Modern Computer Vision Applications for Plant Phenotyping in Agriculture

by

Vaishnavi Thesma

(Under the Direction of Javad Mohammadpour Velni)

Abstract

The rapidly growing world population challenges farmers to meet the rising food demand. Monitoring crop phenotypes, or the physical plant traits, is useful in tracking plant development, maintaining plant health, and increasing yield. However, phenotyping efforts are traditionally manual and become tedious for large scale farms. Thus, it is imperative to develop autonomous solutions to monitor plants accurately, remotely, and timely. To meet this objective, computer vision techniques have been used by researchers to perform automatic plant phenotyping on video and image data collected from either indoor, controlled environments or from the field. Furthermore, these methods have focused on using traditional pixel-based processing, machine learning, and deep learning for plant phenotyping. In this study, various modern computer vision techniques are implemented to automatically phenotype plants for agriculture applications, thereby reducing manual labor while accurately detecting important traits to help increase yield.

Index words: Plant Phenotying, Crop Monitoring, Image Processing, Machine Learning, Deep Learning

Modern Computer Vision Applications for Plant Phenotyping in Agriculture

by

Vaishnavi Thesma

B.S., University of Georgia, 2020

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree

Master of Science

Athens, Georgia

2022

Modern Computer Vision Applications for Plant Phenotyping in Agriculture

by

Vaishnavi Thesma

Major Professor: Javad Mohammadpour Velni

Committee: Glen C. Rains
Mark Trudgen

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
August 2022

Acknowledgments

I would like to first thank my advisor, Dr. Javad Mohammadpour Velni, for his guidance, encouragement, and support in my graduate studies. I would also like to thank my committee members, Dr. Glen C. Rains and Dr. Mark Trudgen, for their guidance in my research and coursework. I am sincerely grateful to my instructors for their teachings and aid throughout my degree. I am also appreciative of lab mates for their help and good times together in the lab. Lastly, my wholehearted thanks goes to my family and friends for their continuous support, encouragement, and faith in me pursuing my goals.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

The world population is expected to increase to nearly 10 billion people by 2050, requiring farmers to produce up to 70% more food and crop to meet the rising demand (Hunter et al., 2017). However, farmers are facing many economical, environmental, and biological challenges to effectively meet this need due to expensive harvesting tools, unskilled workforce, reliance on manual labor, lack of arable land, water scarcity, and disease prevalence. Thus, researchers are interested in developing reliable, accurate, informative, and autonomous solutions to help farmers maintain crop health, strategize methods to treat fields, and increase yield (Chandra et al., 2020).

Specifically, crop monitoring and data collection from fields is an effective method to address these challenges and manage farms. Observing and tracking plant phenotypes, known as plant phenotyping, provide clear indications of plant health, development, and yield potential. Plant phenotyping involves characterizing the physical traits of plants based on their genetic constitution and environmental influences over time. The physical traits of plants include both above soil phenotypes, such as leaves, flowers, and fruits, and below soil phenotypes, such as plant root morphology (Li et al., 2020). Understanding plant phenotypes allow for farmers to make informed decisions to manage their crops to monitor plant health, reduce disease impact, or optimize treatments to increase yield

Historically, plant breeders have manually selected desired qualities of plants based on observed phenotypes during plant growth. These desired plants were cultivated and their seeds were used to grow future generations of the same plant to maintain desired phenotypes (Pieruschka & Schurr, 2019). Furthermore, conventional plant phenotyping beyond observation included invasive measures such as hand-picking or plant uprooting, which is destructive for plants. Moreover, manual crop monitoring in large-scale farms is not reliable or effective as it is arduous, time-consuming, lacks scalability, and may be invasive to the crop. Thus, it is necessary to use autonomous solutions that can effectively phenotype plants accurately, reliably, timely, and with high-throughput.

In the past three decades, researchers have applied computer vision techniques to achieve automatic plant phenotyping for above and below soil traits. Computer vision methods typically use video and image data collected from cameras in indoor, controlled environments or fields for processing to extract meaningful information. In the scope of plant phenotyping, computer vision methods can be used to

extract valuable information on crops such as leaf, fruit, or weed count, root system architecture, or disease segmentation. These techniques have developed in complexity over time and have included pixel-based image processing, machine learning, deep learning, or a combination of these methods (Chandra et al., 2020).

The first use of imaging systems for plant phenotyping began in the 1990s using digital cameras to measure single features, such as leaf growth or root length over time (Pieruschka & Schurr, 2019). For example, the authors in (Schmundt et al., 1998) used a charged coupled device (CCD), a type of traditional digital camera, for phenotyping the growth of dicot leaves in a lighting controlled, indoor environment and processed each image over several minutes. Also, the authors in (Biskup et al., 2007) developed a stereo imaging system for phenotyping spatial orientation of plant canopies. While these traditional imaging systems allowed for a breakthrough in autonomous plant phenotyping, they were unable to process vast amounts of data quickly, used low-level mathematical image processing, and used low resolution images due to high hardware and software costs. Thus, these challenges must be addressed for high-throughput plant phenotyping.

The recent surge and affordability of high quality RGB, multi-spectral, and hyper-spectral cameras, unmanned aerial and ground vehicles, storage capabilities, and open source software in the late 2000s to early 2010s have resulted in large collections of plant data for processing (Chandra et al., 2020). As such, it is necessary to process the vast amounts of data autonomously using machine learning, a sub-field of artificial intelligence where computer system learn from data without explicitly being programmed, instead of simple imaging systems and slow processing. There are three main types of machine learning algorithms: supervised, semi-supervised, unsupervised, and reinforcement learning.

Supervised learning involves feeding the machine learning algorithm information about the training data, such as extracted features or class labels, to help the model learn to make correct predictions on specific data. Semi-supervised learning only use partially labelled data, whereas in unsupervised learning, models are not fed labelled data and must learn to identify patterns independently. Reinforcement learning models learn from its environment via trial-and-error and is rewarded if the experience is favorable or not. For plant phenotyping, supervised learning methods are popularly used.

Plant phenotyping using supervised machine learning typically follows three steps: (1) data collection, (2) data processing for feature extraction, and (3) training a classification or regression algorithm to correctly predict the desired plant phenotype. Various algorithms have been developed for supervised machine learning including support vector machines (SVM), Bayesian neural networks (BNN), artificial neural networks (ANN), and k-nearest neighbors (k-NN) as discussed in (Singh, Thakur, et al., 2016). These methods utilize labelled information to make predictions or classifications for plant phenotyping (Singh, Ganapathysubramanian, et al., 2016). For example, the authors in (Behmann et al., 2014) used SVM to predict effects of drought in hyper-spectral images of barley plants. Additionally, the authors in (Bauer et al., 2011) used k-NN and BNN classify disease prevalence in RGB and multi-spectral images of sugar beet plants.

While these methods have addressed processing vast amounts of data autonomously and with improved algorithmic performance in terms of speed and efficacy, these methods still rely on feature extrac-

tion being done manually to highlight the phenotypes of interest. Specifically, the collected video and image data must be processed in a streamlined fashion to extract important plant features of interest, such as leaves, fruits, flowers, or disease, for automatic plant phenotyping. Thus, feature extraction methods must be done correctly to ensure proper, high-throughput phenotyping, else the machine learning models fail to perform well to new forms of plant data including different plants, environments, and lighting. The lack of robust feature extraction resulted in the emergence of deep learning (Chandra et al., 2020).

Deep learning, a subset of machine learning, addresses the manual feature extraction issue automatically and can be used for large scale data. The recent surge of graphical processing unit (GPU)-computing has dramatically increased the computational capabilities of computer vision. Supervised deep learning models extend machine learning models and are used for object detection, counting, classification, semantic and instance segmentation, and scene understanding. This is possible since deep learning models contain layers of ANNs where nonlinear transformations or processing is done on input data. The layers of ANNs are used for automatic feature extraction and pattern learning, but may require extensive hyperparameter tuning for good model performance.

Deep learning models use convolutional neural networks (CNN), initially developed by (Fukushima, 1988) in the late 1980s but not yet widely adopted, for learning spatial structure in visual input data. CNNs contain several layers of convolutional layers, pooling layers, or fully connected layers. Convolutional layers transform images into feature maps. Pooling layers typically follow convolutional layers by reducing their spatial dimensionality and maintaining important extracted features. Fully connected layers usually follow a stack of convolutional and pooling layers in CNN and are used to convert the extracted feature maps into a single dimensional feature vector that can be fed forward for classification. These layers together serve as a mapping of input to output by transforming visual input data to a feature vector (Chandra et al., 2020). Each deep learning model combines each of these layers differently unique to applications and performance goals.

The process of training deep learning models differs, however, to traditional machine learning models where there are typically two main steps: (1) data collection and labelling, and (2) model training for predictions. For plant phenotyping, this process is useful for variety of plants grown in either indoor, controlled environment or in the field since the model itself will learn to extract features automatically provided the labelled input data. For example, the authors in (J. Ubbens et al., 2018) used CNN for counting leaves in rossette plants and used publicly available plant datasets in indoor, controlled environment. Furthermore, the authors in (Aravind et al., 2018) also used CNN and transfer learning for disease detection of grape images.

While deep learning models have gained popularity for phenotyping large datasets, acquiring and labeling large datasets is not always possible due to limitations in storage, costs, time, and required expertise in manually labelling the collected data. Also, training deep learning models on smaller datasets may result in poor performance or model overfitting, thus lacking generalizability. Thus, data augmentation methods, such as image transformations and generative models, can be used for limited datasets to increase the size and quality of datasets to avoid model overfitting. Also, some generative models, like generative adversarial networks (GAN), allow for the introduction in variations in training data to increase gener-

alizability of some models for introductions in variations in data (Sampath et al., 2021). Lastly, transfer learning methods can be used for limited datasets and pretrained models being retrained on a different, but related task (Chandra et al., 2020).

The motivation of this thesis is implementing modern computer vision techniques for applications in plant phenotyping in agriculture. Specifically, the studies are conducted using various plant images such as *Arabidopsis thaliana* leaves and roots, and cotton field images for phenotyping. Additionally, the datasets used in this study are collected from both indoor, controlled environment and from the field. This thesis contains six chapters. The first chapter provides an introduction and motivation for this research. Chapter 2 presents real-time leaf counting using state-of-the-art deep object detection networks deployed onto a ground vehicle and transfer learning. Chapter 3 provides an approach to create an offline spatio-temporal map of cotton bloom appearance using a combination of state-of-the-art deep object detection networks, transfer learning, and image post-processing. Chapter 4 examines classifying and tracking nutrient deficient cotton plant leaves over time using support vector machine. Chapter 5 addresses semantically segmenting extreme pixel-imbalance plant root images using conditional generative adversarial networks for data augmentation and a state-of-the-art deep segmentation network. Lastly, the final chapter concludes the works presented in this thesis and acknowledges potential future works.

Chapter 2

Real-time Plant Leaf Counting on Ground Vehicle Using Deep Learning

2.1 Introduction

The world population is expected to increase to nearly 10 billion people by the year 2050 (Davoodi et al., 2018). Farmers are pressured to increase their food and crop by at least 70% to meet the rising demand. However, there are many challenges in meeting this demand such as water scarcity, lack of arable land, reliance on manual labor, expensive harvesting tools, an unskilled workforce, and even the prevalence of crop disease. Thus, it is imperative for researchers to develop autonomous solutions that address these challenges to meet the rising demand.

Crop monitoring efforts and data collection methods have proven to be effective in addressing these issues (Darwin et al., 2021). Crop monitoring efforts involve plant phenotyping, which is the characterization of a plant's physical traits based on their genetic background and environmental influences throughout the plant's lifetime. Researchers are interested in both above soil phenotypes, including leaves, flowers, and fruits, and below soil phenotypes, such as the root system architecture. Understanding plant phenotypes will allow farmers to manage their crops to monitor plant health, reduce disease impact, and optimize treatments to increase yield.

Plant leaf phenotyping, in particular, provides much insight in the plant's overall health such as its growth and development, flowering time, and yield potential (Koornneef et al., 1995; Walter & Schurr, 1999). However, it is difficult to manually count plant leaves are there may be many leaves per plant. Furthermore, there may be occlusion issues from layers of plant leaves or even illumination variations. Additionally, deep learning methods for plant phenotyping cannot be deployed onto smaller hardware than desktop computers and graphical processing units (GPUs) for near real-time processing. This issue forces researchers to perform model training and processing offline. Thus, it is imperative to develop

a solution where autonomous image capture and processing with deep learning on-board a deployable robot is possible.

The objectives of this work is to use a deep learning model to perform accurate leaf detection, localization, and counting in real-time and deploy the trained model onto an unmanned ground vehicle for near real-time leaf counting.

This chapter refers to the research conducted in (Buzzy et al., 2020). The remaining sections of this chapter are as follows: section 2 describes deep object detection networks and the two state-of-the-art models used in the methodology; section 3 details the methodology used to conduct the experiments for this work; section 4 presents the experimental results based on the methodology; section 5 describes a robotic implementation where the best performing model is deployed on an unmanned ground vehicle for near real-time leaf counting; section 6 acknowledges potential lines of future work; and section 7 concludes the work presented in this chapter.

2.2 Background

The task of counting the number of leaves in the scope of machine learning may belong to one of two categories (Dobrescu et al., 2017): (i) learning a direct image-to-count regression model (Giuffrida et al., 2016); or (ii) obtaining a per-leaf detection and segmentation (Scharr et al., 2016).

Counting via direct regression methods: In these methods, the deep convolutional neural networks are used to integrate image feature extraction with regression in a single pipeline. For example, the authors of (J. R. Ubbens & Stavness, 2017) introduced Deep Plant Phenomics, an open-source deep learning tool, that implements deep convolutional neural networks for the leaf counting, mutant classification, and age regression from top-down images of plants. Additionally, the authors of (Giuffrida et al., 2018) proposed Pheno-Deep Counter, which is a multi-input deep network, that combines information coming from different imaging sources and can predict leaf count in rosette-shaped plants.

Counting via object detection and segmentation methods: Object detection algorithms operate by simultaneously preforming object classification and localization. Deep object detection networks utilize convolutional neural networks (CNNs) that perform automatic feature extraction using layers of convolutional, pooling, and fully connected layers. These object detection networks have popularly been used for accurate plant phenotyping. Moreover, these networks can perform object detection in a single-pass or use multiple networks in conjunction with one another. Furthermore, object detection networks offer superior accuracy as they can perform individual leaf counting.

Several works implement object detection or segmentation networks to address the leaf counting problem. For example, You Only Look Once (YOLO) (Redmon et al., 2016) detect objects quickly but struggle with densely packed groups of objects. Conversely, networks like Region-based Convolutional Neural Network (R-CNN) (S. Ren et al., 2015) are slower but can more easily discern tight groups of objects; however, the amorphous shapes of leaves may result in double detection of a leaf.

On the other hand, the authors in (M. Ren & Zemel, 2016) use Recurrent Neural Network (RNN) architecture with an attention mechanism to compute instance segmentation jointly with counting. The performance of the method was shown on the CVPPP plant leaf dataset (Minervini et al., 2016), as one of the instance segmentation benchmarks. Additionally, the authors in (L. Xu et al., 2018) used Mask R-CNN for leaf segmentation and counting. Lastly, the authors in (Kuznichov et al., 2019) used data augmentation methods to build their training set for a Mask R-CNN network.

In this work, we consider using deep object detection networks to address near real-time leaf counting on robot.

2.3 Methodology

2.3.1 Plant selection

The candidate plant for our dataset was *Arabidopsis thaliana* because of its unique phenotypic traits during growth as well as its robustness to growing in colder environments (Vongs et al., 1993). Specifically, the candidate plant grows easily, quickly, and without high expense. Thus, collecting large amounts of data on a fast-growing plants is feasible and will result in variability in the dataset. In this application, we grew our dataset within a few weeks.

Additionally, the *Arabidopsis thaliana* plants are tolerant to cold temperatures. This tolerance indicates that the plant will continue to grow in indoor environment with fixed temperature. Thus, the selection of a robust plant allows us to build our dataset without greenhouse environment.

2.3.2 Dataset acquisition

We grew our *Arabidopsis thaliana* plants indoor under red/blue LED grow lamps as shown in Figure 2.6. A total of 60 plants were grown in 10×6 batches. The grow lamps were kept on daily for 24 hours to speed up the growth rate. Additionally, the plants were watered every other day during weekdays. Once the first leaves began to appear, the plants were photographed with a Canon Rebel XS camera and later stored as JPG format. The Canon camera was used to build both the training and test datasets to feed into deep object detection networks for lea detection and counting.

Our data collection period spanned four weeks, where we captured top-down view images of the plants twice every weekday. This ensured that each photograph contained distinct plant position and rotation. We individually photographed each plant about six inches away from the camera. We set the camera settings as follows: 1/5", F5.6, ISO800, and the camera was manually focused. The camera setup for our data acquisition methods is shown in Figure 2.1.

