Bertandos Forest of North-West Portugal. The method involved the manual delineation of individual trees, followed by their classification based on vegetation indexes and a DSM (Digital Surface Map) textural feature using logistic regression. They achieved a classification accuracy of 75%. Abdollahnejad & Panagiotidis (2020) applied a Support Vector Machine to similar features to classify dead, healthy and infected coniferous trees in Kostelec, Czech Republic, achieving an overall accuracy of 85%. While manual tree delineation would be less than ideal, these studies suggest vegetation indexes and DSMs could be valuable features for classification.

2.4.2 Deep Learning

Sani-Mohammed et al. (2022) proposed an instance segmentation model for the detection and segmentation of dead trees in the Bavarian Forest National Park, Germany. The dataset contained images consisting of NIR, red and green bands, collected by a manned aircraft from an altitude of 2918 meters. Their pretrained Mask RCNN based model, pretrained on the Microsoft COCO dataset (Lin et al. 2015), achieved an AP^{50} of 0.85, a COCO AP of 0.54 and an F^1 score of 0.87 for the binary classification and segmentation of dead trees. A similar model could provide value to Quintis, however, they are more concerned about the 5 value Health Score rather than simply an indication whether a tree is alive or dead.

Yarak et al. (2021) proposed a similar model, though using RGB images taken at 100 meters altitude for the purpose of detecting healthy and unhealthy oil palm trees, as well as for detecting and counting the trees. Their Faster RCNN based model, using a ResNet50 backbone, was able to detect oil palms and then healthy and unhealthy individuals to a high accuracy, with F_1 scores of 0.95, 0.92 and 0.87 respectively. However, in a secondary test the model only achieved and F_1 score of 0.57 for unhealthy trees. The tree counting feature of this method would certainly add value to the imagery already taken by Quintis. Moreover, the study suggests that a comparable accuracy is possible using only RGB images rather than the hyper-spectral bands applied in the previously mentioned study.

Şandric et al. (2022) proposed an individual tree health detection model that applied a Mask RCNN segmentation model to detect the individual trees, then calculated to average values of two RGB derived vegetation indexes over the canopy of the individual trees and classified their health based on where they sit in the distribution of all the trees in the image. While this method would provide value to Quintis for the purpose of continual health monitoring, the health classifications would not align with the Health Status used in their inventory calculations.

Very little research has been done into the automatic volume or biomass prediction of individual trees. The majority of the related studies found in the literature present models for coarse, region-based predictions of volume/biomass, rather than fine predictions on a per-tree basis (Tanase et al. 2014, Pertille et al. 2023). However, a study by Iizuka et al. (2022) shows a close relationship between orthomosaic derived crown size estimations and diameter at breast height (DBH), achieving an adjusted R^2 value of 0.83 using a Support Vector Regressor. This study suggests a relationship between tree crown and DBH, thereby implying that it may be possible to predict the DBH from the orthomosaic in Sandalwood also.

2.4.3 Common Design Amongst Health/Volume Prediction Models

When reflecting on the most successful studies encountered in the review of the literature on the topic of individual tree health and volume prediction, some patterns emerged. Firstly, the individual detection or delineation of individual trees had to be achieved. Of the techniques applied amongst the studies, Mask RCNN was certainly the most prevalent and successful. After tree delineation, the predictions or classifications of individual trees are then performed using a model trained on either RGB images, vegetation indexes, structural features or some combination of all of these. It is from this observation that the methodology and design proposed by this study was founded on.

2.5 Related Techniques in Agriculture

Ramos-Giraldo et al. (2020) proposed a method using DenseNet to predict a 1-5 index of water-stress present in unprocessed images of soybean plants for the purpose of preventing yield loss. Using a dataset of 3000 annotated images, the system achieved 88% accuracy,

the confusion matrix showing only small levels of confusion between adjacent classes, stemming from borderline cases. An et al. (2019) conducted a similar study on images of Maize, comparing greyscale and RGB images as well as ResNet50 and ResNet152 both with pretrained and untrained initial weights. They also tested Gradient Boosted Decision Trees (a shallow learning model) for comparison. They found pretrained ResNet50 with RGB images to be the most successful model, achieving an accuracy of 96%.

While not directly applicable, these studies provided useful insight into the behaviour of various machine learning models in related tasks.

2.6 Application of These Techniques in Sandalwood Plantations

After a comprehensive survey of the literature body, no health, volume or even tree detection related studies could be found on natural sandalwood forests or man-made forest plantations. Moreover, the direct applicability of many of the aforementioned studies is limited due to the non-homogeneous format of sandalwood plantations as a result of the species' hemiparasitic nature. Thus furthering the need for more research into the area.

Chapter 3

Research Methodology

In discussing the needs and opportunities of the Quintis research department and a potential topic of this study with Dr. Precila Gonzales and other Quintis staff, it became clear that both yearly, full inventory, as well as the monitoring of the trees' individual health over time were costly and time-consuming yet vitally important to the functioning and profitability of the company. After reviewing the literature body, it was evident that, while evidence is present that it can be done, no research had been completed into the application of machine learning to automating the detection of individual tree health, bole height and DBH. Nor exists any research into the application of anything remotely similar to sandalwood plantations specifically. Thus, the opportunity to develop a system that could save Quintis a substantial amount of both time and money, by utilising their existing imagery to automate the inventory and health monitoring processes emerged.

3.1 Design Objectives

In accordance to this opportunity, the available resources and the findings in the review of the greater literature body, the following design objectives were identified:

1. Use modern machine learning technology to increase the utility of existing imagery
2. Enable the automation of the inventory process using only drone imagery
3. Enable the automatic monitoring of individual tree health across entire plantations using only drone imagery

3.2 System Architecture

Below, in Figure 3.1, is the proposed architecture of the final system, in accordance with the design objectives.

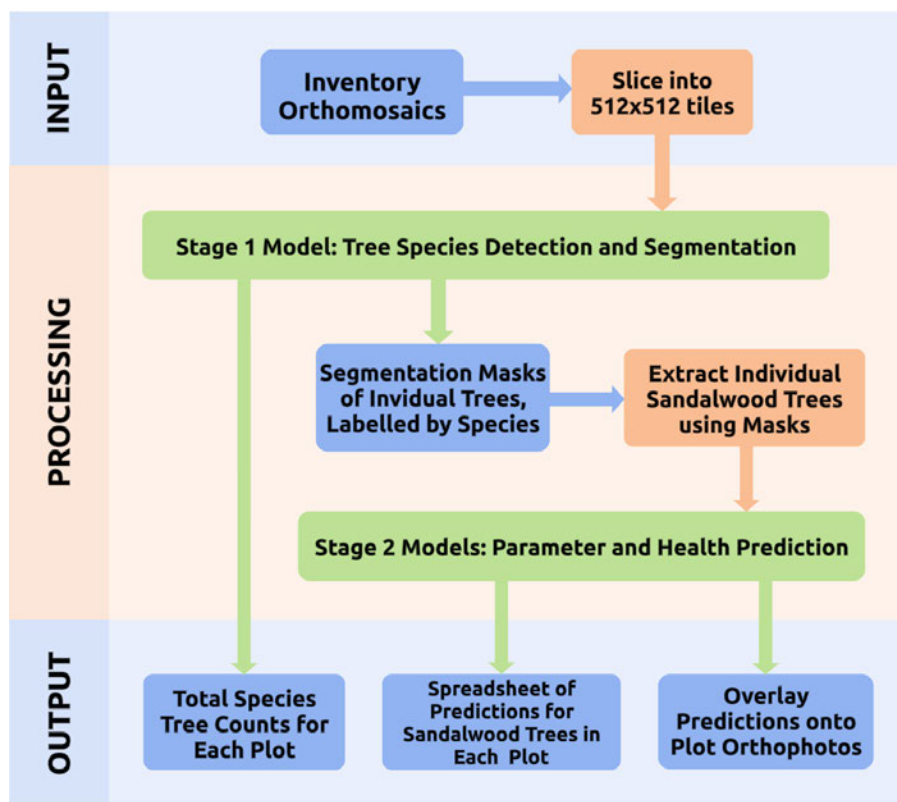


Figure 3.1: Final Proposed System

For the purpose of this study, and in conformity with its time restraints, only the design and training of the Stage 1 & 2 models were covered in order to investigate the viability of such a system.

3.3 Data Collection

The images used for training the models were collected as part of Quintis' normal, yearly inventory. The images were taken using a DJI Mavic 3 Multi-spectral, from an altitude of approximately 225 meters, in a snaking pattern across the plantations, with 65% front and side overlap and a 0° (nadir) gimbal angle. The Drone simultaneously captures RGB images, several Infra-Red bands and records highly accurate (± 0.1 meters) GPS information for each image.

The tree parameters were recorded using tree diameter tape and an 8-meter telescopic measuring pole, and the individual tree health was assessed by qualified foresters or trained contractors. Field notes and GPS coordinates were also recorded to assist in locating the rows and trees recorded within the orthomosaics. The data was collated into a CSV file.

The recorded data was then manually linked to the corresponding tree and those trees were segmented in CVAT. A Python script was used to extract both a masked image and a square crop of each individual tree and save them under a directory corresponding to their health status. Furthermore, the script also saved the individual tree images in a separate directory with their bole height and DBH encoded into their filename.

3.4 Data Preprocessing

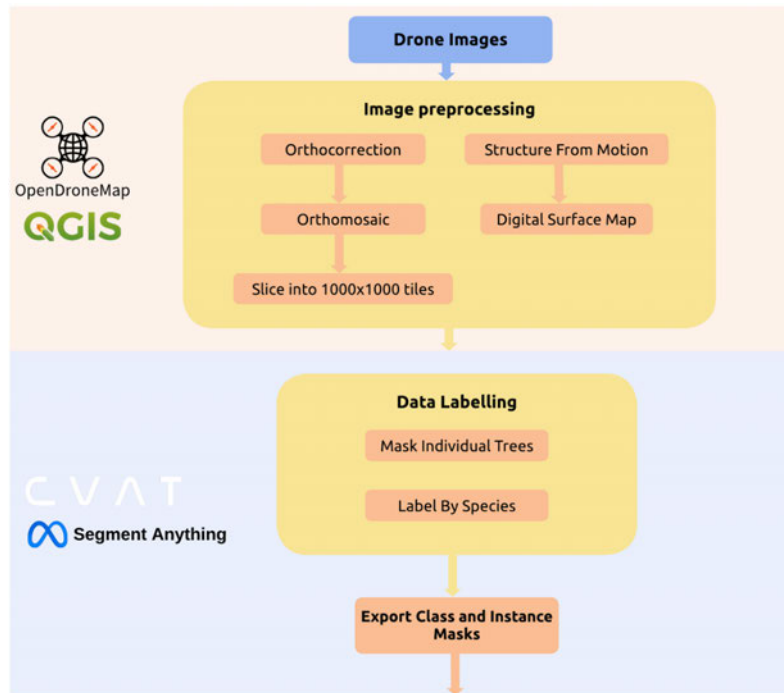


Figure 3.2: Flowchart of the Stage 1 Data Preprocessing Workflow

The workflow of the data preprocessing is depicted in Figure 3.2. Firstly, the raw images are converted into a single, geo-referenced orthomosaic for each plantation block, as seen in Figure 3.3, using the open source image processing software “Open Drone Map” .



Figure 3.3: Orthomosaic of Kingston Rest Plantation 1, Block 47

The Open Drone Map tool was also used to calculate a DSM (Digital Surface Map) for

each plot using SfM (Structure from Motion) algorithms, as seen in Figure 3.4. The resulting file could then be used to visualise and retrieve the height of individual trees. However, due to the height of the drone during imaging, and therefore, the lower resolution in pixels per meter, the algorithm often omits smaller trees entirely and miss-identifies ground regions, rendering the DSM an unreliable source of absolute height information.

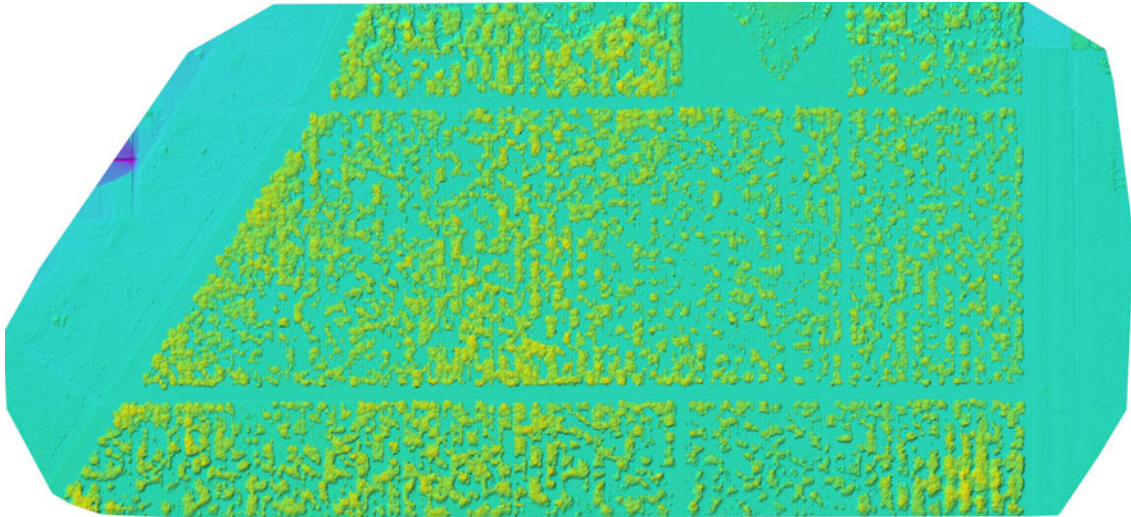


Figure 3.4: Orthomosaic of Kingston Rest Plantation 1, Block 47

The error caused by the misidentified ground regions was corrected by subtracting the entire image by the value of a true ground region, as seen in Figure 3.5 below.

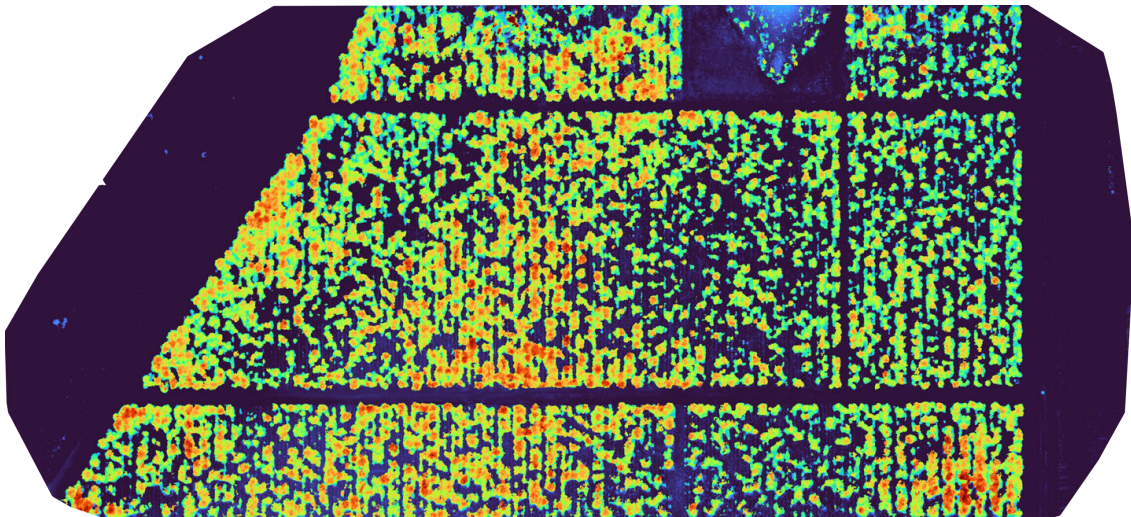


Figure 3.5: Orthomosaic of Kingston Rest Plantation 1, Block 47

3.5 Data Labelling

Due to the size and, thus, the memory requirements of these orthomosaics, the orthomosaics were split into tiles of 1000x1000 pixels to allow them to be passed to the Segment Anything Model to accelerate the labelling process within CVAT (Computer Vision Annotation Tool). Each tree was individually labelled by species, and unrecognisable regions or unidentifiable trees were ignored. The aforementioned unrecognisable regions are normally artefacts caused by the orthorectification and stitching process, which are worsened by the altitude (approximately 225 meters) used by the inventory team for the drone imagery. The labelled tiles were then exported, creating a class mask and an instance mask for each tile.

3.6 Summary of Final Datasets

Below, in Table 3.1, is a breakdown of each of the datasets used in this study.

Stage 1: Tree Detection Dataset	Sandalwood		Senna		Dalbergia		Total	
	2,063		478		750			
							3,291	
Stage 2: Health Classifier Dataset	H1	H2	H3	H4	H5		Total	
	95	44	75	92	164			
							470	
Stage 2: Parameter Regressor Dataset	Bole Height (m)						DBH (cm)	
	Min	Mean	Max		Min	Mean	Max	Total
	1.3	1.79	3.73		7.8	10.25	21.6	
							277	

Table 3.1: Breakdown of Final Datasets

3.7 Training

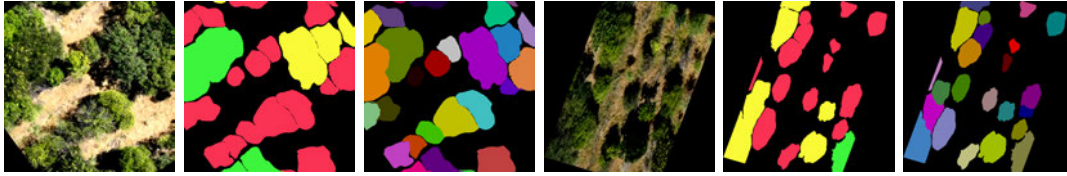
3.7.1 Environment

Training was completed on an NVIDIA 3080, with 10 GB of GRAM and 8704 CUDA cores and run within the official Docker images of PyTorch (Stage 1) and TensorFlow (Stage 2) with WSL2 (Windows Subsystem for Linux 2) for compatibility with the latest releases of the packages and CUDA (Compute Unified Device Architecture).

3.7.2 Stage 1: Automatic Individual Tree Detection and Segmentation



(a) Original Images, Class Masks and Instance Masks



(b) Augmented Images, Class Masks and Instance Masks

Figure 3.6: Examples of Training Data Augmentations

After the aforementioned image preprocessing, the images were sliced into 512x512 images and split into training, validation and test datasets (70/15/15 split). Random augmentations, such as those in Figure 3.6 above, were performed on the training data and the system was trained to maximise validation performance. Hyperparameters were tuned to further maximise model performance. The training workflow can be seen below in Figure 3.7.

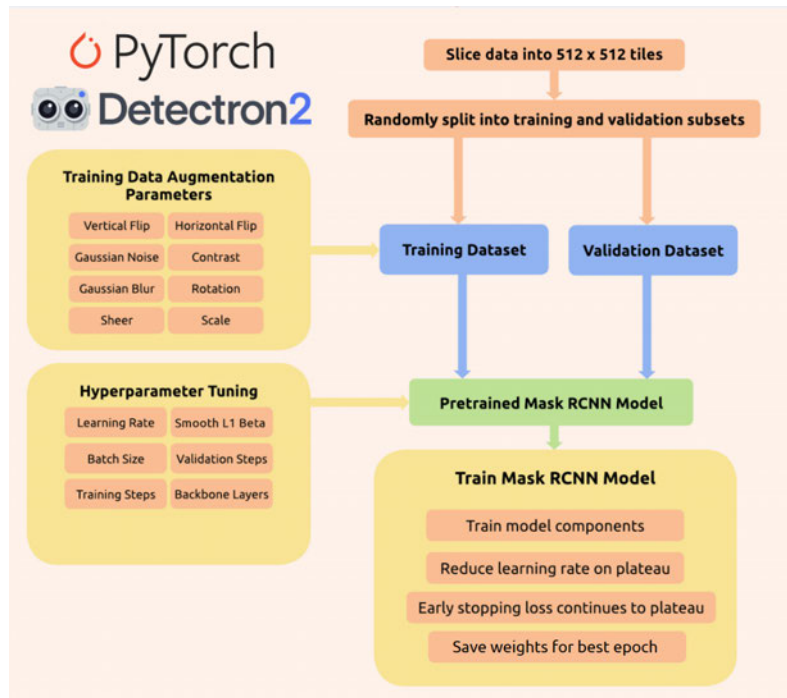


Figure 3.7: Stage 1 Training Workflow

3.7.3 Stage 2: Automatic Tree Health and Parameter Prediction

An individual model was built for each parameter (Health, bole height and DBH) and the corresponding data for each model was loaded and split into training, validation and test subsets (80/10/10 split). The model was trained, applying simple data augmentations on-the-fly until validation performance was maximised. Hyperparameters were tuned to further maximise model performance. The training workflow can be seen below in Figure 3.8.

