ITErRoot: High Throughput Segmentation of 2-Dimensional Root System Architecture

A thesis submitted to the
College of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Kyle Seidenthal

Permission to Use

Disclaimer

Abstract

Root system architecture (RSA) analysis is a form of high-throughput plant phenotyping which has recently benefited from the application of various deep learning techniques. A typical RSA pipeline includes a segmentation step, where the root system is extracted from 2D images. The segmented image is then passed to subsequent steps for processing, which result in some representation of the architectural properties of the root system. This representation is then used for trait computation, which can be used to identify various desirable properties of a plant's RSA. Errors which arise at the segmentation stage can propagate themselves throughout the remainder of the pipeline and impact results of trait analysis.

This work aims to design an iterative neural network architecture, called ITErRoot, which is particularly well suited to the segmentation of root structure from 2D images in the presence of non-root objects. A novel 2D root image dataset is created along with a ground truth annotation tool designed to facilitate consistent manual annotation of RSA. The proposed architecture is able to take advantage of the root structure to obtain a high quality segmentation and is generalizable to root systems with thin roots, showing improved quality over recent approaches to RSA segmentation. We provide rigorous analysis designed to identify the strengths and weaknesses of the proposed model as well as to validate the effectiveness of the approach for producing high-quality segmentations.

Acknowledgements

I would like to extend my gratitude to my supervisor, Dr. Mark Eramian. Your guidance and mentorship has shaped me as a researcher, and I sincerely appreciate your patience and expertise which have made this project what it is today. I have truly enjoyed working with you.

Thank you to the members of the Global Institute for Food Security (GIFS) and Agriculture and Agri-Food Canada (AAFC) who have worked hard to collect the image data-set that made this work possible, and for sharing their expertise on the plant side of things. I would like to thank Dr. Zhigang Liu at GIFS for providing the Cucumber images, Dr. Rahul Chandnani and Tongfei Qin at GIFS for providing the Wheat and Soybean images, and Hasan Ahmed and Adrian Cabral at AAFC for providing the Canola images.

To Cai Lowe, thank you for being supportive and helping me keep on track.

Thank you to Peggy Anderson, for being a great friend every step of the way, and for listening to my good and bad ideas. Your support during this time has been invaluable.

Thank you to my parents, for encouraging me to follow my passions.

Thanks to the members of the IMG lab, and the members of the lab formerly known as DISCUS. You've been helpful and kind, and I am sure I would have had a lot less fun without you.

Contents

List of Tables

List of Figures

List of Abbreviations

AAFC	Agriculture and Agri-Food Canada
AUC	Area Under the ROC Curve
CNN	Convolutional Neural Network
CSV	Comma Separated Value
CLAHE	Contrast Limited Adaptive Histogram Equalization
DAT	Days After Transplanting
EM	Expectation Maximization
XML	Extensible Markup Language
FN	False Negative
FP	False Positive
GAN	Generative Adversarial Network
GDL	Generalized Dice Loss
GMM	Gaussian Mixture Model
GTC	Ground Truth Composite
GUI	Graphical User Interface
IoU	Intersection Over Union
P2IRC	Plant Phenotyping and Imaging Research Centre
PCA	Principal Component Analysis
PC	Predicted Composite
ReLU	Rectified Linear Unit
RSML	Root System Markup Language
RSA	Root System Architecture
RTP	Root-Tip Path
SSL	Sensitivity-Specificity Loss
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
WCE	Weighted Cross-Entropy Loss

1 Introduction

Analysis of root system architecture (RSA) is a branch of high-throughput plant phenotyping which makes use of 2D and 3D image datasets of plant root systems to examine quantitative traits of different species and breeds of plants. Plant breeders make use of these traits in studies to understand the properties and behaviours of different plant species in order to improve crop development. Understanding of the topological structure of root systems can help to identify properties that increase drought tolerance, nutrient uptake, and effects of different soil conditions on the growth of the plant [46, 56]. As such, phenotyping of RSA is a crucial step in the process of developing sustainable agriculture and food security methods.

RSA characterizes the properties of a root system as it grows in the soil. The physical traits, or phenotypes, of a crop's RSA can be matched with particular genotypes of the plant of interest. For plant breeders, this means that plants can be designed with genotypes that produce desired root phenotypes, optimizing the plant's physical properties to produce increased yield. Deeper growing roots, for example, have been shown to increase yield and present moisture management solutions even in agricultural areas which experience water stress such as drought [55]. Understanding the root traits that produce desirable crop properties is essential to efforts in global food security. High-throughput root phenotyping pipelines are a key element to improving the ability for breeders to link root phenotypes with plant genotypes, allowing the design and improvement of crops in adverse conditions to produce high yield. This work explores the importance of image segmentation in the context of high-throughput root phenotyping pipelines to help improve identification of RSA traits.

Studies on RSA can generate large databases of root images which need to be processed in order to extract the root structure and compute desired traits. Images of roots exhibit different properties depending on the chosen apparatus for imaging. One such apparatus is the rhizotron [37], which allow the root to grow in soil while being visible through a glass pane for image acquisition. Another method is to grow the roots in thin folders, forcing the roots to grow on a 2D plane. At imaging time, the roots can be taken from the folder and placed on a black background, allowing the entire root system to be entirely visible from the perspective of the camera. Both of the aforementioned imaging systems can be used to procure 2D images of roots. While 3D images can also be useful for the phenotyping process, this work focuses on the use of 2D images of root systems.

One of the first steps in the extraction process is segmentation of the root system from the image background, which must be done consistently for all images involved in a study. Segmentation operations must be generalizable to different plant species while maintaining a high quality segmentation output. While many approaches to image segmentation exist, none of these approaches are particularly well suited to extracting

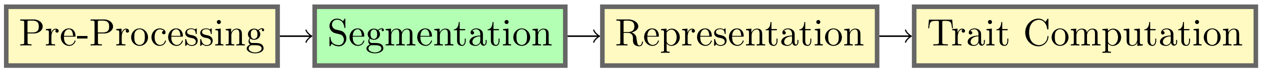


Figure 1.1: The overall RSA pipeline which consists of the image pre-processing step, the root segmentation step, the representation step, and the trait computation step. This work focuses on the segmentation step.

the thin branching structure of RSA in the presence of other non-root objects, which are considered to be noise in the segmentation. This work aims to address the problems with current segmentation approaches for 2D RSA and design an automated and generalizable approach that is well suited to high-throughput segmentation of root systems.

RSA analysis can be distilled into a number of steps which define a larger pipeline, depicted in Figure 1.1. Once imaging is complete, the images go through a pre-processing step. This can be as simple as a normalization operation, which ensures that all images have similar histogram properties before further processing. Cropping the image to remove non-root objects is another useful pre-processing operation, as it can reduce the amount of processing to only that necessary to extract the root from the image, and can also improve the accuracy of certain segmentation algorithms in the next step of the pipeline. Other operations can be applied to the images to attempt to enhance certain features, such as brightness or contrast adjustments. Depending on the approach used in the pipeline these can be done automatically, or by a human annotator just before the segmentation step. The purpose of this step is to ensure that all images have properties that will enhance the information that can be used in later steps in the pipeline.

Once images have been prepared by the pre-processing step, the root system is segmented from the image. The goal of the segmentation step is to distinguish the pixels in the image which contain root from all other pixels in the image. A higher quality segmentation will allow better information to be passed to subsequent steps. Ideally, we want a segmentation that perfectly represents every pixel of the RSA in the input image, though in reality this is quite difficult to achieve. Limitations in imaging systems and image representation can make it difficult to perfectly represent the root system, and while pre-processing steps can help to bring out qualities which give a more accurate segmentation, many segmentation algorithms make assumptions about the input data in order to classify each pixel as either foreground or background. In most cases we can determine a sufficiently high quality for a segmentation to be able to represent the root system, where quality can be determined by various metrics, though the higher the quality of the segmentation, the more accurate further computations will be. Segmentation can be done manually by human annotators, or make use of various image processing algorithms and neural network architectures to achieve an automatic process. These different approaches to segmentation have various advantages and drawbacks, which will be further explored in Chapter 2.

Next, the extracted root segmentation must be stored in a format which allows easy understanding of the structure. The representation step encodes the root structure in a chosen format which can then be stored for later use. An appropriate method for RSA representation will allow further computation of traits based on the root structure in an efficient and extensible manner. Portability of the representation is also an important factor, to allow for extracted RSA to be used in different studies and pipelines without loss of information. Many approaches to this step exist, each with their own advantages and drawbacks. Some common approaches to representation will be discussed in Section 2.2.4.

Finally, the trait computation step makes use of the stored representation to calculate various local and general traits of the whole RSA. This step should be customizable to any given study so that breeders can produce measurements specific to their research goals. By specifying a set of desired traits the representation from the previous step is used to perform calculations on the root system. Some examples of traits are *total root length*, *average root diameter*, and *convex hull*. There are many possible root traits of interest which vary depending on the study, any of which should be achievable at this step.

While each step in the pipeline is important in processing a given root image, the segmentation step in the RSA pipeline is the first point where data is being extracted from the input images. This step is crucial for representing the RSA in later steps, and can even improve the results of the trait computation step. An incorrect segmentation will lead to misrepresentation of the root system and cause errors in trait computation, while a very accurate segmentation gives us all the information needed to understand the paths and properties of each root in the system. It is for this reason that we must develop an accurate segmentation process that is robust in identifying roots of any size and can be generalized to any species of root which may need analysis. The complex structure of RSA makes this a difficult problem, where many classical image processing algorithms cannot achieve a level of detail sufficient for representation of the root system while remaining robust to noise. In this thesis we show that our iterative neural network architecture, ITErRoot, is capable of producing high quality segmentation of RSA in the presence of non-root objects.

1.1 Research Objectives

The objective of this work is to design and evaluate a generalizable and accurate automated approach to root segmentation. The approach will make use of image datasets obtained as part of the Plant Phenotyping and Imaging Research Centre (P2IRC) project to facilitate further study of root systems. Specifically, improvements will be made on the current state of neural network architectures for segmenting RSA by taking into consideration how these architectures might leverage the inherent properties of root systems and the presence of non-root objects in the image.

1.2 Contributions

This work will make the following contributions to the current state of RSA analysis:

- A novel tool for annotation of root systems for producing ground truth segmentations
- A novel dataset of 2D images of root systems of different species, along with ground truth segmentations

- A neural network architecture ITErRoot, that is well suited to segmentation of root systems in the presence of non-root objects

In order to train and evaluate the segmentation network, ground truth segmentations must be created for the root images to be used. Due to the size of the images and the root systems in them, the annotation process can be time consuming and error prone, and inconsistencies can arise between different annotators. Images can be difficult to navigate via panning and zooming at full scale, and the roots can be too complex for a simple outline or paintbrush tool to be effective in all cases. For these reasons an annotation tool designed to reduce the complexity involved in annotating these images is introduced. The tool will allow the user to focus on patches of the image and provide a simple array of tools to help the user attain the level of detail needed for annotation. These tools should allow more control over the output segmentation in localized areas, helping to reduce inconsistencies among many annotators. Unlike other current annotation tools, this tool will provide semi-automated annotation of localized neighbourhoods in the image, giving the user the fine grained control over the annotation process that is demanded by the high-resolution details common to RSA images.

Imaging efforts for the P2IRC project have created new sets of root images for analysis. Plants are being grown in thin folders to ensure that roots grow on a 2D plane and then imaged at different stages in the growth process. At imaging time, plants are placed on a well lit dark background before being imaged at high resolution. Once acquired, the images can be used in various studies of RSA, however accurate ground truth annotations have not yet been produced for these datasets. To facilitate further study and evaluation of RSA pipelines, the ground truth tool designed in this project will be used to produce segmentation masks to match the images acquired for the P2IRC project. This new dataset will be used to train and evaluate the proposed neural network architecture, and to compare results with other approaches to root segmentation.