We grew another group of plants separately using the same methodology to create the evaluation dataset. We used a different group of *Arabidopsis thaliana* plants than the training dataset to introduce more variability and ensure our deep object detection networks were not overfitted to the plants from the

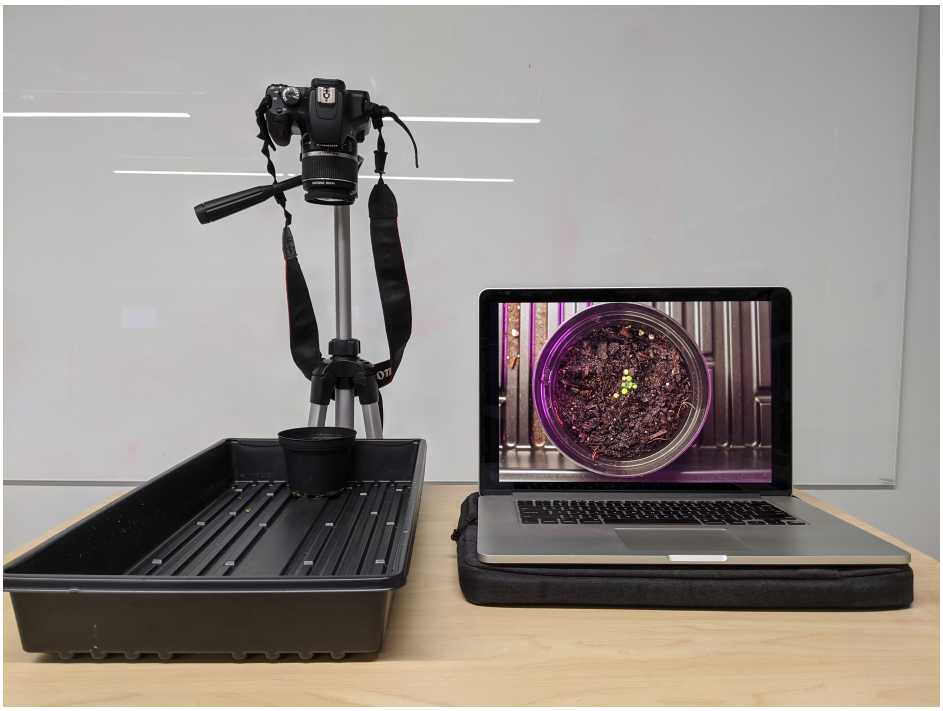

Figure 2.1: Data acquisition setup.

training dataset. Thus, by using a new batch of plants for the testing dataset, we ensured that any bias from the plants in the training dataset can be avoided.

2.3.3 Image Pre-processing

For each image collected, we pre-processed them the same day to account for any adjusted settings during the data collection period.

Specifically, each image was first cropped to increase the size of the leaves with respect to the entire image. We proportionally cropped each image per data collection batch and reduced the cropping factor as the leaves naturally grew larger in each image. For example, we set the cropping factor for the first batch of images to 0.5 on both the x and y axes as the leaves were very small and young. The last batch of images contained large, mature leaves, as such, we did not crop these images at all.

After cropping each image, we padded each image with black pixels, of value 0, to ensure that it is square in shape.

Lastly, we resized each image to dimensions 410×410 using a Python script and *OpenCV*.

2.3.4 Data Labeling

After our pre-processing methods, we proceed to label our images. These labelled images will be fed into our deep object detection networks for feature extraction and classification.

We labelled our pre-processed images using OpenLabeler and the bounding boxes of each label were saved in a corresponding XML file in the popular VOC formatting. Thus, each image has a corresponding XML file. The versatility of VOC formatting allows for seamless conversion into different formats for different object detection networks.

We drew bounding boxes around every visible leaf. If there was uncertainty in labelling a leaf, we defaulted to only drawing bounding boxes that would result in fewer leaves rather than more. After we labelled all the images, we generated an additional text file that contains every annotated images' bounding boxes' coordinates. An example of our labelling method and text file is seen in Figure 2.2.

Object-Class	x	y	width	height
0	0.261	0.376	0.205	0.122
0	0.409	0.361	0.1	0.137
0	0.329	0.501	0.137	0.095
0	0.389	0.584	0.124	0.188
0	0.532	0.643	0.107	0.183
0	0.521	0.512	0.071	0.063
0	0.594	0.457	0.105	0.085
0	0.627	0.354	0.171	0.141
0	0.495	0.31	0.078	0.078

Figure 2.2: One sample of plant and its annotation from our generated dataset.

Our final training dataset contains 1,000 labelled images of *Arabidopsis thaliana* plants and several thousands of labelled leaves. Our evaluation set contains a total of 36 labelled *Arabidopsis thaliana* plant images.

2.3.5 Model Selection

For our experiments, we consider two popular state-of-the-art deep object detection models, namely YOLO (You Only Look Once) and Faster Region-based Convolutional Neural Network (Faster R-CNN).

YOLO is a single object detection and network that performs localization and identification in a single pass (Redmon et al., 2016). YOLO performs object detection by spliting up an input image into an $S \times S$ grid. For each cell in the grid, the YOLO proposes a potential bounding box, class probability map, and a confidence score for how certain that the box contains an object. The output of YOLO is a proposed bounding box with a confidence score value and the class label that coincides with that bounding box on the class probability map. The YOLO architecture we used for leaf counting is shown in Figure 2.3.

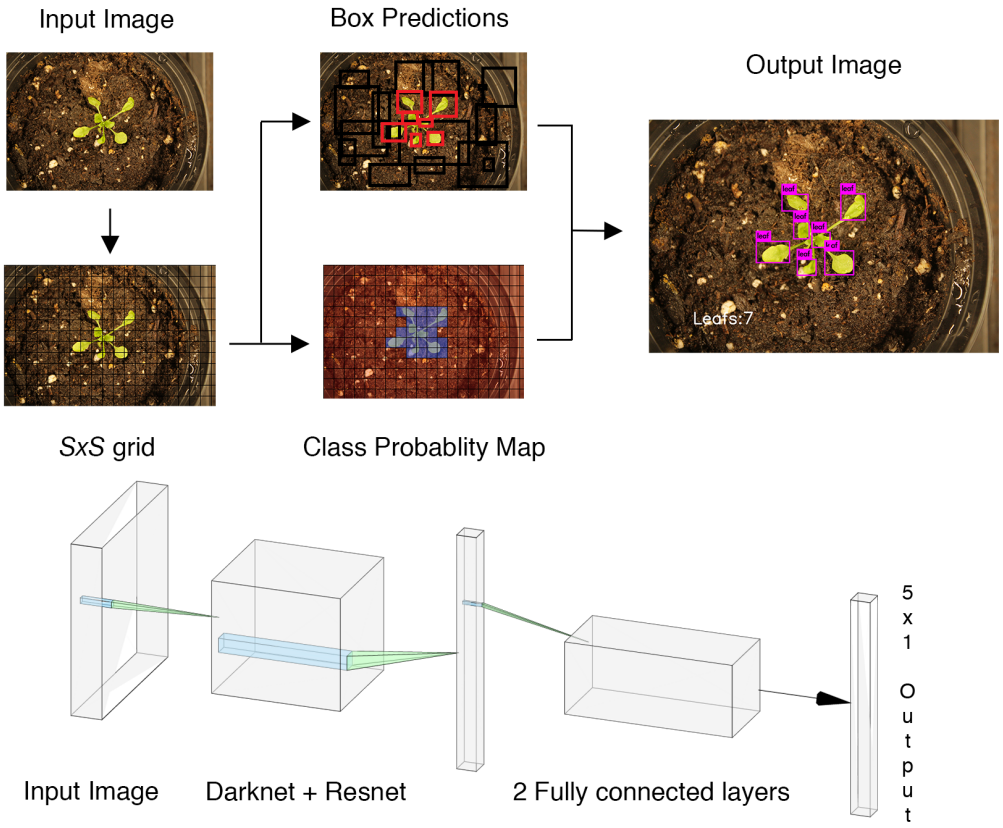

Figure 2.3: Our YOLO framework leaf counting. The YOLO network takes an image input into a deep convolutional neural network and outputs the leaf detection, where the bottom part of the figure consists of a diagram of the YOLO network architecture.

For our experiments, we chose the third generation of the YOLO architecture, namely YOLOv3 (Redmon & Farhadi, 2018a). The choice of YOLO version is significant as previous generations of YOLO showed poor performance in detecting densely clustered objects, like leaves(Redmon et al., 2016). YOLOv3 has superior performance in comparison to its predecessors as it creates three finer mesh grids instead of one. Finer mesh allows for improved detection performance for densely clustered objects, like our leaf dataset.

Moreover, we chose to use tiny-YOLOv3 (Yi et al., 2019) as this version uses fewer layers than the full third generation YOLO architecture. The use of fewer layers allows for the model to perform object detection and localization faster as a slight cost of accuracy. Additionally, since tiny-YOLOv3 has fewer layers than the full YOLO archiectue, it can be deployed on lower end hardware and makes implementation both accessible and versatile. For our objectives, this versatility is imperative for deploying a deep object detection model on an unmanned ground vehicle for near real-time leaf counting.

On the other hand, the Faster R-CNN network has two main parts including the Region Proposal Network (RPN) and Fast R-CNN detector (S. Ren et al., 2015). The RPN is a full convolutional network that generates object proposals on an input image without using selective search. The selective search method was used in both R-CNN and Fast R-CNN to generate regions of interests and was computationally expensive (Girshick, 2015; Girshick et al., 2014). As such, Faster R-CNN shows increased computational speed to generate region proposals. The region proposals from the RPN is fed into the Fast R-CNN detector for classification. Faster R-CNN benefits for classification tasks with densely clustered objects, like leaves.

Since tiny-YOLOv3 is a smaller, single network that does localization and identification, we expect Faster R-CNN to have slower training and inference time, while having comparable performance in accuracy.

2.3.6 Training Procedure

Our tiny-YOLOv3 model was trained for a total of 160,000 batches for two days. We preformed training using a batch size of 24 with subdivisions of 8 to accommodate for our low GPU memory. Other hyperparameters include a momentum of 0.9, weight decay of 0.0005, burn in of 1000, and a learning rate of 0.001.

For Faster R-CNN, we trained the RPS and Fast R-CNN detector separately to ensure accurate performance. The RPN was first trained for 50 epochs for 1000 iterations per epoch. Then, the RPN proposals were used to train the Fast R-CNN network for 50 epochs for 500 iterations per epoch. The total training time was approximately 2.5 days. We used the default training hyperparameters as this Github repository.

2.3.7 Implementation Details

To train and evaluate our tiny-VOLOv3 model, we use Darknet deep learning framework (Redmon, 2013–2016). Darknet is an open source framework written in C, making it fast and portable on any device with C compile. Additionally, we use a Python script and *OpenCV* to overlay the total number of leaves counted per image by our trained mode.

To train and evaluate our Faster R-CNN model, we use Keras, another popular open source deep learning framework built upon Tensorflow.

We trained all of our models on a Quadro P2000 computer with 5GB of GDDR5 memory. The CPU is an Intel CORE i7-7800x with 32 GB of memory.

2.3.8 Evaluation Metrics

We evaluate the performance of our models and their effectiveness in leaf counting using the following evaluation metrics used in (Giuffrida et al., 2016):
(i) Difference in count (DiC): $\frac{1}{N}\sum_{i=1}^{N}\epsilon_i$;
(ii) Absolute difference in count (|DiC|): $\frac{1}{N}\sum_{i=1}^{N}|\epsilon_i|$;
(iii) Mean squared error (MSE): $\frac{1}{N}\sum_{i=1}^{N}\epsilon_i^2$;
(iv) Percentage agreement (%): $\frac{1}{N}\sum_{i=1}^{N}\mathbf{1}[\epsilon_i = 0]$;

where $\epsilon_i = y_i - \hat{y}_i$ is the difference between the ground truth and algorithmic prediction and $\mathbf{1}[\cdot]$ is the indicator function, which returns zero if the error $\epsilon_i \neq 0$, otherwise returns one (Valerio Giuffrida et al., 2019).

Figure 2.4: The leaf counting output.

Ideally, our object detection models will produce bounding boxes around each visible leaf in each image. The total number of bounding boxes must match the total number of leaves present in the image. The output bounding boxes may be classified as one of the following: true positive (*TP*)-correctly classifying an object as a leaf; false positive (*FP*)-incorrectly classifying a background object as a leaf or multiple detections of the same leaf; and false negative (*FN*)-incorrectly classifying a leaf as a background object.

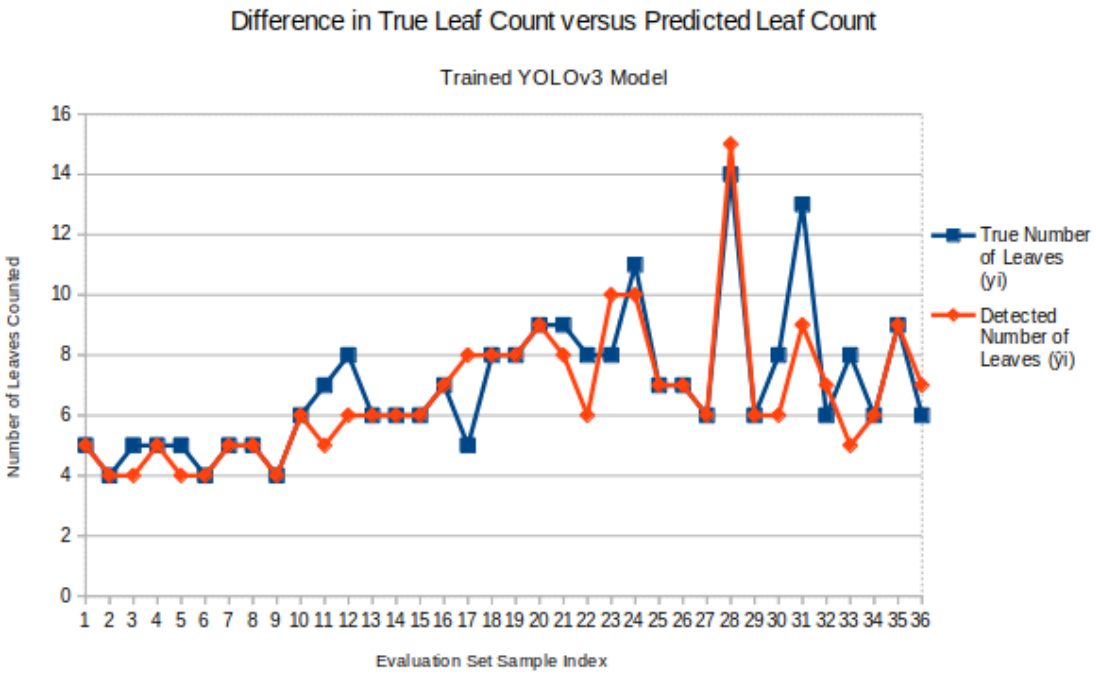

Figure 2.5: Scatter plot comparison between true leaf count vs estimated leaf count from YOLO model.

We quantify the values *TP*, *FP* and *FN*, using the average precision (AP) metric over the intersection over union (IOU) threshold of 0.5. The IOU metric is used to determine the intersection area of predicted and ground truth bounding boxes divided by the union area, which quantifies how close the predicted results are to ground truth labels. In this experiment, we consider a threshold above 0.5 on IOU as a good detection, while under 0.5 is considered as a poor detection. This metric is also known as the area under the recall-precision curve (Sokolova et al., 2006), where

$$\text{precision} = \frac{TP}{TP+FP}, \tag{2.1}$$

$$\text{recall} = \frac{TP}{TP+FN}. \tag{2.2}$$

Additionally, we visually compare our trained tiny-YOLOv3 model's performance on how well it detected leaves by using a scatter plot, as seen in Figure 2.5. Each colored point represents the number of leaves detected (red) and the true number of leaves present (blue) for each image from the evaluation dataset. The scatter plot compares the true number of leaves (blue) and the detected number of leaves (red) for each image in the evaluation dataset. We connected each point with a line to clearly see where the blue and red points overlap, indicating that our trained model was able to accurately identify the true number of leaves. If the red line, which represents the algorithmic prediction, does not align with the

blue line, which is the ground truth, then our model may have overestimated or underestimated the true leaf count.

In addition to computing precision and recall, we calculate the average accuracy and precision of the models' performances, where

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}, \tag{2.3}$$

$$\text{Precision} = \frac{TP}{TP+FP}. \tag{2.4}$$

Using the precision and recall metrics above, we can calculate F1 score of each model. The F1 Score is another metric of accuracy for deep learning models. Ideally, the value of F1 Score should be high and close to 1. Below is the formula for evaluating F1 Score in terms of precision and recall:

$$\text{F1 Score} = \frac{2*precision*recall}{precision+recall}. \tag{2.5}$$

Additionally, we also calculate the true positive rate (TPR) and false positive rate (FPR) for both of our trained models. These metrics provide more insight to how well the trained model can correctly identify and localize the leaves in an image, and how often double detection or falsely identifying background as leaves occur, respectively. Ideally, the TPR should be very close to 100% and the FPR should be low and close to 0%.

Finally, we compute the average inference time, in seconds, that our trained models take to process one image. However, our the time to load and initialize the model is not considered when evaluating the average inference time.

The whole pipeline of our proposed architecture is illustrated in Figure 2.6.

2.4 Experimental Results

2.4.1 Final Results and Comparison

The above evaluation metrics obtained using our platform are summarized in Table 3.1, where compare the results of the trained tiny-YOLOv3 model with Faster R-CNN. We also used the same Quadro P2000 desktop (that was also employed for training) to obtain the results shown in Table 3.1.