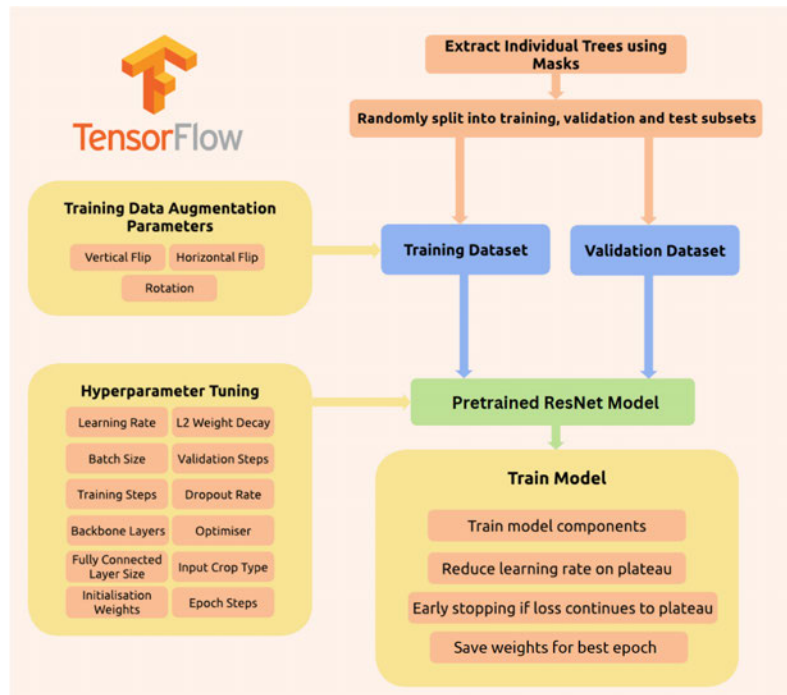


Figure 3.8: Stage 2 Training Workflow

For the tree health classifier model, Categorical Cross Entropy Loss was used as the training loss and validation performance was monitored such that F1 score was maximised. For both the bole height and DBH regressors Mean Squared Error (MSE) was used as the loss function and the validation performance monitored such that Root Mean Squared Error was minimised.

3.8 Model Evaluation

After training, the models were assessed based on relevant metrics to determine their viability for use on real-world data. These metrics are as seen in Table 3.2 below.

Stage 1: Tree Detection Model	Stage 2: Tree Parameter Preditcion	
	<i>Tree Health Classifier</i>	<i>Bole Height and DBH Regressors</i>
Overall Average Precision	Accuracy	Mean Squared Error
Per-class Average Precisions	Precision	Accuracy
Area Under Overall Precision Recall Curves	Recall	Root Mean Squared Error
Area Under Per-class Precision Recall Curves	Area Under Precision Recall Curve	Mean Absolute Error
	Overall F_1 Score (<i>macro</i>)	R^2
	Per-class F_1 Scores	

Table 3.2: Metrics Used For Model Evaluation

Chapter 4

System Design

4.1 Pilot Studies

Before the final system architecture was established, several preliminary pilot studies were conducted, investigating the viability of existing technologies for use as the foundation of the components required by this study. The most prominent and developed of which are included in this section.

4.1.1 Matterport Mask RCNN Individual Tree Detection Model

Mask RCNN (He et al. 2018) is an object detection model, based on Faster R-CNN, extended for object segmentation. The model was first proposed in 2017 and outperformed all existing segmentation models at the time, including the COCO 2016 challenge winners. The most widespread and adopted implementations of this model is Abdulla (2017)’s open source implementation (referred to as “Matterport”), using the Python TensorFlow package, which is the version used for this pilot study.

A pretrained Mask RCNN model, consisting of a region proposal network followed by a ResNet50 backbone, was edited by way of removing the classification head containing the original classes of the MS COCO dataset and replacing them with a new classification head for the classes of this study’s dataset. Additionally, dropout layers and L2 regularisation was added to the model.

Early stopping, learning rate reduction and dropout were added to the existing model to enhance its performance and stability. From here, training began, and the model's hyperparameters, including the newly added dropout rate and L2 regularisation loss, were adjusted with each iteration to improve the model's classification and mask accuracy. As seen below in Figure 4.1, many iterations of changes to the data preprocessing and the model hyperparameters were implemented, resulting in a gradual convergence to a reasonably fitted model, granted the size and limitations of the dataset.

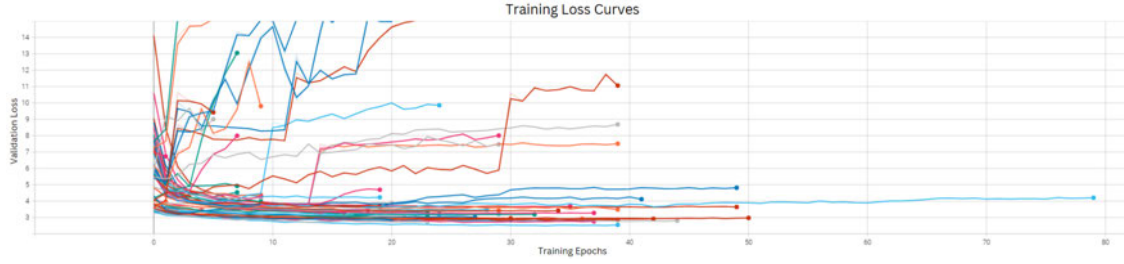


Figure 4.1: Epoch Validation Loss Curves of Model Training Attempts

The prediction results, while reasonably accurate, were deemed unsatisfactory, and furthermore, the outdated code was highly inefficient and took a long time to train. Thus, a more current and efficient model was sought.

4.2 Final System Overview

The overall system architecture was divided into two specialised stages, each applying their own unique deep learning architecture. The output of stage one becomes the image component of the second stages input.

4.2.1 Stage 1: Individual Tree Detection and Segmentation Model

Detectron2 is a modern object detection model, written using the Python PyTorch package and providing the option of the still popular Mask RCNN backbone as well as several other model architectures (Wu et al. 2019). The model applied more modern and efficient algorithms and provided better accuracy after training, substantially better training times and more options for model architectures and pretrained weights, thus, it was selected as the foundation for the final individual tree detection model. After some preliminary

testing, the Mask RCNN 101 backbone, pretrained on the LVIS (Large Vocabulary Instance Segmentation) v0.5 dataset (Gupta et al. 2019) was found to produce the best results, which was to be expected because the dataset has the most classes (1000) out of those available for Detectron2 and, therefore, networks trained on it will learn a larger and more diverse range of features.

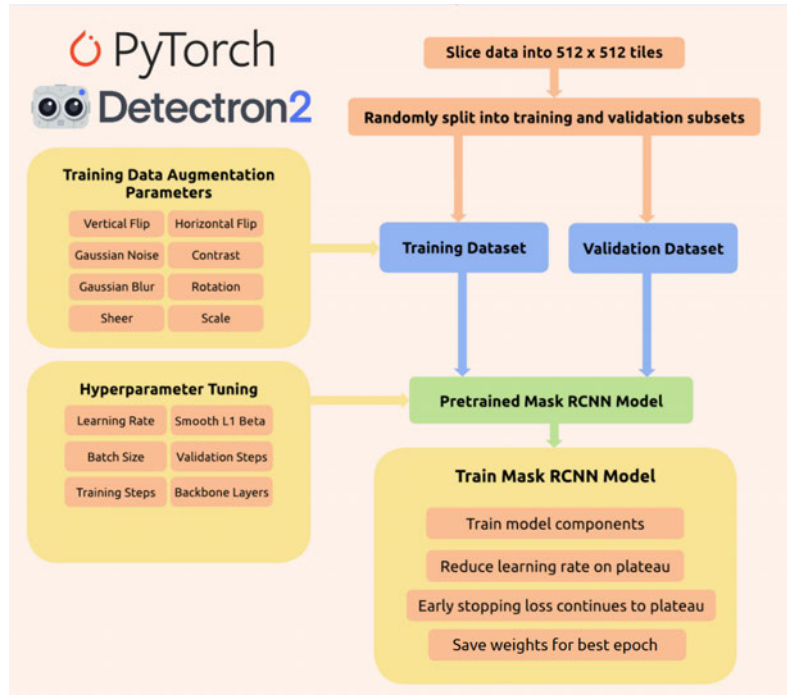


Figure 4.2: Flowchart of the Stage 1 Model Workflow

The workflow for the training of the model can be seen in Figure 4.2 above. The data was further prepared before reaching its final state for training in the following ways. To speed up training and maximise training data, the 1000x1000 tiles (both original images and their corresponding masks) were sliced into 512x512 pixel images. These images were padded, when needed, to fit the required size. After slicing, the images were then randomly divided into training, validation and test subsets, with a 70/15/15 split. The training subset was then augmented such that for each image, seven additional augmented versions were added, resulting in 648 training images containing 16538 instances in total, including 10413 Sandalwood instances. This helped prevent early over-fitting and improved the final accuracy of the model. Examples of the augmentations can be seen below in Figure 3.6. The datasets were then converted to the COCO 1.0 format for storage efficiency and

compatibility with the model used for detection.

The model was then trained on the data, plotting the validation results every hundredth training steps. The model was trained until it began to overfit (i.e. validation metrics began to deteriorate) and hyperparameters were tweaked to maximise performance.

4.2.2 Stage 2: Tree Parameter Prediction Model

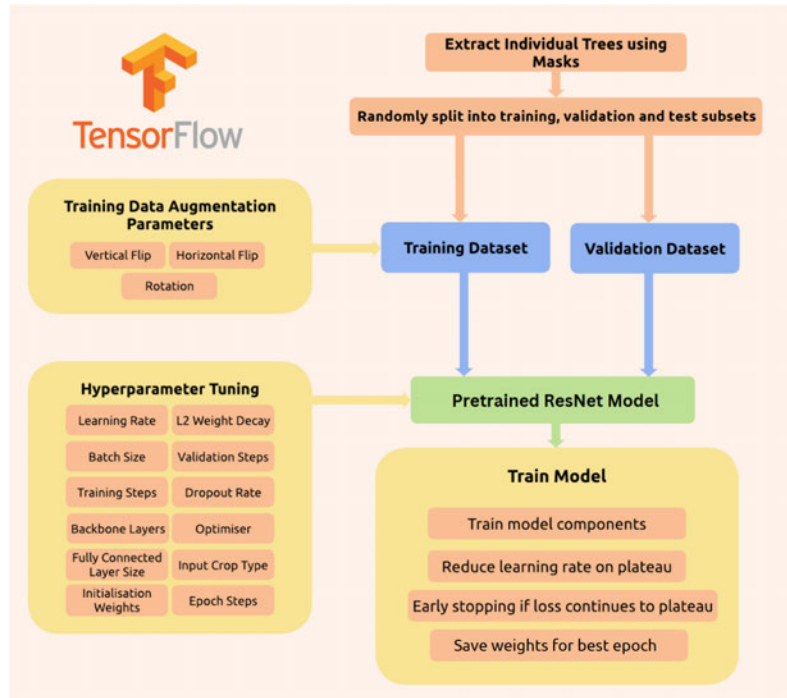


Figure 4.3: Flowchart of the Stage 2 Workflow

In a final, end-to-end implementation, the masked generated by the Sandalwood class segmentation results from Stage 1 would be fed into Stage 2 as inputs, however, to prove the concept and ensure quality training, the input data for this stage was manually curated for the purpose of the study. Due to the very small datasets, the training, validation and test datasets for the Stage 2 models were divided at an 80/10/10 split.

The second stage consisted of 3 parts: the tree health classifier model and 2 regressor models for bole height and DBH individually. The 2 regressors could be combined into one model with 2 outputs, however, during early testing, the 2 output model had greatly diminished performance compared to the individual regressors.

Tree Health Classifier

The classifier was written in Python using the TensorFlow package. The architecture uses a ResNetRS101 backbone, which is an improved version of ResNet101 proposed in 2021 by Bello et al. (2021). The backbone is loaded with pretrained weights from training on the ImageNet dataset (Russakovsky et al. 2015). The classification head of the backbone was then removed and replaced with a pooling layer, followed by a fully connected layer, then a dropout layer and finally a softmax classification layer with outputs matching the number of classes. Additionally, a simple input augmentation layer was added before the backbone to enable on-the-fly random flip and rotation augmentation to help prevent early overfitting, given the small dataset size. The training loss function was weighted such that each class contributed a proportion that coincides with the class's representation in the dataset. Thus, classes with fewer samples effected the loss more than those with more. This helped ensure the model “paid attention” to classes with a smaller number of samples during training as the dataset is heavily unbalanced.

Multiple values were selected for each hyperparameter, attempting to cover a suitable range of potentially viable values for each and then a loop to test every combination of these hyperparameter values was executed, the results and training curves of each combination being logged to TensorBoard (a visualisation tool for model training logs). Thousands of tests were run, including around 1,440 different combinations and new hyperparameters were added as the performance trends were assessed throughout the process. Using TensorBoard, the results were assessed and the best performing configuration was selected.

Bole Height and Diameter at Breast Height Regressors

The architecture for both regressor models was almost identical to the classifier, except the pooling layer was flattened and, instead of a softmax classification layer at the output, a linear activation layer with a single output was used. Additionally, the loss weighting did not apply in this case, as there are no classes. The hyperparameters had to be tweaked manually because, due to the acquisition of the full dataset later than planned and then to some errors in the data set not found until very late in the project timeline, there was not enough time for the full hyperparameter evaluation process like that which was applied to

the classifier model. However, many of the findings from the classification model's testing were found to be transferable.

Chapter 5

Results

5.1 Stage 1: Individual Tree Detection and Segmentation Model

The final Overall Average Precision — also referred to as Mean Average Precision (mAP) — and per class Average Precision (AP) values are listed in Table 5.1 below. Normally in machine learning circles, AP would be the Area Under Curve of the Precision Recall (PR) curve at Intersection Over Union (IoU) = 0.5. However, for relevance and comparison, the Stage 1 model was evaluated using the Microsoft COCO (Common Object in Context) Challenge standards. This means the following AP values listed for this model refer to the average of APs over 10 IoU thresholds between 0.5 and 0.95. Additionally, the Overall AP at IoU 0.5 is included under AP^{50} for further comparison. For reference, the current leader on the MS COCO challenge leaderboard, listing innovative models trained on the MS COCO dataset, has an AP of 0.531 and AP^{50} of 0.768.

	Overall AP	Overall AP^{50}	AP: Sandalwood	AP: Senna	AP: Dalbergia
<i>Bounding Box</i>	0.424	0.712	0.419	0.461	0.392
<i>Segmentation</i>	0.448	0.720	0.436	0.504	0.402

Table 5.1: Bounding Box and Segmentation Test Results

The overall and per-class Precision Recall (PR) curves for various scenarios can be seen below in Figure 5.1, along with the corresponding Area Under PR Curves (AUPRCs)

in the legends, as per the MS COCO challenge standard. For reference, a completely random classifier would plot a horizontal line at the precision proportional to the number of positive examples within the dataset (0.5 for a class-balanced dataset) with an AUPRC of 0.5. The greater AUPRC, the better the model. The scenarios plotted in each PR curve are as follows. The first curve, coinciding with the first black line from the origin and the bottom legend entry named “C75”, is the PR curve at IoU 0.75. The next curve, “C50”, is the PR curve at IoU 0.5. Following (on the far side of the blue section, named “Loc”) is the PR curve with localisation errors removed, or in simple terms, at IoU 0.1; all other curves are also calculated at IoU 0.1. “Sim”, the next on the legend, is not relevant to this study, and removes detections where the subclass is wrong but the superclass is correct (the COCO data set has superclasses grouping similar classes together). The next curve, called “Oth”, is the PR curve after all class confusions are removed, in other words, if a detection is the wrong class but is still a valid object, it is no longer considered an error. The “BG” curve is the PR curve after all error associated with false positive detections on the background are also removed. Finally, “FN” is a trivial curve after all errors are removed which is simply a constant 1 on both axes.

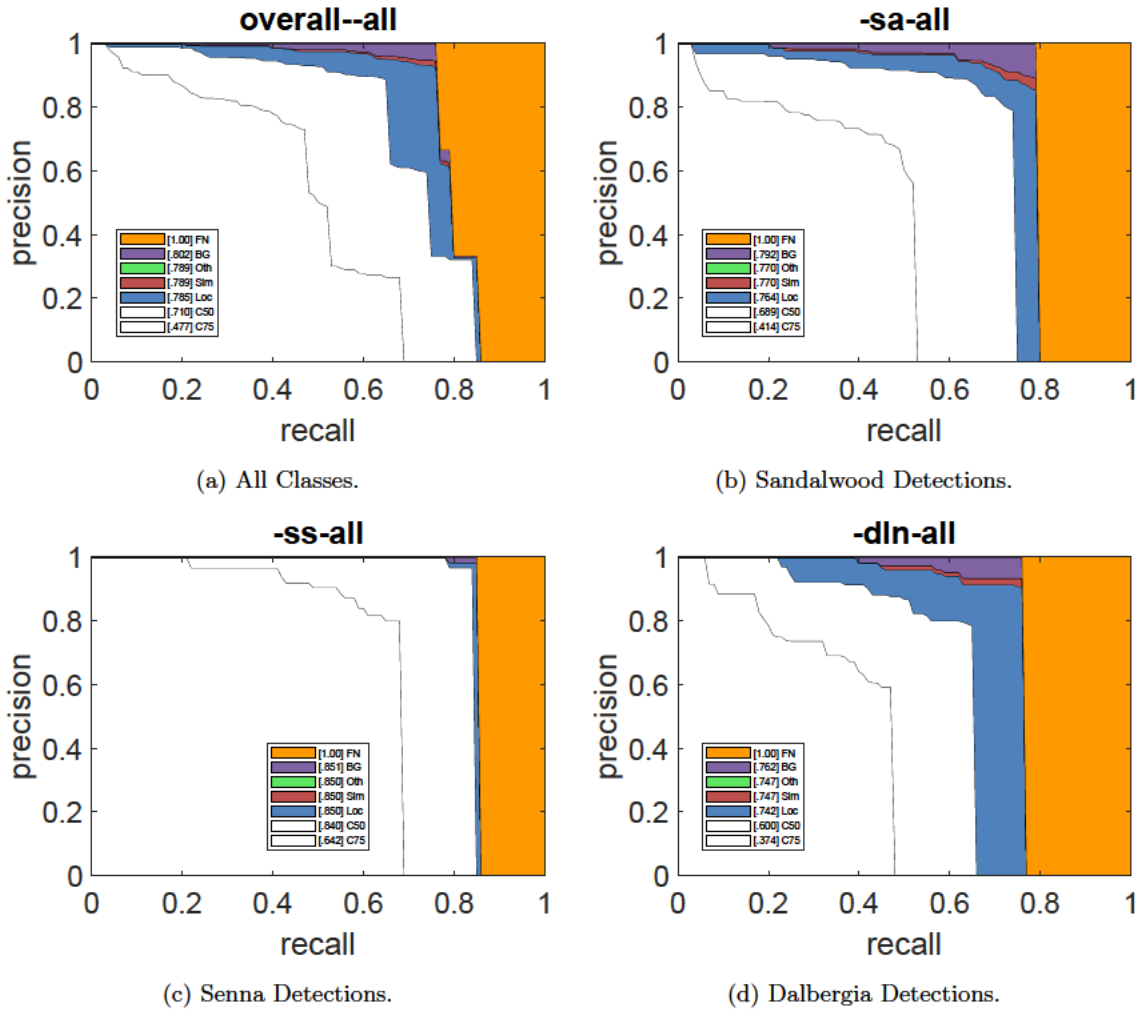


Figure 5.1: Evaluation Precision Recall Curves for Stage 1.

The confusion matrix at a confidence threshold of 0.7 and IoU of 0.5 for the Stage 1 model can be seen below in Figure 5.2. The “null” class denotes false detections on the background. In a perfect confusion matrix, predictions and ground truth are always the same, forming a diagonal line from the top left of the chart to the bottom right.