Finally, a neural network architecture called ITErRoot is designed and evaluated on the annotated ground truth segmentations. This architecture will improve upon the designs of current segmentation approaches by making use of recent deep learning techniques to incorporate properties of the root system in the segmentation process. ITErRoot will be evaluated for accuracy in segmentation and generalizability to species of root that have not been used in the training process. This segmentation method will provide segmentations which retain the RSA in the presence of non-root objects, which can later be used to improve the overall RSA pipeline.

2 Review of Literature

Image segmentation is the process of identifying the pixels in an image that belong to a set of classes, usually background and foreground. Segmentation can be done in many different ways, including manual annotation, use of classical image processing algorithms, or by convolutional neural networks (CNNs). Different algorithms and approaches have different requirements for image properties and may make assumptions about the structure of the objects within the image. In this work, we seek to identify pixels in an image as either root or non-root. Interestingly, the problem of segmenting plant roots is very similar to the problem of vessel segmentation in medical imaging, and crack or road segmentation from satellite or drone imagery. All of these problems deal with identifying thin, branching structures in scenarios where background and image noise cause interference. As such, many applications used in the domain of one of these problems can be used to help solve problems in another, with slight modification. With tools such as CNNs and deep learning we can make use of architectures that leverage small important details to improve the accuracy of a segmentation. The rest of this chapter outlines works related to RSA and the segmentation of thin branching structures to outline the current state of the art. Section 2.1 gives a brief overview of image segmentation, Section 2.2 discusses approaches to quantification and study of RSA traits that are commonly used by plant breeders for their research, and Section 2.3 covers details of more general approaches to segmentation via deep neural networks.

2.1 A Brief Overview of Image Segmentation

The goal of image segmentation in general is to separate an image into regions which identify objects from their background. Often this means classifying each pixel in an image as either background or foreground, though multi-class or semantic segmentation problems do exist [3]. Segmentation approaches can take many different forms. One traditional approach is image thresholding, whereby a single pixel intensity value is determined which splits the image histogram such that pixels with higher intensity are labeled as foreground, and pixels with lower intensity than the threshold value are labeled as background. Many threshold based segmentation approaches exist, such as Otsu's method [39] which attempts to automatically infer the threshold value based on the image histogram, or adaptive thresholding methods which operate on smaller neighbourhoods of the image to obtain a full segmentation [57].

Other approaches to image segmentation include edge-based and region-based approaches. Edge-based approaches make use of edge detection methods such as Canny edge detection [12, 47] to identify the pixels in the image which represent the edges of the objects. The edge information is then used to determine how to label the remaining pixels in the image. Region-based approaches make use of initial points to expand or grow the regions which make up the segmented objects [1]. Each of these approaches have benefits and drawbacks for image segmentation. Edge-based segmentation benefits from images where edges of objects have sharp contrast, making them well defined for edge detection algorithms, however when edges within the image become ambiguous, such approaches can suffer. Region-based approaches are ideal if there is a clear method for determining starting or seed points. Finally, there are deep learning approaches to image segmentation, which leverage recent neural network architectures and training techniques. These approaches are further discussed in the remaining sections of this chapter.

2.2 Current Image-Processing Based Approaches to Root System Analysis

There have been a number of approaches that apply image processing techniques to extracting RSA traits from plant roots. This section presents an overview of current software that exists to measure RSA traits from root images, with a focus on 2D imaging systems. There are three possible approaches to this research problem: manual, semi-automatic, and automatic. Manual approaches require heavy user interaction to achieve accurate results, semi-automatic approaches require user guidance to properly initialize algorithms that can automatically compute the desired traits, and automatic approaches present an almost user-free pipeline that can process large amounts of data with little to no user interaction. The major differences between these approaches manifests itself as a resource-accuracy trade-off. In general, manual pipelines require considerable amounts of labour and time resources, as they operate with a “human in the loop” paradigm. The major bottleneck of these systems is the speed that a human annotator can make measurements and annotations, which reduces the throughput of the phenotyping process. Additionally, inconsistencies and errors can arise among different human annotators, introducing another point of failure in such pipelines. The upside to manual approaches however, is that a human has the opportunity to validate the results as they progress through the pipeline. This can lead to higher accuracy in some cases, as an expert annotator can identify areas in the RSA which need to be annotated, though this is not a guarantee and in many cases bias and inconsistencies will occur across a large dataset. Quite the opposite, fully automated approaches reduce the time required to process an image to the theoretical limits of the algorithms and hardware used to implement them. This increases the throughput of the analysis and reduces the amount of tedious work that is required. While we can process images more quickly this way, it is very difficult to ensure that an algorithm or pipeline can truly generalize to any input set of images, and thus we see a reduction in overall accuracy of the pipeline. These approaches are discussed in more detail below along with recent approaches to RSA analysis tools.

2.2.1 Manual Approaches

Manual approaches considered here are software tools that provide a manual point-and-click interface for a user to measure RSA traits. Many of these tools can be considered to be annotation tools, though many perform calculations based on the annotations provided by the user. Manual approaches often perform most or all of the steps in the RSA pipeline and are designed to facilitate direct computation of RSA traits for a specific study.

EZ-Rhizo, described by Armengaud *et al.* [4], provides a graphical user interface (GUI) that allows the user to perform a series of steps to process and analyze the input image. First they must convert the image to a binary image using a manually selected threshold. The image is then cropped and optionally de-noised using a selected algorithm. A dilation step is automatically performed after de-noising to fill any accidental root gaps. Next, a skeletonization process is initialized, which can be touched up manually after completion to remove any errors. Contiguous white pixels forming a connected component are used to identify root objects in the image. Then the user is presented with a series of dialog boxes, one for each fully connected component, to confirm or reject the object as root. From these fully connected components, the RSA traits are calculated and stored in a text file, which can be further processed by other software. The text files may also be entered back into the program to be imported into a MySQL[1] database. This system is a very simple approach to measuring RSA traits and allows the user to guide the process to avoid errors, however for large datasets this system can be unwieldy. The paper by Armengaud *et al.* [4] provides an analysis of the computed traits with principal component analysis (PCA) to show the use of their software.

In many cases it is useful to look at time series images of roots to identify traits such as growth rate and to observe root branching positions. *DART*, implemented by LeBot *et al.* [31], is a manual approach which models the roots as a set of links that represent the coordinates of each root segment. A link incorporates date information for when the segment started to grow and the hierarchy of links that precede it. The user must manually identify the links on each root image over the entire time series. Once the root has been annotated, the software can compute basic traits such as branching order, date of emergence, distance between branching points and parent base, and root length at observation date. These data are output in a table file for subsequent use.

RootScape, developed by Ristova *et al.* [45], provides a landmark based approach to RSA, which requires the user to place a series of predefined landmarks on the image to denote the root structure. Once the landmarks have been placed on the image, a *MATLAB*[2] script is used to place secondary landmarks on the image. These landmarks are used to create a "40-dimensional space where each axis represents variation in one of the coordinate values [45]." This information is used to define a point cloud over many different examples, the centre of which represents a mean root shape. The study by Ristova *et al.* uses PCA to see how the software compares to individual traits that were extracted for Arabidopsis plants.

[1]https://www.mysql.com/
[2]https://www.mathworks.com/products/matlab.html

Moller *et al.* [36] have developed a *FIJI*[3] plugin that allows the user to manually annotate time series images that come from minirhizotrons. The annotation tool includes a set of configurable labels that can be applied to any node to represent some information (eg. "Living" or "Dead"). The software uses the image file names to sort out timestamps for the time series so that it can be analyzed, which means that image files must conform to their naming standard before analysis. An already imported time series can be added to by importing a new set of images, which allows the same time series to be continuously analyzed as more data is acquired. The user adds nodes and segments to the image via mouse clicks to annotate the root structure. The annotations are then used to represent the root structure based on a series of treelines, a series of nodes and segments, which can be updated by moving the base node to a new position. When a user moves an existing treeline, the segments of that line are automatically updated. Treelines can be merged together or split apart at any node specified by the user to make editing simple. Root diameter at each node is annotated as necessary. The whole root system can be exported to Root System Markup Language (RSML) for further processing (see section 2.2.4 for more details on RSML).

The tools discussed here perform most of the steps in the RSA pipeline with direct input from the user. Regardless of how these tools operate on a given image, the user is marking which parts in the image are root (foreground) from background, identifying root tips and other landmarks, and obtaining some form of computed traits. Many of these approaches do not directly output a representation of the root system, but rather use the annotations of the user as the representation to directly compute traits. This high-coupling of the RSA pipeline steps makes it difficult to directly compare and improve upon specific steps in the pipeline. While the manual annotation style does allow the user to monitor annotations on the image to ensure the root system is being represented properly, it does not scale to large datasets due to the time investment required, and many of these tools focus on a set of specific traits making it difficult to generalize a single tool to other studies of RSA.

2.2.2 Semi-Automatic Approaches

Semi-Automatic tools require interaction from the user, but offer some form of guided or automatic features that make annotation and measurement less tedious. Many of these tools require the user to input some parameters (eg. a threshold value) to operate. As with manual approaches, semi-automatic approaches can perform most or all of the RSA pipeline, though the added benefit of guidance allows for an (usually) easier annotation experience.

SmartRoot, as presented by Lobet *et al.* [34], is an analysis tool that provides a GUI with four layers that will be operated on: the image, line segments and vectors, topological information, and annotations. The user can click on a root and the software will automatically determine its midline so that it can construct a segmented line that progresses to the tip and base of the root. As nodes are added to the line, root diameter is estimated and stored, and lateral roots are detected so they can be traced to their parent root. In the event

[3]https://imagej.net/Fiji

of an error the user can correct by dragging the detected line segments. The interface provides a way for users to annotate the image in any way they like. The annotations and measurements can then be exported to an SQL database for further processing. Time series data can be imported into the software, however they must be analyzed as individual images rather than as a group.

Galkovskyi *et al.* [18] have developed *GiARoots* to allow for fast analysis of root images. The user can import a folder of images which are treated as a stack that can be scrolled through for easy navigation. The user may apply a preprocessing step to crop, rotate, or calibrate the scale of the image to ensure consistent image properties for ease of analysis. Image scale is used to convert pixel units into real world units, which is done by placing a ruler on the image and measuring a known distance. Next, a segmentation algorithm is selected and a threshold value is chosen via a slider to obtain a binary image. The values selected by the user for various steps in the analysis can be saved for later automated segmentation on other images. Global, adaptive, and double adaptive thresholding are provided by *GiARoots* as options for segmentation algorithms. Additionally, the tool provides a command line interface which can be used to create an automated pipeline using previously determined inputs, where all intermediate computations are exported and saved for quality assurance. This automated pipeline assumes that chosen input values will work equally well for all images. The system calculates a series of RSA traits from the segmentation and exports them as a comma separated value (CSV) file. They compare their software to another approach by Iyer-Pascuzzi *et al.* [24] and achieve a very high correlation ($R^2 \approx 1.0$), though they did modify the definitions of some of the trait calculations used. In particular, they modify the *network length distribution* and *root thickness* definitions, which may add a small bias to the result. They also provide a comparison of the different thresholding algorithms and how they perform on their image dataset.

Basu *et al.* [7] have improved the user experience with time series data by allowing the user to annotate a single image in the time series and propagating the information across the entire series. The user may downsample and crop the input images before processing. Segmentation of the root system is computed using the *livewire* [6] algorithm across the time series by treating it as a 3D volume. Once the segmentation is computed, the user must select the root tips on the segmentation, so that they can be used to compute the growth of the root over time. The tool was evaluated on synthetic images, which allowed them to compare the output of the software with a pre-determined set of measurements for each image. While this does provide a benefit, there is no sense for the accuracy on real-world images given from this approach.

Pound *et al.* [42] have designed *RootNav*, which is a top-down approach to RSA extraction. The tool applies a pre-defined set of models which make use of expectation maximization (EM) and A* search [21] to identify root paths. Users may modify and create their own models if they desire. The user is able to guide and correct root paths by dragging points on the path to the desired location. The user must specify the root tips and sources before the A* search can be applied, and once finished a Gaussian mixture model (GMM) is applied to classify the pixels into background and foreground components. To deal with problems caused by noise, the image is split into smaller patches and the GMM classification is applied to the patches.