The results of tiny-YOLOv3 indicate that there is better overall real-time counting of leaves, as seen in Figure 2.4. Specifically, using our trained tiny-YOLOv3 model, a lower mean-squared error (MSE), higher F1 Score, higher TPR, and lower FPR is achieved compared to Faster R-CNN. While Faster R-CNN provided a higher AP score, which is a good indication of object detection and localization, tiny-YOLOv3 has a significantly lower $|DiC|$ value. Moreover, this demonstrates that tiny-YOLOv3 has higher accuracy and less false positive classifications than Faster R-CNN. Most importantly, our trained tiny-YOLOv3 model can detect and localize leaves nearly 100 times faster than Faster R-CNN.

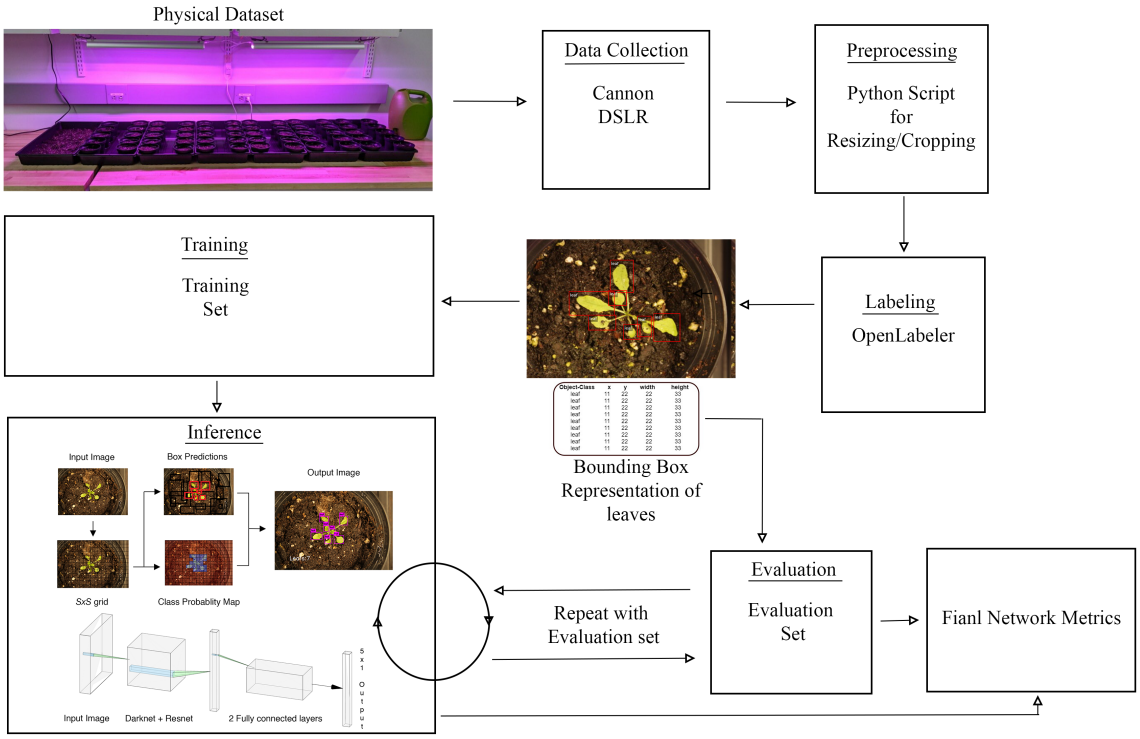

Figure 2.6: The general pipeline of our proposed architecture from generating the dataset to training the model.

As such, we chose our trained tiny-YOLOv3 model to deploy on an unmanned ground vehicle to perform near real-time leaf counting on low-cost hardware.

2.4.2 Transfer Learning

In addition to training the tiny-YOLOv3 network to detect *Arabidopsis thaliana* leaves, we also implemented a transfer learning method using tiny-YOLOv3. Transfer learning refers to a machine learning method where a model trained on one problem, called the source task, is used to solve a different but related problem, called the target task. In this application, the source task is to detect smaller leaves that were grown in the plants' early stages and the target task is to detect larger leaves that were grown in the plants' later, more mature stages. The goal of transfer learning is to limit the time in retraining a model from scratch as the overall domain, the *Arabidopsis thaliana* plant leaves, changed in size over time as the plant grew. This way, we are able to utilize the already learned features of the source task and apply them to the target task with minimal training time.

Table 2.1: Network evaluation metrics.

Metric	Tiny-YOLOv3	Faster R-CNN
DiC	0.25	0.0556
|DiC|	0.8056	1.2778
MSE	2.0833	2.8889
%Agreement	56%	27.78%
AP (@.5)	0.583	0.600
Accuracy	0.88846	0.83088
Precision	0.97059	0.91129
F1 Score	0.94467	0.89866
TPR	91.304%	90.4%
FPR	24.138%	47.826%
Inference time (s)	0.009225	0.917535

To accomplish transfer learning, we first partitioned our original dataset into two main domains: the source domain and target domain, organized by timestamp. The source domain dataset is used to detect and count small, young leaves. The target domain dataset is used to detect and count large, mature leaves. The source domain contained a total of 600 images that were further divided into a training and testing set, with 480 and 120 images, respectively. Similarly, the target domain contained a total of 100 images divided into a training and testing set, with 80 and 20 images, respectively.

The source domain's training set was trained for a total of 160,000 batches, with a batch size of 24, subdivisions of 8, momentum of 0.9, weight decay of 0.0005, burn in of 100, and a low learning rate of 0.001. The evaluation metrics obtained from using this platform are shown in Table 2.

Table 2.2: Source domain network evaluation metrics.

Metric	Tiny-YOLOv3
|DiC|	0.575
MSE	1.075
TPR (%)	93.4%
FPR (%)	11.7%
F1 Score	0.961

Next, the source domain's trained model was tested on the 100 target training images to validate that the two source and target domains are different but still similar. Additionally, the source model should result in slightly worse results in the target training set if the two domains are different. The evaluation metrics obtained from using this platform are shown in Table 3.

Based on the results in Tables 2 and 3, it is observed that there is a slight reduction in accuracy and performance between the trained source model on its own test data and the target training data. Specifically, the $|DiC|$ is nearly doubled and the false positive rate (FPR) is also doubled. Also, the F1 Score has

Table 2.3: Network evaluation metrics for determining differences between both source and target domains.

Metric	Tiny-YOLOv3
|DiC|	0.938
MSE	1.788
TPR (%)	91%
FPR (%)	23%
F1 Score	0.94

decreased slightly. This signifies that the source and target domains are, in fact, slightly different but still similar. Thus, we can proceed to retrain and fine-tune the source model to perform better when presented with larger and more mature leaves.

The target domain's training set was trained for an additional 10,000 batches on the already trained source model, for a total of 170,000 batches. We chose to retrain all layers of the source model on the target training set without freezing any layers. This allows for the model to have more flexibility to improve its overall accuracy while not being trained for a long period of time. The total retraining time was less than 2 hours. The evaluation metrics obtained from using this platform are shown in Table 4.

Table 2.4: Target domain network evaluation metrics.

Metric	Tiny-YOLOv3
|DiC|	1.15
MSE	1.15
TPR (%)	87%
FPR (%)	5%
F1 Score	0.93

From the results given in Table 4, it is observed that there is a significant reduction in the mean-squared error (MSE) and FPR. This indicates that the model that was trained on the target domain via transfer learning results in better detection of leaves. Thus, we can see that transfer learning does indeed result in better detection and localization of large, older leaves when using a model trained on smaller, young leaves. As such, training time can be significantly reduced even if the dataset is modified with the addition of new and similar images, since the features already learned from the source model can be used.

2.5 Robotic Implementation

To deploy the tiny-YOLOv3 model to a low-cost platform, we use an Honor 7x Android phone and attach it to an iRobot Create2 ("iRobot Create2", n.d.), as shown in Figure 2.7. The robot follows a predefined path towards a plant and the Android phone is responsible for capturing the images with its on-board

camera and preforming leaf counting. The inference time for the deployed network was 5 seconds per image. Thus, this experimental setup shows the versatility of using Darknet and Tiny-YOLOv3, as our network is able to be effectively deployed on a very low-cost robot to preform a near real-time analysis on several plants arranged in a typical row pattern that would be found in a greenhouse setting. This demo shows a proof of concept of deploying tiny-YOLOv3 for on-board plant leaf counting in near real-time and can be extended for processing several plants. We also acknowledge that this robot implementation was not used for creating the training and testing datasets, rather was used to deploy the completed experiment in a controlled environment.

Figure 2.7: Robotic implementation of deployed tiny-YOLOv3 model on Android Honor 7x phone and iRobot2 for data acquisition and real-time data processing.

2.6 Discussion

In this work, an autonomous robotic platform was developed to detect and count the number of leaves in an image in near real-time. However, the proposed approach has several potential improvements that can be made to develop an intelligent phenotyping system.

Firstly, the growth conditions of the *Arabidopsis thaliana* plants training and testing sets were identical. Specifically, all of the plants were grown in an indoor, controlled environment under LED lamps and were all watered at the same frequency. As such, there are limited variations in light reception, dryness, insect infestation, and nutrition among each plant. Thus, the model is biased towards the plants having constant and identical growing conditions. Increases in plant variation may be seen if more plants were grown in both indoor and outdoor lighting or with different mineral nutrition. The presence of variations in the datasets would force the model to try to detect potential differences in leaf shape, count, and even damage from dryness. As such, the model would be subject to further performance analysis from these variations.

Secondly, we only tested one type of plant, *Arabidopsis thaliana*, in this experiment. These plants are characterized by their small, round leaves that grow outward. Due to these phenotypes, the leaves are easy to localize. If a different plant species was used in this experiment that had different leaf shapes, sizes, clustering, and growth behavior, then the trained model would also be subject to further performance analysis on the model's ability for learning different plants' leaves (Gao et al., 2020).

Thirdly, only a limited amount of data was used to train the deep object detection networks. Additional data can improve overall model performance and generalizability. However, acquiring more data by growing more plants may not be possible. Thus, implementing data augmentation techniques, such as traditional image transformations like flipping and rotation or generative models, can drastically increase the size of a dataset to improve model performance. A significant increase in training data may result in the trained model being subject to further performance analysis as well.

Consequently, this proof-of-concept study can be eventually used in a wide range of applications by stakeholders ranging from farmers to plant phenotyping researchers. Applications of this work include estimating the number of plant leaves, evaluating a plant's growth stage, final yield prediction, and crop improvement methods.

2.7 Conclusion and Future Works

In this work, we presented an approach for near real-time leaf detection and counting on an unmanned ground vehicle. To achieve that, we train two state-of-the-art a deep object detection models using our annotated *Arabidopsis thaliana* plant leaf images and chose the best performing model to deploy on the ground vehicle. Our experiments show prospect in accurate real-time processing on low-cost hardware in both greenhouse and field environments. Additionally, it is evident that when given a moderate amount of data on top-view images of plants, our trained model, tiny-YOLOv3, is able to learn to localize and predict the number of plant's leaves without any prior knowledge on that specific plant. Lastly, the annotated dataset has been made publicly available, with the goal of promoting the use of object detection deep learning models within the plant phenotyping community.

Future works include creating more robust algorithms to help automate the leaf counting process by using more powerful platforms with more computational resources and powerful cameras. These improvements will lead to the development of an autonomous plant phenotyping system that can first perform localization in real-time to detect potential crops to monitor, then navigate the unknown terrains

to map them while providing real-time feedback on crop status. The acquired information will then be processed on-board in real-time and either be used to deploy another robot to address any issues or to inform farm managers for manual/visual inspection.

Chapter 3

Spatio-temporal Mapping of Cotton Blooms Appearance Using Deep Learning[1]

[1]Vaishnavi Thesma, Canicius Mwitta, Glen C. Rains, and Javad Mohammadpour Velni. To be presented at the *IFAC AgriControl* Conference 2022.

3.1 Abstract

In this paper, we present an approach to create spatio-temporal maps using deep learning to visualize cotton bloom appearance over time. Specifically, we manually annotate cotton flower image data and train three state-of-the-art fast deep neural network models to count cotton blooms and their frequency over time prior to harvesting. We use the detection results of the best performing model combined with traditional pixel-based image analysis methods to create a map of where past and future blooms grow on a mid-stage cotton plant. The training results of our best model show a visual understanding of how many cotton flowers grow with high F1 Score of more than 0.95, a true positive rate of 98%, false negative and false positive rates both under 10%, and millisecond-scale inference time for real-time processing.

3.2 Introduction

Farmers are facing many challenges to produce sufficient food and fiber for the rapidly growing world population. The effects of climate change, reduced natural resources, and an unskilled workforce make efficient and optimal production a high priority. The use of traditional and state-of-the-art computer vision methods can help farmers analyze their farms faster and more accurately (Ahn et al., 2018). Furthermore, these solutions can help farmers to rely less on manual labor and analysis, give insight into farm status remotely, help with decision making strategies, and predict farm yield (Huang et al., 2020).

Plant feature detection in controlled and field environments are popular applications of modern computer vision methods in agriculture. Typical plant features of interest are characteristics of leaves, fruits, and flowers. These phenotypic traits are difficult to manually count accurately due to their size, location, or high frequency (Darwin et al., 2021). Flowers, in particular, are useful features that can help predict the crop's yield and spatial distribution. Flower features have been detected by using pixel-based image processing and deep neural networks. For instance, (Biradar & Shrikhande, 2015) used Gaussian low-pass filters to remove flower background and segment individual flowers. Also, (Lim et al., 2020) used Faster Region-based Convolutional Neural Networks (Faster R-CNN) and Single Shot Detectors (SSD) to detect kiwi fruit flowers during different seasons and locations in an outdoor farm. Furthermore, (J. Wang et al., 2022) used tiny-YOLOv4 and a stereo camera to detect flowers and localize them on a 3D point cloud.

Cotton farmers specifically experience challenges in yield prediction as the number of cotton bolls and their locations determine yield count and profit. However, it is difficult to manually count cotton bolls because they grow in high density (Huang et al., 2020). Additionally, cotton harvesters are very expensive and many farmers cannot afford them, resulting in cotton not being harvested in a timely manner to maximize profit (Kadeghe et al., 2018). On the other hand, cotton blooms typically determine the number of cotton bolls much earlier and do not grow as densely as bolls (Jiang et al., 2020). The cotton blooms each only last a few days and grow approximately one to two months prior to harvesting. Thus, frequent manual counting of blooms is very difficult. Furthermore, cotton blooms also can be occluded by the dense cotton plant foliage surrounding them. This makes noticing cotton flowers extremely difficult

and arduous for large-scale farms. Therefore, automated methods of data collection and analysis must be developed to make this information available for management of new harvesting platforms that use multi-harvest platforms, plant growth regulators, or fertilizers.

Several works have addressed the aforementioned challenges in detecting and counting cotton bolls and blooms to predict yield using traditional pixel-based methods, deep learning models, and unmanned vehicles. Traditional pixel-based methods have been used to mask cotton farm images to detect cotton bolls, as they are white. (Kadeghe et al., 2018) developed computer vision methods with OpenCV to track and localize cotton bolls captured on images using a low-cost camera installed on a ground vehicle. Additionally, modern object detection networks can be trained to learn how to identify cotton bolls specifically with fewer image preprocessing steps. For example, (K. G. Fue et al., 2021) used tiny YOLOv2, a lighter version of state-of-the-art YOLOv2 object detection network, for cotton boll detection in real time. Moreover, (Tedesco-Oliveira et al., 2020) performed a study to predict cotton yield using three state-of-the-art deep learning methods: Faster R-CNN, SSD, and SSDLite in commercial fields. (R. Xu et al., 2018) developed an algorithm to count cotton blooms using the images captured from an unmanned aerial vehicle (UAV). (Jiang et al., 2020) developed "DeepFlower," an image acquisition and deep learning algorithm to detect cotton flowers in field environment using Faster R-CNN. Lastly, (Huang et al., 2020) used a combination of three neural networks for counting cotton bolls using density level classification.

While there exists few works on detecting and counting cotton blooms accurately, they lack providing a visual estimate of the spatial distribution of high and low yielding areas for farmers. **The contribution of this work is to create a spatio-temporal map of cotton blooms for providing a visual understanding of how many blooms grow and their location prior to harvesting using state-of-the-art fast and deep learning models**. This way, farmers have insight about the spatial distribution of blooms that appear and their frequency of appearance. These efforts will provide better cotton yield prediction in advance of harvesting. Additionally, this research will provide same season information that would assist in management of inputs and planning for harvesting.

The remaining sections of this paper are organized as follows: Section 2 details our methodology in achieving our objectives; Section 3 presents our experimental results based on our methodology; Finally, Section 4 provides a summary of our work presented in this paper.

3.3 Methodology

3.3.1 Cotton farm details

The cotton bloom data was collected using an autonomous ground vehicle equipped with stereo camera from the University of Georgia Tifton campus in Tifton, GA, USA. The farm we used for data collection consisted of 40 rows of cotton plants on a one-acre field. An aerial view of the field is shown in Figure 4.1.

Figure 3.1: Aerial view of the cotton field in Tifton, GA.

3.3.2 Cotton data collection

The same ground vehicle as in (K. Fue et al., 2020) was used in our work to collect video streams of the cotton plant canopy. The hydrostatic rover was purchased from West Texas Lee Corp. and customized for remote control and data collection. The dimension of the rover is 340 cm long and 212 cm wide. The vehicle uses an NVIDIA Jetson AGX Xavier embedded computer as the on-board main controller for the vision and navigation systems and can be controlled remotely. The vehicle is powered by a 20 HP Koehler engine to power the hydrostatic drive system of the rover. A Predator 3500 Inverter generator and two 12V car batteries are used to power the electronics including the two inertial measurement units (IMUs), a ZED2 RGB stereo camera with two lenses (each 120 cm apart), embedded NVIDIA computer, and RTK-GPS. The camera was placed on the rover 220 cm above the ground facing downward. The rover is equipped with a navigator control system that uses an extended Kalman filter for robot localization. A front view of the ground vehicle and camera is shown in Figure 4.2.