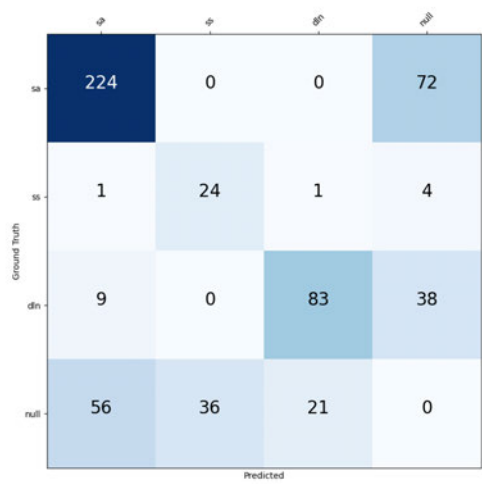
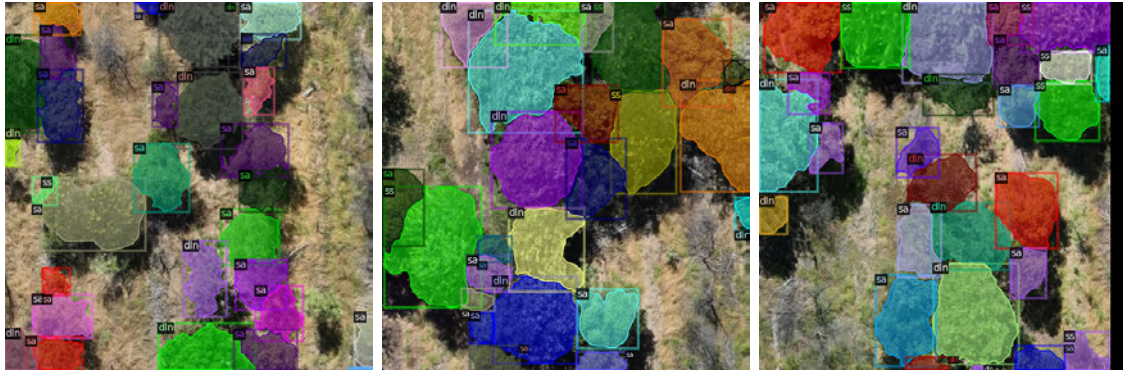
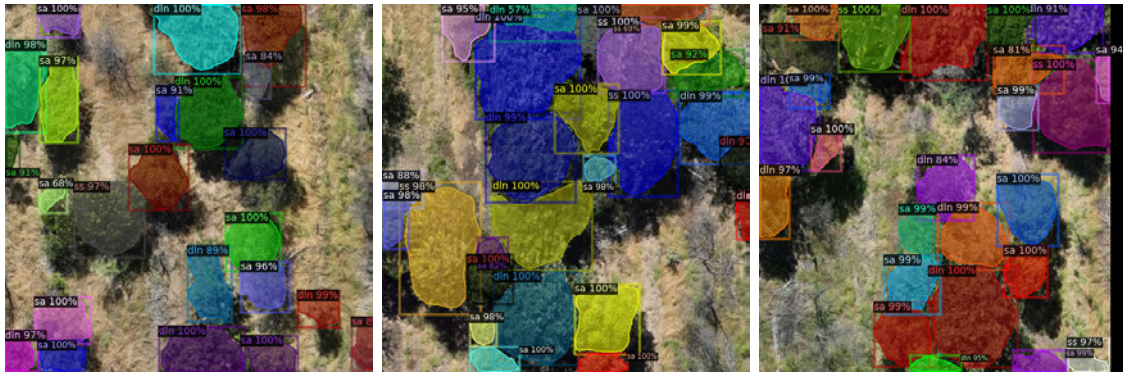


Figure 5.2: Confusion Matrix for The Final Model on Test Dataset

Below, in Figure 5.3, are three comparisons between the ground truth and the model's detections on some images from the test dataset.



(a) Ground Truth Annotations



(b) Model Detections

Figure 5.3: Examples of Stage 1 Detections

5.2 Stage 2: Individual Tree Parameter Prediction

5.2.1 Tree Health Classifier Model

Given that the heartwood estimation formula applied by Quintis is only impacted by classes 1 and 2 (classes 3-5, conversely, only having no negative effect on the output), the health classifier model was tested on both the full 5 class version and a condensed version where classes 3-5 are combined.

The final results of both models can be seen below in Table 5.2. The precision and recall values were calculated at a confidence threshold of 0.5. The AUPRC is calculated on the overall curve, considering all classes. The accuracy metric is a simple proportion

of correct predictions (true positives and true negatives). It should be noted that in the case of an unbalanced dataset, accuracy is typically considered a misleading metric for assessing a model’s performance alone due to the effect said unbalances have on the result. For example, if 90% of the data belongs to one class, a classifier that always predicts that class, no matter the input, will have an accuracy of 90%. The Precision, Recall and F_1 score give a more complete picture of a classifier’s performance. The precision metric measures the proportion of correct positive predictions; a high precision means the classifier is less likely to make false positive predictions, thus can be at risk of missing actual positive borderline cases. The recall metric is the proportion of actual positive class samples correctly identified by the model. Models with high recall can be at risk of incorrectly including borderline samples resembling positive, when they would, in actuality, be correctly classified as a negative. The F_1 score represents the harmonic mean of precision and recall. A perfect classifier would have a value of 1.0 for all of these metrics. The overall F_1 score listed is the “*sample-weighted F_1* ”, or the weighted average of the class-wise F_1 scores.

	Categorical Crossentropy Loss	Accuracy	Precision	Recall	AUC Precision Recall Curve	F_1 Score Overall (<i>weighted</i>)	Per Class F_1 Scores
5 Class Model	<i>Test</i>	0.25	0.50	0.65	0.27	0.56	0.86/0.22/0.31/0.57/0.53
	<i>Validation</i>	1.21	0.41	0.52	0.24	0.49	0.86/0.44/0.31/0.29/0.73
	<i>Training</i>	1.61./0.96	0.61	0.77	0.37	0.70	0.88/0.48/0.38/0.29/0.72
3 Class Model	<i>Test</i>	0.25	0.90	0.91	0.88	0.97	1/0.44/0.93
	<i>Validation</i>	0.29	0.89	0.91	0.87	0.96	0.86/0.55/0.93
	<i>Training</i>	0.99/0.29	0.88	0.88	0.87	0.96	0.93/0.53/0.93

Table 5.2: Results for test, validation and training datasets

The confusion matrices (CMs) for both classifiers can be seen below in Figure 5.4. Notice the absence of the “null” class seen in the Stage 1 detector’s CM as background detections do not apply to image classification.

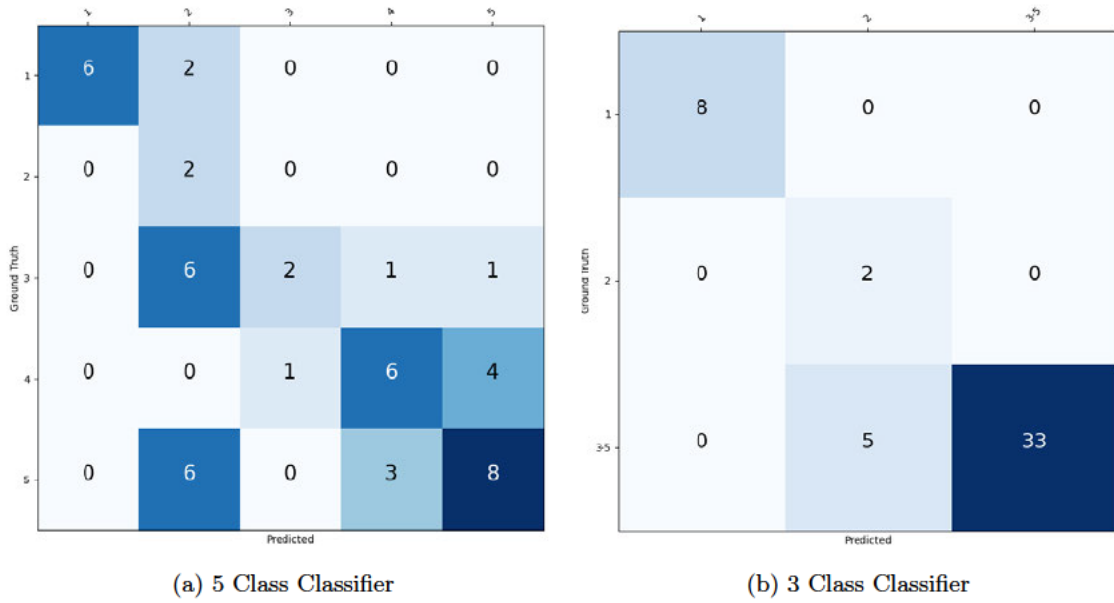


Figure 5.4: Confusion Matrices

The class-wise PR curves for both the 3 class and 5 class classifier models can be seen below in Figure 5.5. Note that, unlike in the Stage 1 detection model, IoU thresholds do not apply in this case, as each image is only classified, there is no bounding box or segmentation detection.

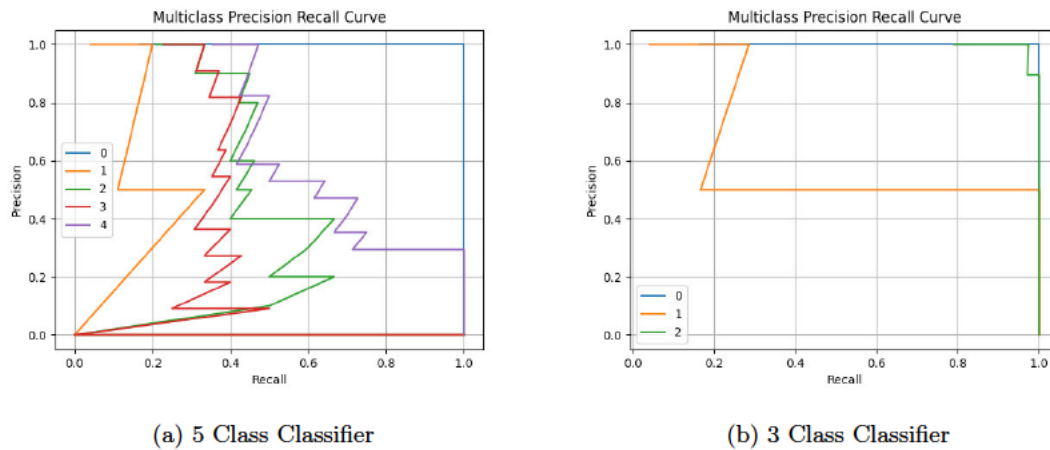


Figure 5.5: Precision/Recall Curves

In Figure 5.6 below are the Receiver Operating Characteristic (ROC) curves for the 5 class and 3 class classifier models. The ROC curve shows a model's performance over the full range of classification thresholds. A perfect ROC curve is a true positive rate of 1.0 across all false positive rates, and thus, an Area Under ROC curve (AUROC) of 1.0. A

completely random classifier would have a straight ROC curve, from coordinates (0,0) to (1,1) and, therefore, a AUROC of 0.5.

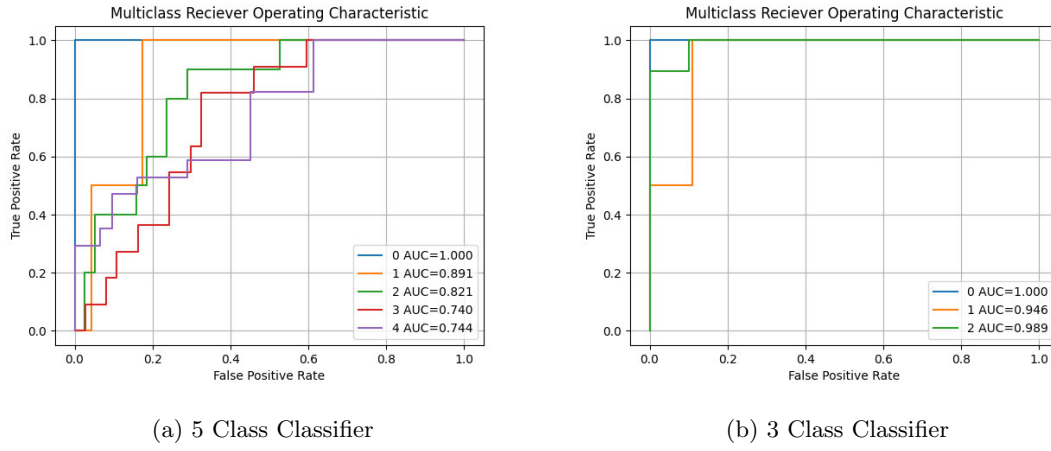


Figure 5.6: Receiver Operating Characteristic Curves

5.2.2 Regressor Models

The final results for the bole height (BH) and diameter at breast height (DBH) regressor models can be seen in Table 5.3 below. It may be noted that the R^2 values are negative. While this is not usually possible in statistical regression (where only values between 0 and 1 can occur), because of the way the metric is computed in this context, it can be less than zero, simply meaning the fit is arbitrarily worse than random.

		Mean Squared Error Loss	Root Mean Squared Error	Mean Absolute Error	R^2
Bole Height	<i>Test</i>	1.37	1.17	0.97	-1.81
	<i>Validation</i>	0.71	0.84	0.69	-1.21
	<i>Training</i>	1.16	1.08	0.87	-3.32
Diameter at Breast Height	<i>Test</i>	13.47	3.67	2.99	-0.34
	<i>Validation</i>	19.09	4.37	3.36	-1.46
	<i>Training</i>	13.50	3.67	2.75	-0.46

Table 5.3: Results for test, validation and training datasets

Below, in Figure 5.7, are the prediction results of both of the regressors on each of the

three datasets. For reference, a perfect regressor would plot in a straight line from the bottom left of the plot to the top right, each prediction value matching the corresponding ground truth value.

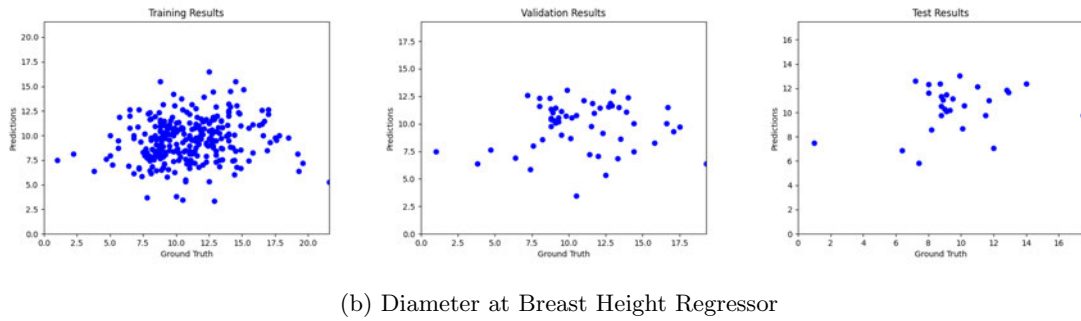
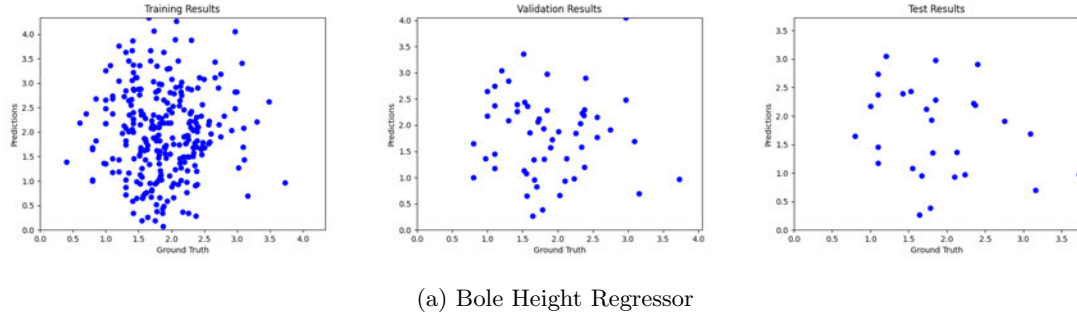


Figure 5.7: Regressor Predictions

5.3 Sources of Error

It should be noted that, while measures were taken to ensure a high degree of randomisation in the division of the training, validation and test datasets, as well as to ensure maximum model generalisation, there will always be the possibility of misrepresentation between the datasets. For example, if the validation and test datasets are sufficiently similar in terms of the features the model has learnt, the evaluation results from the test dataset would be higher than usual and would not accurately represent the model's performance on real-world data. This is particularly more likely when the dataset is relatively small, as is the case for the Stage 2 models.

Additionally, it is important to note that all images used in this study come from 2 plots in the same plantation in near Kununurra, Western Australia of plots that use the same host tree configuration and are approximately the same age. Furthermore, the images were taken with the same drone, on the same day, in the same season. While

Stage 1 model should not be sensitive to lighting, image quality or minor changes in camera parameters (ISO, exposure, etc.) thanks to the extensive use of augmentation, augmentation was not used as much in the Stage 2 models. All models may not perform as well on images from other plantations (where, for example, the canopy is much denser due to a kinder climate) or other seasons, as the trees and their canopies will appear differently. For example, in spring or times of higher rainfall, dead or dying trees (health status 1 or 2) can have a sudden growth of new, bright green leaves that will die off again when the season passes. Furthermore, the Senna host trees were flowering at the time the images were taken, making them easily identifiable by their bright yellow flowers. To mitigate these errors, the training datasets would have to be bolstered with images from a broad range of configurations, plantation ages, climates, seasons and locations.

After examining of the segmentation results from Stage 1, it became clear that there were errors present in the ground truth labelling. In these cases, even though the model has correctly identified a tree, if the tree is missing or incorrectly segmented in the “ground truth” data, the detection will count as an error and negatively impact the evaluation metrics. Furthermore, artefacts of the orthorectification and stitching process are present across the images, worsened by the high imaging altitude and resultant, lowered definition of the individual trees. These artefacts made the identification of individual trees difficult (and, in some cases, impossible), and, therefore, will have had an effect on the training and evaluation of the models.

The tree health assessments, while mostly completed by Dr. Precila Gonzales, were sometimes done by contractors given brief training by Dr. Gonzales. In cases where both Dr. Gonzales and the contractors had recorded their own health status assessments for a row, there were some discrepancies between the two versions. In these cases, Dr. Gonzales’ assessments were used for the final value, however, there are some rows where the assessments were only completed by the contractors. These rows were rigorously validated against the imagery to ensure there were no obvious mistakes, however, there is likely some inconsistency in these rows compared to those assessed by Dr. Gonzales, likely resulting in error.

Finally, for the training data of the Stage 2 models, trees that had been measured had to be manually identified in the orthomosaic and linked with their corresponding data. Initially, GPS coordinates were planned to be used to cross-reference the images with the data, however, the GPS module available on site was not sufficiently accurate (± 3 meters).

Therefore, the trees were identified using field notes taken by Dr Precila Gonzales and the measurements themselves. While the identifications were rigorously checked for consistency with the data, it is possible that some misidentifications or incorrect delineations between adjacent trees exist in the data.

Chapter 6

Discussion

6.1 Analysis of Results

6.1.1 Stage 1: Individual Tree Species Detector

The Stage 1 detector displayed exceptional performance for the given task. Achieving a bounding box mAP of 0.424 and an mAP^{50} of 0.712, the model had comparable performance to the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2017 winners (Russakovsky et al. 2015), who achieved an mAP^{50} of 0.731 (though, of course, on a different dataset with different classes). Moreover, Stage 1's bounding box mAP and mAP^{50} are also comparable to the current leading model in the ongoing, open COCO detection challenge (Lin et al. 2015), which achieved an mAP of 0.588 and an mAP^{50} of 0.766 (on the COCO dataset). Furthermore, the segmentation mAP and mAP^{50} achieved by Stage 1 (0.448 and 0.720) are also highly comparable to the COCO leaders at time of writing, which are 0.531 and 0.768 respectively.

It may be noted that the AP results were significantly higher for the Senna class. This is most likely due to the distinctive yellow flowers decorating the Senna trees in the imagery used for the study. Were the dataset to be extended to include example of the trees across the seasons, it would be expected that the APs for each class would become more similar.

The large locality error for Sandalwood and Dalbergia, visualised by the blue portion of the PR curves in Figure 5.1, is likely caused by labelling errors as, for those two species, it

is quite difficult to delineate between individual trees consistently. With careful auditing of the data's annotations this error should be reducible to something closer to the Senna class's, which is has much more accurate tree delineations in the current data annotations.

The confusion matrix in Figure 5.2 suggests that most of the error occurred due to false detections on the background. However, given the aforementioned annotation errors in the dataset, particularly where trees are missed, it is possible, perhaps even likely, that were these omissions to be corrected, the model may be performing better than it would appear in its current state. Moreover, if it were to be re-trained on the corrected dataset, a further performance increase may be observed.

Overall, the Stage 1 model performed well and is already at a stage where it would be highly useful with regard to the design objectives. Moreover, with some further error-correction and additions to the dataset, the model utility would be further increased.

6.1.2 Stage 2a: Tree Health Classifier

5 Class Version

When reviewing the 5 class version of the health classifier model's evaluation results, the first item to note is its success in classifying trees of health score 1 (H1) with an F_1 score of 0.86 and perfect ROC and PR curves. This is not surprising, as most of the trees of that class are distinctive in color and texture, being entirely dead. The remaining error in this class is most likely due to the epicormic branches that the trees tend to grow when dying as a final effort for survival. Though having some living, green leaves, foresters will still classify a tree as dead (H1) if the main stem has expired. Because these epicormic branches make the dead trees appear more similarly to those of a higher health status, the model would require more data to learn the differentiation more reliably. However, these results are comparable to those achieved for dead tree classification by the aforementioned Sani-Mohammed et al. (2022) and Yarak et al. (2021) studies.