This allows localized illumination to be used to avoid large changes in brightness or contrast over the whole image. Once the likelihoods of the pixel classes have been calculated, they are used to convert the image pixels into a weighted graph, where the edge weights are a function of the class likelihood. Root classes are given a lower cost, and background classes a hight cost. The A* search finds the optimal path from each root tip to the source. The points along this line can be manually corrected if necessary, followed by fitting of a spline to allow for easier calculations of RSA traits.

Pierret *et al.* [41] proposed *IJ-Rhizo*, an *ImageJ*[4] macro designed to measure root length and diameter, and have compared the outputs of this software with those of *WinRhizo*[5]. Their paper does not discuss the methods they use for segmentation, measurement, or any internal representation of root structure. The software focuses on computing root length and diameter traits, but there is little detail on how this is accomplished. Overall the software performed comparable to *WinRhizo*, with high-positive correlation of root length features.

ARIA, developed by Pace *et al.* [40] is a tool that uses a single image of roots at a time to extract RSA traits. The software automatically converts 2D images of plant roots into a graph notation, which allows computation of various RSA traits. The background is first segmented from the image using Otsu's method [39], which results in a binary image where the root pixels are determined to be the largest connected component of foreground pixels; all other foreground pixels are discarded as noise. Once the largest connected component has been identified, it is skeletonized and the remaining points on the skeleton are used as nodes in a graph, with edges between adjacent pixels. This representation seems to be extendable and effective for representing root structures, especially because of the well understood nature of graph structures. The *ARIA* framework is also "extendable to 3D tomography image data" [40]. In their paper, they compare measurements with *WinRhizo* and find that they have similar results. *ARIA* was used to phenotype maize seeds that were grown in a controlled environment for study and was able to compute 27 different root traits. No segmentation results were presented, which makes it difficult to say how the segmentation method affects their results.

DynamicRoots, designed by Symonova *et al.* [53] accomplishes RSA by constructing a time series of 3D shapes from 2D images of root systems. The shapes are then aligned to define a depth function, and each shape is decomposed into a hierarchy of branches using its own depth function. At this point, a time function is constructed and used to re-organize the hierarchy by "repairing inconsistencies between depth and time of growth" [53]. Various traits of the root system are then computed from the distance and time functions on the branch hierarchy. The images used in the paper were acquired by placing a container holding a plant on a rotating table. The images taken during rotation are used to construct the 3D volume, which is repeated at different moments in time so that the growth of the system can be analyzed. The analysis relies on three major assumptions: The root system can only grow by generating a new branch at a fork or by elongating an

[4]https://imagej.net/Welcome
[5]https://regent.qc.ca/assets/winrhizo_about.html

existing branch at the tip, the system is connected and contractible, and the time series is dense enough to observe the correct root hierarchy. The proposed software can handle small violations of these assumptions, but will fail if there is a large deviation from them.

Rellan-Alvarez *et al.* [44] proposed *GLO-Roots*, an integrated imaging system that enables the study of root architecture. The system uses luminescent reporters to image root systems grown in rhizotrons. The software associated with the system, named *GLO-RIA* allows four types of analysis: local, global, shape, and directionality. Local analysis performs computations on each root pixel to determine root length, direction, and position, while global analysis takes into account the entire root system and computes the visible surface, convex hull, width, and depth. Shape analysis makes use of frequency domain landmarks that perform similarly to the approach taken by Ristova *et al.* in *RootScape* [45]. Finally, directionality analysis computes directional features of the root system in specified regions of the image. All of these methods are automated, but allow for manual adjustments if necessary. The software also allows detection of gene reporters and displays various structure and soil attributes, which can be exported to the *RSML* format. The software was tested for accuracy using manual measurements from *ImageJ*, as well as a large set of generated images of root systems.

Reeb *et al.* [43] have developed *MorphoSnake*, a tool that can automatically compute traits at different levels of thallus for analysis. Existing root analysis cannot achieve this because thalloid plants do not have a straightforward branching pattern, making it difficult to find a branch to base analysis off of. When imaging, they manually fixed overlaps in the plant by physically moving roots so that they could avoid errors in processing. The processing is done by computing a skeletonization of the thallus and converting to a graph structure. No segmentation is done by the software, the input must be a binary mask image. The graph structure is then used to compute and visualize the traits and structure of the thallus.

RootSnap![6] is a manual root annotation program designed to work with touch screen devices. It allows the user to draw curves on an image to represent the root structure while automatically snapping points on the curve to the centre of the root. Once the structure has been defined by a series of curves, the software can compute the following traits: root length, area, volume, average diameter, root angle, branch angle, start tip angle, and end tip angle. During the annotation phase, the user can activate an "auto detect" feature, which takes the currently selected curve and tries to extend it to the root tip automatically. This can work well, but only after carefully configuring image brightness, contrast, and gamma using the tool's image adjustment features. The user must also adjust the scale of the area under their finger (or under the cursor when using a mouse) so that the estimated diameter of points along the curve can be correct, however it is possible to set this scale too small such that it will obtain incorrect diameter estimates along a given curve. Overall the annotation process is tedious and error prone, particularly if the image does not have desirable brightness and contrast properties. Once complete, the software can export the computed values to a CSV file.

Similar to manual RSA approaches, many semi-automatic approaches aim to be a full RSA pipeline

[6]https://www.quantitative-plant.org/software/rootsnap

in one component. While the tools discussed here make improvements on the user experience to speed up annotation of root images, they still require human intervention which makes them difficult to scale to larger datasets for high-throughput phenotyping. Many of the automatic algorithms used in these approaches require image specific input parameters to be accurate. Reliance on such specific parameters makes these approaches accurate when the input data meets the requirements, though a fair amount of effort still must be made to achieve accurate results.

2.2.3 Automatic Approaches

Automatic approaches to RSA are processes that require minimal to no input from the user, and can operate on large batches of images to obtain a segmentation or result. Many of the programs described in this section are GUI programs that require the user to do some initial setup and optional manual adjustments to the outputs. These are still automatic in the sense that the algorithm can perform without intervention from the user, though it may benefit from small tweaks specific to the current images. There are two main groups of approaches in this section, those that attempt to automatically compute a set of RSA traits, and those that focus on segmentation, classification, and representation of root structure from the images. In the latter cases, the output is some representation that can be used with other software to compute RSA traits.

Automatic Trait Computation

The automatic trait computation approaches discussed here combine all of the steps in the RSA pipeline into a single piece of software. These tools make use of various image processing based segmentation and classification techniques to automatically identify the root system in each image, followed by some form of representation of the structure for computing desired traits.

Bucksch *et al.* [9] outline a portable process for extracting RSA traits of excavated plants in the field. This analysis technique is destructive as it requires the plants to be removed from the soil, cleaned, and then placed on a black imaging board. The imaging board used was designed with specific reflective qualities to ensure ease of digital processing. Experiment tags and scale markers are placed on the imaging board along with the root. Once an image has been captured using this set-up, the PAL system is used to convert the image to grayscale. A global threshold is found using Otsu's method [39], which is used as the input to an adaptive thresholding algorithm to segment the root system and experiment tags. Next, a connected component labelling is used to remove noise from the thresholding process and the root, experiment tag, and scale marker are identified using knowledge about their expected shape and size relative to other objects on the imaging board. The segmentation mask is used to compute a number of RSA traits and the root system mask is input into an algorithm that computes a series of root-tip paths (RTPs) for the system. Each RTP is added to an overall skeleton of the root system, which is represented as a graph where nodes are pixels and edges between nodes represent neighbouring pixels. This graph can be used to compute additional root traits, as well as to identify root tips and branching points based on the number of neighbours a given pixel has. This study outlines an approach that allows high-throughput phenotyping in the field, as well as defining new traits that could not previously be measured via manual measurement methods, though the processing pipeline is highly dependent on the imaging process and settings that were used in the study. This illustrates the benefits of automated phenotyping systems while showing that improvements can be made to make such automated pipelines generalizable.

Iyer-Pascuzzi *et al.* [24] introduced an automatic imaging system that images root systems grown in transparent tubes. The camera is rotated around the plant to acquire images from different angles, which are then sent to a processing pipeline. The first step is to pre-process the image and break it up into 100×100 pixel squares, each of which are thresholded using the mean pixel intensity of the square. Specifically, pixels that are 5% (a configurable value) above the median intensity are determined to be a part of the root system. Connected components of the root are found from these results and determined to be the foreground of the whole image. Next, a series of 16 traits are computed from the foreground pixel values of the image. They then use a support vector machine (SVM) to analyze which traits are significant in distinguishing a pair of specific genotypes. The SVM finds a separation of the 16-dimensional space (determined by the 16 traits) that classifies the images into their respective genotypes. Though this analysis is quite useful, it does require pre-labelled sets of images for each genotype along with the required features, and has only been used in this paper to distinguish between a given pair of genotypes; there is no reference for how extendable this approach is to classifying between larger sets of genotypes.

BRAT, implemented by Slovak *et al.* [50], is a *FIJI* plugin that automatically segments and analyzes root traits from images acquired by flatbed scanners. There is an automatic mode which can be parallelized, and a quality control mode that allows the user to correct the segmentations in areas where the algorithm may have failed. *BRAT* uses a combination of thresholding and marching squares algorithms to obtain segmentations, followed by skeletonization of the root system. The skeleton is then used to construct a graph data structure which can be used to calculate various RSA traits and the user interface allows manual assignment of genotypes to the root structure. The resolution of the images from the flatbed scanner is too low to compute root width, and some smaller local artifacts in the image are not captured in the analysis. Slovak *et al.* [50] report a false negative rate of 21% in the Quality Control Mode, and 25% in the automatic mode. This might be improved by a more rigorous set of preprocessing steps.

Colombi *et al.* [16] devised an imaging tent setup that can be used in a field to acquire 2D root images, and developed a *MATLAB* based software, called *REST*, so they could process the images. Each image has a label placed in it to identify the sample, which is later detected by the software during processing and used to rename the image file. In order to reduce errors caused by outlying roots, the image is cropped to remove 5% of the root pixels from the bottom of the image and 2.5% of the root pixels from the left and right sides. Otsu's method [39] is applied to get a segmentation of the remaining root system, which is used to directly compute RSA traits.

Regent Instruments Inc. have developed a commercially licensed product called *WinRHIZO*[7]. The software is designed to work alongside a flatbed scanner used to image plant roots that have been washed, but it is also possible to use TIFF or JPEG image files. Once the image has been acquired, *WinRHIZO* will process the image automatically, with a small number of adjustments by the user, to identify the visible root systems. Once the roots have been identified, a number of morphological and topological features can be extracted and stored as text files.

GT-Roots, developed by Borianne *et al.* [8], uses adaptive thresholding and morphological operations to automatically segment the root system in the image and ensure a single connected component. The segmentation is then used to define a "house shaped polygon", which defines a global structure of the root system. This structure can be used to compute global traits of the RSA. The process requires a calibration step before processing, using an empty image of a rhizobox to help with the determination of the house shaped polygon in the input images. The software outputs the numerical data and the intermediate images from the processing steps to allow for retrospective checks on the analysis.

Many of the approaches discussed will scale well to larger datasets of root images, making them ideal for high-throughput phenotyping of root systems. The automatic nature of these tools make them ideal for large scale analysis of RSA, though in many cases it is not clear how well each individual step in the RSA pipeline achieves its goal. Many of these tools are designed with a specific plant species or image input in mind, making them difficult to apply generally to any RSA study.

Automatic Segmentation and Classification

Automatic segmentation and classification approaches do not attempt to compute any RSA traits, rather they focus on accurately extracting root structure from images and return the structure in a data format which allows the user to calculate RSA traits later. By focusing on the extraction process, many of these tools can be used in various types of RSA study, as they attempt to more generally represent the RSA for customizable trait computation, rather than calculating a specific set of traits.