The ZED2 camera has 4M pixel sensor for each of the lenses with large 2-micron pixels. The camera comes with a ZED SDK, compatible with ROS and OpenCV, both used to capture and process cotton plant image data. The camera captured video streams of each row every two or three days per week from June 2021 to October 2021, if it was not raining. The frequency of data collection was chosen because the blooms initially grow white, but quickly change color within 24 to 48 hours to a red-orange color and fall off the plant. Each video stream was stored as a ROS bag file data chronologically. Thus, each row can be located by both its stored name and modification time. The video streams were extracted using both *rospy* and *CvBridge* Python packages. An example image frame acquired on July 8, 2021 is shown in Figure 4.3.

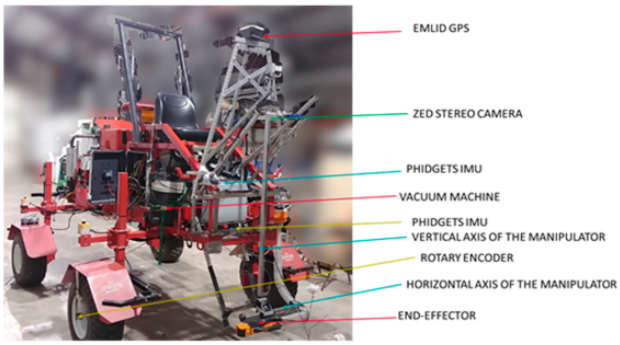

Figure 3.2: Front view of the rover (see (K. Fue et al., 2020) for further details) to collect video streams of cotton plants in Tifton, GA.

Figure 3.3: An example of extracted image frame from ZED2 camera video stream.

3.3.3 Dataset creation

The camera has two lenses and captures an image of the left view and of the right view of the cotton plants as the rover moves forward in each row. A total of 765 image frames were extracted from a different row on each data collection day between July 14, 2021 and August 4, 2021 to build the training data set. There were a total of 7 collection days in this time frame. We chose the July 14, 2021 date as the start since the earliest blooms began to appear in mid-July.

The extracted frames were labelled in ascending numeric order to match the corresponding video stream. Additionally, the 765 image frames included both left and right lens views and were split in half to separate the left and right lens views using Python Image Library (PIL). After splitting each image frame, we had a total of 1,530 frames, each labelled whether it was from the left or right lens. Since the video streams were selected on different rows on each collection day, the training dataset consisted of high variation in the cotton plant data, location, and treatments.

Upon examination of the extracted images, it was realized that the size of the cotton blooms is much smaller in comparison to their background. This can make detecting the flowers very difficult as the small size can be easily mistaken as background, soil, or foliage. To increase the size of the cotton blooms with respect to their background, we cropped the image frames focusing on the center two rows and sliced the cropped images into five equally-sized slices using PIL. The top left and bottom right coordinates of each of these slices were stored as part of the new sliced image names, as these slices will be stitched together

to build the spatio-temporal map. This resulted in 9,649 cropped and sliced images. An example of the image slicing is shown in Figure 4.4.

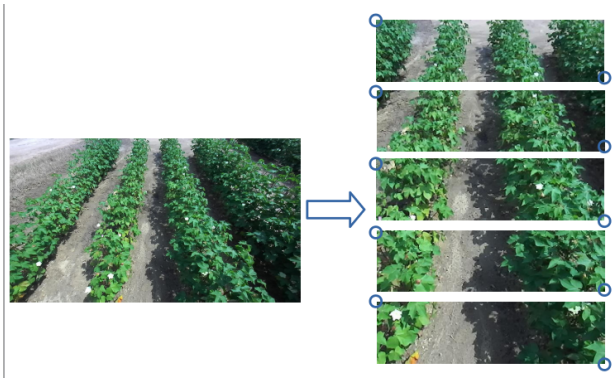

Figure 3.4: An example of a left view image that is cropped and sliced to produce 5 smaller equal-sized images to increase the cotton flower size with respect to its background. The blue circles on the top left and bottom right of each of the image slices are respective coordinates that were stored along with the image name to stitch the images together to build the spatio-temporal map.

3.3.4 Image labeling

We used OpenLabeler to annotate 2,000 images out of the 9,649 cropped and sliced images. Only 2,000 images were annotated since cotton blooms were not present in some of the cropped and sliced images. We decided to annotate the blooms that were clearly visible in each slice since the ground vehicle only provided a top-down view of each row. Additionally, the background soil and the sunlight reflection on the leaves may be misclassified as cotton blooms, so occluded blooms were not annotated. The annotations consist of a bounding box containing the location of a cotton bloom. These annotations are of VOC format that is widely used for state-of-the-art deep learning models. As such, the bounding box annotations were populated in an XML file for each annotated image. Figure 3.5 is an example of the annotated data.

Figure 3.5: An example of the annotated dataset using OpenLabeler to label only clearly distinguishable cotton flowers.

3.3.5 Model selection

For our experiments, we consider three deep learning model architectures: tiny-YOLOv3, tiny-YOLOv4, and Faster R-CNN. These models are popular object detection models that can perform in near real-time as discussed in (Buzzy et al., 2020).

The YOLO architecture does object localization and identification in a single pass (Redmon & Farhadi, 2018b). The tiny versions of YOLO employ fewer layers than full YOLO architecture but still produce accurate results in near real-time. We used both the third and fourth generations of tiny-YOLO for our experiments. Specifically, we used the pre-trained tiny-YOLOv3 weights from K. Fue et al., 2020, that were trained to detect cotton bolls, to perform transfer learning to detect cotton blooms on our annotated data. This way, we did not have to train a model from scratch to learn the cotton bloom features.

The pre-trained tiny-YOLOv3 weights perform well on close-up cotton boll data, but fail to detect detailed cotton blooms in close-up images in this year's data. Since we are interested in cotton bloom detection exclusively, we can utilize transfer learning to learn bloom features up close and farther away from the rover's camera.

YOLOv4 is different from its predecessor in that it uses a mosaic data augmentation technique during training (Bochkovskiy et al., 2020). Specifically, YOLOv4 tiles four images from the training dataset into a single image to further increase the variability of the data during training. Furthermore, this allows for the model to learn to detect variations in class sizes. Particularly, in these experiments, the cotton bloom size may vary due to its position with respect to the camera or natural orientation. As such, we use tiny-YOLOv4 to address this issue to improve the model's performance using fewer layers.

Faster R-CNN is another popular object detection network that is useful for densely packed objects (S. Ren et al., 2015). The architecture has two parts: the Region Proposal Network (RPN) and the Fast R-CNN detector. The RPN creates regions of interest on an input image and the Fast R-CNN detector classifies the regions of interest from the RPN. It is possible to train both the RPN and detector end-to-end, but is very difficult. Thus, we chose to train both the RPN and Fast R-CNN detector separately. The criteria used for training and evaluating each model is described in the ensuing section.

3.3.6 Training procedure and implementation

The pre-trained tiny-YOLOv3 model was initially weakly trained by K. Fue et al., 2020 for 2,000 iterations on 2,085 images of cotton bolls collected in field environment. We implemented transfer learning using Darknet by training the model for an additional 40,000 iterations without freezing any layers. This results in the model having more flexibility in learning cotton bloom features with some prior knowledge on cotton boll features. Instead of using the entire annotated training dataset, we chose only 280 images collected between July 14, 2021 and August 4, 2021 to perform transfer learning. This subset of training data was created by using 40 images for the 7 data collection dates within the aforementioned time frame. The model was trained using a batch size of 128, with 32 subdivisions, momentum of 0.9, and a learning rate of 0.001. The model was trained for a total of nearly 48 hours.

Contrastingly, we trained the tiny-YOLOv4 model from scratch using Darknet for 6,000 iterations on all 2,000 annotated images. Since tiny-YOLOv4 incorporates mosaic data augmentation during training, we decided to reduce the total number of iterations in comparison to transfer learned tiny-YOLOv3 model. The batch size was set to 64, with 16 subdivisions, a momentum of 0.9, and a learning rate of 0.001. The model was trained for a total of only 2.5 hours by speeding up the GPU three times faster with the Compute Unified Device Architecture (CUDA).

Lastly, the Faster R-CNN model was trained from scratch using Keras and TensorFlow. The RPN was trained for 50,000 iterations and the Fast R-CNN classifier was trained for 10,000 iterations. The RPN and classifier were trained sequentially and the trained weights from the RPN were fed into the classifier for training. The momentum was 0.9, the decay was 0.005, and the learning rate was 0.00001. The model training took approximately 72 hours since the RPN and classifier were trained separately.

The performance of all three models was tested on 101 image slices from August 6, 2021 to August 16, 2021 on different rows than the training data. The testing data was created similar to the preprocessing steps as the training data by cropping and slicing the extracted images to focus on the center two rows. All the models were trained and tested individually using a Quadro P2000 with 5 GB of GDDR5 Memory. The CPU was an Intel CORE i7-7800x with 32 GB of memory.

3.3.7 Evaluation metrics

To evaluate the effectiveness and performance of our proposed approach, we use the following evaluation metrics as in (Buzzy et al., 2020):
(i) Difference in count (DiC)= $\frac{1}{N}\sum_{i=1}^{N}\epsilon_i$;
(ii) Absolute difference in count (|DiC|)= $\frac{1}{N}\sum_{i=1}^{N}|\epsilon_i|$;
(iii) Mean squared error (MSE)= $\frac{1}{N}\sum_{i=1}^{N}\epsilon_i^2$,
where $\epsilon_i = y_i - \hat{y}_i$ is the difference between the ground truth and algorithmic prediction, respectively. The value y_i is defined by the true number of blooms present in a single image and the value $\hat{y}_i$ is defined by the number of blooms detected by the three trained algorithms individually. The systematic error DiC, absolute DiC, and MSE metrics should all ideally be near zero.

The detection results of our trained models can result in any of the following situations: true positive (*TP*); false positive (*FP*); true negative (*TN*); and false negative (*FN*). Precision and recall can be used to quantify these metrics:

$$\text{precision} = \frac{TP}{TP+FP}, \tag{3.1}$$

$$\text{recall} = \frac{TP}{TP+FN}. \tag{3.2}$$

The values of precision and recall should ideally both be high and near one. Using the precision and recall metrics above, F1 Score can be calculated as another way to analyze the accuracy of a model. Below

is the formula for evaluating F1 Score in terms of the precision and recall:

$$\text{F1 Score} = \frac{2 * precision * recall}{precision + recall}. \tag{3.3}$$

The value of F1 Score should also be near one for a good performing model. Additionally, the true positive rate (TPR), false positive rate (FPR), and false negative rate (FNR) are also calculated.

$$\text{TPR} = \frac{TP}{TP+FN}, \tag{3.4}$$
$$\text{FPR} = \frac{FP}{FP+TN}, \tag{3.5}$$
$$\text{FNR} = \frac{FN}{FN+TP}. \tag{3.6}$$

Ideally, the TPR should be very close to 100% and both the FPR and FNR should be low and close to 0%.

Finally, the inference time will be used to quantify the amount of time, in seconds, each model takes to detect cotton blooms in one image.

3.4 Classification results

The above evaluation metrics are calculated using the methodology described in the previous section and are summarized in Table 3.1 below.

Table 3.1: Network evaluation metrics.

Metric	Tiny-YOLOv3	Tiny-YOLOv4	Faster R-CNN
DiC	0.22	-0.059	-3.02
|DiC|	0.26	0.119	3.06
MSE	0.26	0.119	26.88
F1 Score	0.868	0.967	0.576
TPR	82.31%	98.52%	93.28%
FPR	3.7%	6.25%	70.22%
FNR	17.69%	1.48%	6.72%
Inference time	0.0083	0.0069	2.2777

Based on the results presented in Table 3.1, our tiny-YOLOv4 model showed better performance in terms of DiC, absolute DiC, MSE, F1 Score, TPR, FNR, and inference time compared to the tiny-YOLOv3 and Faster R-CNN models. These results are also significant since our tiny-YOLOv4 model was trained for only 2.5 hours from scratch whereas the other two models were trained for several days combined. Figure 3.6 shows our tiny-YOLOv4 prediction results on an illustrative test image.

While Faster R-CNN was able to detect many blooms correctly, some background data such as shadow and foliage of similar shapes as blooms were mistakenly detected as blooms, and this attributed to its

Figure 3.6: The detection results using our tiny-YOLOv4 model for a test image containing three blooms, each detected with the probability of over 50%.

high FPR. Furthermore, the FPR associated with the tiny-YOLOv4 model is slightly higher than the FPR for tiny-YOLOv3. We concluded that the FPR for tiny-YOLOv4 is still an acceptably low value to build our spatio-temporal map since the other counting metrics of tiny-YOLOv4 are superior to tiny-YOLOv3. A higher FNR as seen with both tiny-YOLOv3 and Faster R-CNN would not result in a visually representative spatio-temporal map as many blooms would have been missed.

In addition, our Faster R-CNN model has a high inference time in comparison to both the third and fourth generations of tiny-YOLO. A high inference time results in slow image processing and lacks potential for running our model online or on a deployable unmanned device, similar to the rover used in our data collection study. Although the inference times for our tiny-YOLOv3 and tiny-YOLOv4 models are very close, tiny-YOLOv4 can still outperform tiny-YOLOv3 when several hundreds or thousands of images are being used for testing. Therefore, we chose to use our trained tiny-YOLOv4 model to build the spatio-temporal map due to its superior performance in our experiments.

3.5 Spatio-temporal map creation

We selected six data collection days to build our spatio-temporal map from the same row, namely, July 14, July 23, August 4, August 11, August 20, and September 2, 2021. August 4 was used as a base date and served as the background of the spatio-temporal map since the cotton plants were at mid-stage of growth. The two dates prior to and the three dates after August 4 were used to overlay their past and future bloom locations onto the base date using different colors. Since all six dates were from the same row, the spatio-temporal map can be used to extrapolate the cotton bloom growth for the remainder of the field.

First, we feed each date's cropped and sliced images into our trained tiny-YOLOv4 model to localize each bloom. Our model was successfully able to detect all the apparent blooms in these dates. The coordinates of each detected bloom per each image were stored into a text file for each date. Each date had approximately 1,100 to 1,300 cropped and sliced images that were fed into our trained model. For each day, it took about 30 minutes to process the images. Thus, a total of 7,370 image slices were fed into our trained model to store the bounding box coordinates of each detected bloom in each image into six text

Figure 3.7: A cropped image frame from the left lens of the stereo camera collected on August 4 with five blooms present.

Figure 3.8: The spatio-temporal map of past and future detected blooms on the same row and frame from the left lens of the stereo camera. Each colored box corresponds to a past or future date of the same row and position in the field. The clear boxes represent the blooms present and detected on August 4.

files. This process of populating the text files with coordinates took three hours to complete due to the large number of image slices to process.

Next, we read and parse one date's text file and store all coordinates per image into a list object using Python. Then, we draw colored boxes onto the corresponding image from August 4 based on the list object's coordinates and save the newly formed image to file. This procedure is repeated for each date. The newly formed image is continuously overwritten to preserve the previous colored box location. Lastly, we stitch the colored image slices into the original frame size to show a snapshot of a row with all the past and future blooms together. Building our spatio-temporal map for the entire row took about 36 seconds.

Figures 3.7 and 3.8 show an example of our spatio-temporal map implementation using these dates. Figure 3.7 is a cropped image frame from August 4 from the left lens of the stereo camera. Five cotton blooms are apparent in the image frame, two in the foreground being the clearest and three in the background slightly occluded by foliage. Figure 3.8 shows our spatio-temporal map implementation result with the colored boxes representing the past and future bloom detection prior to and after August 4. The clear black-outlined boxes represent the five blooms detection present in the image. Blooms on July 14, July 23, August 4, August 20, and September 2 are shown by colors dark blue, red, light blue, yellow, and pink, respectively. There are fewer blue and red boxes as mid- to late-July are when blooms first appear. There are more light blue, yellow, and pink blooms as these are when most blooms appear in higher frequency.

3.6 Concluding Remarks

In this paper, we present an approach to create a spatio-temporal map to monitor cotton bloom growth and frequency prior to harvesting. To achieve that, we train state-of-the-art deep learning models using our annotated cotton bloom images and choose the best performing model to build the spatio-temporal map. Our experiments show prospect in accurate real-time processing to accurately detect cotton blooms online, building the spatio-temporal map offline, and the potential of early yield estimation based on bloom count.

Chapter 4

Classification and Tracking of Nutrient Deficient Leaves in Cotton Plants Using Support Vector Machine

4.1 Introduction

Farmers are facing many challenges to produce enough food and crop for the rapidly growing population (R. Xu et al., 2019). Some of these challenges include water scarcity, lack of arable land, disease prevalence, and difficulty in maintaining vast farmland. However, the use of traditional and modern computer vision methods can help mitigate these challenges. These solutions can also help farmers gain insights to their farm statuses remotely and accurately. Also, they will allow for strategizing and decision making to tackle apparent issues in the farm to maximize crop yield and profits (Buzzy et al., 2020).