The fact that the H2 class has the lowest F_1 score is not at all surprising, considering it is the most under-represented class, with only 44 instances in the entire dataset. The classification accuracy for this class would likely benefit from more instances to train on. Moreover, much like the case in H1, trees in this class also tend to grow epicormic

branches as they are in the process of dying, further increasing the need for more data to represent this class.

The remaining three classes have relatively low F_1 scores (though not as low as H2). Trees in these classes tend to appear very similar to their adjacent classes, which is the suspected reason for the lower scores. This theory is supported by the confusion matrix where it is observed that 3 trees from H5 were classified under H4 and 5 trees belonging to H4 were classified under either H5 or H3.

When reflecting on these results, it is also important to note that, due to health status being a qualitative classification, there are often border cases where even an experienced forester is uncertain whether a tree would be more appropriate for one health status or another. In these cases, either class could be considered correct. Therefore, it is possible that if the classification threshold were to be lowered some, and 2 class suggestions (granted are over the threshold) be allowed from the model, that the outputs may be useful. In this scenario, trees where the health status is obvious would still be given a single class and border cases would be given the 2 most likely options. Though further investigation would be required to validate this claim, the ROC curve may attest to its plausibility. The sharp incline of the problematic classes would suggest that if the classification threshold were to be reduced such that the false positive rate were around 0.4, the true positive rates for H2 would be 1.0, around 0.9 for H3 and H4 and around 0.7 for H5.

3 Class Version

It is immediately clear that the 3 class version was highly successful. With an AUPRC (which approximates to the AP) of 0.97 and F_1 score of 0.79, the model has performed especially well considering the small, unbalanced and challenging dataset. The fact that the classification performance on both H1 and H2 has increased compared to the 5 class model, even though the architecture and training, validation and testing datasets are identical, is an unexpected finding. The only conceivable explanation is that, in combining the latter 3 classes such that most of the class confusion was removed, the optimisation algorithm was able to better minimise the loss function and form more class separation by optimising important features without overfitting. The confusion matrix also attests to the anomaly. Where in the 5 class version 2 H1 trees were misclassified as H2 and

12 H3 and H5 trees misclassified as H2, in the 3 class version, those exact same trees were correctly classified at a much higher rate, with no misclassifications in the H1 class and only 5 trees from H3-H5 misclassified as H2. Moreover, the H1 classification results from this version significantly exceed the dead tree classification accuracy of the Sani-Mohammed et al. (2022) and Yarak et al. (2021) studies.

The F_1 score and PR curve for the H2 class suggest there is still some way to go before the model could be considered highly accurate for this class, however, this is not surprising due to the combination of the difficulty of the class and its gross underrepresentation. The ROC curve would seem to suggest that if the classification threshold were to be slightly reduced, such that false positives occur at a rate of around 10%, then the true positive rate would become 100%. This could mean the aforementioned 2 class suggestion arrangement for fringe cases could further extend the utility of this model as it stands.

6.1.3 Stage 2b: Tree Parameter Regressors

The evaluation results from both of the tree parameter regressors (for bole height and DBH) were regrettably inconclusive. While the models were able to reduce the error function somewhat, early in the training process, the R^2 values and the plotted predictions in Figure 5.7 do not show evidence of a fit. If anything, the models appear to have general range their respective tree parameters tend to reside within, and, particularly in the case of the DBH regressor, have begun to make predictions around the average value within the dataset.

While the results would indicate the model was unable to learn any relationship between the masked tree images and the bole height and DBH values, it is important to consider the size of the dataset. With only 277 data points in total (even fewer than the health classifier model) and considering the challenging nature of the task, it should not come as a surprise the models were not able to learn to predict with any suitable degree of accuracy. It is still possible that this method could still achieve a usable accuracy with the addition of more data.

6.2 Limitations of the Methods

The first and most notable limitation of the methodology applied in this study is the size of the datasets, particularly for the Stage 2 models. The small datasets make it difficult to be indisputably certain of the system's performance on real-world data. To mitigate this uncertainty somewhat (though not completely, as a larger dataset would be preferable), K-Fold cross validation could have been applied to the final models to average their test performance across the entire dataset. This would remove the error associated with any unintentional biases that may be present within the test subsets after the randomised subdivision of the dataset. However, due to the time constraints of the project, the K-Fold cross validation could not be implemented in time.

Secondly, very little augmentation was performed on the training data for the Stage 2 models. If more augmentation had been used there may have been an increase in model performance and certainly would have been an increase in generalisation. Regrettably, debugging for the Stage 2 models, particularly the regressors, continued until quite late in the project timeline. This was mostly due to some unit errors in the bole height data. As a result, the models weren't as polished as originally planned.

Finally, the Detectron2 framework that was applied for the Stage 1 model did not allow for as much customisation as the TensorFlow models or the original Mask RCNN Stage 1 model mentioned in the Pilot Studies section. Namely, regularisation weight decay could not be adjusted and dropout could not be applied at all. While the model clearly performed well without these features, the addition dropout in particular would increase the model's generalisation and robustness.

6.3 Limitations of the Study

The main limitation of this study, aside from those stemming from the methodology, is the narrow dataset. The fact that all data points came from 2 plantations and were recorded at around the same time of year means the results can't describe with certainty how the models would perform on data from other seasons or plantations. Moreover, the datasets for the Stage 2 regressor models were extremely small, making it difficult to draw meaningful conclusions from their evaluation results.

6.4 Significance of the Study

The performance of the tree detection model and the health classifier are both at a stage where they would provide value to Quintis by tracking the counts of trees of the three species in the dataset and tracking the health of the sandalwood trees to an acceptable level of accuracy in plantations similar to those in the dataset. At the very least, the models could be applied in their current state to the rest of the Kingston Rest plantations.

The results are sufficient to conclude that, with the addition of more data representing trees from other locations, seasons and planting configurations, the tree detection model and the health classification model would be capable of even higher accuracy across the entirety of Quintis's Sandalwood plantations. Furthermore, it is conceivable that the architectures would be useful in plantations of other species as well and would only need a solid dataset from such plantations to achieve comparable performance.

Additionally, the datasets collected in this study are possibly the first of their kind and will be able to be used in further research in the area. They are already being used by Dr. Precila Gonzales for her own research. The segmentation dataset in particular is a suitable size for detection models to train on and could be added to other datasets as part of a larger project.

6.5 Further Work

The most obvious task to further the concept proposed in this study would be to expand the dataset. With more data, the existing models could be re-trained and will certainly achieve higher accuracy.

Furthermore, as alluded to by the literature, the NDVI (or other vegetation indexes) and DCM (Digital Canopy Map) could be added to the input data as additional bands to further increase the performance of the models, particularly the Stage 2 models. This was originally planned as part of the study, however, had to be excluded due to time restraints. The drones used for the imagery are also capable of recording several hyperspectral bands which could also increase performance. NIR in particular has been found to be useful in vegetation health assessment and is used in some vegetation indexes. These extra features

can be calculated directly from the existing imagery and would require no additional measurements or data. The addition of augmented data to the training set would also further increase the performance of the model but had to be omitted due to lack of time.

It would also be worth researching the impact of loading the tree parameter regressor model with the weights from the tree health classifier before training. It is conceivable that the regressor could make use of the sandalwood (with tree health) specific features learned by the classifier to predict the bole height and DBH accurately.

Finally, in 2021 a new and unique form of CNN architecture was proposed, called the Vision Transformer, or ViT (Dosovitskiy et al. 2021). While there are hardly any papers testing this architecture in any applications yet, the architecture grants CNNs a new ability: attention to global image context. Based on the highly successful architecture behind GTP, the architecture has shown improved performance over other CNN architectures. At time of writing the tools and pretrained weights for the application of such an architecture were limited, however, it would likely be worth investigating their application in this system as future work.

Chapter 7

Conclusion

The preceding chapters 1 through 6 describe the development and evaluation process of a deep learning based method for the automatic detection of trees within Sandalwood plantations, classification of Sandalwood trees into 5 health status classes and prediction of the bole height and DBH of the same trees. After a survey of the literature, an architecture was proposed, informed by the findings of previous research. A novel dataset was collected, collated and annotated for the purpose of the study. The architecture was then implemented and critically evaluated on a previously unseen test dataset. The results prove the system to be effective for the detection and classification of trees within the plantation, as well as for the classification of Sandalwood tree by their health status. With the addition of more representative data, the system would be suitable for application to any sandalwood plantation. Regrettably, the results from the bole height and DBH regressor components were inconclusive due to the small dataset. Aside from the addition of more data, recommendations to add vegetation index and DCM bands to the input images, as well as to load the regressors with the pretrained weights from the health classifier and add augmentation to the Stage 2 models were made as further work to increase the accuracy and utility of the system without violating the design objectives. Overall, the proposed system addressed all research questions and, with exception to the further work required on the regressor models, met all design objectives by greatly increasing the utility of Quintis' existing drone imagery and showcasing the use of deep learning for automated tree counting and health monitoring, contributing to the full automation of the inventory process.

References

- Abdollahnejad, A. & Panagiotidis, D. (2020), ‘Tree Species Classification and Health Status Assessment for a Mixed Broadleaf-Conifer Forest with UAS Multispectral Imaging’, *Remote Sensing* **12**(22), 3722.
- Abdulla, W. (2017), ‘Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow’.
- An, J., Li, W., Li, M., Cui, S. & Yue, H. (2019), ‘Identification and Classification of Maize Drought Stress Using Deep Convolutional Neural Network’, *Symmetry* **11**(2), 256.
- Anon, D. & Chittapur, B. M. (2021), ‘Sandalwood Plantations – Points to Ponder’, *Current Science* **120**(7), 1184.
- Aslam, Y. & N, S. (2019), A Review of Deep Learning Approaches for Image Analysis, *in* ‘2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)’, pp. 709–714.
- Bechberger, L. (2020), ‘What are “Convolutional Neural Networks”? – Lucas Bechberger’s Website’, <https://lucas-bechberger.de/2020/11/12/what-are-convolutional-neural-networks/>.
- Bello, I., Fedus, W., Du, X., Cubuk, E. D., Srinivas, A., Lin, T.-Y., Shlens, J. & Zoph, B. (2021), ‘Revisiting ResNets: Improved Training and Scaling Strategies’.
- Brand, J., Norris, L. & Dumbrell, I. (2012), ‘Estimated heartwood weights and oil concentrations within 16-year-old Indian sandalwood (*Santalum album*) trees planted near Kununurra, Western Australia’, *Australian Forestry* **75**, 225–232.
- Cover, T. & Hart, P. (1967), ‘Nearest neighbor pattern classification’, *IEEE Transactions on Information Theory* **13**(1), 21–27.

- Das, S. C. (2021), Silviculture, Growth and Yield of Sandalwood, *in* T. Pullaiah, S. C. Das, V. A. Bapat, M. K. Swamy, V. D. Reddy & K. S. R. Murthy, eds, ‘Sandalwood: Silviculture, Conservation and Applications’, Springer, Singapore, pp. 111–138.
- Dash, J. P., Watt, M. S., Pearse, G. D., Heaphy, M. & Dungey, H. S. (2017), ‘Assessing very high resolution UAV imagery for monitoring forest health during a simulated disease outbreak’, *ISPRS Journal of Photogrammetry and Remote Sensing* **131**, 1–14.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. & Houlsby, N. (2021), ‘An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale’.
- Durkee, M. S., Abraham, R., Ai, J., Fuhrman, J. D., Clark, M. R. & Giger, M. L. (2021), Comparing Mask R-CNN and U-Net architectures for robust automatic segmentation of immune cells in immunofluorescence images of Lupus Nephritis biopsies, *in* ‘Imaging, Manipulation, and Analysis of Biomolecules, Cells, and Tissues XIX’, Vol. 11647, SPIE, pp. 109–115.
- Ecke, S., Dempewolf, J., Frey, J., Schwaller, A., Endres, E., Klemmt, H.-J., Tiede, D. & Seifert, T. (2022), ‘UAV-Based Forest Health Monitoring: A Systematic Review’, *Remote Sensing* **14**(13), 3205.
- Ege, T. & Yanai, K. (2018), ‘Image-Based Food Calorie Estimation Using Recipe Information’, *IEICE Transactions on Information and Systems* **E101.D**(5), 1333–1341.
- FPC: Sandalwood Establishment Guide* (2020), <https://www.wa.gov.au/government/publications/fpc-sandalwood-establishment-guide>.
- Fukushima, K. (1980), ‘Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position’, *Biological Cybernetics* **36**(4), 193–202.
- Girshick, R., Donahue, J., Darrell, T. & Malik, J. (2013), ‘Rich feature hierarchies for accurate object detection and semantic segmentation’, <https://arxiv.org/abs/1311.2524v5>.
- Global And United States Sandalwood Market Insights, Forecast To 2027* (2021), Global Market Report QYR-18683557, Industry Research.

- Guerra-Hernández, J., Díaz-Varela, R. A., Álvarez-González, J. G. & Rodríguez-González, P. M. (2021), ‘Assessing a novel modelling approach with high resolution UAV imagery for monitoring health status in priority riparian forests’, *Forest Ecosystems* **8**, 61.
- Gupta, A., Dollár, P. & Girshick, R. (2019), ‘LVIS: A Dataset for Large Vocabulary Instance Segmentation’.
- He, K., Gkioxari, G., Dollár, P. & Girshick, R. (2018), ‘Mask R-CNN’.
- Hebb, D. O. (1949), *The Organization of Behavior; a Neuropsychological Theory*, The Organization of Behavior; a Neuropsychological Theory, Wiley, Oxford, England.
- Ho, T. K. (1995), Random decision forests, in ‘Proceedings of 3rd International Conference on Document Analysis and Recognition’, Vol. 1, pp. 278–282 vol.1.
- Huang, G., Liu, Z., van der Maaten, L. & Weinberger, K. Q. (2018), ‘Densely Connected Convolutional Networks’.
- Iizuka, K., Kosugi, Y., Noguchi, S. & Iwagami, S. (2022), ‘Toward a comprehensive model for estimating diameter at breast height of Japanese cypress (*Chamaecyparis obtusa*) using crown size derived from unmanned aerial systems’, *Computers and Electronics in Agriculture* **192**, 106579.
- Ivakhnenko, A. G. (1967), *Cybernetics and Forecasting Techniques*, American Elsevier Pub. Co., New York.
- Kingma, D. P. & Ba, J. (2017), ‘Adam: A Method for Stochastic Optimization’.
- Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollár, P. & Girshick, R. (2023), ‘Segment Anything’.
- Köhl, M. (2004), ‘INVENTORY — Forest Inventory and Monitoring’, *Encyclopedia of Forest Sciences* pp. 403–409.
- Krug, T. & dos Santos, J. (2004), ‘RESOURCE ASSESSMENT — Forest Change’, *Encyclopedia of Forest Sciences* pp. 989–997.
- Lawson, S. A., McDonald, J. M. & Pegg, G. S. (2008), ‘Forest health surveillance methodology in hardwood plantations in Queensland, Australia’, *Australian Forestry* **71**(3), 177–181.

- Li, Y. (2022), Research and Application of Deep Learning in Image Recognition, *in* ‘2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA)’, pp. 994–999.
- Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L. & Dollár, P. (2015), ‘Microsoft COCO: Common Objects in Context’.
- Mali, S. (2023), Sandalwood Market Report 2023 (Global Edition), Global Market Report CMR487184, Cognitive Market Research.
- McCulloch, W. S. & Pitts, W. (1943), ‘A logical calculus of the ideas immanent in nervous activity’, *The bulletin of mathematical biophysics* **5**(4), 115–133.
- Muchoney, D. & Haack, B. N. (1994), ‘Change Detection for Monitoring Forest Defoliation’, *Photogrammetric Engineering & Remote Sensin* **60**(10), 1243–1251.
- Myers, A., Johnston, N., Rathod, V., Korattikara, A., Gorban, A., Silberman, N., Guadarrama, S., Papandreou, G., Huang, J. & Murphy, K. (2015), Im2Calories: Towards an Automated Mobile Vision Food Diary, *in* ‘2015 IEEE International Conference on Computer Vision (ICCV)’, IEEE, Santiago, Chile, pp. 1233–1241.
- OpenDroneMap Authors (2023), ‘ODM - A command line toolkit to generate maps, point clouds, 3D models and DEMs from drone, balloon or kite images.’.
- Pertille, C., Nicoletti, M. & Jr, M. (2023), ‘Estimating the commercial volume of a Pinus taeda L. plantation using active and passive sensors’, *CERNE* **29**.
- Popescu, S. C., Wynne, R. H. & Nelson, R. F. (2003), ‘Measuring individual tree crown diameter with lidar and assessing its influence on estimating forest volume and biomass’, *Canadian Journal of Remote Sensing* **29**(5), 564–577.
- Rahaman, M., Li, C., Yao, Y., Kulwa, F., Rahman, M., Wang, Q., Qi, S., Kong, F., Zhu, X. & Zhao, X. (2020), ‘Identification of COVID-19 samples from chest X-Ray images using deep learning: A comparison of transfer learning approaches’, *Journal of X-ray science and technology* **28**.
- Ramos-Giraldo, P., Reberg-Horton, C., Locke, A. M., Mirsky, S. & Lobaton, E. (2020), ‘Drought Stress Detection Using Low-Cost Computer Vision Systems and Machine Learning Techniques’, *IT Professional* **22**(3), 27–29.

- Redfern, D. B. & Boswell, R. C. (2004), 'Assessment of crown condition in forest trees: Comparison of methods, sources of variation and observer bias', *Forest Ecology and Management* **188**(1), 149–160.
- Ronneberger, O., Fischer, P. & Brox, T. (2015), 'U-Net: Convolutional Networks for Biomedical Image Segmentation'.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. & Fei-Fei, L. (2015), 'ImageNet large scale visual recognition challenge', *International Journal of Computer Vision (IJCV)* **115**(3), 211–252.
- Sani-Mohammed, A., Yao, W. & Heurich, M. (2022), 'Instance segmentation of standing dead trees in dense forest from aerial imagery using deep learning', *ISPRS Open Journal of Photogrammetry and Remote Sensing* **6**, 100024.
- Schapire, R. E. (1990), 'The strength of weak learnability', *Machine Learning* **5**(2), 197–227.
- Smith, D., Smith, I., Collett, N. & Elms, S. (2008), 'Forest health surveillance in Victoria', *Australian Forestry* **71**(3), 188–195.
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. (2015), 'Rethinking the Inception Architecture for Computer Vision'.
- Tanase, M. A., Panciera, R., Lowell, K., Aponte, C., Hacker, J. M. & Walker, J. P. (2014), 'Forest Biomass Estimation at High Spatial Resolution: Radar Versus Lidar Sensors', *IEEE Geoscience and Remote Sensing Letters* **11**(3), 711–715.
- Wang, J., Sammis, T. W., Gutschick, V. P., Gebremichael, M., Dennis, S. O. & Harrison, R. E. (2010), 'Review of Satellite Remote Sensing Use in Forest Health Studies~!2010-01-27~!2010-04-05~!2010-06-29~!', *The Open Geography Journal* **3**(1), 28–42.
- Widyaningrum, R., Candradewi, I., Aji, N. R. A. S. & Aulianisa, R. (2022), 'Comparison of Multi-Label U-Net and Mask R-CNN for panoramic radiograph segmentation to detect periodontitis', *Imaging Science in Dentistry* **52**(4), 383–391.
- Windrim, L., Carnegie, A. J., Webster, M. & Bryson, M. (2020), 'Tree detection and health monitoring in multispectral aerial imagery and photogrammetric pointclouds

using machine learning’, *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* **13**, 2554–2572.

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y. & Girshick, R. (2019), ‘Detectron2’.

Yarak, K., Witayangkurn, A., Kritiyutanont, K., Arunplod, C. & Shibasaki, R. (2021), ‘Oil palm tree detection and health classification on high-resolution imagery using deep learning’, *Agriculture (Switzerland)* **11**(2), 1–17.

Zeng, Y., Hao, D., Huete, A., Dechant, B., Berry, J., Chen, J. M., Joiner, J., Frankenberg, C., Bond-Lamberty, B., Ryu, Y., Xiao, J., Asrar, G. R. & Chen, M. (2022), ‘Optical vegetation indices for monitoring terrestrial ecosystems globally’, *Nature Reviews Earth & Environment* **3**(7), 477–493.

Şandric, I., Irimia, R., Petropoulos, G. P., Anand, A., Srivastava, P. K., Pleşoianu, A., Faraslis, I., Stateras, D. & Kalivas, D. (2022), ‘Tree’s detection & health’s assessment from ultra-high resolution UAV imagery and deep learning’, *Geocarto International* **37**(25), 10459–10479.

Appendix A

Project Specification

ENG4111/ENG4112 Research Project
Project Specification

For: Isaac Humber

Title: A Deep Learning Solution for the Detection of Health and Productivity Metrics in Sandalwood Forest Plantations Using Drone Imaging

Major: Computer Engineering

Supervisor: Dr. Tobias Low

Enrolment: ENG4111 - EXT S1, 2023
ENG4112 - EXT S2, 2023

Project Aim: To develop a machine learning model for detecting general health, bole height and DBH in Sandalwood trees using drone multi-spectral imaging.