Kumar *et al.* [30] have developed a *MATLAB* script that can automatically identify root tips and distinguish between primary and lateral roots. They take a statistical learning approach using Zernike features to identify properties of the roots that can be used to classify them. The system operates on 2D images and performs well on plants that have been grown in transparent gel as well as plants that have been harvested and imaged. This software can only detect and count primary and lateral roots in a root image.

In *RootGraph*, Cai *et al.* [11] also distinguish between primary and lateral roots, building on the usefulness of the outputs from Kumar *et al.* [30]. Root images are segmented using the surface fitting approach presented by Cai *et al.* [10], skeletonized, and converted into a weighted graph representation, as this process proves to be robust to noise in root images. They achieve improved results over the approach by Kumar *et al* [30] and *WinRHIZO*, having comparable or better correlation with manual measurements.

[7]https://regent.qc.ca/assets/winrhizo_about.html

Douarre *et al.* [17] use a machine learning approach to segment images of roots on soil from X-Ray tomography images. They generate a set of synthetic images modelled after a set of ground truth images such that the distributions of pixels and classes match in each set. They first use a CNN to extract localized image features on $n \times n$ sized patches of the image, then pass those features into a SVM to classify each pixel as root or soil. They trained the model on the generated synthetic images, and tested on real images of Maize roots in two different types of soil. For evaluation, they defined a quality measure which does not seem to represent any standard image segmentation measure. The quality measure is based on a ratio of true positive classifications, calculated by multiplying sensitivity with specificity, for which 1 is a segmentation with no errors. They used this quality measure to evaluate the outputs of the model. Due to the unconventional metric they use for evaluation, it is difficult to compare this approach to other known segmentation methods.

In some cases, a segmentation of root images will contain disjoint components, where a single fully connected component is desired. Chen *et al.* [13] have proposed an approach to fix these incomplete segmentations via machine learning propelled inpainting. Their model makes use of a generative adversarial network (GAN), which is used to generate data based on a previously learned set of features. They apply a U-Net style architecture and treat the problem as a classification of root versus background pixels. The model makes use of a global discriminator as well as a local discriminator where the local discriminator is used to inform the generator on the quality of patches of inpainted root segments, while the global discriminator computes a similarity score between the generated images and the ground truth. This model is able to achieve an improvement over previous attempts at inpainting and may be a useful tool for obtaining a good segmentation in noisy images.

RootNav 2.0, implemented by Yasrab *et al.* [58], is an improvement on the *RootNav* software proposed by Pound *et al.* [42]. In this iteration of the software, machine learning is applied to classify root tips and produce a segmentation of the root structure in one automatic tool. Their model is based on an encoder-decoder network that classifies and segments the root structure in the image. The network benefits from training on both segmentation and detection by making use of similar image features learned in the earlier parts of the network. This is a bit different from transfer learning as the network weights are updated relative to both tasks, rather than just one. This means that any learned features useful to the segmentation of roots could be propagated to the tip detection task, and *Vice Versa*. This approach also reduces memory and computation costs, as only one network is required to perform both tasks, however the simple encoder-decoder architecture is not specially suited to the task of thin structures, and could thus be improved. After segmentation and detection, A* search [21] is applied to find the paths each root take to their sources. The model was trained on a wheat root dataset, and the authors have used transfer learning to apply the model to arabidopsis. The software is able to extract the architecture from the segmentation and store it in the RSML format for export. This output can be used in other software to compute root structure traits.

SegRoot, proposed by Wang *et al.* [54], makes use of the SegNet architecture developed by Badrinarayanan *et al.* [5] to perform pixel-wise segmentation. They train the model using a modified Dice similarity coefficient as the loss function to improve segmentation results. The network is used to create a prediction matrix of every pixel between 0 and 1, and threshold the results at 0.99 to get a binary image. The model is trained on dilated binary images of roots so that the model can pick up on smaller features of the root system. After thresholding, the dilation is undone by eroding the resulting segmentation mask with an equivalent filter. One issue with dilation is that it could unintentionally close small holes in the foreground object, particularly those caused by crossing roots. The use of SegNet skip connections rather than U-Net skip connections allows the model to be more memory efficient. The model was trained and tested on soybean roots, but could be trained to work on other species.

As with the automatic trait computation methods, the automatic segmentation approaches detailed here provide a scalable way to process large root image datasets. While these approaches do not perform the entire RSA phenotyping pipeline on their own, they do achieve high-quality segmentation results which will further improve subsequent steps of the overall pipeline. It seems clear from the approaches described in this section that machine learning approaches to segmentation are to be a major component of future solutions to the problem of segmenting root systems, due to their ability to learn generalizable feature filters for images and their ability to perform automatically.

2.2.4 Abstract Representations of Root Systems

After the root system has been extracted from the image, it must be stored in some way that facilitates further computation. Having a consistent format for representing the RSA is important in facilitating future research studies as well as allowing retroactive analysis of root system datasets. In this section three main methods for storing RSA are briefly described.

The first and simplest method for storing RSA is to use the segmentation output. Generally, a segmentation of a root image will result in some form of mask which details which pixels are part of the root system, and occasionally other information such as root tip locations and primary or lateral roots can be indicated. The mask can be stored as an image which can later be read in to a trait computation program. While it is simple to store the segmentation mask as an image, the mask will likely need to be processed into an intermediate format to facilitate computation. Depending on the desired traits, directly using the saved image masks may not be ideal.

The second method is to convert the segmentation into a graph data structure. Root pixels in the segmentation can be stored as vertices in the graph with additional information such as whether a root tip exists at its location, as well as pre-computed traits such as diameter. The major benefits to this approach are the direct applicability of graphs to trait computation algorithms. To compute a desired trait, one may simply define the trait as a series of operations on a graph. The graph can be customised to represent any additional information that is required, and can be updated at various steps in a larger pipeline to keep track of various operations. One downside to this approach is the abstract nature of graphs, whereby we lose the intuitive representation of the image and root system making it difficult to easily see and understand the RSA. Great care must be taken in encoding the graph to avoid loss or misrepresentation of structural information which may be caused by inconsistent or conflicting graph representations.

The final method described here is the RSML format. RSML is a standardized representation for RSA, presented by Lobet *et al.* [35]. The current specification of the language can be found publicly on Github[8]. This format follows an extensible markup language (XML) standard that allows easy and portable representation of root systems. It allows two types of data: metadata and scene data. Metadata contains basic information about the images that were used in the analysis, such as if they are part of a time series, and any specific image properties or transforms that may have been applied. Scene data contains the topological information about the root system. RSML presents a hierarchical structure that has a parent *plant* object, containing children. If a *plant* has one primary root with many lateral roots, the language can represent this structure with a parent object for the primary root and a series of child objects representing the lateral roots. Each root object can contain properties (pre-calculated values or general observations), geometry (represented as polylines or optionally more arbitrary geometric representations), and functions (values which depend on a position in the domain of the root's geometry). This allows a simple structure that can be extended to contain any desirable properties. While this format loses the intuitive representation of the segmentation image format, it does retain the hierarchical structure of the RSA making it slightly more intuitive that the graph representation. The structure of this format is not as straightforward or as well understood as graphs, making the computation process a bit more involved than using a graph, however the extensible nature of RSML combined with its ability to store functional information make it quite easy to do complicated computations.

2.2.5 Discussion

The software discussed in the above sections outline a number of advantages and deficiencies across the three approaches to RSA analysis. Manual approaches present a pipeline that allow the user full control over the analysis process, which allows the user to provide manual quality assurance on each measurement, and to selectively re-analyze data when they suspect an error. Semi-automatic approaches grant the same abilities to the user, with the added benefit of computer guidance on more tedious parts of the analysis. This guidance can be implemented as a tool that automatically does part of the analysis with the press of a button, or by literally guiding the user along in their annotations, making adjustments as the user adds information to achieve a more accurate result. Depending on the nature of the guidance, this can add reliability to the output of the analysis. Semi-automated approaches also provide a level of consistency between different trials and studies that purely manual approaches cannot offer. With a guided annotation software there is a defined standard for what constitutes an acceptable annotation, thereby reducing some (though not all) of the variance caused by different annotation conditions. Automatic approaches provide mostly user-free analysis of images, which increases the throughput of analysis of a given dataset. This reduces subjective

[8]http://rootsystemml.github.io/

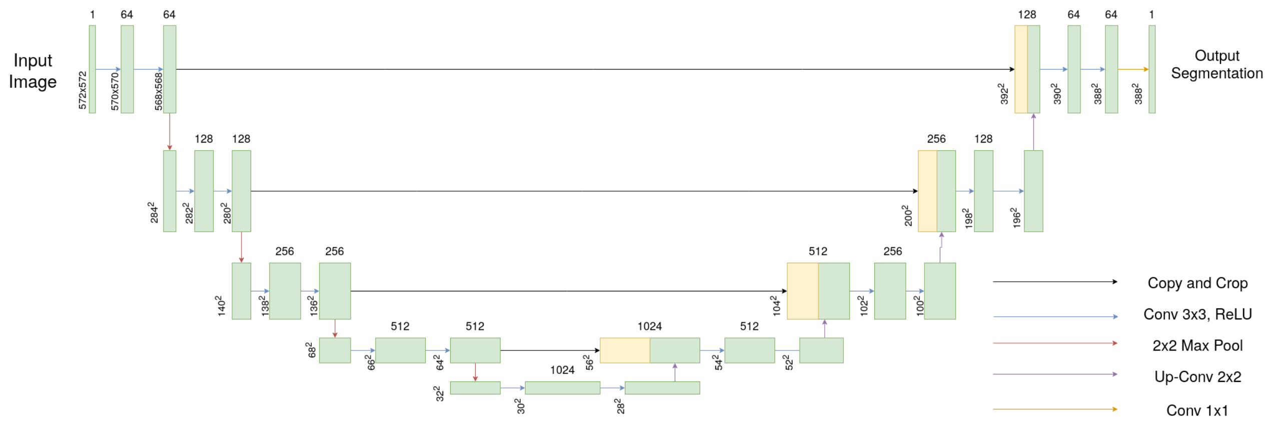
errors caused by different human annotators in a manual or semi-automatic approach, and can introduce more objective results as the pipeline processes each image using the same process. Automatic pipelines remove tedious analysis done by humans, and reduce the amount of labour required by a human doing a study. That said, automatic approaches can be less generalizable and introduce errors of their own. Many of the automatic approaches discussed make assumptions about their input data that allow them to be accurate, and if one or more of these assumptions are broken one can expect a reduction in accuracy and reliability. It is for this reason that one may prefer to use a semi-automatic approach over an automatic approach. This highlights the importance of further research into more reliable automated methods.

One major contributor to the reduced reliability of automatic methods is the choice of algorithm for segmentation of the root system. Many of the tools described here use some form of thresholding to separate the root (foreground) from the background of the image. This can work well if the background of the image is uniform and contrasting with the desired foreground pixels, in which case a global threshold using something like Otsu's method [39] will perform quite well. In many cases, however, the imaging setup can not capture a perfectly uniform background and foreground, and there is often a variation in the brightness of the image as well as noise. This can cause a global threshold to underperform. Much of this problem can be solved with some form of adaptive or local thresholding algorithm, which usually works by splitting the image in to smaller sections and performing individual thresholding on each. Local thresholding can result in a much better segmentation, and can overcome the challenges posed by the varying image properties. These approaches can be optimized to work with specific image setups and species of plants at the cost of generality of the algorithm. Another issue with threshold based methods is that small details can be lost if they are too close to the threshold limits. It is possible that segmentation networks, such as the approach by Wang *et al.* [54], can be applied to obtain a better segmentation of the root, without losing as many of the smaller image details. It may also be possible to adapt certain segmentation networks to account for the thin branching structure of the root to make them more efficient. These approaches do, however, require more annotated data to develop than thresholding methods, which can be an obstacle in some cases.