Plant feature detection in controlled and field environments are popular applications of traditional and modern computer vision methods in agriculture. Typical plant features of interest are characteristics of leaves, fruits, and flowers. These phenotypic traits are difficult to manually count and analyze accurately due to their size, color, location, or high frequency (Darwin et al., 2021). Leaves, in particular are useful features that are indicative of a plant's health, development to maturity, flowering time, and estimation of yield (Buzzy et al., 2020). Leaf features have been detected by using pixel-based processing, support vector machines (SVMs), and deep neural networks. For example, the authors in (Gavhale et al., 2014) used color space transformations to convert RGB images of citrus leaves to highlight diseased spots. Also, (Mokhtar et al., 2015) used color space conversion and SVM to detect two tomato leaf viruses from field environment. Additionally, (Ozguven & Adem, 2019) used Faster Region-based Convolutional Neural Networks (Faster R-CNN) to detect leaf spot diseases on sugar beet images and scaled the severity of disease progression.

Specifically, cotton is a significant crop in the US and monitoring its status and harvesting is very arduous and expensive as discussed in (K. G. Fue et al., 2021). Good cotton plant health correlates to higher yield and profitability of the crop (Commission, 2021). Contrastingly, poor cotton plant health drastically decreases the yield and profitability for the farmer, resulting in wasted crop and money. Typically, cotton plant health is observed from the plant leaves and monitored throughout the growing season, as the leaves begin to defoliate once many bolls appear (Commission, 2021). Some diseases that are apparent in cotton grown include Cotton Leaf Roll Dwarf, Red Spot, White Spot, and Crumple Leaf (Commission, 2021; Patki & Sable, 2016). These diseases are seen by dark, circular spotting across the top of cotton leaves or leaf rolling and curling. Also, deficiencies in potassium, magnesium, and nitrogen are very common in cotton plants and negatively impact cotton yield at harvest time (Commission, 2021; Gormus & Kanat, 1998). These deficiencies are typically caused by runoff from rainfall and discoloration of leaves from green to yellow or from green to red. Cotton plants themselves cannot regulate their nutrient content and require the application of treatments to maintain good health (Commission, 2021). Since cotton plants defoliate near harvesting time, it is imperative to detect cotton diseases and nutrient deficiencies early to increase yield. Thus, automated computer vision methods must be utilized to quickly identify areas of poor health in cotton fields and help farmers strategize the use of treatments.

Several works have addressed detecting and monitoring cotton plant health from leaves using traditional pixel-based methods, SVMs, and deep learning models. The authors in (Revathi & Hemalatha, 2012) used several stages of pixel-based image processing including color space conversion, filtering, masking green areas, edge detection for classifying cotton leaf spot diseases and magnesium deficiency. Moreover, the authors in (Bhimte & Thool, 2018) detected leaf spot diseases of cotton leaf images captured from a digital camera from field environment up close by using K-means clustering for segmentation and SVM for classification. Lastly, (Zekiwos, Bruck, et al., 2021) developed and trained a custom convolutional neural network to extract and cotton leaf disease features and classify each image as one of four diseases.

Pixel-based image processing methods may be fast to implement but are not robust to variations in lighting or textures in uncontrolled environments. Similarly, SVMs are may also fast and can classify data based on extracted features. However, SVMs do not perform well for high dimensional data where feature variation is high such as classifying animals. Furthermore, robust SVMs require more data samples. Nonetheless, deep learning models can have the most robust performance with variations in input data and can learn high dimensional features. But, training accurate deep learning models require large datasets, long training time, and extensive parameter tuning.

While there has been work done to use various computer vision methods for disease and nutrient deficiency classification of cotton plants accurately on single, up-close images, they lack in tracking the progression of these anomalies from field environment over time. **The contribution of this work is to classify and track the progression of nutrient deficient cotton plant leaves from field environment over time using SVM.** We chose to use SVM for our work since our dataset is small in size and nutrient deficiencies are exhibited by color changes in leaves, which are lower dimension features. Inspecting cotton plant health via the plant's leaves will help farmers have better understanding of cotton plant

health throughout the growing season, strategize changes to improve crop health, and increase crop yield with preventative measures (Ji et al., 2010).

The remaining sections of this chapter are as follows: section 2 briefly describes the theoretical background of support vector machine and its use for classification; section 3 is the methodology for the collection and preparation of the cotton plant image data as input for SVM training; section 4 shows our results and provides discussion on each model; and section 5 summarizes the contributions of this work.

4.2 Background

SVMs are a type of supervised learning model popularly used for binary classification (James et al., 2013). SVMs are a generalized version of the maximal margin classifier (MMC) and support vector classifier (SVC), where training samples are classified into two classes, c_1 or c_2, and separable by a hyperplane. Assuming that the training samples can be linearly separable on the hyperplane, we can easily classify the testing samples to be on either side of the hyperplane.

However, if the training samples are not linearly separable, then our classifier will perform poorly to classify the samples to c_1 and c_2. Thus, two solutions can be implemented. First, a cost parameter, *C*, can be introduced that provides a soft margin width where some training samples may be misclassified to maintain a classifier's generalizability. If we set *C* to be large, then the margin of the hyperplane will narrow and allow in more testing samples to be incorrectly classified. Otherwise, if we set *C* to be small, then the margin width will widen and will not tolerate more testing samples to be incorrectly classified. Second, we must convert the linear classifiers used in MMC and SVC to nonlinear decision boundaries automatically by enlarging the feature space using *kernels*. A kernel function, which is the key point of SVM, is a computational method that determines the similarity between two training samples numerically and is represented as

$$K(x_i, {x_i}'), \tag{4.1}$$

where K is a function of inner products, and where x_i and x_i' are training samples.

There are three popular kernels that may be used for SVM including linear, polynomial, and radial. The linear kernel can be expressed as

$$K(x_i, {x_i}') = \sum_{j=1}^{p} x_{ij} x_{i'j}, \tag{4.2}$$

since the features of the training samples are linear and is the same as using standard SVC.

Secondly, the polynomial kernel can be used to fit SVC on higher dimension feature space and increase the flexibility of the decision boundary. The polynomial kernel can be written as

$$K(x_i, {x_i}') = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d, d > 0, \tag{4.3}$$

resulting in a nonlinear hyperplane. Higher values of d increase the nonlinearity of the decision boundary.

Lastly, the radial kernel can also be used to fit SVC on higher dimension feature space similar to the polynomial kernel. This kernel is useful when training samples are in separating clusters or near each other. However, the proximity of training samples influences the classification of a testing sample. Lower values of γ allows for farther samples to have higher influence whereas higher values of γ allows for nearby samples to have higher influence. The radial kernel can be written as

$$K(x_i, x_i') = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2, \gamma > 0, \right. \tag{4.4}$$

resulting in even more flexibility and nonlinearity of the decision boundary.

In our work, we use these three kernels to fit four SVC models to detect nutrient deficiencies in cotton plant leaves.

4.3 Methodology

4.3.1 Cotton farm details

The cotton bloom data was collected by using an autonomous ground vehicle equipped with stereo camera from the University of Georgia Tifton campus in Tifton, GA, USA. The farm consisted of 40 rows of cotton plants on a one-acre field. An aerial view of the farm is shown in Figure 4.1.

Figure 4.1: Aerial view of the cotton field in Tifton, GA.

4.3.2 Cotton data collection

The same ground vehicle as in (K. Fue et al., 2020) was used in our work to collect video streams of the cotton plant canopy. The hydrostatic rover was purchased from West Texas Lee Corp. and customized for remote control and data collection. A Predator 3500 Inverter generator and two 12V car batteries are used to power the electronics including the two inertial measurement units (IMUs), a ZED2 RGB stereo camera with two lenses (each 120 cm apart), embedded NVIDIA computer, and RTK-GPS. The camera was placed on the rover 220 cm above the ground facing downward. A front view of the ground vehicle and camera is shown in Figure 4.2.

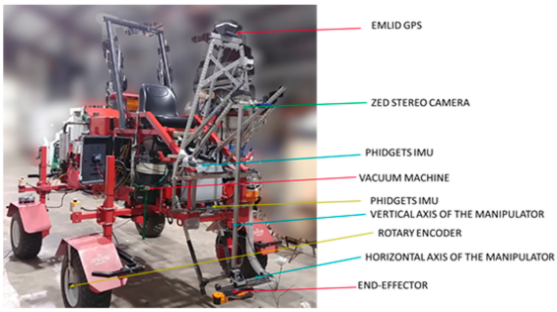

Figure 4.2: Front view of the rover (see (K. Fue et al., 2020) for further details) to collect video streams of cotton plants in Tifton, GA.

The ZED2 camera has 4M pixel sensor for each of the lenses with large 2-micron pixels. The camera comes with a ZED SDK, compatible with ROS and OpenCV, both used to capture and process cotton plant image data. The camera captured video streams of each row every two or three days per week from June 2021 to October 2021, if it was not raining. Each video stream was stored as a ROS bag file data chronologically and image frames were extracted using both rospy and CvBridge Python packages. An example image frame extracted from July 8, 2021 is shown in Figure 4.3.

Figure 4.3: An example of extracted image frame from ZED2 camera video stream.

Upon examination of the extracted image data, the size of the cotton plant features are much smaller in comparison to its background. This can make detecting the features very difficult as the small size can be easily mistaken as background, soil, or foliage. To increase the size of the cotton features with respect to its background, we cropped the image frames focusing on the center two rows and sliced the cropped images into five equal-sized slices using Python Image Library (PIL). An example of the image slicing is shown in Figure 4.4.

From this sliced dataset we acquire the images of nutrient deficient leaves.

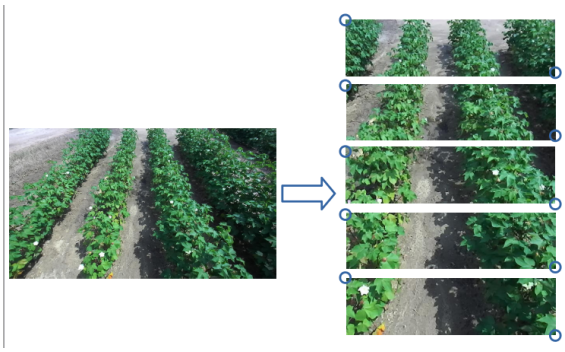

Figure 4.4: An example of a left view image that is cropped and sliced to produce 5 smaller equal-sized images to increase the cotton plant feature sizes with respect to its background. The blue circles on the top left and bottom right of each of the image slices are respective coordinates that were stored along with the image name.

4.3.3 Image Acquisition

We manually searched and cropped images of leaves that have reddish discoloration visible. A total of 90 images were cropped for our SVM experiments. An example of a cropped image from our dataset can be seen in figure 4.5.

Figure 4.5: An example of a cropped image of a nutrient deficient leaf from our collaborator's field.

4.3.4 Image Preprocessing

Many of the cropped images of our nutrient deficient leaves were dark in color due to shadows from surrounding foliage. This will make it difficult to extract the features that distinguish nutrient deficient leaves from healthy leaves. Thus, we resized the images to be of size 100x100 pixels and increased the sharpness, brightness, and contrast using PIL. Examples of our preprocessed images can be seen in figure 4.6.

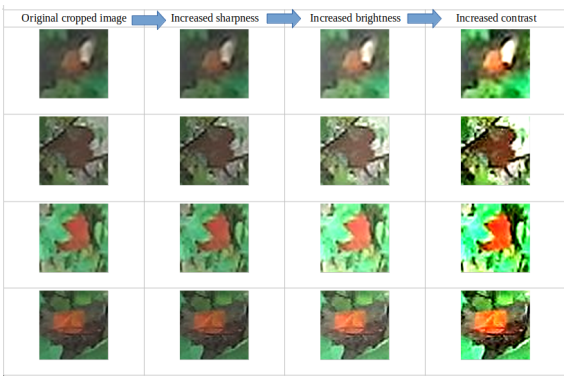

Figure 4.6: Examples of our preprocessing procedure to increase the contrast between the nutrient deficient and healthy leaves.

4.3.5 Feature Extraction

Once we preprocess the images, we use Grey Level Co-occurrence Matrix (GLCM) for feature extraction as discussed in (Bhimte & Thool, 2018). The GLCM matrix contains information that describes texture variations present in images. The parameters in GLCM include mean color channel value, standard deviation, contrast, energy, energy, entropy, homogeneity, and correlation. After we evaluated the parameters for the GLCM, we notice that the mean red and mean green channels had the highest relation to the presence of a nutrient deficient leaf. Thus, we introduce a categorical variable that would threshold the mean red channels to be either high nutrient deficient (1) or healthy (0). A total of 70 images in our dataset were categorized as high nutrient deficient and the remaining 20 were categorized as healthy. We use the mean red, mean green, and deficiency categorical variable features to feed into our four SVM models.

4.3.6 Classification

We randomly split the training data and extracted features as 70% training and 30% testing. The training and testing set includes 63 and 27 images, respectively. The training data was fed into three SVM models prior to hyper-parameter tuning, each with different kernels. The first model was trained using a linear kernel, the second using a radial kernel, and the last using a polynomial kernel. The cost parameter C was set to 100 for all three models to reduce the width of the margin. Also, we chose γ to be 0.01 for the radial kernel. Lastly, we did not specify a hard limit on the maximum iterations for training. The results of our three trained models are described in section 4. We use the sklearn package for training our models.

4.3.7 Hyperparameter Tuning using Cross Validation

For our fourth SVM model, we use 10-fold cross validation to tune the cost parameter, gamma, and the choice of kernel. Our cross validation resulted in a tuned value of C to be 1, γ of 0.0001, and linear kernel.

We train our fourth SVM model on these tuned parameters. The results of our final trained model is described in section 4. We also use the sklearn package for cross validation and training the final model.

4.3.8 Evaluation Metrics

We feed our testing set to each of our four trained models to evaluate their performance using five metrics: precision, recall, F1 score, accuracy, and support vector (SV) count.

As discussed in (Buzzy et al., 2020), model predictions can result in true positives (TP), false positives (FP), or false negatives (FN). We use these metrics to define precision and recall:

$$\text{precision} = \frac{TP}{TP+FP}, \tag{4.5}$$

$$\text{recall} = \frac{TP}{TP+FN}. \tag{4.6}$$

Moreover, precision and recall can be used to define F1 Score, another metric for analyzing a model's accuracy:

$$\text{F1 Score} = \frac{2 * precision * recall}{precision + recall}. \tag{4.7}$$

Ideally, precision, recall, F1 score, and accuracy should be high and near 1.00 to indicate a well performing model.

Since our objective is to classify cotton plant leaves as either healthy or nutrient deficient, our results show evaluation metrics for both classes. The best performing model will be used to track the progression of nutrient deficient leaves from the same row over time.

4.4 Results

The following sections are our results for training four SVM models with different kernels and cost parameters. For each trained model, we present their performance on the testing dataset in tabular format and the plot of each kernel's classification. The plot's horizontal axis represents the mean red channel feature and the vertical axis represents the mean green channel feature. The red points correspond to the data that was categorized as highly nutrient deficient and the blue points correspond to the data that was categorized to be healthy. In terms of a generic SVM, the red area corresponds to the +1 class and the blue area corresponds to the -1 class.

Furthermore, we choose the best performing model to track the progression of nutrient deficient leaves over time from the same row. We plot the true nutrient deficient count with our best model's predictions and to visualize the tracking.

4.4.1 Linear Kernel

The performance metrics and results for training our first SVM model using a linear kernel are seen in table 4.1 and figure 4.7, respectively. These results show that our dataset is nearly linearly separable and our model has high F1 score for classifying both healthy and nutrient deficient leaves. However, there was 1 mistake training observation where a nutrient deficient leaf was mistakenly classified as a healthy leaf. This indicates that some leaves that are near our mean red channel threshold may be misclassified. But since our selection of C was large, the margin of the hyperplane is narrow, allowing for some misclassifications while maintaining generalizability.

Table 4.1: Evaluation metrics of linear kernel SVM.

Metric	Healthy	Nutrient Deficient
Precision	0.86	1.00
Recall	1.00	0.95
F1 Score	0.92	0.98
Accuracy	0.96	
SV Count	3	

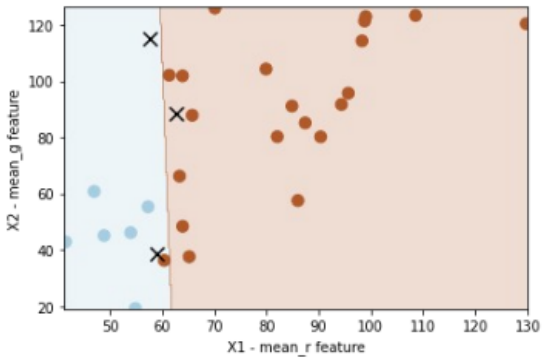

Figure 4.7: Our results using a linear kernel.

4.4.2 Polynomial Kernel

The performance metrics and results for training our second SVM model using a polynomial kernel are seen in table 4.2 and figure 4.8, respectively. In comparison to the performance of our linear kernel model, the precision and F1 score for detecting healthy leaves is worse for our polynomial kernel model. Additionally, the F1 score for detecting nutrient deficient leaves is also slightly worse in comparison to the linear kernel model. This slight decrease is attributed by the 3 mistake training observations where nutrient deficient leaves with similar mean red features as healthy leaves were misclassified as healthy leaves. Despite the slight decrease in performance in comparison to the linear kernel, the overall accuracy of the polynomial kernel model is still acceptably high and shows potential to be generalizable.