Programme: Version 2, 21th September 2023

1. Research existing methods for similar use-cases and determine promising machine learning models to test.
2. Collect and annotate aerial images of trees in the forest plantations, labelled with each tree health, bole height and DBH (as determined by Quintis staff using appropriate instruments) and location.
3. Review tree detection methods and develop code to detect trees within images and link each tree to data.
4. Develop and train machine learning models in Python using extracted features.
5. Review suitable model accuracy metrics and model comparison techniques and implement them in code.

If time and resources permit:

6. Refine the most promising model using advanced techniques found or inspired by those within the literature body
7. Develop code to create a visual representation of individual tree health over a large geographical area.

Project Resources

Table 1: Anticipated Resource Requirements

Item	Source	Cost	Comment
Personal Computer with Python installed	Student	nil	
Labelled dataset	Quintis	nil	1,000 or more data points
Machine learning and image processing Python packages	Web or package manager download	nil	
Visual Studio Code	https://code.visualstudio.com/	nil	For LaTeX and Python Development
Personal Computer with LaTeX installed	Student	nil	

Appendix B

Risk Assessment

Risk Assessment

Probability of Occurrences			Impact				
			Catastrophic	Critical	Moderate	Minor	Negligible
Definition	Meaning	Value	(A)	(B)	(C)	(D)	(E)
Frequent	<ul style="list-style-type: none"> Occurs frequently Will be continuously experienced unless action is taken to change events 	5	5A	5B	5C	5D	5E
Likely	<ul style="list-style-type: none"> Occurs less frequently if corrective action is taken Documented through surveillance 	4	4A	4B	4C	4D	4E
Occasional	<ul style="list-style-type: none"> Occurs sporadically Discovered through surveillance 	3	3A	3B	3C	3D	3E
Seldom	<ul style="list-style-type: none"> Unlikely to occur Rarely, if ever, reported 	2	2A	2B	2C	2D	2E
Improbable	<ul style="list-style-type: none"> Highly unlikely to occur Never previously reported 	1	1A	1B	1C	1D	1E

Risk Levels: Risk is High for codes 5A, 5B, 5C, 4A, 4B, 3A (in red); Risk is Medium High for codes 5D, 5E, 4C, 3B, 3C, 2A, 2B (in orange); Risk is Medium Low for codes 4D, 4E, 3D, 2C, 1A, 1B (in yellow); Risk is Low for codes 3E, 2D, 2E, 1C, 1D, 1E (in blue).

Figure 1: Risk Assessment Matrix

Table 2: Risk Assessment Matrix

Hazard	Minimisation	Risk Level Before	Risk Level After
Posture-related injuries from computer use	Take hourly breaks and use an ergonomic workstation.	5D	2D
Eye strain from computer use	Work in a well-lit area with monitors in an ergonomic layout	5D	2D
Dataset not available when needed	Request Quintis to send portions of the dataset as they are recorded to enable earlier design using smaller subsets of the data	3C	2C
Unhealthy stress levels	Stay strictly on or ahead of schedule. Communicate progress and issues with supervisor.	4C	2C
Loss of data	Backup all data on cloud service (OneDrive). Use correct storage media protocol.	2A	1A

Appendix C

Ethical Clearance

There are no Ethical Clearances applicable to this project.

Appendix D

Data

D.1 Introduction to this Appendix

The raw data as collected from various plantations and locations across Quintis' Western Australia plantations. Data collection and drone imaging is still ongoing.

D.2 Raw Data

Field Tree Number	Row	HEALTH STATUS	DBH	BOLE HEIGHT	TRIAL
1	65	5	11.4	2.335	KR1_MULCH_R1
2	65	2	10.5	1.510	KR1_MULCH_R1
3	65	3	12.3	1.415	KR1_MULCH_R1
4	65	1	0.0	0.0	KR1_MULCH_R1
5	65	1	0.0	0.0	KR1_MULCH_R1
6	65	3	9.2	1.400	KR1_MULCH_R1
7	65	5	19.3	1.41	KR1_MULCH_R1
8	65	5	15.5	1.39	KR1_MULCH_R1
9	65	1	0.0	0.0	KR1_MULCH_R1
10	65	5	17.6	1.400	KR1_MULCH_R1
11	65	5	15.8	1.30	KR1_MULCH_R1
12	65	3	11.6	1.35	KR1_MULCH_R1
1	69	3	11.0	1.66	KR1_MULCH_R1
2	69	5	12.3	1.28	KR1_MULCH_R1
3	69	4	11.6	1.36	KR1_MULCH_R1
4	69	2	9.5	1.28	KR1_MULCH_R1
5	69	4	19.2	1.47	KR1_MULCH_R1
6	69	3	11.1	1.875	KR1_MULCH_R1
7	69	5	16.6	2.385	KR1_MULCH_R1
8	69	3	10.2	1.420	KR1_MULCH_R1
9	69	4	11.0	1.520	KR1_MULCH_R1
10	69	5	19.6	1.530	KR1_MULCH_R1
2	73	3	10.7	1.80	KR1_MULCH_R1
3	73	3	12.9	1.410	KR1_MULCH_R1
4	73	4	12.3	2.255	KR1_MULCH_R1
5	73	3	10.0	1.975	KR1_MULCH_R1
6	73	3	14.4	2.070	KR1_MULCH_R1
7	73	3	7.7	1.380	KR1_MULCH_R1
9	73	4	17.8	1.560	KR1_MULCH_R1
10	73	4	11.9	2.97	KR1_MULCH_R1
11	73	5	10.5	2.135	KR1_MULCH_R1
12	73	5	16.7	2.065	KR1_MULCH_R1
1	129	4	8.8	2.97	KR1_MULCH_R2
2	129	5	13.4	2.25	KR1_MULCH_R2
3	129	5	9.8	2.325	KR1_MULCH_R2
4	129	5	14.0	1.775	KR1_MULCH_R2
5	129	5	10.7	1.600	KR1_MULCH_R2
6	129	5	21.6	1.250	KR1_MULCH_R2
7	129	1	0.0	0.0	KR1_MULCH_R2
8	129	3	9.3	1.780	KR1_MULCH_R2
9	129	5	8.3	1.68	KR1_MULCH_R2
10	129	5	13.3	1.535	KR1_MULCH_R2
11	129	1	0.0	0.0	KR1_MULCH_R2
1	133	2	14.4	0.850	KR1_MULCH_R2
2	133	3	17.1	1.070	KR1_MULCH_R2
3	133	2	10.5	1.510	KR1_MULCH_R2
4	133	3	7.8	1.735	KR1_MULCH_R2
5	133	2	11.5	2.130	KR1_MULCH_R2
6	133	5	12.4	2.995	KR1_MULCH_R2
8	133	5	13.4	2.030	KR1_MULCH_R2
9	133	5	12.6	2.010	KR1_MULCH_R2
10	133	2	7.6	1.350	KR1_MULCH_R2
11	133	4	11.0	0.970	KR1_MULCH_R2
12	133	5	18.5	1.57	KR1_MULCH_R2
1	137	3	8.9	0.85	KR1_MULCH_R2
2	137	5	10.3	2.58	KR1_MULCH_R2
3	137	4	9.0	1.3	KR1_MULCH_R2
4	137	5	9.1	2.1	KR1_MULCH_R2
5	137	5	12.5	1.9	KR1_MULCH_R2
6	137	1	7.0		KR1_MULCH_R2
7	137	5	13.8	3.3	KR1_MULCH_R2
9	137	4	9.3	1.65	KR1_MULCH_R2
11	137	5	8.5	1.3	KR1_MULCH_R2
12	137	5	11.3	2.14	KR1_MULCH_R2
4	188	2		0.5	KR1_MULCH_R3
5	188	5	9.1	1.58	KR1_MULCH_R3
6	188	4	14.0	1.93	KR1_MULCH_R3
7	188	3	8.8	1.51	KR1_MULCH_R3
8	188	1			KR1_MULCH_R3
9	188	2	12.7	1.78	KR1_MULCH_R3
10	188	2	5.0	0.8	KR1_MULCH_R3
11	188	5	13.4	2.56	KR1_MULCH_R3
12	188	1	5.0		KR1_MULCH_R3
13	188	1	7.0		KR1_MULCH_R3
14	188	4	10.5	1.77	KR1_MULCH_R3
15	188	1	4.5		KR1_MULCH_R3
16	188	5	11.0	1.3	KR1_MULCH_R3
17	188	5	11.0	2.23	KR1_MULCH_R3
1	192	5	15.1	2.38	KR1_MULCH_R3
2	192	5	9.8	1.82	KR1_MULCH_R3
3	192	5	10.7	2.48	KR1_MULCH_R3
4	192	5	12.1	2.18	KR1_MULCH_R3
5	192	4	9.8	2.5	KR1_MULCH_R3
6	192	1	6.1		KR1_MULCH_R3
7	192	2	8.2	1.42	KR1_MULCH_R3
8	192	5	12.7	2.72	KR1_MULCH_R3
9	192	5	7.2	1.0	KR1_MULCH_R3
10	192	4	11.4	2.56	KR1_MULCH_R3
11	192	5	14.0	2.6	KR1_MULCH_R3
12	192	1			KR1_MULCH_R3
1	196	5	8.2	1.89	KR1_MULCH_R3
2	196	2	12.5	1.6	KR1_MULCH_R3
3	196	4	11.5	1.52	KR1_MULCH_R3

4	196	4	13.0	2.03	KR1.MULCH_R3
5	196	4	9.2	2.35	KR1.MULCH_R3
6	196	3	8.0	1.51	KR1.MULCH_R3
7	196	4	17.0	3.07	KR1.MULCH_R3
8	196	4	9.2	2.33	KR1.MULCH_R3
9	196	4	8.1	1.47	KR1.MULCH_R3
10	196	4	9.8	1.6	KR1.MULCH_R3
1	10	2	5.0	0.8	KR1.BIOCHAR_R1
2	10	1	7.7	1.1	KR1.BIOCHAR_R1
3	10	5	8.0	1.64	KR1.BIOCHAR_R1
4	10	4	11.5	2.26	KR1.BIOCHAR_R1
5	10	4	1.0	2.35	KR1.BIOCHAR_R1
6	10	3	9.5	1.81	KR1.BIOCHAR_R1
7	10	4	11.7	2.21	KR1.BIOCHAR_R1
8	10	1	8.0	1.68	KR1.BIOCHAR_R1
9	10	2	2.3		KR1.BIOCHAR_R1
10	10	4	8.4	1.7	KR1.BIOCHAR_R1
11	10	2	3.1		KR1.BIOCHAR_R1
12	10	5	16.5	3.48	KR1.BIOCHAR_R1
13	10	4	8.8	2.4	KR1.BIOCHAR_R1
14	10	5	11.6	2.39	KR1.BIOCHAR_R1
1	14	5	12.8	2.28	KR1.BIOCHAR_R1
2	14	5	14.1	3.1	KR1.BIOCHAR_R1
3	14	5	10.0	2.19	KR1.BIOCHAR_R1
6	14	1	3.5	0.4	KR1.BIOCHAR_R1
7	14	5	12.1	3.16	KR1.BIOCHAR_R1
8	14	5	6.5	1.88	KR1.BIOCHAR_R1
9	14	5	11.7	2.41	KR1.BIOCHAR_R1
11	14	2	11.9	1.8	KR1.BIOCHAR_R1
12	14	2	8.3	2.28	KR1.BIOCHAR_R1
13	14	1	7.0	0.9	KR1.BIOCHAR_R1
14	14	5	11.8	2.45	KR1.BIOCHAR_R1
1	18	2	6.8	1.0	KR1.BIOCHAR_R1
2	18	2	11.0	1.68	KR1.BIOCHAR_R1
3	18	2	11.7	2.66	KR1.BIOCHAR_R1
4	18	1			KR1.BIOCHAR_R1
5	18	1	11	1.0	KR1.BIOCHAR_R1
6	18	5	10.4	1.5	KR1.BIOCHAR_R1
7	18	5	9.3	1.69	KR1.BIOCHAR_R1
8	18	2	2.2	1.0	KR1.BIOCHAR_R1
9	18	3	5.2	0.8	KR1.BIOCHAR_R1
10	18	5	8.1	1.85	KR1.BIOCHAR_R1
11	18	2	7.8	1.3	KR1.BIOCHAR_R1
12	18	4	7.1	1.1	KR1.BIOCHAR_R1
1	22	5	11.9	2.15	KR1.BIOCHAR_R1
3	22	5	8.0	1.47	KR1.BIOCHAR_R1
4	22	5	8.8	1.88	KR1.BIOCHAR_R1
5	22	5	8.1	1.76	KR1.BIOCHAR_R1
6	22	1	4.5	0.9	KR1.BIOCHAR_R1
7	22	5	8.5	1.75	KR1.BIOCHAR_R1
8	22	5	8.8	1.84	KR1.BIOCHAR_R1
9	22	5	13.0	2.38	KR1.BIOCHAR_R1
10	22	1	8.0	1.0	KR1.BIOCHAR_R1
11	22	5	10.2	1.83	KR1.BIOCHAR_R1
12	22	1	9.6	1.42	KR1.BIOCHAR_R1
1	26	4	8.4	1.85	KR1.BIOCHAR_R1
2	26	2	8.8	1.71	KR1.BIOCHAR_R1
3	26	2	9.4	1.75	KR1.BIOCHAR_R1
4	26	1			KR1.BIOCHAR_R1
5	26	1	6.4	1.1	KR1.BIOCHAR_R1
6	26	5	17.0	2.66	KR1.BIOCHAR_R1
7	26	5	10.4	1.97	KR1.BIOCHAR_R1
8	26	1	2.0		KR1.BIOCHAR_R1
9	26	5	14.0	2.23	KR1.BIOCHAR_R1
10	26	5	8.7	1.93	KR1.BIOCHAR_R1
11	26	5	8.8	1.53	KR1.BIOCHAR_R1
12	26	2	7.2	1.2	KR1.BIOCHAR_R1
13	26	5	13.0	2.47	KR1.BIOCHAR_R1
1	30	5	9.6	1.69	KR1.BIOCHAR_R1
2	30	5	17.5	3.26	KR1.BIOCHAR_R1
3	30	4	6.4	0.8	KR1.BIOCHAR_R1
5	30	5	12.0	1.54	KR1.BIOCHAR_R1
6	30	5	9.5	1.93	KR1.BIOCHAR_R1
7	30	3	8.8	1.42	KR1.BIOCHAR_R1
8	30	1	4.4	1.2	KR1.BIOCHAR_R1
9	30	5	12.8	2.24	KR1.BIOCHAR_R1
10	30	5	7.9	1.73	KR1.BIOCHAR_R1
11	30	5	8.4	1.81	KR1.BIOCHAR_R1
1	34	5	11.0	1.3	KR1.BIOCHAR_R1
2	34	5	14.8	2.75	KR1.BIOCHAR_R1
3	34	5	14.5	2.35	KR1.BIOCHAR_R1
4	34	5	9.9	1.86	KR1.BIOCHAR_R1
6	34	1	12.6	1.7	KR1.BIOCHAR_R1
7	34	3	13.0	2.0	KR1.BIOCHAR_R1
8	34	1	10.8	1.62	KR1.BIOCHAR_R1
9	34	1	7.3	1.64	KR1.BIOCHAR_R1
10	34	5	8.8	1.78	KR1.BIOCHAR_R1
11	34	2	9.8	1.35	KR1.BIOCHAR_R1
1	69	5	12.9	2.46	KR1.BIOCHAR_R2
2	69	4	10.6	2.49	KR1.BIOCHAR_R2
3	69	4	13.5	2.36	KR1.BIOCHAR_R2
5	69	5	12.7	1.8	KR1.BIOCHAR_R2
6	69	1	2.4		KR1.BIOCHAR_R2
7	69	4	10.5	1.96	KR1.BIOCHAR_R2
8	69	2	9.8	1.43	KR1.BIOCHAR_R2
9	69	1	10.3	1.8	KR1.BIOCHAR_R2
10	69	4	10	1.59	KR1.BIOCHAR_R2
11	69	4	9.0	1.72	KR1.BIOCHAR_R2
14	69	4	12.8	1.76	KR1.BIOCHAR_R2
2	73	5	8.5	1.7	KR1.BIOCHAR_R2
3	73	5	9.5	2.2	KR1.BIOCHAR_R2
4	73	5	9.7	2.28	KR1.BIOCHAR_R2
5	73	3	8.2	1.57	KR1.BIOCHAR_R2
6	73	1	4.9		KR1.BIOCHAR_R2
7	73	5	14.7	2.19	KR1.BIOCHAR_R2
8	73	1			KR1.BIOCHAR_R2
9	73	1	9.8		KR1.BIOCHAR_R2
10	73	1	13.6		KR1.BIOCHAR_R2
11	73	1	4.5		KR1.BIOCHAR_R2
12	73	2	16.5	2.25	KR1.BIOCHAR_R2
13	73	1	3.0		KR1.BIOCHAR_R2
14	73	3	14.3	1.45	KR1.BIOCHAR_R2
15	73	2	5.6	0.6	KR1.BIOCHAR_R2
16	73	4	10.8	1.58	KR1.BIOCHAR_R2
2	81	3	8.4	1.35	KR1.BIOCHAR_R2
3	81	4	8.5	1.52	KR1.BIOCHAR_R2
4	81	4	6.8	1.0	KR1.BIOCHAR_R2
5	81	5	9.5	1.9	KR1.BIOCHAR_R2
6	81	5	14.0	2.3	KR1.BIOCHAR_R2