2.3 Deep Segmentation Networks

With the recent success of CNNs, many new network architectures have been designed to benefit the task of segmentation. These networks can have many layers to help localization of image features, giving us so-called deep segmentation networks. Deep segmentation networks have been largely applied to the field of medical imaging, where it can be important to distinguish thin structures in patient scans such as blood vessels, arteries, and catheters. Furthermore, they have been applied to segmenting road structures from satellite and drone imagery, as well as segmentation of images of roots in soil to aid in RSA studies. These approaches are continuously updated and refined to provide improved results and are a promising approach to the problem of root system segmentation.

Figure 2.1: U-Net architecture presented by Ronneberger *et al.*. The input image passes through a series of convolutional layers on the contracting side (left half of the network) followed by max pooling layers for downsampling. Convolutional layers on the expanding side (right half of the network) are upsampled and concatenated with the outputs of the corresponding layer on the contracting side.



The U-Net architecture proposed by Ronneberger *et al.* [48] is a commonly used approach to image segmentation. It was designed to work with small amounts of training images for research areas where it is difficult to acquire data. Figure 2.1 depicts the architecture of U-Net. The architecture is made of two sides: a contracting side and an expanding side. On the contracting side of the network, the image is sent through a series of layers that perform 3×3 convolution, followed by rectified linear unit (ReLU) activation, further followed by a 2×2 max pooling unit to downsample the image. After each max pooling unit, the layer is repeated with the downsampled data, and the number of feature channels is doubled. At the end of the contracting side, the data is passed to the expanding side, where it undergoes a similar process. The expanding side consists of a series of layers that perform 2×2 upsampling, 3×3 convolutions, and a ReLU activation, repeated until the output layer, which is a 1×1 convolution unit that produces the final segmentation. After each upsampling layer the number of feature channels is halved. To propagate localized features through the network, the output of each layer on the contraction side before the max pooling layer is cropped and passed to the corresponding layer on the expanding side, where it is concatenated to the feature channels. This allows the feature map found in each layer of the contraction side to be combined with the features propagated by the upsampling layers in the expanding side, providing better local features throughout the network for various scales.

Cherukuri *et al.* [14] have explored the use of multi-scale filter representation in deep segmentation networks. They have designed a network architecture made up of two parts: a representation network and a task network. The representation network is trained to develop filters for geometric features at multiple scales to account for variation in vessel thickness, while the task network uses these filters to learn to perform segmentation. The two networks are optimized together, such that the task network is indirectly influenced by the optimization of the weights of the representation network. For the segmentation part of the network, the U-Net architecture [48] is used, modified such that the input layer matches the output layer of the representation network. They have also further modified U-Net to use regression loss instead of cross entropy loss for multiple output channels. Due to the diverse orientation of retinal vessel structure, the representation filters must be sensitive to changes in orientation. Thus, they have modified the loss function of the network to include an orientation diversity term. The orientation diversity term is formulated such that it will have a strong positive response to a pattern with similar orientation, and a weak response to patterns of orthogonal orientation. This allows the network to identify the thin structures of the vessels at various scales (defined at the time the network is trained) and to then perform segmentation with better accuracy.

Chitta *et al.* [15] have designed a CNN architecture that produces a quad-tree representation of a segmentation, which compared to other deep segmentation approaches reduces the amount of memory and compute power required when training the network on segmentation tasks. The quad-trees in their architecture are stored using a hash-table, keyed on a tuple (l, x, y) containing the quad-tree level l, and the coordinates x and y. The value stored at the key is the class that has been assigned to that quadrant, which can be any of the possible classes in the ground truth image, or a composite class. The composite class is used to label a quadrant whose pixels are not all of the same class. The ground truth segmentation are represented in T-pyramid format, which is a general-form quad-tree where each inner node has exactly four children, and the leaf nodes are all on the same level. This way, we have a standardized structure with which to compare the predictions of the network with, and we can quantify the difference of the predicted quad-tree and the ground truth T-pyramid. Each level in the prediction quad-tree has its own loss function, which is a normalized cross-entropy loss between the given quad-tree level and the same level in the T-pyramid. The overall loss of the network is then calculated to be a weighted sum of the losses for each level of the quad-tree. In their study, Chitta *et al.* evaluate two different methods of weighted loss, a fixed rate using a hyper-parameter, and an adaptive weighting method; the two methods give similar segmentation accuracy and mean intersection over union (IoU). The proposed architecture also allows flexible propagation schemes, giving flexibility in the way activations are passed through the layers of the network. They evaluate three such schemes: All, ground truth composite (GTC), and predicted composite (PC). The All propagation scheme simply passes all activations through the network, which uses the maximum possible memory available to the network. The GTC approach makes use of the ground truth quad-tree representation to determine which pixels are part of the composite class and only propagates those pixels to the next layer, which reduces the amount of memory required to store the activations being used. This approach is only useful when training, as the ground truth quad-tree is not available when performing inference on new data. The PC approach is similar to GTC, except it uses the network to predict which parts of the image are part of the composite class, and only propagates those. This is not useful during initial training of the network as the predictions will be unreliable, however it can be used during inference on a pre-trained network to reduce the computational resources required. The proposed architecture gives comparable results to dilation based segmentation networks with decreased usage of memory and compute resources.

Guo *et al.* [20] have developed a modification of the U-Net architecture that is specialized for guidewire tip segmentation from X-ray images. Guidewires are used by physicians to treat stenosis in patients and can be difficult to see during operation. The modification suggested by Guo *et al.* improve real-time viewing of guidewire tips during the treatment process by segmenting the guidewire in an X-ray fluoroscopy image. They have introduced a new layer block to the U-Net architecture that makes use of summation operations rather than concatenation operations to propagate features between layers of the network. This allows the network to operate with reduced memory costs and allow reuse of features between layers. In the last layer of the network, a connectivity cube [27] is applied to account for discontinuities in the segmentation. The loss function is split into two components: the segmentation loss, which is computed as the Dice overlap score, and the connectivity score that is calculated as the binary cross-entropy loss for the prediction and the ground truth. These two components are combined together to produce the overall network loss function. Compared to the original U-Net architecture, reduced dense U-Net, and connectivity U-Net, the approach by Guo *et al.* shows an improved robustness for segmentation of guidewires as measured by sensitivity, F1, Jaccard index, and Hausdorff distance scores.

U-Net has been used in conjunction with a random forest tree bagger classifier by Kassim *et al.* [28] to optimize segmentation of blood vessels from epifluorescent images. Their pipeline first applies contrast limited adaptive histogram equalization (CLAHE) to obtain consistent and increased contrast properties for each image. The enhanced images are then used to train a U-Net model, modified with additional dropout layers, to output a regression likelihood map based on the green channel of the input image. The regression likelihood map output from the trained U-Net is then added to a set of handcrafted features that are used to train a random forest tree bagger classifier for segmentation. The output of the random forest algorithm is a segmentation that requires some post-processing to remove blobs from segments. This process combines the spatial preservation properties of the U-Net architecture with the ability to specialize feature extraction via handcrafted features to allow improved segmentation of thin vessels. Kassim *et al.* have compared their pipeline to a standard U-Net and an optimized U-Net approach using the sensitivity, precision, specificity, accuracy and Dice similarity metrics. They have shown a small increase in the sensitivity and accuracy, while U-Net outperforms in precision and specificity. The largest improvement is in the Dice similarity metric, which may justify the small decrease in the precision and specificity.

Smith *et al.* [51] have applied U-Net to segment chicory roots from soil in 2D images. They make use of a chicory image data set that was used in a previous study to manually measure root intensity and length. 50 images were annotated by an experienced agronomist to obtain ground truth for training and evaluation of the segmentation process, 2 of which were removed due to other objects being present in the image. 10 images were set aside for testing, and the remaining 38 images were split into training and validation sets. The validation images were selected by ordering the images by root intensity and choosing 9 images with even spacing to ensure a wide range of root intensities were included in both training and validation. Images were mirrored along the edges to provide better context for segmentation around the edges of the image. Input images were randomly cut into 90 patches at the beginning of each epoch, and were filtered down to 40 patches by removing patches that did not contain root. These input patches were then normalized and subjected to augmentation via colour jitter and elastic grid deformation. A Dice loss combined with cross-entropy loss was used, multiplied by 0.3 as found by trial and error to improve results of training and validation phases. The U-Net approach was compared with a Frangi filter approach which was used as a baseline. Segmented images from both approaches were skeletonized and pixels were counted to estimate root lengths, though this method of length estimation may distort results based on the orientation of the root in the image. An F1 score was computed for the U-Net and Frangi methods to compare, as well as an accuracy measure. In both approaches the accuracy measure was greater than 0.99, which is mostly due to the high classification accuracy of background, due to an imbalance in the examples of background and foreground pixels. More interestingly, the U-Net approach resulted in much higher F1 scores (0.701 for U-Net vs 0.462 for Frangi), specifically having higher performance in recall. Additionally, a high positive correlation (Spearman rank correlation of 0.9748 ($P < 10^{-8}$) and r-squared 0.9217) was found with root intensity found from manual study and the estimated skeleton length from U-Net segmentation.

Choice of loss function when training deep segmentation networks can have a large impact on the results of the segmentation, particularly when there is a high class imbalance (between foreground and background pixels) as is the case when dealing with root images. Sudre *et al.* [52] have evaluated four commonly used loss functions designed for deep segmentation networks and datasets with high class imbalance: weighted cross-entropy loss (WCE), Dice loss, sensitivity-specificity loss (SSL), and generalized Dice loss (GDL). All of these loss functions are presented in their two-class form and evaluated via usage in four different network architectures. Two 2D networks, U-Net [48] and TwoPathCNN [22], are trained with each of the four loss functions to perform segmentation of tumors in medical images, while two 3D networks, DeepMedic [26] and HighResNet [33], are trained to segment age-related white matter hyperintensities. These different segmentation tasks represent a highly imbalanced class problem, as tumors and white matter hyperintensities can vary in size and location and images contain a large proportion of background pixels. The networks were trained multiple times, changing the learning rate through 10^{-3}, 10^{-4}, and 10^{-5}, and changing the sizes of input patches from small, medium, and large to outline results coming from choices of these parameters. Data augmentation was not applied during training. Results from the tests show that GDL preforms well on small patches across the different learning rates when used with U-Net, while WCE performed better on larger input patches. Overall GDL and Dice loss remain robust across the different architectures and learning rates.

Another consideration when training a network architecture for segmentation is the depth of the network. Zhou *et al.* [60] explore this further by studying the effects of U-Net network depth on segmentation problems, and propose a new architecture that combines many U-Nets into an ensemble network. They conclude from their study that the appropriate depth of a network is highly depended on the problem and the availability of data for training. With this in mind, they propose U-Net++, a modification of the U-Net architecture that allows high-level features from encoder layers of all of the networks in the ensemble to influence the feature maps in the decoder networks of each network in the ensemble. The benefits of this type of architecture are also prevalent during back-propagation, where high-level features from decoder layers are propagated to the encoder layers of shallower networks in the system. Zhou *et al.* also introduce deep supervision into the network, establishing an overall loss function for their system by way of a weighted sum of the individual losses of the networks in the ensemble. To reduce computational costs at inference time, it is possible to prune the network by dropping networks with a depth greater or equal to a specified depth. This reduces the number of networks being used to compute the segmentation, but also reduces performance of the system. Due to the connections of decoder layers to shallower encoder layers mentioned above, pruning the network will not lose all of the features learned by the deeper networks in the ensemble, allowing a reasonable trade-off between performance and inference time. The results are compared with a classical U-Net and VNet (on 3D datasets), along with wide versions of these networks that have the same number of parameters as the U-Net++ architecture, allowing comparison without bias on the number of parameters in the networks. It is shown that U-Net++ performs consistently better than the original U-Net and VNet architectures on six biomedical image segmentation tasks, as determined by Dice similarity and IoU scores. With this in mind, it is not clear exactly how much more information is actually being utilized in this new architecture and whether it is worth the large increase in resources to train.