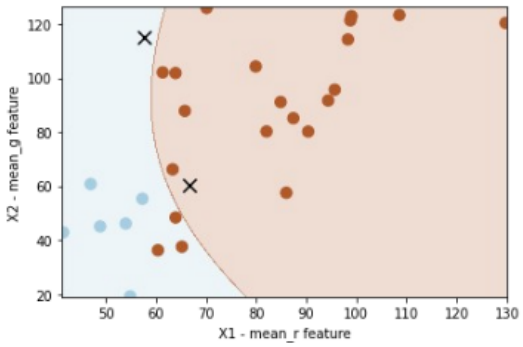

Figure 4.8: Our results using a polynomial kernel.

Table 4.2: Evaluation metrics of polynomial kernel SVM.

Metric	Healthy	Nutrient Deficient
Precision	0.67	1.00
Recall	1.00	0.86
F1 Score	0.80	0.92
Accuracy	0.89	
SV count	3	

4.4.3 Radial Kernel

The performance metrics and results for training our third SVM model using a radial kernel are seen in table 4.3 and figure 4.9, respectively. The precision, recall, and F1 score associated with the healthy leaf predictions are significantly lower than the results from the linear and polynomial kernel models. However, the precision, recall, and F1 score associated with the nutrient deficient leaf predictions are similar to the results from the linear and polynomial kernel models. Nonetheless, the large number of support vectors indicate that the radial kernel model was overfitted due to the selection of γ and the influence of nutrient deficient leaves being higher than healthy leaves.

Table 4.3: Evaluation metrics of radial kernel SVM.

Metric	Healthy	Nutrient Deficient
Precision	0.62	0.95
Recall	0.83	0.86
F1 Score	0.71	0.90
Accuracy	0.96	
SV count	45	

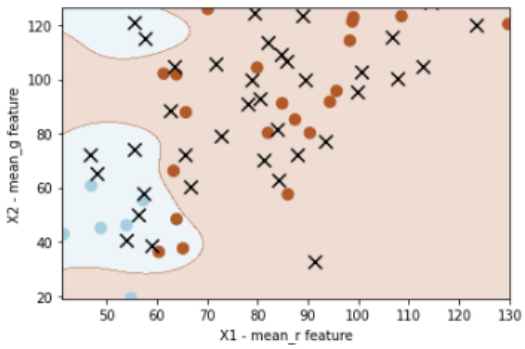

Figure 4.9: Our results using a radial kernel.

4.4.4 Cross Validated Linear Kernel and Tuned Hyperparameters

The performance metrics and results for training our fourth SVM model using cross-validated hyperparameters and linear kernel are seen in table 4.4 and figure 4.10, respectively. The performance results are identical to the results in 4.7, but there is similar potential for some leaves to be misclassified if their mean red value is near our threshold.

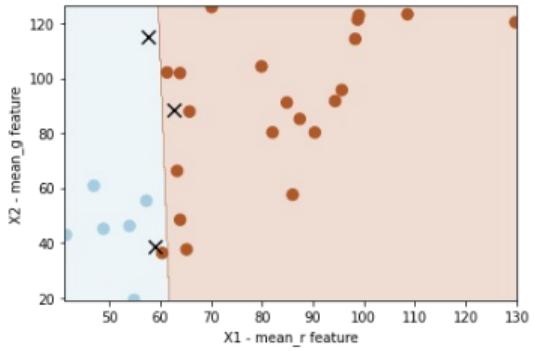

Figure 4.10: Our results using tuned hyperparameters from 10-fold cross validation.

4.4.5 Tracking progression of nutrient deficient leaves

Based on the results and analysis from our experiments, our cross validated model had the best performance to classify healthy and nutrient deficient leaves. We used this trained SVM model to track the progression of nutrient deficient leaves from the same row over time. We plot the performance of both our best model's

Table 4.4: Evaluation metrics of tuned hyperparameter SVM.

Metric	Healthy	Nutrient Deficient
Precision	0.86	1.00
Recall	1.00	0.95
F1 Score	0.92	0.98
Accuracy	0.96	
SV Count	3	

prediction count of nutrient deficient leaves with the true nutrient deficient leaf visualize the accuracy of tracking the progression of leaf discoloration caused by nutrient deficiency.

We chose five data collection days to build our dataset to track nutrient deficient leaves. Specifically, July 26, August 4, August 11, August 23, and September 2. These dates were chosen approximately one week apart from each other to allow for potential new nutrient deficient leaves to appear prior to defoliation. For each date, we manually search for discolored leaves similar to our methodology for creating our training dataset. In our dataset, we observed that fewer nutrient deficient leaves appeared in late July in comparison to early September. A total of 151 images of discolored leaves were found within this date range.

Next, we preprocessed the cropped images from each date, extracted features, and categorized the leaves as either healthy or nutrient deficient similar to our methodology when creating our training dataset. From the 151 images, 108 were classified as nutrient deficient based on our feature extraction. We store the number of nutrient deficient leaves from each date for comparison with our model's predictions. We then fed each date's enhanced images and extracted features into our trained cross validated linear kernel model. We stored the number of leaves that were predicted to be nutrient deficient.

Lastly, we plot each date's true nutrient deficient count along with our model's count of nutrient deficient leaves. The plot can be seen in figure 4.11. The results of our tracking show that our cross validated model correctly identified 103 out of 108, an accuracy of over 95%, nutrient deficient leaves in this date range.

4.5 Conclusion

In this chapter, we present an approach to classify cotton plant leaves as either healthy or nutrient deficient. To achieve that, we trained and tested four SVM models on our dataset of leaves from field images. We chose the best performing model to track the progression of nutrient deficient leaf appearance from the same row over time. Our experiments show prospect in accurately classifying and tracking the appearance discolored leaves from field images.

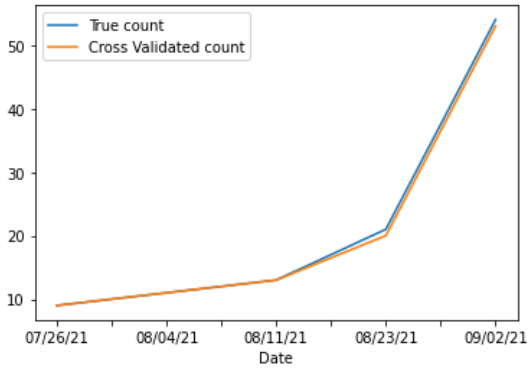

Figure 4.11: Plot of tracking results.

Chapter 5

Binary Semantic Segmentation for Root Phenotyping using Deep Conditional GANs

5.1 Introduction

Monitoring plant root morphology, also referred to as root phenotyping, is imperative in the analysis of the plant behaviors such as nutrient absorption, growth, and response to environmental changes in soils (Gong et al., 2021; T. Wang et al., 2019). Root phenotyping involves the characterization of a plant's root system architecture (RSA) throughout plant growth. Roots help anchor plants above the ground and provide insight on a plant's development and survival potential as variations in crop genotypes are developed, soil fertility changes, and efficient resource absorption becomes a priority to meet the food and crop demand of the rapidly growing world population. Therefore, root phenotyping allows for a comprehensive understanding of plant fitness under adverse conditions and for yield prediction (Bucksch et al., 2014; Gaggion et al., 2020).

Manual root phenotyping is very arduous as roots are usually small, thin, transparent, and most importantly, underground. Traditional root phenotyping had often been conducted by physical uprooting plants manually or by using unmanned ground vehicles (UGVs) for visual analysis. However, removing plants from the ground can easily damage roots. Furthermore, roots that are cored from the soil are later washed, which can result in root drying (Smith et al., 2020). These damages on roots impede proper analysis of the plants health. Thus, it is necessary to develop alternative, nondestructive analytic methods to automatically phenotype plant roots.

Nondestructive crop monitoring typically utilizes a combination of traditional and modern computer vision methods, especially with the surge of high quality cameras, sensors, and computational hardware for processing (Buzzy et al., 2020). These methods include pixel-based image processing, magnetic resonance imaging (MRI), X-ray, and machine and deep learning. Moreover, these methods are popularly combined together to completely phenotype RSA automatically. Additionally, plants grown in controlled

environments also have been placed in clear containers or gel media to easily examine RSA using imaging systems (Bucksch et al., 2014). Specifically, segmenting roots from their background is mainly studied to accurately visualize root health and temporal development, and to gain a comprehensive understanding of RSA. Several works have addressed segmenting roots from its background using a combination of traditional and modern computer vision methods.

The authors in (Gong et al., 2021) used pixel-based preprocessing on rice root images to discard background and maintain images with majority roots. Also, they used a sliding window approach to select smaller patches of these majority root images and fed them into two segmentation models. Furthermore, the authors in (T. Wang et al., 2019) developed a convolutional neural network (CNN) based off of SegNet, a popular segmentation network, to segment soybean roots from dense background soil. Similarly, the authors in (Smith et al., 2020) used U-Net, a classic segmentation model, to segment images of chicory roots growing in clear containers filled with soil. Similar to the efforts in (Gong et al., 2021) and (T.-C. Wang et al., 2018), the authors in (Gaggion et al., 2020) compared various deep semantic segmentation models by performing traditional image transformations and randomly searching for patches with reduced pixel imbalance. They used their augmented dataset to train several segmentation models for comparison. Lastly, the authors in (Möller et al., 2021) used the same dataset as in (Gaggion et al., 2020) to perform a study on using various loss functions and parameters while training SegNet and U-Net for segmenting main and lateral roots from their background.

While several works have addressed root phenotyping via image segmentation, they rely heavily on traditional data augmentation methods to reduce pixel imbalance caused by the sparsity of roots in images themselves or only training small datasets. While patch creation can drastically increase the size of the dataset, it requires extensive storage and may result in segmentation results not providing a comprehensive visual on how well the RSA has been segmented from background, as the model was trained on patches of the roots instead of the entire RSA. Furthermore, these trained models are not generalizable to different datasets of root images that contain complete RSA instead of patches. As such, additional pre- and post-processing would always be required for these models to create patches to reduce pixel imbalance and interpreting RSA development (Sampath et al., 2021). Therefore, it is necessary for the segmentation models to provide generalizable results that contain complete RSA so that the root phenotyping can be achieved quickly.

The contribution of this work is in segmenting root images that contain complete RSA while reducing pixel-wise class imbalance. To achieve that, we use a high definition conditional GAN, Pix2PixHD, to generate realistic and high resolution images with complete RSA and their corresponding annotations to reduce the pixel-wise class imbalances between root and background of *Arabidopsis thaliana* root images. Furthermore, we use our generated dataset to perform binary semantic segmentation using SegNet. Our approach involves training two deep learning models to increase our dataset, reduce pixel-wise class imbalance, and perform semantic segmentation for the root phenotyping. This work aims to provide more generalizable segmentation results of plant root images that contain the complete RSA compared to current methods that use patches of root images.

The remaining sections of this chapter are as follows: Section 2 provides a background to generative adversarial networks and its purpose for root phenotyping; Section 3 details our methodology for generating realistic images using cGAN and binary semantic segmentation; Section 4 shows our results and analysis of our experiments; and Sections 5 concludes our efforts presented in this chapter and provides discussion for future work.

5.2 Background

Ideal class balance in datasets is present where there exists an even distribution for every class sample. However, realistic datasets do not always maintain perfect class balance and some classes may be more prevalent than others. It is also possible that non-desired classes are more prevalent than desired ones such as background. Datasets with class imbalance used for deep learning tasks, such as classification or segmentation, result in poor model performance.

In root phenotyping, it is common for root datasets to have class imbalance in terms of scarce amounts of roots in comparison to background as roots are typically thin. This class imbalance is crucial to address when developing root phenotyping models. Traditional data augmentation techniques such as cropping and patch creation are not sufficient for improving segmentation tasks since these efforts do not adequately represent RSA. However, generative models have shown prospect in reducing class imbalance even for semantic segmentation tasks (Sampath et al., 2021).

Generative modeling is a type of unsupervised learning task that learns patterns from input data to create new samples similar to the input data. Generative adversarial networks (GANs) are a type of generative modeling that contains two submodels: a generator model, *G*, and a discriminator model, *D* (Goodfellow et al., 2014). The generator is trained to create new samples similar to the input data and labels. The discriminator is simultaneously trained to classify if the input from the original dataset or the generator is real or not. The goal for the generator during training is to maximize the likelihood that the discriminator fails to determine the correct classification of the input data. This would indicate that the generator is creating plausible examples nearly indistinguishable from the original input data. Therefore, the relationship between these two models represents a two-player min-max game as

$$\min_G \max_D V(D,G) = E_x[\log D(x)] + E_z[\log(1 - D(G(z)))], \tag{5.1}$$

where E_x is the expected value over all samples in the dataset, $\log(D(x))$ is the probability that the discriminator has determined that a sample is real, and E_z is the expected value of the random input samples being fed into the generator. Ideally, the loss is minimized when both expected values are equivalent, indicating that the generator is creating nearly indistinguishable samples from the original data and the discriminator has a 50% chance of correctly determining if the sample is real or not.

A subset of GANs include conditional GANs (cGANs) where the input data being fed into the generator model is conditionally coupled with auxiliary metadata (Mirza & Osindero, 2014). The coupled

metadata may include a class label, numerical values, or images. The discriminator model is similarly conditioned where its input is now both the auxiliary metadata and original or generated data. This type of GAN allows for the generator to create data belonging to a particular domain. Similarly, cGANs also play a two-player min-max game as

$$\min_G \max_D V(D,G) = E_x[\log D(x|y)] + E_z[\log(1 - D(G(z|y)))], \tag{5.2}$$

where y is the auxiliary metadata coupled with the input samples, $\log(D(x|y))$ is the probability that the discriminator has determined that a sample is real given the concatenated conditional attribute y, and $G(z|y)$ is the generator function for a sample z given the concatenated conditional attribute y.

The benefits of GANs are primarily their use for data augmentation by increasing the size and quality of an original dataset. Data augmentation usually increases the performance of models in terms of accuracy and generalizability. GANs, specifically, can also perform data augmentation by modelling higher dimensional data such as high resolution images, artwork, and image-to-image translation. cGANs for image-to-image translation are done by transforming an image from one domain to another while maintaining the content of the source image and modifying some visual attributes (Isola et al., 2017; Pang et al., 2021). These types of cGANs must be trained to learn a mapping that can generate a new image similar to a target image while maintaining the content in the source image. In our work, we use an image-to-image translation cGAN to generate photorealistic and high resolution images of roots to reduce pixel-wise class imbalance in our root dataset.

5.3 Methodology

5.3.1 Dataset Acquisition

We use the dataset from the root segmentation challenge and the research conducted by the authors of (Gaggion et al., 2020). The authors' dataset consists of *Arabidopsis thaliana* plants growing in controlled, indoor environment inside clear gel Petri boxes. The growing periods varied between two to four weeks, and each Petri box contained four *Arabidopsis thaliana* plants. The authors used Raspberry Pi and four infrared cameras to capture RGB image frames of the plants' roots growth over time in near-infrared lighting every twelve hours. An example of the growing conditions captured by the RGB camera used is seen in Figure 5.1. The resolution of each image is 3280×2464.

A portion of the captured image frames were annotated for training segmentation models by the authors in (Gaggion et al., 2020). In our experiments, we used 198 of the annotated images for binary segmentation. An example of the binary annotations that correspond to Figure 5.1 is shown in Figure 5.2.

The binary annotations were stored as MRI medical image format and were extracted using ITK-SNAP (Yushkevich et al., 2006). We use these 198 image frames and their corresponding binary anno-

Figure 5.1: Example of a captured image frame of *Arabidopsis thaliana* plant growing in a controlled, indoor environment.

Figure 5.2: Example of the annotation corresponding to Figure 5.1.

tations to feed into our cGAN and segmentation models for training. The average root to background pixel ratio of these annotations is 1:100, which indicates severe pixel-wise class imbalance for this dataset.

5.3.2 Semantic Map Creation

To train both our cGAN and segmentation models, we converted the binary annotations of each image into semantic maps, where each pixel is labelled as belonging to a particular class from 0 to $N - 1$, where N is the total number of classes. In our work, we have labelled two classes using Python Image Library (PIL), which include the background pixels as 0 and root pixels as 1. The semantic maps are created by first converting the RGB image annotations from 8-bit color to 8-bit gray-scale. Next, each white pixel corresponding to the roots in the gray-scale binary annotation is set to the value of 1. We store these semantic label maps for each annotation as a new image label file. An example of our semantic map creation is seen on a small patch of a root image in Figure 5.3.

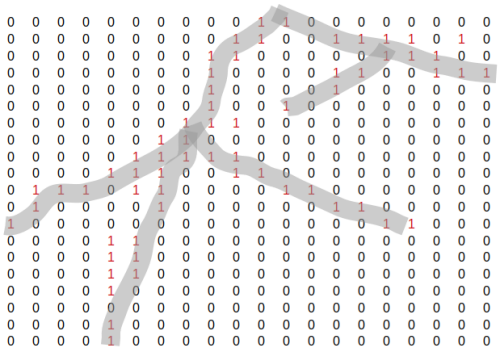

Figure 5.3: An illustrative example of semantic label map on a patch of a root image. The patch example is seen as a matrix, where indices with the value of 0 correspond to the background class and those with the red value of 1 correspond to the root class. Since the width of roots' annotations is between 1 and 3 pixels, we highlight the red indices here in light gray so the form of the root is clearly visible.