7	81	2	6.5	1.1	KR1.BIOCHAR_R2
8	81	5	10.7	1.88	KR1.BIOCHAR_R2
9	81	4	13.3	2.33	KR1.BIOCHAR_R2
10	81	2	8.0	1.3	KR1.BIOCHAR_R2
1	85	5	12.8	1.94	KR1.BIOCHAR_R2
2	85	5	9.8	2.04	KR1.BIOCHAR_R2
3	85	1	5.8	1.0	KR1.BIOCHAR_R2
4	85	5	11.5	2.23	KR1.BIOCHAR_R2
5	85	3	5.6	1.0	KR1.BIOCHAR_R2
6	85	5	7.3	1.42	KR1.BIOCHAR_R2
7	85	4	9.1	1.53	KR1.BIOCHAR_R2
8	85	5	11.2	1.92	KR1.BIOCHAR_R2
10	85	5	9.2	1.92	KR1.BIOCHAR_R2
11	85	5	12.1	1.81	KR1.BIOCHAR_R2
13	85	1	4.7		KR1.BIOCHAR_R2
14	85	2	8.0	1.3	KR1.BIOCHAR_R2
1	89	4	16.9	1.82	KR1.BIOCHAR_R2
2	89	5	6.5	1.1	KR1.BIOCHAR_R2
3	89	3	9.0	1.77	KR1.BIOCHAR_R2
4	89	4	11.2	2.12	KR1.BIOCHAR_R2
5	89	5	11.0	2.27	KR1.BIOCHAR_R2
6	89	5	14.0	2.42	KR1.BIOCHAR_R2
7	89	5	12.7	1.43	KR1.BIOCHAR_R2
8	89	5	12.5	2.0	KR1.BIOCHAR_R2
10	89	5	14.0	2.57	KR1.BIOCHAR_R2
1	93	1	10.0	1.6	KR1.BIOCHAR_R2
3	93	5	14.5	3.09	KR1.BIOCHAR_R2
4	93	4	8.4	1.67	KR1.BIOCHAR_R2
5	93	5	16.3	2.91	KR1.BIOCHAR_R2
6	93	5	14.9	3.73	KR1.BIOCHAR_R2
7	93	4	9.5	1.72	KR1.BIOCHAR_R2
8	93	4	10.7	1.61	KR1.BIOCHAR_R2
9	93	3	7.2	1.1	KR1.BIOCHAR_R2
10	93	3	11.0	1.4	KR1.BIOCHAR_R2
11	93	5	12.3	2.39	KR1.BIOCHAR_R2
1	77	5	14.9	1.93	KR1.BIOCHAR_R2
2	77	4	9.5	2.3	KR1.BIOCHAR_R2
3	77	5	9.1	1.42	KR1.BIOCHAR_R2
4	77	5	15.8	2.45	KR1.BIOCHAR_R2
5	77	4	5.7	1.0	KR1.BIOCHAR_R2
6	77	4	8.7	1.82	KR1.BIOCHAR_R2
7	77	3	8.8	1.62	KR1.BIOCHAR_R2
8	77	3	7.8	1.64	KR1.BIOCHAR_R2
9	77	5	8.1	1.7	KR1.BIOCHAR_R2
10	77	3	10.1	1.85	KR1.BIOCHAR_R2
1	133	3	9.0	1.42	KR1.BIOCHAR_R3
3	133	5	13.4	2.39	KR1.BIOCHAR_R3
4	133	4	9.0	1.42	KR1.BIOCHAR_R3
5	133	4	10.8	2.37	KR1.BIOCHAR_R3
6	133	4	10.2	1.8	KR1.BIOCHAR_R3
7	133	3	10.2	1.63	KR1.BIOCHAR_R3
8	133	5	8.9	1.62	KR1.BIOCHAR_R3
9	133	4	11.5	1.98	KR1.BIOCHAR_R3
10	133	3	7.6	1.2	KR1.BIOCHAR_R3
1	137	5	11.4	1.99	KR1.BIOCHAR_R3
4	137	5	12.0	2.17	KR1.BIOCHAR_R3
5	137	1	7.0		KR1.BIOCHAR_R3
6	137	3	11.7	1.55	KR1.BIOCHAR_R3
7	137	1	9.3		KR1.BIOCHAR_R3
8	137	3	8.1	2.0	KR1.BIOCHAR_R3
9	137	5	12.0	2.1	KR1.BIOCHAR_R3
10	137	2	8.5	1.86	KR1.BIOCHAR_R3
11	137	2	10.6	2.24	KR1.BIOCHAR_R3
12	137	3	6.8	1.1	KR1.BIOCHAR_R3
13	137	1	7.2	0.8	KR1.BIOCHAR_R3
1	141	5	7.5	1.2	KR1.BIOCHAR_R3
2	141	4	9.1	1.81	KR1.BIOCHAR_R3
3	141	1	9.5	1.46	KR1.BIOCHAR_R3
4	141	4	10.0	1.83	KR1.BIOCHAR_R3
5	141	3	7.4	1.2	KR1.BIOCHAR_R3
6	141	1	5.9	1.05	KR1.BIOCHAR_R3
7	141	5	13.5	2.06	KR1.BIOCHAR_R3
8	141	5	12.5	2.06	KR1.BIOCHAR_R3
9	141	5	13.3	2.14	KR1.BIOCHAR_R3
11	141	5	13.2	2.27	KR1.BIOCHAR_R3
2	145	2	8.5	1.3	KR1.BIOCHAR_R3
3	145	5	13.5	1.88	KR1.BIOCHAR_R3
4	145	5	13.9	2.13	KR1.BIOCHAR_R3
5	145	5	14.1	2.14	KR1.BIOCHAR_R3
6	145	3	9.9	1.8	KR1.BIOCHAR_R3
7	145	5	8.8	1.55	KR1.BIOCHAR_R3
8	145	5	12.8	2.8	KR1.BIOCHAR_R3
9	145	3	9.2	2.13	KR1.BIOCHAR_R3
10	145	5	13.0	3.11	KR1.BIOCHAR_R3
1	149	5	14.4	2.22	KR1.BIOCHAR_R3
2	149	2	3.8	0.7	KR1.BIOCHAR_R3
3	149	3	8.9	2.03	KR1.BIOCHAR_R3
4	149	2	9.9	1.92	KR1.BIOCHAR_R3
5	149	5	15.5	2.47	KR1.BIOCHAR_R3
6	149	5	14.1	2.38	KR1.BIOCHAR_R3
7	149	4	11.1	2.15	KR1.BIOCHAR_R3
8	149	5	13.1	2.3	KR1.BIOCHAR_R3
9	149	1	8.8	1.47	KR1.BIOCHAR_R3
10	149	1	4.9	0.4	KR1.BIOCHAR_R3
1	153	4	14.3	2.29	KR1.BIOCHAR_R3
2	153	3	11.2	1.97	KR1.BIOCHAR_R3
3	153	3	11.1	1.82	KR1.BIOCHAR_R3
4	153	4	13.0	1.77	KR1.BIOCHAR_R3
5	153	4	11.0	1.72	KR1.BIOCHAR_R3
6	153	4	5.0		KR1.BIOCHAR_R3
7	153	5	11.7	2.02	KR1.BIOCHAR_R3
8	153	5	10.0	2.12	KR1.BIOCHAR_R3
9	153	5	11.9	2.75	KR1.BIOCHAR_R3
10	153	5	12.5	2.71	KR1.BIOCHAR_R3
1	157	5	12.3	1.7	KR1.BIOCHAR_R3
2	157	4	11.2	1.42	KR1.BIOCHAR_R3
3	157	1	4.9		KR1.BIOCHAR_R3
4	157	1	9.8	0.8	KR1.BIOCHAR_R3
5	157	3	14.6	2.37	KR1.BIOCHAR_R3
6	157	5	13.0	1.81	KR1.BIOCHAR_R3
7	157	4	14.6	3.02	KR1.BIOCHAR_R3
8	157	1	7.3		KR1.BIOCHAR_R3
9	157	4	14.9	2.97	KR1.BIOCHAR_R3
10	157	1	8.5	1.4	KR1.BIOCHAR_R3
11	157	1		1.2	KR1.BIOCHAR_R3
12	157	5	4.7	0.4	KR1.BIOCHAR_R3
14	157	5	8.9	1.55	KR1.BIOCHAR_R3
15	157	2	9.0	1.72	KR1.BIOCHAR_R3
1	39	4			KR3.BIOCHAR_R1

2	39	5			KR3.BIOCHAR_R1
3	39	4			KR3.BIOCHAR_R1
4	39	5			KR3.BIOCHAR_R1
5	39	1			KR3.BIOCHAR_R1
6	39	5			KR3.BIOCHAR_R1
7	39	5			KR3.BIOCHAR_R1
8	39	4			KR3.BIOCHAR_R1
9	39	5			KR3.BIOCHAR_R1
10	39	4			KR3.BIOCHAR_R1
1	43	3			KR3.BIOCHAR_R1
2	43	3			KR3.BIOCHAR_R1
3	43	3			KR3.BIOCHAR_R1
4	43	4			KR3.BIOCHAR_R1
5	43	3			KR3.BIOCHAR_R1
6	43	4			KR3.BIOCHAR_R1
8	43	5			KR3.BIOCHAR_R1
9	43	5			KR3.BIOCHAR_R1
11	43	1			KR3.BIOCHAR_R1
12	43	5			KR3.BIOCHAR_R1
1	47	1			KR3.BIOCHAR_R1
2	47	4			KR3.BIOCHAR_R1
3	47	5			KR3.BIOCHAR_R1
4	47	5			KR3.BIOCHAR_R1
5	47	3			KR3.BIOCHAR_R1
6	47	1			KR3.BIOCHAR_R1
7	47	4			KR3.BIOCHAR_R1
8	47	4			KR3.BIOCHAR_R1
9	47	4			KR3.BIOCHAR_R1
10	47	1			KR3.BIOCHAR_R1
11	47	3			KR3.BIOCHAR_R1
1	51	1			KR3.BIOCHAR_R1
2	51	3			KR3.BIOCHAR_R1
3	51	1			KR3.BIOCHAR_R1
4	51	4			KR3.BIOCHAR_R1
5	51	4			KR3.BIOCHAR_R1
6	51	1			KR3.BIOCHAR_R1
7	51	4			KR3.BIOCHAR_R1
8	51	4			KR3.BIOCHAR_R1
9	51	3			KR3.BIOCHAR_R1
10	51	5			KR3.BIOCHAR_R1
11	51	5			KR3.BIOCHAR_R1
12	51	2			KR3.BIOCHAR_R1
13	51	3			KR3.BIOCHAR_R1
1	55	1			KR3.BIOCHAR_R1
2	55	4			KR3.BIOCHAR_R1
3	55	3			KR3.BIOCHAR_R1
4	55	3			KR3.BIOCHAR_R1
5	55	4			KR3.BIOCHAR_R1
6	55	1			KR3.BIOCHAR_R1
7	55	1			KR3.BIOCHAR_R1
8	55	1			KR3.BIOCHAR_R1
9	55	5			KR3.BIOCHAR_R1
10	55	3			KR3.BIOCHAR_R1
11	55	2			KR3.BIOCHAR_R1
12	55	4			KR3.BIOCHAR_R1
1	59	3			KR3.BIOCHAR_R1
2	59	3			KR3.BIOCHAR_R1
3	59	5			KR3.BIOCHAR_R1
4	59	3			KR3.BIOCHAR_R1
5	59	3			KR3.BIOCHAR_R1
6	59	5			KR3.BIOCHAR_R1
7	59	3			KR3.BIOCHAR_R1
8	59	3			KR3.BIOCHAR_R1
9	59	4			KR3.BIOCHAR_R1
11	59	5			KR3.BIOCHAR_R1
1	63	5			KR3.BIOCHAR_R1
2	63	3			KR3.BIOCHAR_R1
3	63	1			KR3.BIOCHAR_R1
4	63	4			KR3.BIOCHAR_R1
5	63	1			KR3.BIOCHAR_R1
6	63	5			KR3.BIOCHAR_R1
7	63	4			KR3.BIOCHAR_R1
8	63	1			KR3.BIOCHAR_R1
9	63	1			KR3.BIOCHAR_R1
10	63	3			KR3.BIOCHAR_R1
1	62	3			KR3.MULCH_R1
2	62	1			KR3.MULCH_R1
3	62	1			KR3.MULCH_R1
4	62	1			KR3.MULCH_R1
5	62	2			KR3.MULCH_R1
6	62	2			KR3.MULCH_R1
7	62	3			KR3.MULCH_R1
8	62	1			KR3.MULCH_R1
9	62	1			KR3.MULCH_R1
10	62	1			KR3.MULCH_R1
11	62	3			KR3.MULCH_R1
13	62	5			KR3.MULCH_R1
1	66	3			KR3.MULCH_R1
2	66	2			KR3.MULCH_R1
3	66	1			KR3.MULCH_R1
4	66	1			KR3.MULCH_R1
5	66	1			KR3.MULCH_R1
6	66	3			KR3.MULCH_R1
7	66	3			KR3.MULCH_R1
8	66	1			KR3.MULCH_R1
9	66	4			KR3.MULCH_R1
10	66	1			KR3.MULCH_R1
11	66	1			KR3.MULCH_R1
12	66	1			KR3.MULCH_R1
13	66	3			KR3.MULCH_R1
1	235	2			KR3.MULCH_R3
2	235	3			KR3.MULCH_R3
3	235	1			KR3.MULCH_R3
4	235	3			KR3.MULCH_R3
5	235	4			KR3.MULCH_R3
6	235	1			KR3.MULCH_R3
7	235	1			KR3.MULCH_R3
8	235	1			KR3.MULCH_R3
9	235	1			KR3.MULCH_R3
10	235	1			KR3.MULCH_R3
11	235	1			KR3.MULCH_R3
12	235	1			KR3.MULCH_R3
13	235	1			KR3.MULCH_R3
14	235	4			KR3.MULCH_R3
15	235	4			KR3.MULCH_R3
16	235	5			KR3.MULCH_R3
17	235	1			KR3.MULCH_R3
18	235	3			KR3.MULCH_R3
1	239	4			KR3.MULCH_R3

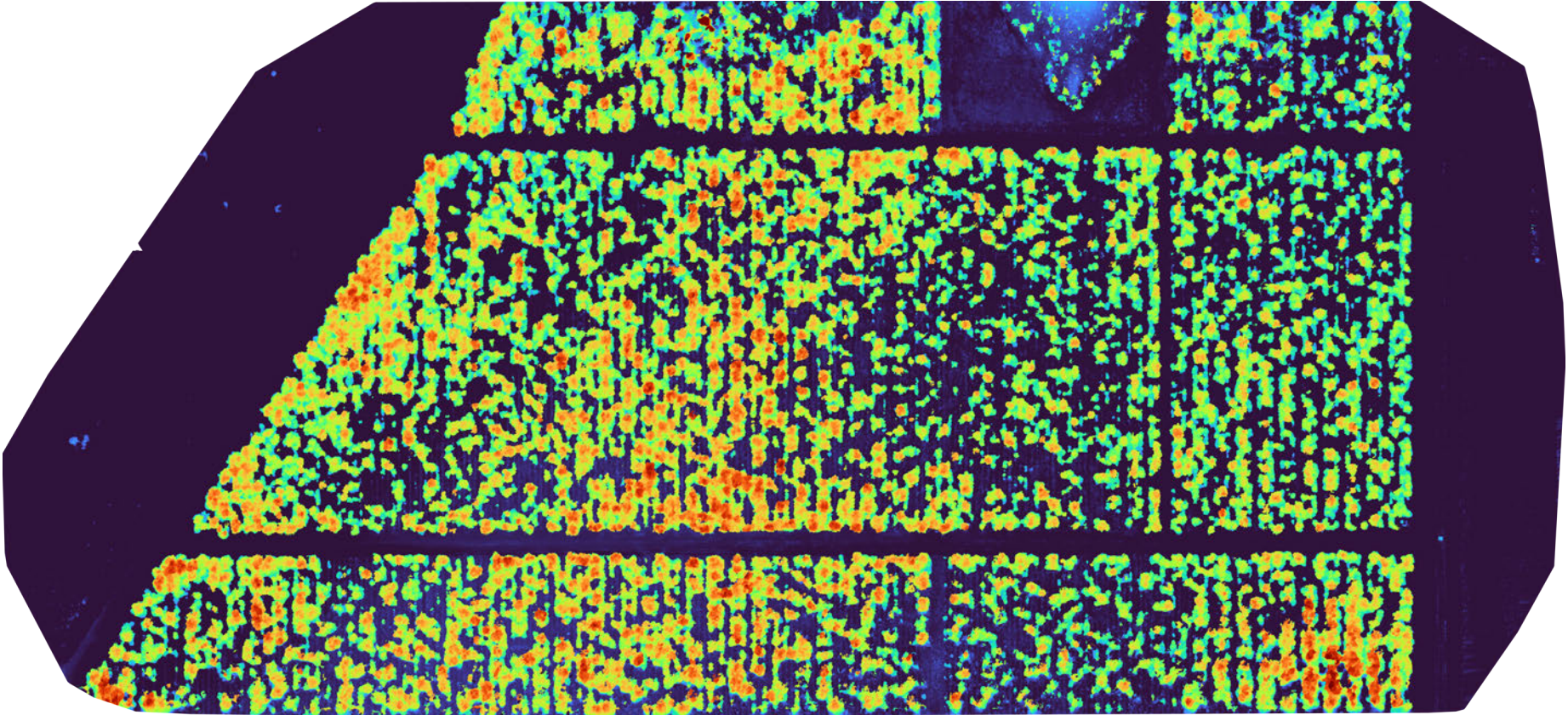
2	239	4			KR3.MULCH_R3
5	239	5			KR3.MULCH_R3
6	239	5			KR3.MULCH_R3
7	239	5			KR3.MULCH_R3
8	239	2			KR3.MULCH_R3
9	239	5			KR3.MULCH_R3
10	239	1			KR3.MULCH_R3
11	239	1			KR3.MULCH_R3
12	239	3			KR3.MULCH_R3
13	239	1			KR3.MULCH_R3
14	239	4			KR3.MULCH_R3
1	243	1			KR3.MULCH_R3
2	243	5			KR3.MULCH_R3
3	243	5			KR3.MULCH_R3
4	243	5			KR3.MULCH_R3
6	243	1			KR3.MULCH_R3
9	243	5			KR3.MULCH_R3
10	243	4			KR3.MULCH_R3
11	243	4			KR3.MULCH_R3
12	243	4			KR3.MULCH_R3

D.3 Drone Imaging

D.3.1 Kingston Rest 1 Block 47



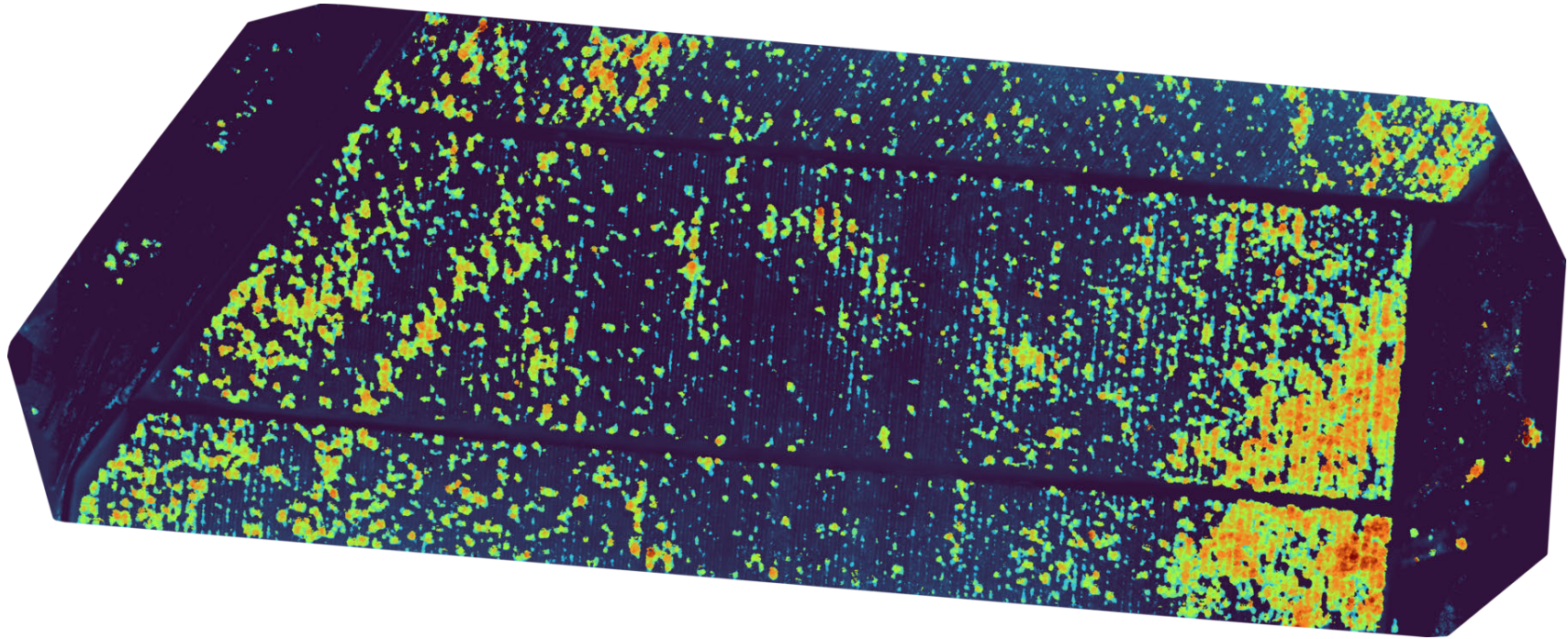
D.3.2 Kingston Rest 1 Block 47 - Digital Surface Model