He *et al.* [23] describe a method for allowing a neural network to encode residual information by inserting skip connections among fully connected layers. By representing the residual the network not only learns the filters to apply to inputs to achieve the desired encoding, but also the residual of the inputs (ie. what the filter removes from the inputs) which can be passed on to future layers. With this new representation of information and the shorter path through the network taken by the residual, the network achieves a higher convergence rate reducing training error, even for deeper network architectures. Zhang *et al.* [59] apply this concept to convolutional layers within a U-Net structure for road extraction, taking advantage of the ability to make the network deeper without losing training value. They make use of the residual block to replace the standard layers in the U-Net architecture. The residual block contains a batch normalization layer, followed by a ReLU and a 3×3 convolution, followed by another set of the same three layers. The output of these layers is added to the original input to the block to form the final output, forming the residual function. The output of each residual block is sent to the next layer in the U-Net, as well as being concatenated to the corresponding expansion layer as is done in standard U-Net practice. Zhang *et al.* evaluate their model with relaxed precision and recall, which are defined as the precision and recall of pixels within a given range of pixels from their target classification. They compare their models using the breakeven point, which is the point on the relaxed precision-recall curve where the precision and recall values are equal to each other. The breakeven point of classical U-Net is reported as 0.9053 for the task of road extraction, while the breakeven point of the residual U-Net is 0.9187, suggesting that the addition of residual units can help to better extract thin road structures.

Many of the approaches discussed here make use of the U-Net architecture as a starting point. While U-Net is quite good for segmentation tasks, some modification is required to improve performance on tasks involving thin structures. These modifications take many different forms, such as advanced skip connections for feature propagation, different types of layers which help the network to pick up on small features, combinations of different network setups, and modification of loss functions. While these methods have not been designed for the problem of 2D root segmentation, it is feasible that many of the ideas presented here may be helpful in designing a method for segmentation of roots.

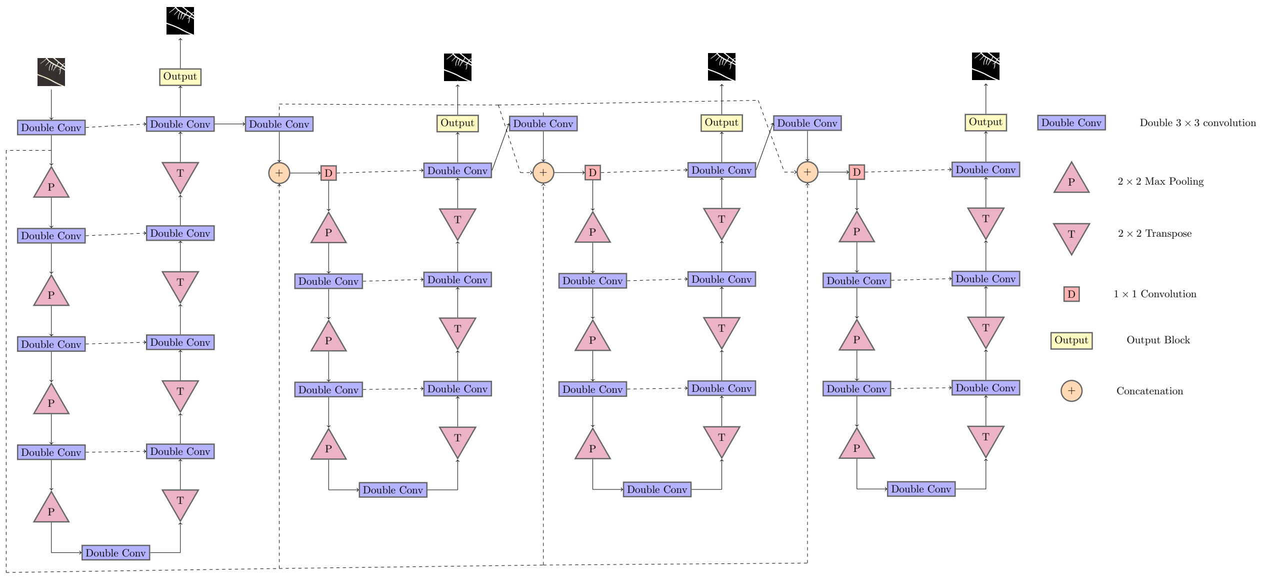
2.3.1 Iterative Neural Network

Neshatpour *et al.* [38] propose the concept of iterative learning as a means for reducing the computational power required for large networks by decomposing deep CNN architectures into a series of smaller networks. The AlexNet [29] architecture is taken as a case study in employing the iterative methodology. By introducing iterative blocks in the overall network, more control is gained over the algorithm at inference time. As each smaller network has its own output, the algorithm can be stopped at any time and the output of the current iteration can be used. This is quite useful in cases of limit computational power, where a decrease in accuracy can be taken to reduce the amount of time spent processing the image. Neshatpour *et al.* suggest that input images be sub-sampled for each iteration, allowing new image features to be presented to each subsequent network, along with the learned features of the previous iteration.

Li *et al.* [32] have proposed IterNet, an architecture that makes use of iterating over a smaller architecture to refine the output of a segmentation for retinal vessel segmentation. The network consists of a standard U-Net, followed by a number of smaller "mini U-Nets" that use the output of the network before them to produce a more accurate segmentation. Each of the mini U-Nets have their own output and loss function which are used to improve its own output. We treat each iteration as one pass through a network in a series of networks, such that each network is independent of the others. This has the opposite effect of an ensemble network, where rather than using the outputs of each network together to improve a result, each network passes its learned representation on to the next to further learn from the representation as iterations continue. At each iteration the output segmentation of that network should be an improvement over the outputs of the previous network. While each iteration does output its own binary segmentation, we do not pass the segmentation as the input to the next iteration, rather the output of the last decoder layer before the output layer. This has the effect of passing the learned encoding of the image on to the next image which will have more information contained within than the final segmentation. This allows the next iteration to produce a segmentation based on the learned features of the previous iteration, rather than just the output segmentation.

Li *et al.* employ three types of skip connections in their architecture to allow high level information to propagate through the network. The first are the standard U-Net skip connections that connect each encoder layer with its corresponding decoder layer, concatenating the output of a layer to the input of the decoder on the opposite side of the network. Skip connections are also made between the first U-Net to the input of

Figure 2.2: IterNet network architecture as presented by Li *et al.* [32]. The first U-Net structure is larger than the subsequent networks, while high level skip connections propagate features to each network in sequence.



each of the mini U-Nets, which are concatenated to the output of the previous iteration, followed by a 1×1 convolution for dimensionality reduction. These type of connections allow for high level features to remain present throughout each iteration in the network. Finally, dense connections between the mini U-Nets are used to allow propagation of features across each iteration. Figure 2.2 depicts the architecture used by Li *et al.* for retina segmentation.

In IterNet [32], each mini U-Net has its own sigmoid cross-entropy loss function. Input images are first split into patches of 128×128 pixels which are chosen randomly during training. At inference time, images are split into overlapping patches and the output segmentations of each patch are averaged together to produce the final output. Li *et al.* use three commonly used retinal vessel data sets, each having a small number (less than 40 or so) images, during their evaluation. To improve results, augmentation of colour, shape, brightness, and position are applied during training. The number of mini U-Nets used is 3. To evaluate their model on each of the three datasets, they split each dataset into training and testing sets, however they do not create a validation set meaning that we have no basis for how the model performs on data it has never seen before. Results are compared with the standard U-Net [48], DenseBlock-U-Net [19], and DeformUNet [25] architectures which were implemented and trained on the data by Li *et al.*. Comparison is also made with other known studies on the same dataset, namely Residual U-Net [2], Recurrent U-Net [2], R2U-Net [2], and Iter-Seg [49]. F1 score, sensitivity, specificity, accuracy, and area under the ROC curve (AUC) were used to compare the results of each method. A marginal increase in accuracy and AUC are shown by IterNet [32], as well as improvement in sensitivity which subsequently increases F1 score. Due to the omission of a validation set, it is difficult to say if these marginal improvements are generalizable to other data, or if this architecture provides a reasonable training time vs performance trade off for this task.

RSA has many properties which are similar to the structure of retinal vessels, both structures being thin structures which branch out in different directions. The results achieved by IterNet [32] suggest that its architecture is suited to detecting thin branching structures. The ability of the network to refine the segmentation at each iteration allows better identification of smaller details in the image as required by the task of segmenting root systems. Adopting the iterative approach to segmentation networks also provides the benefit of fine tuning network performance in low resource computing systems. Basing the overall structure of the model on the U-Net architecture also allows training to be successful with small amounts of data, which is common in RSA due to the amount of work required to produce an annotated dataset of root images.

3 Methods

In order to develop and evaluate a root segmentation algorithm we must first obtain a set of root images along with ground truth annotation data describing the ideal segmentation that should be produced by the algorithm. As this work will make use of a deep neural network architecture for segmentation, it is important that enough data is acquired to train the proposed model while still maintaining a reasonably sized testing and validation set. Ideally the training set of images will contain many images of various root species, however in order to test the generalizability of the model we must be careful to maintain a specific set of images of a species which is not used in the training set. Model training is also dependent on a number of hyper-parameters which can dramatically change the effectiveness of a model. With this in mind, we define a series of simple trials for obtaining a high quality segmentation model before evaluation on a final set of validation images. These trials also aim to identify possible weaknesses and biases that are inherent to the model architecture and training set.

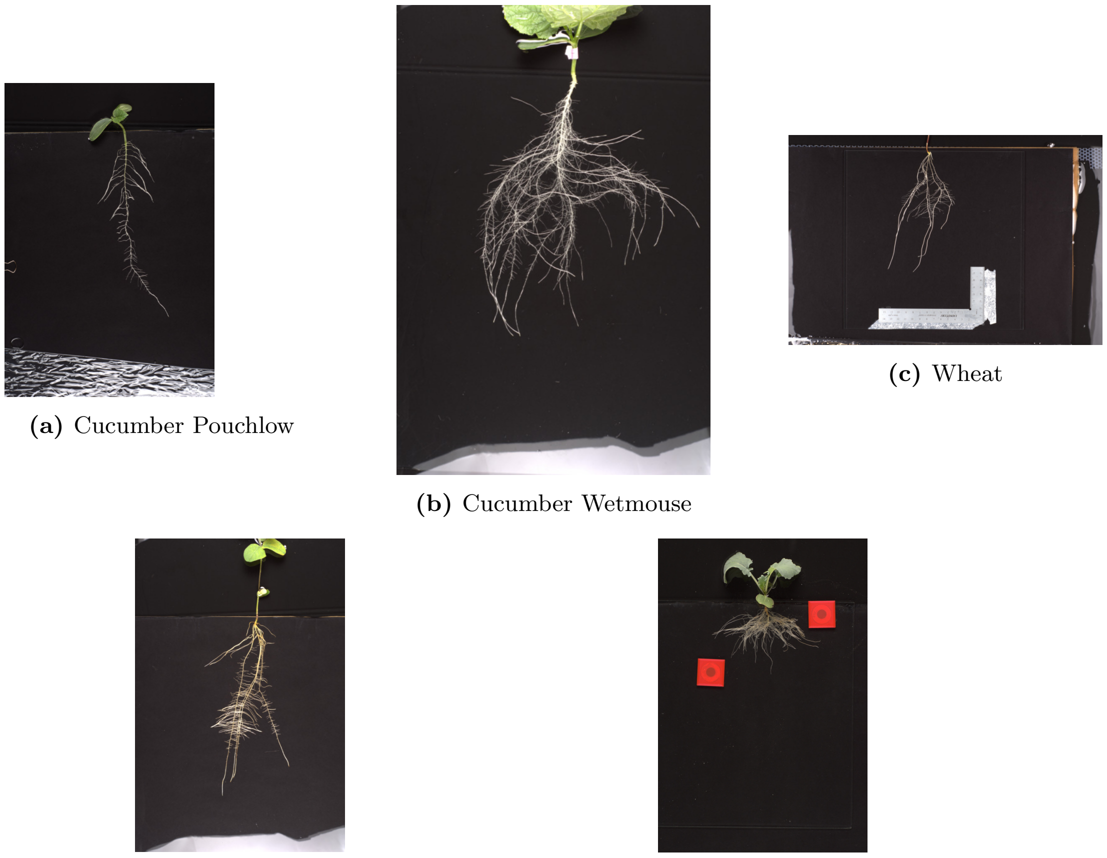
Section 3.1 below outlines the images which we will acquire from previous imaging efforts in the P2IRC program, along with how ground truth annotations will be obtained and used to create a dataset suitable for training and evaluating the proposed network architecture. Section 3.2 describes the proposed network architecture in detail, Section 3.3 details the experimental framework which will be used to evaluate and determine the ideal parameters of the model, and Section 3.4 defines common evaluation metrics used in image segmentation and how they will be used to determine the success of the model.