5.3.3 cGAN Model Selection and Training

For our cGAN, we chose to use the Pix2PixHD architecture developed by (T.-C. Wang et al., 2018) to generate new realistic images to augment our root dataset. This model is based off of Pix2Pix cGAN developed by (Isola et al., 2017), where the generator model learns to translate semantic label maps to realistic images and the discriminator model tries to distinguish the real images from the generated translated images (T.-C. Wang et al., 2018). Specifically, the Pix2Pix cGAN uses both the original image and its corresponding semantic label map as its auxiliary metadata for training. The model uses a U-Net architecture as the generator model and produces low resolution images. For our experiments, we require high resolution images since the roots are thin and sparse with respect to the background.

Pix2PixHD builds on Pix2Pix by improving photorealism and resolution (T.-C. Wang et al., 2018). Specifically, Pix2PixHD incorporates a coarse-to-fine generator model, a multi-scale discriminator, and a robust adversarial loss function. The coarse-to-fine generator contains two subnetworks that are jointly trained on high resolution images. The multi-scale discriminator contains three discriminator models that are trained on different image sizes by downsampling its input images. The motive for the multi-scale discriminator is to reduce computational complexity of using one discriminator on high resolution images. Lastly, the robust adversarial loss stabilizes the generator during training. The architecture of Pix2PixHD used for this work is shown in Figure 5.4.

We use 163 of our 198 images from our dataset to train the Pix2PixHD model from scratch for 118 epochs. During each epoch, our model randomly cropped each image to reduce the amount of empty background and increase the ratio of root to background pixels. We set the batch size to 1 to reduce training time, the learning rate to 0.0002, and used the Adam optimizer. Lastly, we used two discriminators during training. Our model took approximately ten hours to train.

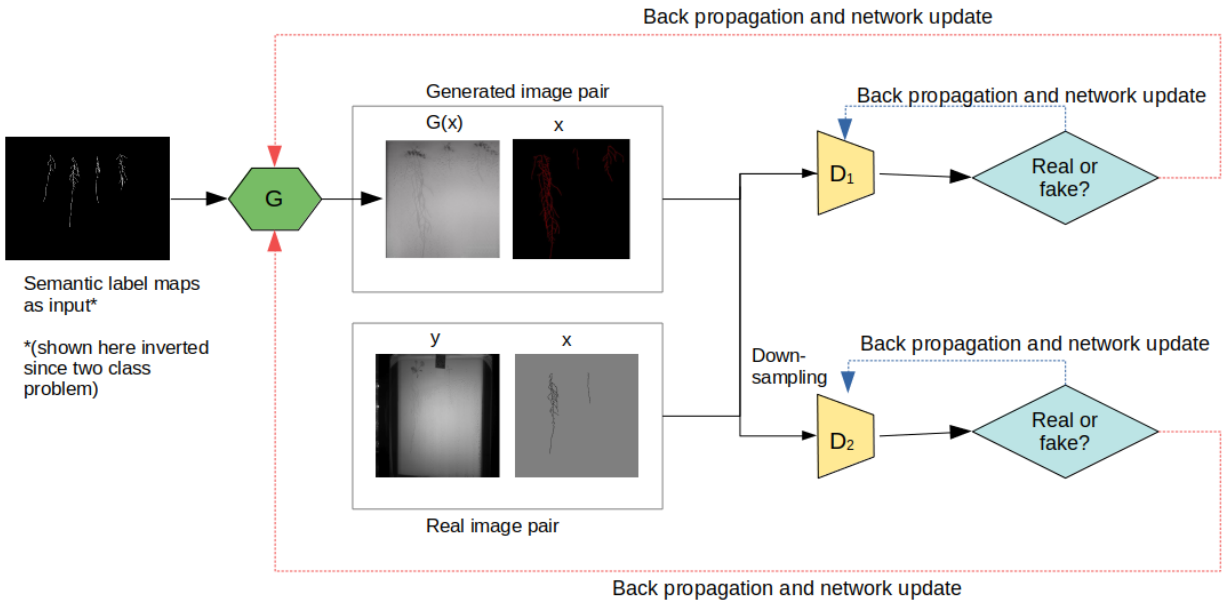

Figure 5.4: Pix2PixHD architecture.

We use our trained cGAN to generate an additional 396 images using the remaining 35 images from our training dataset and by vertically flipping our original training data to be different than the data used to train the cGAN. Similar to the cGAN training procedure, the trained cGAN randomly cropped and flipped each image to generate the fake images and corresponding labels. Thus, we increased our original dataset by three folds and it finally consisted of 594 images and their corresponding annotations. Our Pix2PixHD codes are inspired from the GitHub repository https://github.com/NVIDIA/pix2pixHD.

Lastly, we processed the images and annotation from our generated dataset and the original dataset to be fed into our SegNet model for semantic segmentation. We resized both our datasets and their corresponding labels to 480 by 360 using PIL, as this is the required input size for training SegNet, and we converted the generated fake labels to segmentation maps using the same methodology discussed in Section 5.3.2.

5.3.4 Semantic Segmentation Model Selection and Training

For our semantic segmentation model, we chose SegNet, a popular state-of-the-art semantic segmentation model that is primarily used for road scene understanding and dense pixel-wise classifications (Badrinarayanan et al., 2017). Similar to various U-Net series, SegNet has an encoder-decoder architecture (Ronneberger et al., 2015). The encoder network is identical to the popular VGG16 convolutional layers, but the fully connected layers are removed to make the SegNet encoder part smaller and easier to train

end-to-end. The encoder network contains encoder blocks that downsample the inputted RGB images using convolutional and max pooling layers. The decoder network contains decoder blocks that upsample the extracted features from the convolutions and finally apply pixel-wise classification. The output of each pooling layer from the encoder network is concatenated with an upsampling layer in the decoder network. Thus, there is 1 decoder block for every 1 encoder block. The architecture of SegNet used for this work is shown in Figure 5.5.

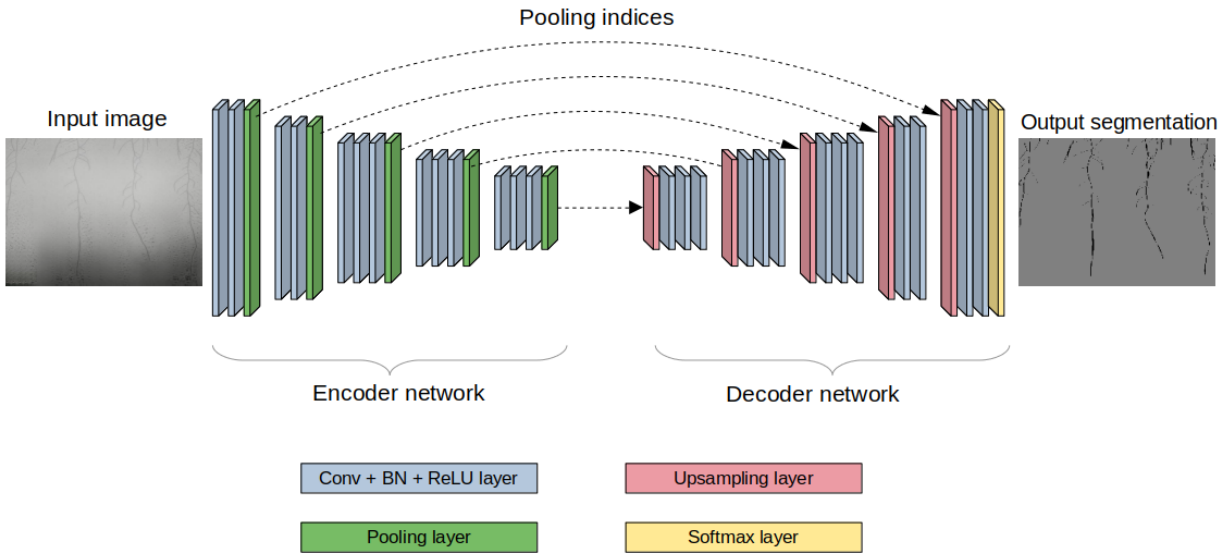

Figure 5.5: SegNet architecture.

We feed the processed generated dataset, the original dataset, and both their corresponding annotations into our SegNet model to train from scratch. We train our model for 50,000 iterations for about two and half days. We set the batch size to 5 to reduce the training time, the momentum to 0.9, the learning rate as 0.0001, and used the Adam optimizer again to automatically adjust the learning rate. Our SegNet codes are inspired from the GitHub repository https://github.com/aizawan/segnet.

5.3.5 Segmentation Postprocessing

Our segmentation results from our trained SegNet model show some gaps along the main root and lateral roots. To address this issue, we manually post-processed the segmentation results to close the gaps between main root and lateral roots. We chose a small image patch from our segmentation results and manually searched for gaps by converting the results to binary matrix similar to Figure 5.3, saved the matrix to a CSV file, and recorded the coordinates of the gaps' endpoints. We defined each branch and main root as arrays of coordinates of endpoints and connected the gaps between the endpoints by drawing white lines using PIL.

5.3.6 Evaluation Metrics

We use different evaluation metrics to gauge the performances of both our trained Pix2PixHD cGAN and SegNet segmentation models. For our cGAN model, we examine the performance based on the objective loss function given in (5.2) and the visual clarity of the generated images during training.

For our semantic segmentation model, we examine its performance using four metrics. The first is the cross entropy loss function at the end of training; the second is the overall accuracy of the model also measured at the end of the training; the third is the mean intersection-over-union (IOU) from testing our trained model on our testing set; and the last is the Dice Score also measured from testing our trained model on our testing set. Cross entropy is another popular loss function used to evaluate the performance of deep learning models. This loss function determines the difference between two probability distributions for a random variable or event (Yeung et al., 2022). For segmentation tasks, cross entropy loss aims to minimize pixel-wise error, especially in high class imbalance scenarios as in our experiments. Specifically, cross entropy loss is defined as

$$L_{CE}(y, \hat{y}) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})), \tag{5.3}$$

where $y, \hat{y} \in \{0, 1\}^N$ and y is the true class label and $\hat{y}$ is the predicted class label. Ideally, the value of (5.3) should be near 0 for a well performing model.

The accuracy metric is a global average of pixels being correctly classified as being root or background. Each pixel can be classified as true positive (TP), false positive (FP), true negative (TN), or false negative (FN) as described in (Buzzy et al., 2020). Ideally, the value of accuracy should be near 1 for a well performing model.

Furthermore, mean IOU is another common evaluation metric to determine the overall performance of a trained semantic segmentation model. Specifically, mean IOU determines the percent overlap of the ground truth and the trained model's prediction. Based on the aforementioned possible pixel classifications, mean IOU can be defined as

$$IOU = \frac{TP}{TP + FP + TN}. \tag{5.4}$$

Ideally, the value of mean IOU should be near 100% for perfect segmentation overlap. However, achieving this is very difficult for root segmentation tasks as roots are very sparse and as thin as 1 pixel in width.

Dice Score is another metric used for evaluating the accuracy of the segmentation models; it is similar to F1 Score but used for segmentation tasks (Bertels et al., 2019). Dice Score can be defined as a function of mean IOU as

$$Dice = \frac{2 * IOU}{1 + IOU}. \tag{5.5}$$

Ideally, the value of Dice Score should be near 1 for a well performing segmentation model.

Lastly, we compute the average inference time for our trained SegNet model to perform semantic segmentation in the testing dataset.

5.4 Results and Discussions

5.4.1 Pix2PixHD Results

Using our trained Pix2PixHD cGAN, we generated an additional 396 images to augment our original 198 root image dataset. An example of a generated image and its corresponding annotation is shown in figures 5.6 and 5.7, respectively. Our generated images show that the *Arabidopsis thaliana* roots are similar to our original dataset as seen in Figure 5.1. Additionally, since our original annotations only contain annotated root pixels and considered the leaves as background, the generated images do not perfectly translate the *Arabidopsis thaliana* leaves. This is acceptable as we are mainly interested in generating photorealistic root images similar to our original dataset and in semantically segmenting the RSA from the background.

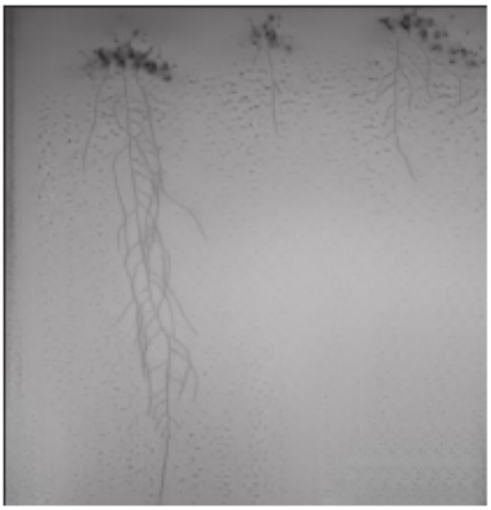

Figure 5.6: Example of a generated image from our trained cGAN. The roots are clearly visible and look similar to our original dataset. The *Arabidopsis thaliana* leaves are not translated in the generated images since semantic label maps were not created for them.

5.4.2 SegNet Results

We tested our trained SegNet model on 30 images comprised both of generated images from our trained cGAN and our original dataset. The performance of our model based on the aforementioned evaluation metrics discussed in Section 5.3.6 is given in Table 5.1. Our trained model shows high global average accuracy and Dice Score. Specifically, most pixels were correctly classified as either root or background. Additionally, the cross entropy loss of our trained model is very low and near 0. While the mean IOU and Dice Scores are not close to their ideal values, they are both still high given that our task involves segmenting thin roots. Lastly, the average inference time to process each image is fast and can be used for near real-time applications.

Figure 5.7: Example of the corresponding annotation from Figure 5.6. The annotations are clear and include the same generated root architecture present in Figure 5.6.

Table 5.1: Evaluation metrics for our trained SegNet model.

Metric	**SegNet Performance**
Cross Entropy Loss	0.020
Accuracy	0.991
Mean IOU	65.87%
Dice Score	0.7942
Inference Time	0.0438 sec.

Our segmentation results (one example shown in Figure 5.8) demonstrate that most of the root architectures can be successfully segmented from the background using our trained model. However, there are visible gaps along the lateral and main roots. The lengths of these gaps are typically less than 10 pixels and thus can be mitigated using traditional pixel-based postprocessing to close the gaps as discussed in Section 5.3.5.

5.4.3 Postprocessing

From Figure 5.8, we select a small patch from the second root system from the left for closing the gaps along the lateral and main roots using pixel-based postprocessing. The selected patch is seen in left-hand subimage in Figure 5.9. We manually process this image and connect the gaps using thin white lines as shown in the right-hand subimage of Figure 5.9. Thus, our postprocessing method can effectively close small gaps from our segmentation results.

Figure 5.8: Example of semantic segmentation results from our trained SegNet model. The main and lateral root architectures are successfully segmented, but there are small gaps along them.

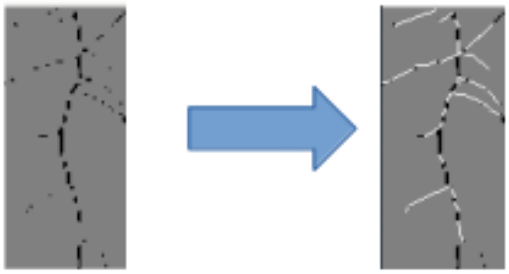

Figure 5.9: Example of postprocessing a patch of our segmentation results from Figure 5.8 using PIL. The gaps along the segmented lateral and main roots are closed using white lines.

5.5 Concluding Remarks

In this chapter, we present an approach to segment *Arabidopsis thaliana* root images from the background using a high resolution cGAN to reduce pixel-wise class imbalance and increase the size of our dataset. Our results show that our trained models can generate photorealistic images of full root architectures with their corresponding annotations and can segment them without the need for creating smaller patches, which do not sufficiently represent RSA. Thus, our experimental results demonstrate potentials in being generalizable to a variety of root images being fed into our model for segmentation.

Chapter 6

Conclusions and Future Work

This thesis focused on implementing plant phenotyping using modern computer vision methods for agriculture applications. Throughout this study, plant datasets originated from both indoor, controlled environments and from the field, including *Arabidopsis thaliana* and cotton plants. Furthermore, great attention was placed on implementing plant phenotyping accurately using datasets of limited size in near real time.

The contribution of the second chapter of this thesis was to develop a robotic platform to navigate between plant rows, capture top-down view images of leaves, and detect and count the number of leaves in real time. Two deep object detection models, namely tiny-YOLOv3 and Faster R-CNN, were trained from scratch on a lab-grown dataset of *Arabidopsis thaliana* plants. The trained tiny-YOLOv3 model had the best real-time performance and was deployed on a robotic platform to capture top-down view images of plants to detect and count leaves. Furthermore, transfer learning was implemented using tiny-YOLOv3 trained on smaller, young leaves to learn to detect larger, mature leaves as the plants grew; thus reducing retraining models as datasets grow or change over time.

In the third chapter, we develop an approach to create a spatio-temporal map of cotton bloom appearance from field images. Three deep object detection models, namely tiny-YOLOv3, tiny-YOLOv4, and Faster R-CNN, were trained on a manually annotated dataset of cotton blooms from field images. The tiny-YOLOv3 model was pre-trained on cotton boll detection and was trained using transfer learning to detect cotton bloom features accurately. Additionally, both the tiny-YOLOv4 and Faster R-CNN models were trained from scratch on cotton bloom data. The detection results of tiny-YOLOv4 were used to create a spatio-temporal map of cotton blooms on the same row at five different dates. The spatio-temporal map showed the spatial distribution of cotton bloom appearance over time on the same row prior to harvesting.