D.3.3 Kingston Rest 3 Block 1



D.3.4 Kingston Rest 3 Block 1 - Digital Surface Model



Appendix E

Source Code

E.1 Tree Detection Module

```

1  # from networkx import omega
2  from zipp import Path
3  from detectron2.data.datasets import register_coco_instances
4  import random
5  import cv2
6  from detectron2 import model_zoo
7  from detectron2.config import get_cfg
8  from detectron2.utils.visualizer import Visualizer
9  from detectron2.data import MetadataCatalog, DatasetCatalog
10 from detectron2.engine import DefaultTrainer
11 from detectron2.engine import DefaultPredictor
12 from detectron2.utils.visualizer import ColorMode
13 from detectron2.evaluation import COCOEvaluator, inference_on_dataset, SemSegEvaluator, DatasetEvaluators
14 from detectron2.data import build_detection_test_loader, DatasetMapper
15 from detectron2.engine.hooks import HookBase
16 from detectron2.evaluation import inference_context
17 from detectron2.utils.logger import log_every_n_seconds
18 import detectron2.utils.comm as comm
19 import os
20 import datetime
21 from pycocotools.cocoeval import COCOEval
22 import torch
23 import time
24 import numpy as np
25 import logging
26
27
28 register_coco_instances("dataset_train", {},
29                        "TreeDetectionData/Training/annotations/instances-default.json",
30                        "TreeDetectionData/Training/images")
31 register_coco_instances("dataset_val", {},
32                        "TreeDetectionData/Validation/annotations/instances-default.json",
33                        "TreeDetectionData/Validation/images")
34 register_coco_instances("dataset_test", {},
35                        "TreeDetectionData/Test/annotations/instances-default.json",
36                        "TreeDetectionData/Test/images")
37
38 # metadata = MetadataCatalog.get("dataset_val")
39 # dataset_dicts = DatasetCatalog.get("dataset_val")
40 # i = 0
41 # for d in random.sample(dataset_dicts, 3):
42 #     img = cv2.imread(d["file_name"])
43 #     visualizer = Visualizer(img[:, :, ::-1], metadata=metadata, scale=0.5)
44 #     out = visualizer.draw_dataset_dict(d)
45 #     cv2.imwrite('image' + str(i) + '.png', out.get_image()[:, :, ::-1])
46 #     i+=1
47
48
49 IM_SIZE = 512
50
51 now = datetime.datetime.now().strftime("%Y-%b-%d-%H%M%S")
52 log_dir = os.path.join('./TDM_Logs/', "TDM-{}".format(now))
53 # log_dir = 'TDM_Logs/TDM-2023-Oct-08-0707-34/'
54
55 # ----- TRAIN -----
56
57 cfg = get_cfg()
58 cfg.merge_from_file(model_zoo.get_config_file("LVISv0.5-InstanceSegmentation/mask_rcnn_R-101_FPN-1x.yaml"))
59 cfg.INPUT.MIN_SIZE_TRAIN = (IM_SIZE,)
60 cfg.INPUT.MAX_SIZE_TRAIN = IM_SIZE
61 # Size of the smallest side of the image during testing. Set to zero to disable resize in testing.
62 cfg.INPUT.MIN_SIZE_TEST = IM_SIZE
63 # Maximum size of the side of the image during testing
64 cfg.INPUT.MAX_SIZE_TEST = IM_SIZE
65 cfg.MODEL.RPN.SMOOTH_L1_BETA = 1.5
66 # cfg.MODEL.RPN.LOSS_WEIGHT = 1.0
67 cfg.INPUT.RANDOM_FLIP = "horizontal"
68 cfg.DATASETS.TRAIN = ("dataset_train",)
69 cfg.DATASETS.TEST = ("dataset_val",)
70 cfg.TEST.EVAL_PERIOD = 100
71 cfg.DATALOADER.NUMWORKERS = 4
72 cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("LVISv0.5-InstanceSegmentation/mask_rcnn_R-101_FPN-1x.yaml") # Let
73 cfg.SOLVER.IMS_PER_BATCH = 4 # This is the real "batch size" commonly known to deep learning people
74 cfg.SOLVER.BASE_LR = 0.00025 # pick a good LR
75 cfg.SOLVER.MAX_ITER = 2#300 # 300 iterations seems good enough for this toy dataset; you will need to train longer
76 cfg.SOLVER.STEPS = [] # do not decay learning rate
77 cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128 # The "RoIHead batch size". 128 is faster, and good enough for this
78 cfg.MODEL.ROI_HEADS.NUM_CLASSES = 3 # only has one class (ballon). (see https://detectron2.readthedocs.io/tutorials
79 # NOTE: this config means the number of classes, but a few popular unofficial tutorials incorrect uses num_classes+1
80 cfg.OUTPUT_DIR = log_dir
81
82 class LossEvalHook(HookBase):
83     def __init__(self, eval_period, model, data_loader):
84         self._model = model
85         self._period = eval_period
86         self._data_loader = data_loader
87
88     def _do_loss_eval(self):
89         # Copying inference_on_dataset from evaluator.py
90         total = len(self._data_loader)
91         num_warmup = min(5, total - 1)
92
93         start_time = time.perf_counter()
94         total_compute_time = 0
95         losses = []
96         for idx, inputs in enumerate(self._data_loader):
97             if idx == num_warmup:

```

```

98         start_time = time.perf_counter()
99         total_compute_time = 0
100     start_compute_time = time.perf_counter()
101     if torch.cuda.is_available():
102         torch.cuda.synchronize()
103     total_compute_time += time.perf_counter() - start_compute_time
104     iters_after_start = idx + 1 - num_warmup * int(idx >= num_warmup)
105     seconds_per_img = total_compute_time / iters_after_start
106     if idx >= num_warmup * 2 or seconds_per_img > 5:
107         total_seconds_per_img = (time.perf_counter() - start_time) / iters_after_start
108         eta = datetime.timedelta(seconds=int(total_seconds_per_img * (total - idx - 1)))
109         log_every_n_seconds(
110             logging.INFO,
111             "Loss on Validation {} done {} / {} {} {:.4f} s / {} img. {} ETA={}".format(
112                 idx + 1, total, seconds_per_img, str(eta)
113             ),
114             n=5,
115         )
116     loss_batch = self._get_loss(inputs)
117     losses.append(loss_batch)
118     mean_loss = np.mean(losses)
119     self.trainer.storage.put_scalar('validation_loss', mean_loss)
120     comm.synchronize()
121
122     return losses
123
124 def _get_loss(self, data):
125     # How loss is calculated on train loop
126     metrics_dict = self._model(data)
127     metrics_dict = {
128         k: v.detach().cpu().item() if isinstance(v, torch.Tensor) else float(v)
129         for k, v in metrics_dict.items()
130     }
131     total_losses_reduced = sum(loss for loss in metrics_dict.values())
132     return total_losses_reduced
133
134
135 def after_step(self):
136     next_iter = self.trainer.iter + 1
137     is_final = next_iter == self.trainer.max_iter
138     if is_final or (self._period > 0 and next_iter % self._period == 0):
139         self._do_loss_eval()
140     self.trainer.storage.put_scalars(timetest=12)
141
142 class MyTrainer(DefaultTrainer):
143     @classmethod
144     def build_evaluator(cls, cfg, dataset_name, output_folder=None):
145         if output_folder is None:
146             output_folder = os.path.join(cfg.OUTPUT_DIR, "inference")
147         return COCOEvaluator(dataset_name, cfg, True, output_folder)
148
149     def build_hooks(self):
150         hooks = super().build_hooks()
151         hooks.insert(-1, LossEvalHook(
152             cfg.TEST.EVAL_PERIOD,
153             self.model,
154             build_detection_test_loader(
155                 self.cfg,
156                 self.cfg.DATASETS.TEST[0],
157                 DatasetMapper(self.cfg, True)
158             )
159         ))
160         return hooks
161
162 # =====
163 os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
164 trainer = MyTrainer(cfg)
165 trainer.resume_or_load(resume=False)
166 trainer.train()
167 # =====
168
169
170
171 # Inference should use the config with parameters that are used in training
172 # cfg now already contains everything we've set previously. We changed it a little bit for inference:
173 cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth") # path to the model we just trained
174 cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.55 # set a custom testing threshold
175 predictor = DefaultPredictor(cfg)
176
177 metadata = MetadataCatalog.get("dataset_test")
178 dataset_dicts = DatasetCatalog.get("dataset_test")
179 i = 0
180 for d in random.sample(dataset_dicts, 10):
181     im = cv2.imread(d["file_name"])
182     outputs = predictor(im) # format is documented at https://detectron2.readthedocs.io/tutorials/models.html#model
183     v = Visualizer(im[:, :, ::-1],
184                   metadata=metadata,
185                   scale=1,
186                   instance_mode=ColorMode.IMAGE # remove the colors of unsegmented pixels. This option is only av
187     )
188     out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
189     cv2.imwrite(log_dir + '/pred' + str(i) + '.png', out.get_image()[:, :, ::-1])
190
191     visualizer = Visualizer(im[:, :, ::-1], metadata=metadata, scale=1)
192     out = visualizer.draw_dataset_dict(d)
193     cv2.imwrite(log_dir + '/groundTruth' + str(i) + '.png', out.get_image()[:, :, ::-1])
194
195     i+=1
196
197 eval_dir = log_dir + '/COCO_Eval/'

```

```
198 os.makedirs(eval_dir, exist_ok=True)
199
200 cocoEval = COCOEvaluator("dataset_test", output_dir=eval_dir, tasks=("segm", "bbox"))
201 evaluators = DatasetEvaluators([cocoEval])
202 val_loader = build_detection_test_loader(cfg, "dataset_test")
203 results = inference_on_dataset(predictor.model, val_loader, evaluators)
204 print(results)
205
206 # pr_curve = PrecisionRecallCurve(task="multiclass", num_classes=5)
207 # precision, recall, thresholds = pr_curve(pred, target)
```

Listing E.1: Final Tree Detection Model.

E.2 Health Classifier Model

```

1 from torchmetrics.classification import MulticlassPrecisionRecallCurve, MulticlassROC
2 import pandas as pd
3 from tensorboard.plugins.hparams import api as hp
4 import itertools
5 from matplotlib import pyplot as plt
6 import io
7 import cv2
8 import tensorflow as tf
9 from tensorflow import keras
10 import datetime
11 import os
12 import numpy as np
13 import sklearn
14 import torch
15 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
16
17 # -----
18 #                               GLOBAL CONSTANTS
19 # -----
20
21 # MODEL CONFIG PARAMETERS
22 LEARNING_RATE = hp.HParam('learning_rate', hp.Discrete([0.005]))
23 NUM_UNITS = hp.HParam('num_units', hp.Discrete([1536]))
24 OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam']))
25 DROPOUT_RATE = hp.HParam('dropout', hp.Discrete([0.6]))
26 L2_PENALTY = hp.HParam('l2_penalty', hp.Discrete([0.05]))
27 WEIGHTS = hp.HParam('weights', hp.Discrete(['imagenet']))
28 IMAGE_TYPE = hp.HParam('image_type', hp.Discrete(['masked']))
29 MODEL_LAYERS = hp.HParam('model_layers', hp.Discrete(['50']))
30 METRIC_ACCURACY = 'top_2_cat_acc'
31 METRIC_ACCURACY2 = 'top_1_cat_acc'
32 METRIC_F1 = 'f1_score'
33
34 IM_SIZE = 300
35 INPUT_SHAPE = (300, 300, 3)
36 OUTPUT_NUM = 5
37 BATCH_SIZE = 4
38 EPOCHS = 100
39 VAL_SPLIT = 0.2
40 CLASSES = 5
41
42
43 data_dir = './images/Stage_2/Masks/Health5Class/'
44 test_dir = './images/Stage_2/Masks/HealthTest/'
45 data_dir_crops = './images/Stage_2/JPEGImages/Health/'
46 now = datetime.datetime.now().strftime("%Y-%b-%d-%H%M%S")
47 log_dir = os.path.join('./THC_Logs/', "THC-{}".format(now))
48 data_cnt = sum([len(files) for r, d, files in os.walk(data_dir)])
49
50 with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
51     hp.hparams_config(
52         hparams=[NUM_UNITS, DROPOUT_RATE, OPTIMIZER, LEARNING_RATE, L2_PENALTY,
53                 MODEL_LAYERS, WEIGHTS, IMAGE_TYPE],
54         metrics=[hp.Metric(METRIC_ACCURACY, display_name='Top2CatagoricalAccuracy'),
55                 hp.Metric(METRIC_ACCURACY2,
56                           display_name='Top1CatagoricalAccuracy'),
57                 hp.Metric(METRIC_F1, display_name='F1Score')],
58     )
59
60 checkpoint_path = os.path.join(log_dir, "TreeHealthClassifier_Best.h5")
61
62 # -----
63 #                               FUNCTIONS
64 # -----
65
66
67 def get_files(dir):
68     return [f for f in os.listdir(dir) if os.path.isfile(os.path.join(dir, f))]
69
70
71 def train_test_model(log_dir, hparams):
72
73     data_augmentation = tf.keras.Sequential([
74         keras.layers.RandomFlip("horizontal_and_vertical"),
75         keras.layers.RandomRotation(1, fill_mode="constant", fill_value=0),
76     ])
77
78     if hparams[WEIGHTS] == 'None':
79         wts = None
80     else:
81         wts = hparams[WEIGHTS]
82
83     if hparams[MODEL_LAYERS] == '50':
84         resnet_type = tf.keras.applications.inception_v3.InceptionV3
85     elif hparams[MODEL_LAYERS] == '101':
86         resnet_type = keras.applications.resnet_rs.ResNetRS101
87
88     resnet_base = resnet_type(
89         include_top=False,
90         weights=wts,
91         input_shape=INPUT_SHAPE,
92         input_tensor=None,
93         pooling=None,
94         classifier_activation='softmax',
95         # include_preprocessing=True
96     )
97

```

```

98     model = keras.models.Sequential()
99     model.add(data_augmentation)
100     model.add(resnet_base)
101     model.add(keras.layers.GlobalAveragePooling2D())
102     model.add(keras.layers.Dense(hparams[NUM_UNITS], activation='relu'))
103     model.add(keras.layers.Dropout(hparams[DROPOUT_RATE]))
104     model.add(keras.layers.Dense(CLASSES, activation='softmax'))
105     model.trainable = True
106
107     # adding regularization
108     regularizer = tf.keras.regularizers.l2(hparams[L2_PENALTY])
109
110     for layer in model.layers[1].layers:
111         for attr in ['kernel_regularizer']:
112             if hasattr(layer, attr):
113                 setattr(layer, attr, regularizer)
114
115     if OPTIMIZER == 'adam':
116         optimizer = tf.keras.optimizers.Adam(
117             learning_rate=hparams[LEARNING_RATE])
118     else:
119         optimizer = tf.keras.optimizers.SGD(
120             learning_rate=hparams[LEARNING_RATE])
121
122     model.compile(
123         optimizer=optimizer,
124         loss='CategoricalCrossentropy',
125         metrics=[
126             tf.keras.metrics.CategoricalAccuracy(name='accuracy'),
127             tf.keras.metrics.Precision(name='precision'),
128             tf.keras.metrics.Recall(name='recall'),
129             tf.keras.metrics.AUC(curve='PR', name='AUC_PR'),
130             tf.keras.metrics.F1Score(name='f1_score', average='macro')],
131         run_eagerly=None,
132         steps_per_execution=None,
133         jit_compile=None,
134         pss_evaluation_shards=0,
135     )
136
137     callbacks = [
138         keras.callbacks.TensorBoard(log_dir=log_dir,
139                                     histogram_freq=10, write_graph=False,
140                                     write_images=False),
141         keras.callbacks.ModelCheckpoint(log_dir + "/TreeHealthClassifier_Best.h5",
142                                         verbose=1, save_weights_only=True,
143                                         save_best_only=True, monitor='f1_score',
144                                         mode='max'),
145         keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
146                                           factor=0.2,
147                                           patience=2,
148                                           verbose=1,
149                                           mode='auto',
150                                           min_delta=0.0001,
151                                           cooldown=0,
152                                           min_lr=0.0000001),
153         keras.callbacks.EarlyStopping(monitor='val_loss',
154                                       min_delta=0.00001,
155                                       patience=5,
156                                       verbose=1,
157                                       mode='auto',
158                                       baseline=None,
159                                       ),
160         keras.callbacks.TerminateOnNaN(),
161         # tf.keras.callbacks.TensorBoard(log_dir), # log metrics
162         hp.KerasCallback(log_dir, hparams), # log hparams
163     ]
164
165     history = model.fit(norm_train_ds,
166                        validation_data=norm_val_ds,
167                        epochs=EPOCHS,
168                        verbose=1,
169                        callbacks=callbacks,
170                        shuffle=True,
171                        initial_epoch=0,
172                        class_weight={0: 1.7, 1: 3.7,
173                                     2: 2.18, 3: 1.78, 4: 1.0},
174                        validation_freq=1,
175                        max_queue_size=10,
176                        workers=1,
177                        use_multiprocessing=False)
178     model.load_weights(log_dir + "/TreeHealthClassifier_Best.h5")
179     _, accuracy, *_ , f1 = model.evaluate(norm_val_ds)
180
181     print('Best_test_F1_scores:{}'.format(f1))
182
183     return history, accuracy, model
184
185
186 def normalise_ds(ds):
187     normalization_layer = tf.keras.layers.Rescaling(1./255)
188     return ds.map(lambda x, y: (normalization_layer(x), y))
189
190
191 def plot_confusion_matrix(cm, class_names):
192     """
193     Returns a matplotlib figure containing the plotted confusion matrix.
194
195     Args:
196         cm (array, shape = [n, n]): a confusion matrix of integer classes
197         class_names (array, shape = [n]): String names of the integer classes

```

```

198 """
199 figure = plt.figure(figsize=(9, 9))
200 ax = plt.gca()
201 ax.tick_params(axis="x", top=True, labeltop=True,
202               bottom=False, labelbottom=False)
203 plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
204 tick_marks = np.arange(len(class_names))
205 plt.xticks(tick_marks, class_names, rotation=45)
206 plt.yticks(tick_marks, class_names)
207 plt.rcParams.update({'font.size': 20})
208 plt.rc('axes', titlesize=18) # fontsize of the axes title
209 plt.rc('axes', labelsiz=18) # fontsize of the x and y labels
210 plt.rc('xtick', labelsiz=18) # fontsize of the tick labels
211 plt.rc('ytick', labelsiz=18)
212 labels = cm
213
214 # Use white text if squares are dark; otherwise black.
215 threshold = np.max(cm) / 2.
216 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
217     color = "white" if cm[i, j] > threshold else "black"
218     plt.text(j, i, labels[i, j], horizontalalignment="center", color=color)
219
220 plt.ylabel('Ground-Truth')
221 plt.xlabel('Predicted')
222 return figure
223
224
225 def plot_to_image(figure):
226     """Converts the matplotlib plot specified by 'figure' to a PNG image and
227     returns it. The supplied figure is closed and inaccessible after this call."""
228     # Save the plot to a PNG in memory.
229     buf = io.BytesIO()
230     plt.savefig(buf, format='png')
231     plt.close(figure)
232     buf.seek(0)
233     # Convert PNG buffer to TF image
234     image = tf.image.decode_png(buf.getvalue(), channels=4)
235     # Add the batch dimension
236     image = tf.expand_dims(image, 0)
237     return image
238
239
240 def log_confusion_matrix(epoch, logs, test_labels, test_preds):
241     cm = tf.math.confusion_matrix(test_labels, test_preds)
242     pd_conf_mat = pd.DataFrame(cm, columns=['1', '2', '3', '4', '5'], index=[
243         '1', '2', '3', '4', '5'])
244     print(pd_conf_mat)
245     cm = np.array(cm)
246     # Log the confusion matrix as an image summary.
247     figure = plot_confusion_matrix(cm, class_names=['1', '2', '3', '4', '5'])
248     cm_image = plot_to_image(figure)
249
250     # Log the confusion matrix as an image summary.
251     with file_writer_cm.as_default():
252         tf.summary.image("epoch_confusion_matrix", cm_image, step=4)
253
254
255 # Define the per-epoch callback.
256 cm_callback = keras.callbacks.LambdaCallback(on_epoch_end=log_confusion_matrix)
257 file_writer_cm = tf.summary.create_file_writer(log_dir + '/cm')
258
259
260 def plot_roc(name, labels, predictions, **kwargs):
261     fp, tp, _ = sklearn.metrics.roc_curve(labels, predictions)
262
263     plt.plot(100*fp, 100*tp, label=name, linewidth=2, **kwargs)
264     plt.xlabel('False_positives_[%]')
265     plt.ylabel('True_positives_[%]')
266     plt.xlim([-0.5, 20])
267     plt.ylim([80, 100.5])
268     plt.grid(True)
269     ax = plt.gca()
270     ax.set_aspect('equal')
271
272
273 def plot_prc(name, labels, predictions, **kwargs):
274     precision, recall, _ = sklearn.metrics.precision_recall_curve(
275         labels, predictions)
276
277     plt.plot(precision, recall, label=name, linewidth=2, **kwargs)
278     plt.xlabel('Precision')
279     plt.ylabel('Recall')
280     plt.grid(True)
281     ax = plt.gca()
282     ax.set_aspect('equal')
283
284 # -----
285 #                               MAIN CODE
286 # -----
287
288
289 session_num = 0
290
291
292 def run(run_dir, hparams):
293     with tf.summary.create_file_writer(run_dir).as_default():
294         hp.hparams(hparams) # record the values used in this trial
295         _, accuracy, model = train_test_model(run_dir, hparams)
296         tf.summary.scalar(METRIC_ACCURACY, accuracy, step=EPOCHS)
297         return model