3.1 Datasets

Here we outline the 2D image dataset that has been compiled for training and evaluating the proposed segmentation network. Our dataset is comprised of many high resolution images of root systems of different plant species taken at different growth stages. To obtain ground truth segmentations for training and evaluation we introduce the *Friendly Ground Truth* tool which is designed to facilitate consistent annotation of root images by annotators. Finally, we describe how the data will be organized to evaluate the model.

3.1.1 Acquired Images

Our dataset is comprised of six sets of plant root images: Cucumber-Pouchlow, Cucumber-Wetmouse, Canola, Wheat, Soybean, and Soybean-assoc. The Cucumber-Wetmouse images were obtained by growing the cucumber plants in an aerated nutrient solution and, prior to imaging, the plants were placed in a shallow glass tray filled with water. The five remaining image sets consist of root systems which were grown in a Plexiglas folder on top of a sheet of black filter paper. Four sheets of germination paper are placed below the chamber which absorb a nutrient solution from a plastic tub during growth. The root system on the black filter paper is covered with a pliable plastic sheet which prevents the RSA from moving as well as preventing the root system from drying out. This system allows the roots to grow freely within a 2-dimensional space, allowing a 2D image to capture the entire RSA. Examples of each of the root species are shown in Figure 3.1.



(a) Cucumber Pouchlow

(b) Cucumber Wetmouse

(c) Wheat

(d) Soybean

(e) Canola

Figure 3.1: Example images from the Cucumber-Pouchlow (a), Cucumber-Wetmouse (b), Wheat (c), Soybean (d), and Canola (e) data sets

Global Institute For Food Security (GIFS) has provided the Cucumber sets, the Wheat set, and the Soybean sets of images. All five of these image sets were imaged using a Nikon D7200 camera. The Cucumber-Pouchlow dataset consists of 40 4016 $\times$ 6016 images of Cucumber (*Cucumis sativus*) roots taken at 7, 8 and 11 days after transplanting (DAT), the Cucumber-Wetmouse dataset consists of 39 4016 $\times$ 6016 images of Cucumber (*Cucumis sativus*) roots imaged at 7, 9, and 11 DAT, the Wheat dataset consists of 301 6016$\times$4016 images of Wheat (*Triticum aestivum*) roots imaged at 7, 9, and 11 DAT, the Soybean dataset is made up of 70 6016$\times$4016 images of Soybean (*Glycine max*) roots imaged at 8, 10, and 12 DAT, and the Soybean-assoc dataset consists of 333 6016 $\times$ 4016 images of Soybean (*Glycine max*) roots imaged at 5 and 8 DAT. The Canola image set was provided by Agriculture and Agri-Food Canada (AAFC) and consist of 495 2180$\times$280 images of Canola (*Brassica napus*) roots, imaged using a Nikon D7200 camera in two different sessions, 4 days apart. All images are RGB colour TIFF images. Table 3.1 gives a summary of the entire dataset.

Table 3.1: Data Set Summary

Data Set	**Species**	**Image Size**	**Repetitions**	**Number of Images**	**Total Images**
Cucumber-Pouchlow	*Cucumis sativus*	4016×6016	7 DAT	13	40
			9 DAT	14	
			11 DAT	13	
Cucumber-Wetmouse	*Cucumis sativus*	4016×6016	7 DAT	14	39
			9 DAT	12	
			11 DAT	13	
Wheat	*Triticum aestivum*	6016×4016	7 DAT	101	301
			9 DAT	100	
			11 DAT	100	
Soybean	*Glycine max*	6016×4016	8 DAT	24	70
			10 DAT	23	
			12 DAT	23	
Soybean-assoc	*Glycine max*	6016×4016	5 DAT	181	333
			8 DAT	152	
Canola	*Brassica napus*	2180×2980	Session 1	249	495
			Session 2	243	
			Session 3	3	
					1278

3.1.2 Ground Truth Annotation

Before a segmentation approach can be developed and evaluated, we must first have ground truth segmentations which identify, as closely as possible, the desired segmentation of the RSA for each image which is to be used in the training and evaluation process. Due to the large sizes of the images and the tediousness of creating such annotations by hand, we have developed *Friendly Ground Truth*, an annotation tool designed to improve consistency of root annotations among annotators and reduce the amount of tedious effort that is required to achieve a high quality annotation.

Friendly Ground Truth allows users to annotate a root image in a series of patches, rather than operating on the whole image at once. This approach allows easy navigation of large images and provides a focus on a particular area of the RSA. Figure 3.2 depicts an example of the tools in use. At any given patch for annotation, the immediately surrounding patches of the image are shown to the user for contextual information during annotation, but cannot be modified without explicitly navigating away from the current patch. Navigation between patches can be done using the arrow keys on the keyboard, or via an image overview window which displays the image with the current annotation layered on top. At the time that the user loads the image, the tool computes a threshold for each patch using Otsu's method [39] and creates a segmentation as a default. This provides the user with a starting point for each patch, as well as helping to identify patches which have no roots in them, as the threshold will be low enough on a completely dark patch to include most (if not all) of the patch pixels as foreground. At any time the user has access to a

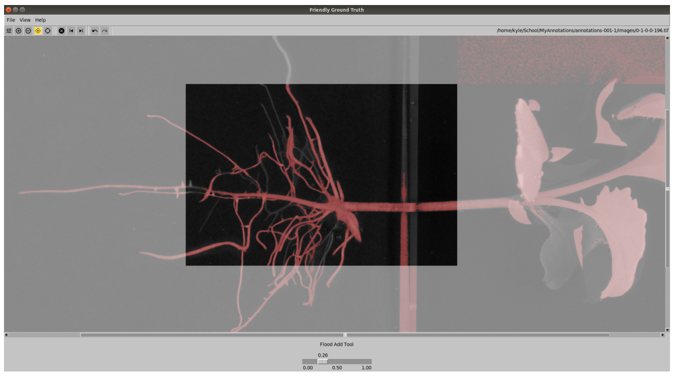


Figure 3.2: A screenshot of an in-progress annotation on a patch in the *Friendly Ground Truth* tool.

thresholding tool, which simply increases or decreases the threshold value to adjust the foreground region in real-time. Often the Otsu threshold is close to the desired output but needs small adjustment with the thresholding tool to include finer details in the RSA. Adjusting the threshold does not affect any patch in the overall image other than the currently focused patch. In some cases, applying a threshold to the entire patch does not allow fine enough control to accurately segment the roots in the patch. In these cases the user has access to a flood fill tool (commonly known as a magic wand) and a brush tool. The flood fill tool can be used to select a region of the current patch based on a similarity threshold, which can be adjusted to segment targeted sections of the RSA. In places where the flood fill tool is not accurate enough, the user can use the brush tool to manually paint or erase any part of the foreground segmentation for fine control over details. Finally, the user has access to a "No Root Tool", which marks the current patch as having no root pixels visible and marks all pixels as background before automatically moving to the next patch. This can greatly reduce annotation time in images with a large background area. The tool was developed in Python and is publicly available on GitHub[1] for easy distribution.

Starting an Annotation When the user first opens an image, the tool will split the image into a 10×10 grid of patches and apply an Otsu [39] threshold to each patch to achieve an initial baseline segmentation. The main annotation panel within the tool will then display the first patch, with the estimated foreground in coloured in red. The immediately surrounding patches of the current patch are also displayed, but they are greyed out and cannot be interacted with, they are provided only for contextual information (see Figure 3.2 for an example). In addition to the main panel, an image preview window is displayed to the user, showing the full image, the division of the image into patches, and the current state of the segmentation coloured red (see Figure 3.3). The individual patches in the preview can be selected to navigate directly to that patch for annotation.

[1]https://github.com/p2irc/friendly_ground_truth

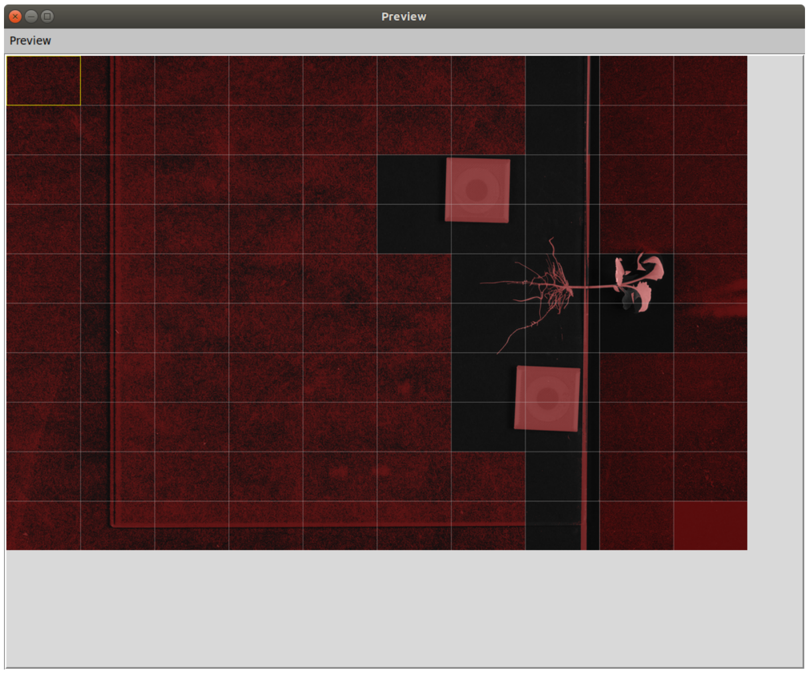
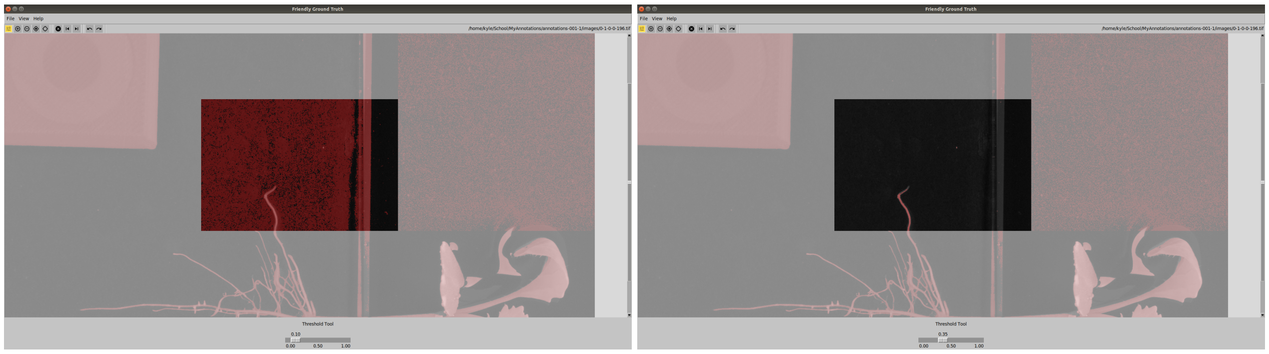


Figure 3.3: The Preview Window displayed by *Friendly Ground Truth*. We can see that it has placed converted the image into a grid of patches and applied a simple threshold to each patch to give a starting estimate for the segmentation.