The focus of the fourth chapter was to classify and track the progression of nutrient deficient cotton plant leaves from field environment over time using SVM. The dataset of red, discolored cotton leaves was manually created and pre-processed for feature extraction based on mean RGB color channels. Four SVM models were trained on the feature extracted dataset to classify images of cotton leaves to be either healthy or nutrient deficient. Progression of nutrient deficiency was also accurately tracked using the

best performing SVM model on the same row over time. The tracking performance showed a quadratic increase in the number of nutrient deficient leaves with high accuracy.

The penultimate chapter's contribution was to perform root phenotyping using deep conditional GAN and binary semantic segmentation on images *Arabidopsis thaliana* roots. An image-to-image translation cGAN, Pix2PixHD, was trained to generate photorealitic and high resolution of root images to reduce pixel-wise class imbalance and increase the size of the original dataset. The generated and original images were both fed into SegNet for binary semantic segmentation. The segmentation results were post-processed to close the thin gaps along main and lateral roots. Moreover, the trained segmentation model can segment root images in near real time.

Future Work:

- As an extension of the second chapter, future works include the development of an autonomous plant phenotyping system where a robotic device can navigate to localize plant leaves and provide real time feedback on crop status on board. Thus, offline processing and analysis will be reduced.
- In addition to the work presented in the third chapter, the spatio-temporal map can be used for early yield estimation based on cotton blooms. As cotton flowers bloom up to two months prior to boll appearance, counting the frequency of cotton blooms can provide early estimation can provide farmers an idea of how many bolls will grow and estimate their profit. Furthermore, the spatio-temporal map of cotton blooms can be used to plot the rate of increase of bloom appearance.
- Using the models developed in the fourth chapter, another line of work includes tracking nutrient deficient leaves on different rows that have different treatments and comparing the rate of increase of nutrient deficient leaves. Also, more feature extraction can be implemented for the SVM models to learn more complex features beyond mean RGB color channels, such as LAB colorspace or texture analysis. Lastly, the trained SVM model can be used to localize nutrient deficient leaves around specific plants to target singular plant treatment.
- Another prospect in addition to the efforts from the fifth chapter include segmenting lateral and main roots separately to analyze RSA depth and width. Additionally, segmented root length can be translated to track plant health temporally.
- Generally, modern computer vision methods can also be deployed to imaging systems in both indoor, controlled and field environments for online plant phenotyping. This effort would address the issue of offline image processing and model retraining, as this is very time-consuming and plants continuously change over time. Additionally, developing a big data pipeline to collect and process large quantities of image and metadata online is another interesting line of work to help reduce offline processing.
- Lastly, modeling based on ensemble learning is becoming increasingly popular where several models trained on different tasks are combined together. An ensemble model would improve plant phenotyping in agriculture as ensemble models are typically more accurate than single models.

Bibliography

Ahn, H. S., Sa, I., & Dayoub, F. (2018). Introduction to the special issue on precision agricultural robotics and autonomous farming technologies. *IEEE Robotics and Automation Letters*, *3*(4), 4435–4438.

Aravind, K., Raja, P., Aniirudh, R., Mukesh, K., Ashiwin, R., & Vikas, G. (2018). Grape crop disease classification using transfer learning approach. *International Conference on ISMAC in Computational Vision and Bio-Engineering*, 1623–1633.

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, *39*(12), 2481–2495.

Bauer, S. D., Korč, F., & Förstner, W. (2011). The potential of automatic methods of classification to identify leaf diseases from multispectral images. *Precision Agriculture*, *12*(3), 361–377.

Behmann, J., Schmitter, P., Steinrücken, J., & Plümer, L. (2014). Ordinal classification for efficient plant stress prediction in hyperspectral data. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *40*(7), 29.

Bertels, J., Eelbode, T., Berman, M., Vandermeulen, D., Maes, F., Bisschops, R., & Blaschko, M. B. (2019). Optimizing the dice score and jaccard index for medical image segmentation: Theory and practice. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 92–100.

Bhimte, N. R., & Thool, V. (2018). Diseases detection of cotton leaf spot using image processing and svm classifier. *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, 340–344.

Biradar, B. V., & Shrikhande, S. P. (2015). Flower detection and counting using morphological and segmentation technique. *Int. J. Comput. Sci. Inform. Technol*, *6*, 2498–2501.

Biskup, B., Scharr, H., Schurr, U., & Rascher, U. (2007). A stereo imaging system for measuring structural parameters of plant canopies. *Plant, cell & environment*, *30*(10), 1299–1308.

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

Bucksch, A., Burridge, J., York, L. M., Das, A., Nord, E., Weitz, J. S., & Lynch, J. P. (2014). Image-based high-throughput field phenotyping of crop roots. *Plant Physiology*, *166*(2), 470–486.

Buzzy, M., Thesma, V., Davoodi, M., & Mohammadpour Velni, J. (2020). Real-time plant leaf counting using deep object detection networks. *Sensors*, *20*(23), 6896.

Chandra, A. L., Desai, S. V., Guo, W., & Balasubramanian, V. N. (2020). Computer vision with deep learning for plant phenotyping in agriculture: A survey. *arXiv preprint arXiv:2006.11391*.

Commission, G. C. (2021). Georgia cotton production guide.

Darwin, B., Dharmaraj, P., Prince, S., Popescu, D. E., & Hemanth, D. J. (2021). Recognition of bloom/yield in crop images using deep learning models for smart agriculture: A review. *Agronomy*, *11*(4), 646.

Davoodi, M., Velni, J. M., & Li, C. (2018). Coverage control with multiple ground robots for precision agriculture. *Mechanical Engineering Magazine Select Articles*, *140*(06), S4–S8.

Dobrescu, A., Valerio Giuffrida, M., & Tsaftaris, S. A. (2017). Leveraging multiple datasets for deep leaf counting. *Proceedings of the IEEE International Conference on Computer Vision*, 2072–2079.

Fue, K., Porter, W., Barnes, E., Li, C., & Rains, G. (2020). Center-articulated hydrostatic cotton harvesting rover using visual-servoing control and a finite state machine. *Electronics*, *9*(8), 1226.

Fue, K. G., Porter, W. M., Barnes, E. M., & Rains, G. C. (2021). Ensemble method of deep learning, color segmentation, and image transformation to track, localize, and count cotton bolls using a moving camera in real-time. *Transactions of the ASABE*, *64*(1), 341–352.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, *1*(2), 119–130.

Gaggion, N., Ariel, F., Daric, V., Lambert, É., Legendre, S., Roulé, T., Camoirano, A., Milone, D., Crespi, M., Blein, T., et al. (2020). Chronoroot: High-throughput phenotyping by deep segmentation networks reveals novel temporal parameters of plant root system architecture.

Gao, J., French, A. P., Pound, M. P., He, Y., Pridmore, T. P., & Pieters, J. G. (2020). Deep convolutional neural networks for image-based convolvulus sepium detection in sugar beet fields. *Plant Methods*, *16*(1), 1–12.

Gavhale, K. R., Gawande, U., & Hajari, K. O. (2014). Unhealthy region of citrus leaf detection using image processing techniques. *International Conference for Convergence for Technology-2014*, 1–6.

Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE international conference on computer vision*, 1440–1448.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 580–587.

Giuffrida, M. V., Doerner, P., & Tsaftaris, S. A. (2018). Pheno-deep counter: A unified and versatile deep learning architecture for leaf counting. *The Plant Journal*, *96*(4), 880–890.

Giuffrida, M. V., Minervini, M., & Tsaftaris, S. A. (2016). Learning to count leaves in rosette plants. *In Proceedings of the Computer Vision Problems in Plant Phenotyping (CVPPP) Workshop*.

Gong, L., Du, X., Zhu, K., Lin, C., Lin, K., Wang, T., Lou, Q., Yuan, Z., Huang, G., & Liu, C. (2021). Pixel level segmentation of early-stage in-bag rice root for its architecture analysis. *Computers and Electronics in Agriculture*, *186*, 106197.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, *27*.

Gormus, O., & Kanat, A. (1998). Yield and quality properties of cotton as affected by potassium fertilization. *Proceedings of the world cotton research conference*, *2*, 6–12.

Huang, Z., Li, Y., & Wang, H. (2020). In-field cotton boll counting based on a deep neural network of density level classification. *Journal of Electronic Imaging*, *29*(5), 053009.

Hunter, M. C., Smith, R. G., Schipanski, M. E., Atwood, L. W., & Mortensen, D. A. (2017). Agriculture in 2050: Recalibrating targets for sustainable intensification. *Bioscience*, *67*(4), 386–391.

iRobot Create2. (n.d.).

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1125–1134.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112). Springer.

Ji, R., Li, D., Chen, L., & Yang, W. (2010). Classification and identification of foreign fibers in cotton on the basis of a support vector machine. *Mathematical and Computer Modelling*, *51*(11-12), 1433–1437.

Jiang, Y., Li, C., Xu, R., Sun, S., Robertson, J. S., & Paterson, A. H. (2020). Deepflower: A deep learning-based approach to characterize flowering patterns of cotton plants in the field. *Plant methods*, *16*(1), 1–17.

Kadeghe, F., Glen, R., & Wesley, P. (2018). Real-time 3-D measurement of cotton boll positions using machine vision under field conditions.

Koornneef, M., Hanhart, C., van Loenen-Martinet, P., & Blankestijn de Vries, H. (1995). The effect of daylength on the transition to flowering in phytochrome-deficient, late-flowering and double mutants of Arabidopsis thaliana. *Physiologia Plantarum*, *95*(2), 260–266.

Kuznichov, D., Zvirin, A., Honen, Y., & Kimmel, R. (2019). Data augmentation for leaf segmentation and counting tasks in rosette plants. *CoRR*, *abs/1903.08583*. http://arxiv.org/abs/1903.08583

Li, Z., Guo, R., Li, M., Chen, Y., & Li, G. (2020). A review of computer vision technologies for plant phenotyping. *Computers and Electronics in Agriculture*, *176*, 105672.

Lim, J., Ahn, H. S., Nejati, M., Bell, J., Williams, H., & MacDonald, B. A. (2020). Deep neural network based real-time kiwi fruit flower detection in an orchard environment. *arXiv preprint arXiv:2006.04343*.

Minervini, M., Fischbach, A., Scharr, H., & Tsaftaris, S. A. (2016). Finely-grained annotated datasets for image-based plant phenotyping. *Pattern Recognition Letters*, *81*, 80–89.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Mokhtar, U., Ali, M. A., Hassanien, A. E., & Hefny, H. (2015). Identifying two of tomatoes leaf viruses using support vector machine. *Information systems design and intelligent applications* (pp. 771–782). Springer.

Möller, B., Schreck, B., & Posch, S. (2021). Analysis of arabidopsis root images–studies on cnns and skeleton-based root topology. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 1294–1302.

Ozguven, M. M., & Adem, K. (2019). Automatic detection and classification of leaf spot disease in sugar beet using deep learning algorithms. *Physica A: statistical mechanics and its applications*, *535*, 122537.

Pang, Y., Lin, J., Qin, T., & Chen, Z. (2021). Image-to-image translation: Methods and applications. *IEEE Transactions on Multimedia*.

Patki, S. S., & Sable, G. (2016). Cotton leaf disease detection & classification using multi svm. *International Journal of Advanced Research in Computer and Communication Engineering*, *5*(10), 165–168.

Pieruschka, R., & Schurr, U. (2019). Plant phenotyping: Past, present, and future. *Plant Phenomics*, *2019*.

Redmon, J. (2013–2016). Darknet: Open source neural networks in C.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.

Redmon, J., & Farhadi, A. (2018a). YOLOv3: An incremental improvement. *arXiv*.

Redmon, J., & Farhadi, A. (2018b). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Ren, M., & Zemel, R. S. (2016). End-to-end instance segmentation and counting with recurrent attention. *CoRR*, *abs/1605.09410*. http://arxiv.org/abs/1605.09410

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 91–99.

Revathi, P., & Hemalatha, M. (2012). Classification of cotton leaf spot diseases using image processing edge detection techniques. *2012 International Conference on Emerging Trends in Science, Engineering and Technology (INCOSET)*, 169–173.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *International Conference on Medical image computing and computer-assisted intervention*, 234–241.

Sampath, V., Maurtua, I., Aguilar Martın, J. J., & Gutierrez, A. (2021). A survey on generative adversarial networks for imbalance problems in computer vision tasks. *Journal of big Data*, *8*(1), 1–59.

Scharr, H., Minervini, M., French, A. P., Klukas, C., Kramer, D. M., Liu, X., Luengo, I., Pape, J.-M., Polder, G., Vukadinovic, D., et al. (2016). Leaf segmentation in plant phenotyping: A collation study. *Machine vision and applications*, *27*(4), 585–606.

Schmundt, D., Stitt, M., Jähne, B., & Schurr, U. (1998). Quantitative analysis of the local rates of growth of dicot leaves at a high temporal and spatial resolution, using image sequence analysis. *The Plant Journal*, *16*(4), 505–514.

Singh, A., Thakur, N., & Sharma, A. (2016). A review of supervised machine learning algorithms. *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 1310–1315.

Singh, A., Ganapathysubramanian, B., Singh, A. K., & Sarkar, S. (2016). Machine learning for high-throughput stress phenotyping in plants. *Trends in plant science*, *21*(2), 110–124.

Smith, A. G., Petersen, J., Selvan, R., & Rasmussen, C. R. (2020). Segmentation of roots in soil with u-net. *Plant Methods*, *16*(1), 1–15.

Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond accuracy, F-Score and ROC: A family of discriminant measures for performance evaluation. In A. Sattar & B.-h. Kang (Eds.), *Ai 2006: Advances in artificial intelligence* (pp. 1015–1021). Springer Berlin Heidelberg.

Tedesco-Oliveira, D., da Silva, R. P., Maldonado Jr, W., & Zerbato, C. (2020). Convolutional neural networks in predicting cotton yield from images of commercial fields. *Computers and Electronics in Agriculture*, *171*, 105307.

Ubbens, J., Cieslak, M., Prusinkiewicz, P., & Stavness, I. (2018). The use of plant models in deep learning: An application to leaf counting in rosette plants. *Plant methods*, *14*(1), 1–10.

Ubbens, J. R., & Stavness, I. (2017). Deep plant phenomics: A deep learning platform for complex plant phenotyping tasks. *Frontiers in plant science*, *8*, 1190.

Valerio Giuffrida, M., Dobrescu, A., Doerner, P., & Tsaftaris, S. A. (2019). Leaf counting without annotations using adversarial unsupervised domain adaptation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 0–0.

Vongs, A., Kakutani, T., Martienssen, R. A., & Richards, E. J. (1993). Arabidopsis thaliana DNA methylation mutants. *Science*, *260*(5116), 1926–1928.

Walter, A., & Schurr, U. (1999). The modular character of growth in Nicotiana tabacum plants under steady-state nutrition. *Journal of Experimental Botany*, *50*(336), 1169–1177.

Wang, J., Gao, Z., Zhang, Y., Zhou, J., Wu, J., & Li, P. (2022). Real-time detection and location of potted flowers based on a ZED camera and a YOLO V4-tiny deep learning algorithm. *Horticulture*, *8*(1), 21.

Wang, T., Rostamza, M., Song, Z., Wang, L., McNickle, G., Iyer-Pascuzzi, A. S., Qiu, Z., & Jin, J. (2019). Segroot: A high throughput segmentation method for root image analysis. *Computers and Electronics in Agriculture*, *162*, 845–854.

Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Tao, A., Kautz, J., & Catanzaro, B. (2018). High-resolution image synthesis and semantic manipulation with conditional gans. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Xu, L., Li, Y., Sun, Y., Song, L., & Jin, S. (2018). Leaf instance segmentation and counting based on deep object detection and segmentation networks. *2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS)*, 180–185.

Xu, R., Li, C., & Paterson, A. H. (2019). Multispectral imaging and unmanned aerial systems for cotton plant phenotyping. *PloS one*, *14*(2), e0205083.

Xu, R., Li, C., Paterson, A. H., Jiang, Y., Sun, S., & Robertson, J. S. (2018). Aerial images and convolutional neural network for cotton bloom detection. *Frontiers in plant science*, *8*, 2235.

Yeung, M., Sala, E., Schönlieb, C.-B., & Rundo, L. (2022). Unified focal loss: Generalising dice and cross entropy-based losses to handle class imbalanced medical image segmentation. *Computerized Medical Imaging and Graphics*, *95*, 102026.

Yi, Z., Yongliang, S., & Jun, Z. (2019). An improved tiny-yolov3 pedestrian detection algorithm. *Optik*, *183*, 17–23.

Yushkevich, P. A., Piven, J., Cody Hazlett, H., Gimpel Smith, R., Ho, S., Gee, J. C., & Gerig, G. (2006). User-guided 3D active contour segmentation of anatomical structures: Significantly improved efficiency and reliability. *Neuroimage*, *31*(3), 1116–1128.

Zekiwos, M., Bruck, A. et al. (2021). Deep learning-based image processing for cotton leaf disease and pest diagnosis. *Journal of Electrical and Computer Engineering*, *2021*.