```

```

298
299
300 for imtype in IMAGE_TYPE.domain.values:
301
302     if imtype == 'masked':
303         data_dir = data_dir
304     else:
305         data_dir = data_dir_crops
306
307     train_ds = tf.keras.utils.image_dataset_from_directory(
308         data_dir,
309         label_mode='categorical',
310         validation_split=VAL_SPLIT,
311         subset="training",
312         seed=77,
313         shuffle=True,
314         image_size=(IM_SIZE, IM_SIZE),
315         batch_size=BATCH_SIZE)
316
317     val_ds = tf.keras.utils.image_dataset_from_directory(
318         data_dir,
319         label_mode='categorical',
320         validation_split=VAL_SPLIT,
321         subset="validation",
322         seed=77,
323         shuffle=True,
324         image_size=(IM_SIZE, IM_SIZE),
325         batch_size=BATCH_SIZE)
326
327     val_batches = tf.data.experimental.cardinality(val_ds)
328     test_ds = val_ds.take((val_batches) // 2)
329     val_ds = val_ds.skip((val_batches) // 2)
330
331     AUTOTUNE = tf.data.AUTOTUNE
332
333     norm_train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
334     norm_val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
335     norm_test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
336
337     for layers in MODELLAYERS.domain.values:
338         for weights in WEIGHTS.domain.values:
339             for l2 in L2_PENALTY.domain.values:
340                 for lr in LEARNING_RATE.domain.values:
341                     for num_units in NUM_UNITS.domain.values:
342                         for dropout_rate in DROPOUT_RATE.domain.values:
343                             for optim in OPTIMIZER.domain.values:
344                                 hparams = {
345                                     NUM_UNITS: num_units,
346                                     DROPOUT_RATE: dropout_rate,
347                                     OPTIMIZER: optim,
348                                     LEARNING_RATE: lr,
349                                     L2_PENALTY: l2,
350                                     WEIGHTS: weights,
351                                     IMAGE_TYPE: imtype,
352                                     MODELLAYERS: layers
353                                 }
354
355                                 run_name = "run-%d" % session_num
356                                 print('——_Starting_trial:_%s' % run_name)
357                                 print({h.name: hparams[h] for h in hparams})
358                                 *, model = train_test_model(
359                                     log_dir + '/hparam_tuning/' + run_name, hparams)
360
361                                 print('——_Final_Training_Results_——')
362                                 loss, accuracy, * \
363                                     -, f1 = model.evaluate(
364                                         norm_train_ds, verbose=2)
365
366                                 print('——_Final_Test_Evaluation_Results_——')
367                                 loss, accuracy, * \
368                                     -, f1 = model.evaluate(
369                                         norm_test_ds, verbose=2)
370
371                                 test_preds = model.predict(
372                                     norm_test_ds, batch_size=BATCH_SIZE)
373                                 train_preds = model.predict(
374                                     norm_train_ds, batch_size=BATCH_SIZE)
375                                 val_preds = model.predict(
376                                     norm_val_ds, batch_size=BATCH_SIZE)
377
378                                 cnt = 0
379                                 preds = []
380                                 test_labels = []
381                                 one_hot_test_labels = []
382
383                                 for pred, [img, label] in zip(test_preds, norm_test_ds.unbatch()):
384                                     pred = np.argmax(pred)
385                                     one_hot_test_labels.append(label)
386                                     label = np.argmax(label)
387                                     preds.append(pred)
388                                     test_labels.append(label)
389                                     cv2.imwrite(log_dir + '/testpred' + str(cnt)
390                                                 + '_pred' +
391                                                 str(pred+1) + '_actual'
392                                                 + str(label+1) + '.png', np.array(img))
393
394                                 cnt += 1
395
396     cnt = 0
397     one_hot_train_labels = []

```

```

398
399
400     for [_, label] in norm_train_ds.unbatch():
401         one_hot_train_labels.append(label)
402         cnt += 1
403
404     cnt = 0
405     one_hot_val_labels = []
406
407     for [_, label] in norm_val_ds.unbatch():
408         one_hot_val_labels.append(label)
409         cnt += 1
410
411     metric = tf.keras.metrics.F1Score()
412     metric.update_state(
413         one_hot_test_labels, test_preds)
414     result = metric.result()
415     print('test_f1s', result.numpy())
416
417     metric = tf.keras.metrics.F1Score()
418     metric.update_state(
419         one_hot_train_labels, train_preds)
420     result = metric.result()
421     print('train_f1s', result.numpy())
422
423     metric = tf.keras.metrics.F1Score()
424     metric.update_state(
425         one_hot_val_labels, val_preds)
426     result = metric.result()
427     print('val_f1s', result.numpy())
428
429     test_preds_torch = torch.from_numpy(
430         np.array(test_preds))
431     test_labels_torch = torch.from_numpy(
432         np.array(test_labels))
433
434     metric = MulticlassPrecisionRecallCurve(
435         num_classes=CLASSES)
436     metric.update(test_preds_torch,
437                  test_labels_torch)
438     fig, ax = metric.plot(score=True)
439     plt.xlabel('Recall')
440     plt.ylabel('Precision')
441     plt.title('Multiclass_Precision_Recall_Curve')
442     plt.savefig(log_dir + '/prc.png')
443
444     prc_image = plot_to_image(fig)
445     with file_writer_cm.as_default():
446         tf.summary.image(" prc", prc_image, step=3)
447
448     metric = MulticlassROC(num_classes=CLASSES)
449     metric.update(test_preds_torch,
450                  test_labels_torch)
451     fig, ax = metric.plot(score=True)
452     plt.xlabel('False_Positive_Rate')
453     plt.ylabel('True_Positive_Rate')
454     plt.title('Multiclass_Reciever_Operating_Characteristic')
455     plt.savefig(log_dir + '/roc.png')
456
457     roc_image = plot_to_image(fig)
458     with file_writer_cm.as_default():
459         tf.summary.image(" roc", roc_image, step=3)
460
461     log_confusion_matrix(
462         epoch=session_num, logs=None,
463         test_labels=test_labels,
464         test_preds=np.argmax(test_preds, axis=1))
465     session_num += 1

```

Listing E.2: Final Health Classifier Model.

E.3 Tree Parameter Regressor Model

```

1 import cv2
2 import numpy as np
3 import tensorflow as tf
4 from tensorflow import keras
5 import datetime
6 import random
7 import pandas as pd
8 import os
9 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
10 from matplotlib import pyplot as plt
11 import scipy.stats
12
13
14 # -----
15 #                               GLOBAL CONSTANTS
16 # -----
17
18 L2.PENALTY = 0.35
19 IM_SIZE = 320
20 INPUT_SHAPE = (IM_SIZE, IM_SIZE, 3)
21 # OUTPUT_NUM = 5
22 BATCH_SIZE = 4
23 # TRIAL_ROWS = {"MULCH_R1": [65, 69, 73], }
24
25 data_dir = './images/Stage_2/Masks/RegressorData/'
26 train_dir = './images/Stage_2/Masks/RegressorData/Train/'
27 val_dir = './images/Stage_2/Masks/RegressorData/Val/'
28 now = datetime.datetime.now().strftime("%Y-%b-%d-%H%M%S")
29 log_dir = os.path.join('./TPR_Logs/', "TPR-{}".format(now))
30 # log_dir = './TPR_Logs/TPR-2023-Oct-14-0918-26/'
31
32 # checkpoint_path = os.path.join(log_dir, "TreeParameterRegressor.*epoch*.h5")
33 # checkpoint_path = checkpoint_path.replace("*epoch*", "{epoch:04d}")
34
35 checkpoint_path = os.path.join(log_dir, 'checkpoint_best.h5')
36
37 # -----
38 #                               FUNCTIONS
39 # -----
40
41 def get_files(dir):
42     return [f for f in os.listdir(dir) if os.path.isfile(os.path.join(dir, f))]
43
44 def create_regressor_model(input_shape, outputs, top='flatten'):
45     if top not in ('flatten', 'avg', 'max'):
46         raise ValueError('unexpected_top_layer_type: %s' % top)
47
48     data_augmentation = tf.keras.Sequential([
49         keras.layers.RandomFlip("horizontal_and_vertical"),
50         keras.layers.RandomRotation(1, fill_mode="constant", fill_value=0),
51         # keras.layers.RandomBrightness(0.2),
52     ])
53
54     # connects base model with new "head"
55     BottleneckLayer = {
56         'flatten': keras.layers.Flatten(),
57         'avg': keras.layers.GlobalAveragePooling2D(),
58         'max': keras.layers.GlobalMaxPooling2D()
59     }[top]
60
61     base = keras.applications.resnet_rs.ResNetRS101(
62         include_top=False,
63         weights='imagenet',
64         # weights=None,
65         input_shape=input_shape,
66         input_tensor=None,
67         pooling=None,
68         # classifier_activation='softmax',
69         include_preprocessing=True
70     )
71
72     # x = BottleneckLayer(base.output)
73
74     # x = keras.layers.Dense(1056, activation='relu')(x)
75     # x = keras.layers.Dropout(0.6)(x)
76
77     # x = keras.layers.Dense(outputs, activation='linear')(x)
78     # model = keras.Model(inputs=base.inputs, outputs=x)
79     # model.trainable = True
80
81     model = keras.models.Sequential()
82     model.add(data_augmentation)
83     model.add(base)
84     model.add(BottleneckLayer)
85     model.add(keras.layers.Dense(256, activation='relu'))
86     model.add(keras.layers.Dropout(0.6))
87     model.add(keras.layers.Dense(outputs, activation='linear'))
88     model.trainable = True
89
90     # adding regularization
91     regularizer = tf.keras.regularizers.l2(L2.PENALTY)
92
93     for layer in model.layers[0].layers:
94         for attr in ['kernel_regularizer']:
95             if hasattr(layer, attr):

```

```

97         setattr(layer, attr, regularizer)
98     return model
99
100 def normalise_ds(ds):
101     normalization_layer = tf.keras.layers.Rescaling(1./255)
102     return ds.map(lambda x, y: (normalization_layer(x), y))
103
104 # def get_data(dir, train_split=0.8):
105 #     files = get_files(dir)
106 #
107 #     total_size = (len(files))
108 #     train_num = int(np.round((len(files) * train_split)))
109 #
110 #     train_set = random.sample(range(total_size), train_num)
111 #
112 #     train_x = np.empty((len(train_set), IM_SIZE, IM_SIZE, 3))
113 #     train_y = np.empty((len(train_set), 2))
114 #
115 #     val_x = np.empty(((len(files) - len(train_set)), IM_SIZE, IM_SIZE, 3))
116 #     val_y = np.empty(((len(files) - len(train_set)), 2))
117 #
118 #
119 #     i = 0
120 #     train_i = 0
121 #     val_i = 0
122 #     for file in files:
123 #         file_path = os.path.join(dir, file)
124 #         img = cv2.imread(file_path)
125 #
126 #         file = os.path.splitext(file)[0]
127 #         parts = file.split('_')
128 #
129 #         if i in train_set:
130 #             train_x[train_i, :, :, :] = img
131 #             train_y[train_i, 0] = np.float32(parts[-2])
132 #             train_y[train_i, 1] = np.float32(parts[-1])
133 #             train_i += 1
134 #         else:
135 #             val_x[val_i, :, :, :] = img
136 #             val_y[val_i, 0] = np.float32(parts[-2])
137 #             val_y[val_i, 1] = np.float32(parts[-1])
138 #             val_i += 1
139 #
140 #     i += 1
141 #
142 #     return train_x, train_y, val_x, val_y
143
144 def get_data(dir, outputs='both', train_split=0.8):
145     files = get_files(dir)
146
147     total_size = (len(files))
148     train_num = int(np.round((len(files) * train_split)))
149
150     train_set = random.sample(range(total_size), train_num)
151
152     train_x = np.empty((len(train_set), IM_SIZE, IM_SIZE, 3))
153
154     val_x = np.empty(((len(files) - len(train_set)), IM_SIZE, IM_SIZE, 3))
155
156     if outputs == 'both':
157         train_y = np.empty((len(train_set), 2))
158         val_y = np.empty(((len(files) - len(train_set)), 2))
159     else:
160         train_y = np.empty(len(train_set))
161         val_y = np.empty(len(files) - len(train_set))
162
163     i = 0
164     train_i = 0
165     val_i = 0
166     for file in files:
167         file_path = os.path.join(dir, file)
168         img = cv2.imread(file_path)
169
170         file = os.path.splitext(file)[0]
171         parts = file.split('_')
172
173         if i in train_set:
174             train_x[train_i, :, :, :] = img
175
176             if outputs == 'both':
177                 train_y[train_i, 0] = np.float32(parts[-2])
178                 train_y[train_i, 1] = np.float32(parts[-1])
179             elif outputs == 'dbh':
180                 train_y[train_i] = np.float32(parts[-2])
181             elif outputs == 'bh':
182                 train_y[train_i] = np.float32(parts[-1])
183
184             train_i += 1
185         else:
186             val_x[val_i, :, :, :] = img
187
188             if outputs == 'both':
189                 val_y[val_i, 0] = np.float32(parts[-2])
190                 val_y[val_i, 1] = np.float32(parts[-1])
191             elif outputs == 'dbh':
192                 val_y[val_i] = np.float32(parts[-2])
193             elif outputs == 'bh':
194                 val_y[val_i] = np.float32(parts[-1])
195
196             val_i += 1

```

```

197         i += 1
198
199     norm_train_ds = tf.data.Dataset.from_tensor_slices((train_x, train_y)).batch(BATCH_SIZE)
200     norm_val_ds = tf.data.Dataset.from_tensor_slices((val_x, val_y)).batch(BATCH_SIZE)
201
202     # norm_train_ds = normalise_ds(train_ds)
203     # norm_val_ds = normalise_ds(val_ds)
204
205
206
207
208     # AUTOTUNE = tf.data.AUTOTUNE
209     # norm_train_ds = norm_train_ds.cache().prefetch(buffer_size=AUTOTUNE)
210     # norm_val_ds = norm_val_ds.cache().prefetch(buffer_size=AUTOTUNE)
211
212     return norm_train_ds, norm_val_ds
213
214 # -----
215 #                               MAIN CODE
216 # -----
217
218
219 # ----- LOAD DATASET -----
220
221 # # Reads an image from a file, decodes it into a dense tensor, and resizes it
222 # # to a fixed shape.
223 # # def parse_image(filename):
224 # def parse_image(filename):
225 #     parts = tf.strings.split(filename, '.png')[0]
226 #     parts = tf.strings.split(parts, '_')
227
228 #     label = parts[-2:]
229 #     label = tf.strings.to_number(label)
230
231 #     image = tf.io.read_file(filename)
232 #     image = tf.io.decode_png(image)
233 #     # image = tf.image.convert_image_dtype(image, tf.float32)
234 #     # image = tf.convert_to_tensor(image, dtype=tf.float32)
235 #     return (image, label)
236
237 # train_ds = list_train_ds.map(parse_image)
238 # val_ds = list_val_ds.map(parse_image)
239
240
241 # train_x, train_y, val_x, val_y = get_data(data_dir, 0.8)
242 norm_train_ds, norm_val_ds = get_data(data_dir, outputs='bh', train_split=0.8)
243
244 val_batches = tf.data.experimental.cardinality(norm_val_ds)
245 test_ds = norm_val_ds.take((val_batches) // 2)
246 norm_val_ds = norm_val_ds.skip((val_batches) // 2)
247
248
249 # train_ds = tf.data.Dataset.from_tensor_slices((train_x, train_y)).batch(BATCH_SIZE)
250 # val_ds = tf.data.Dataset.from_tensor_slices((val_x, val_y)).batch(BATCH_SIZE)
251
252 # norm_train_ds = normalise_ds(train_ds)
253 # norm_val_ds = normalise_ds(val_ds)
254
255
256
257 # AUTOTUNE = tf.data.AUTOTUNE
258 # norm_train_ds = norm_train_ds.cache().prefetch(buffer_size=AUTOTUNE)
259 # norm_val_ds = norm_val_ds.cache().prefetch(buffer_size=AUTOTUNE)
260
261 # ----- CREATE MODEL -----
262
263 # MODEL CONFIG PARAMETERS
264 LEARNING_RATE = 0.000015
265
266 def r_squared(y_true, y_pred):
267     tss = tf.reduce_sum(tf.square(y_true - tf.reduce_mean(y_true)))
268     rss = tf.reduce_sum(tf.square(y_true - y_pred))
269     r_squared = 1 - (rss / tss)
270     return r_squared
271
272
273 model = create_regressor_model(INPUT_SHAPE, outputs=1, top='flatten')
274
275 model.compile(
276     optimizer=tf.keras.optimizers.SGD(learning_rate=LEARNING_RATE),
277     # loss=tf.keras.losses.MeanAbsoluteError(),
278     # loss=tf.keras.metrics.RootMeanSquaredError(),
279     loss=tf.keras.losses.MeanSquaredError(),
280     metrics=[
281         # tf.keras.metrics.R2Score(
282         #     class_aggregation=None,
283         #     num_regressors=0,
284         #     name='r2_score',
285         #     dtype=np.float32
286         # ),
287         r_squared,
288         tf.keras.metrics.RootMeanSquaredError(name='rmse'),
289         tf.keras.metrics.MeanAbsoluteError(name='mae'),
290         tf.keras.metrics.MeanSquaredError(name='mse'),
291     ],
292     loss_weights=None,
293     weighted_metrics=None,
294     run_eagerly=None,
295     steps_per_execution=None,
296     jit_compile=None,

```



```

297         pss_evaluation_shards=0,
298     )
299
300     #-----TRAIN MODEL-----
301
302     callbacks = [
303         keras.callbacks.TensorBoard(log_dir=log_dir, write_graph=False, write_images=False),
304         keras.callbacks.ModelCheckpoint(checkpoint_path,
305                                         verbose=1, save_weights_only=True,
306                                         save_best_only=True, monitor='val_mse',
307                                         mode='min'),
308         keras.callbacks.ReduceLROnPlateau(monitor='mse',
309                                           factor=0.2,
310                                           patience=5,
311                                           verbose=1,
312                                           mode='auto',
313                                           min_delta=0.0001,
314                                           cooldown=0,
315                                           min_lr=0.0000001),
316         # keras.callbacks.EarlyStopping(monitor='mse',
317         #                               min_delta=0.00001,
318         #                               patience=5,
319         #                               verbose=1,
320         #                               mode='auto',
321         #                               baseline=None,
322         #                               restore_best_weights=False),
323         keras.callbacks.TerminateOnNaN(),
324     ]
325
326     # model.load_weights('./THC_Logs/THC-2023-Oct-10-0444-34-3_Class_Test/TreeHealthClassifier_Best.h5')
327
328     history = model.fit(norm_train_ds,
329                        validation_data=norm_val_ds,
330                        epochs=60,
331                        verbose='auto',
332                        callbacks=callbacks,
333                        shuffle=True,
334                        initial_epoch=0,
335                        # validation_split=0.2,
336                        # steps_per_epoch=20,
337                        # validation_steps=None,
338                        # validation_batch_size=None,
339                        validation_freq=1,
340                        max_queue_size=10,
341                        workers=1,
342                        use_multiprocessing=False)
343
344     # model.build(input_shape=(BATCH_SIZE, IM_SIZE, IM_SIZE, 3))
345
346     model.evaluate(test_ds, verbose=2)
347
348     y_true = [] # store true labels
349     for [-, label] in test_ds.unbatch(): # use dataset.unbatch() with repeat
350         # append true labels
351         y_true.append([label])
352
353     model.evaluate(test_ds, verbose=2)
354     preds = model.predict(test_ds, batch_size=BATCH_SIZE)
355     metric = tf.keras.metrics.R2Score()
356     metric.update_state(np.array(y_true), np.array(preds))
357     result = metric.result()
358     print('Test_R2_Score: ', result.numpy())
359
360     # pd_conf_mat = pd.DataFrame([np.transpose(np.array(y_true)), np.transpose(np.array(preds))], columns=['Y_true', 'Y_pred'])
361     # print(pd_conf_mat)
362     # print(y_true)
363     # print(preds)
364
365     maxval = np.max([np.max(preds), np.max(y_true)])
366     plt.scatter(y_true, preds, c="blue")
367     plt.xlabel("Ground Truth")
368     plt.ylabel("Predictions")
369     plt.title("Test Results")
370     # plt.annotate("r-squared = {:.3f}".format(result.numpy()), (0, 1))
371     ax = plt.gca()
372     ax.set_xlim([0, maxval])
373     ax.set_ylim([0, maxval])
374
375     plt.savefig('scattestbh.png')
376
377     y_true = [] # store true labels
378     for [-, label] in norm_val_ds.unbatch(): # use dataset.unbatch() with repeat
379         # append true labels
380         y_true.append([label])
381
382     model.evaluate(norm_val_ds, verbose=2)
383     preds = model.predict(norm_val_ds, batch_size=BATCH_SIZE)
384     metric = tf.keras.metrics.R2Score()
385     metric.update_state(np.array(y_true), np.array(preds))
386     result = metric.result()
387     print('Val_R2_Score: ', result.numpy())
388
389     maxval = np.max([np.max(preds), np.max(y_true)])
390     plt.scatter(y_true, preds, c="blue")
391     plt.xlabel("Ground Truth")
392     plt.ylabel("Predictions")
393     plt.title("Validation Results")
394     # plt.annotate("r-squared = {:.3f}".format(result.numpy()), (0, 1))
395     ax = plt.gca()
396     ax.set_xlim([0, maxval])

```

```

397 ax.set_ylim([0, maxval])
398
399 # To show the plot
400 plt.savefig('scatvalbh.png')
401
402
403 y_true = [] # store true labels
404 for [-, label] in norm_train_ds.unbatch(): # use dataset.unbatch() with repeat
405     # append true labels
406     y_true.append([label])
407 model.evaluate(norm_train_ds, verbose=2)
408 preds = model.predict(norm_train_ds, batch_size=BATCH_SIZE)
409 metric = tf.keras.metrics.R2Score()
410 metric.update_state(np.array(y_true), np.array(preds))
411 result = metric.result()
412 print('Train_R2_Score:~', result.numpy())
413
414 maxval = np.max([np.max(preds), np.max(y_true)])
415 plt.scatter(y_true, preds, c="blue")
416 plt.xlabel("Ground_Truth")
417 plt.ylabel("Predictions")
418 plt.title("Training_Results")
419 # plt.annotate("r-squared = {:.3f}".format(result.numpy()), (0, 1))
420 ax = plt.gca()
421 ax.set_xlim([0, maxval])
422 ax.set_ylim([0, maxval])
423
424 plt.savefig('scattrainbh.png')

```

Listing E.3: Final Tree Parameter Regressor Model.