Manual Thresholding The threshold tool allows the user to adjust the pre-computed Otsu threshold for the currently selected patch. The value can be increased or decreased to accommodate the currently visible area of the root system. Thresholding in this way is restricted only to the current patch, and so will not affect the annotations of other patches in the image. Figure 3.4 shows how a patch's threshold can be increased to obtain a better segmentation.

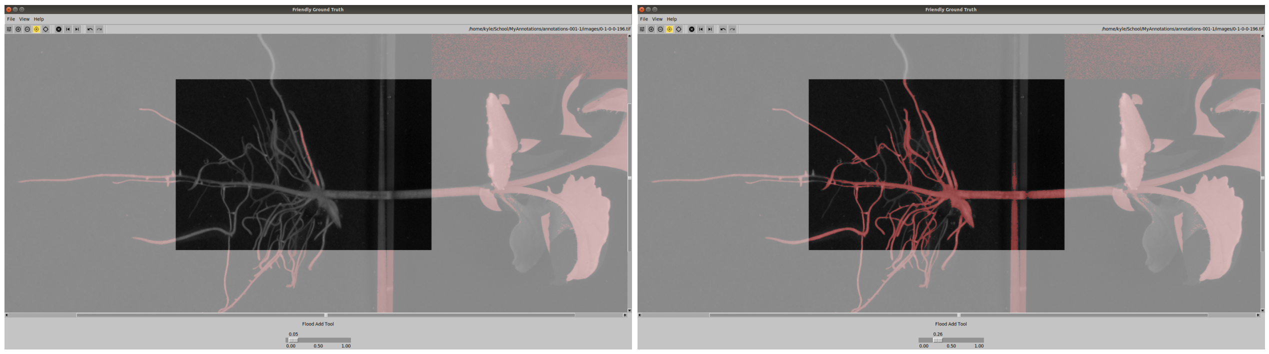


(a) A patch with a poor initial threshold. **(b)** The same patch after adjusting the threshold value.

Figure 3.4: A simple threshold operation on a patch within *Friendly Ground Truth*.

Brush Tools Brush tools allow the user to interactively add or remove regions of the current patch from the annotation. The user can use the scroll wheel or the input box at the bottom of the screen to adjust the brush size. This allows intuitive addition or removal of areas in the annotation that cannot be addressed with the other tools.

Flood Fill Tools The flood fill tools allow a user to select a pixel in the current patch as a starting point. The user can then adjust a similarity threshold value to grow or shrink the foreground region in the annotation. This can be very helpful for annotating many branching roots in the image, as much of the root can be automatically selected with a well chosen similarity threshold. The ease of use of this tool can reduce fatigue on the user, and is particularly useful for annotating thin areas between two roots. Figure 3.5 displays an example of the tool before and after use.



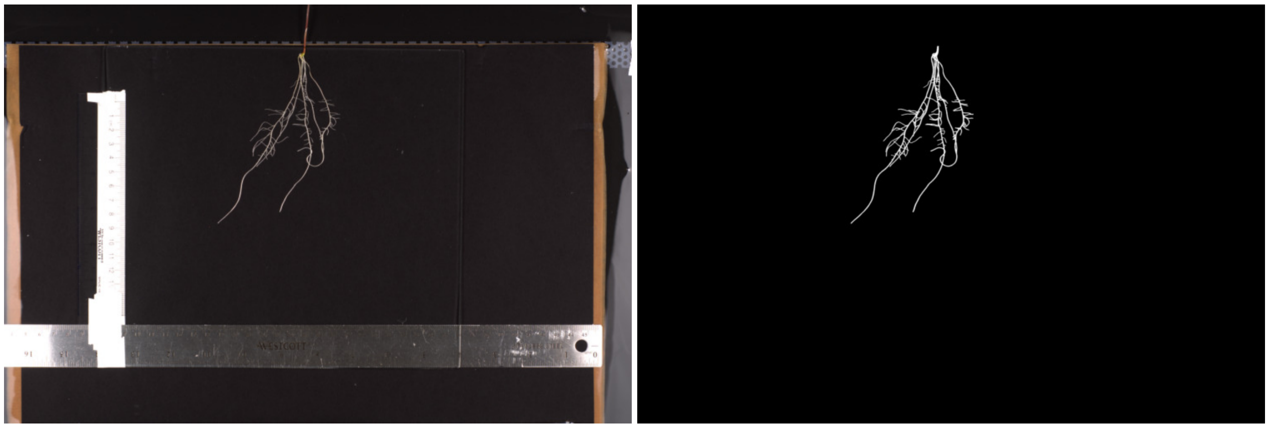
(a) A patch before using the flood fill tool. **(b)** The same patch after using flood fill.

Figure 3.5: An example of the flood fill tool within *Friendly Ground Truth*.

Other Useful Tools Many of the patches within the image will not contain root system, and thus do not need to be processed in detail. To make this process simpler, the user is provided with a "No Root" tool, which simply sets the threshold for the current patch such that no pixels are identified as foreground. Using this tool will also automatically navigate the user to the next patch in the image, allowing quick processing of such patches. The user is able to use the arrow keys, or the preview window, to navigate between different patches, as well as basic pan and zoom tools for positioning the current patch for easier annotation. Undo and Redo tools are also available to easily edit annotation actions in the event of a mistake.

Using the tools detailed above, *Friendly Ground Truth* simplifies the root annotation process while allowing the user to focus on finer details in the root system. The initial automatic thresholding of the patches gives the user a place to start, which can often make it much easier to quickly identify which areas of the current patch need to be edited to be correct. The splitting of the image into smaller patches also makes the process of annotating the large images much less overwhelming, as the user can focus on a small area of the image at one time. The flood fill tools in particular are well suited to the types of annotation motions that are common to RSA annotations.

The *Friendly Ground Truth* tool was used to create ground truth annotations from the images described in Section 3.1.1. Annotations were completed by volunteer computer science students who had no previous experience with root annotations. Annotators were provided with a manual describing the desired segmentation properties as well as outlining possible cases of uncertainty to reduce variance in annotation quality (See the Appendix for the provided manual). Each volunteer annotator was first trained on a pre-determined image (the same image for each annotator) and assessed to ensure understanding of the annotation process. Once the annotator was familiar with the annotation process they were given a set of images to produce ground truth segmentations for. The resulting segmentations from each annotator were processed to keep only the largest single connected component of foreground pixels to ensure that small single pixel mistakes were not included in the final ground truth. An example of an input image and its associated ground truth segmentation are depicted in Figure 3.6. In total 142 images were annotated using the *Friendly Ground Truth* tool.



(a) Original Image **(b)** Image Mask

Figure 3.6: Example image (a) with its human annotated ground truth mask (b)

3.1.3 Experimental Organization

To facilitate experimental analysis of the model's performance the 142 annotated images were transformed into a number of datasets which facilitate experimental trials. All images of Cucumber roots were removed as a hold-out validation set to test the generalizability of the model to a species it had never seen before. The remaining images were split into training, testing, and validation datasets using a $70\% - 15\% - 15\%$ split. The resulting split images give a set of annotated root images with 90 training images, 16 testing images, 18 validation images, and 18 hold-out cucumber validation images. This image dataset is referred to as the *Full Image Set*, as it contains the full sized images of entire root systems.

Derivative Datasets

In addition to the *Full Image Set*, five additional datasets are created which make use of the images from the *Full Image Set* to allow further analysis of model performance.

Hyper-Parameter Patch Training Set Images from the training portion of the *Full Image Set* were cropped around the root system to reduce the amount of background. Next, 256×256-pixel size patches were extracted randomly from the cropped images to create a set of patched images suitable for training the network. $3,282$ patches were created as a result of this process, taken from 90 full sized images. This dataset is used as the training set in Trial 1 (described below in Section 3.3.1) to facilitate search for training hyper-parameters that produce a high-quality model.

Expanded Patch Training Set The second trial we conduct is designed to build upon the results of the hyper-parameter search conducted in Trial 1. This is done by including additional patches containing only non-root objects that appear in the images. $1,069$ 256×256-pixel sized patches identified as containing no roots were added to the to the *Hyper-Parameter Patch Training set* to create the *Expanded Patch Training Set*. The patches were hand chosen by a single individual to include image features that the network had difficulty identifying in the first trial, such as plant stem, water droplets, or dust particles. Trial 2 (described below in Section 3.3.2) makes use of this dataset for training the model.

Testing Patch Set For testing the model during training we created the *Testing Patch Set* from the images in the *Full Image Set*. This dataset was created by extracting 256×256-pixel sized non-overlapping patches from the images in the testing portion of the *Full Image Set*, resulting in $1,077$ patches taken from 16 full sized images. The *Testing Patch Set* is used for testing the models trained in Trial 1 and Trial 2 (described in Sections 3.3.1 and 3.3.2, respectively).

Validation Patch Set The *Validation Patch Set* is used for final validation of the model once it has been finalized through the trials we have designed. Similar to the *Testing Patch Set*, this dataset was created from the validation portion of the *Full Image Set* by extracting non-overlapping patches of 256×256 pixels. The full sized images were not cropped, ensuring that non-root artifacts appear in the images to allow examination of the performance of the model when such artifacts are present. This dataset consists of $1,864$ patches taken from 18 full sized images.

Cropped Image Set Finally, due to the large amount of background in the images, we create the *Cropped Image Set* which consists of the same images as the *Full Image Set* manually cropped to remove a majority of the non-root area in the image. The images were manually cropped to ensure that root tips do not appear on the edge of the image. The manual nature of this cropping makes it difficult to specify an amount of pixels used for padding, though a post-cropping analysis shows that a mean number of 153 pixels was allowed for padding on each side. The stem of the plant shoot was cropped from the images so that only roots were visible. This dataset is used in Trial 4 (described in Section 3.3.4), which is designed to evaluate the model when a reduced number of non-root objects are present, as well as to reduce the amount of processing required for a single image by removing known background areas. Identically to the *Full Image Set*, this dataset is divided into testing, validation, and hold-out cucumber validation sets.

Table 3.2 gives a summary of all dataset used in this work.

Table 3.2: Summary of the datasets used for evaluating the network in different trials. The last column indicates which trial each dataset is used in.

Dataset Name	**Description**	**Num Images**	**Trial**
Hyper-Parameter Patch Training Set	Size 256 × 256 patches created from the full image set.	3,282	1
Expanded Patch Training Set	Size 256 × 256 patches created from the full image set, with an additional 1,069 "empty" patches.	4,351	2
Testing Patch Set	Size 256 × 256 patches created from the full image set.	1,077	1 & 2
Validation Patch Set	Size 256 × 256 patches created from the full image set.	1,864	Validation
Full Image Set	Full size images including a testing, validation, and hold-out cucumber subset.	16 testing, 18 validation, and 18 cucumber hold-out.	3
Cropped Image Set	Images from the Full Image Set croped to various sizes tightly to the root system.	16 testing, 18 validation, and 18 cucumber hold-out.	4

3.2 Proposed Model

Our proposed neural network architecture is an improvement on the iterative network proposed by Li *et al.* [32]. The architecture makes use of a number of refinement U-Net networks to iteratively improve upon the segmentation of an image. As Li *et al.* have applied this architecture to segmentation of retinal arteries, which are thin and highly branched structures, this architecture seems to lend itself to the problem of segmentation of root systems, though we make a number of modifications to the model. The first of these modifications is to replace the standard U-Net hidden layers with residual units as introduced by He *et al.* [23]. These residual units preserve feature information as the network becomes deeper, and while each iteration of the network is in reality a new network with its own set of learned features, as we progress through iterations of the network the feature inputs will begin to degrade as they would in more standardized deep network architectures. This modification helps to reduce the effects of the degradation of features. Figure 3.7 depicts a high level view of the proposed architecture. The first iteration of the network is larger than the subsequent

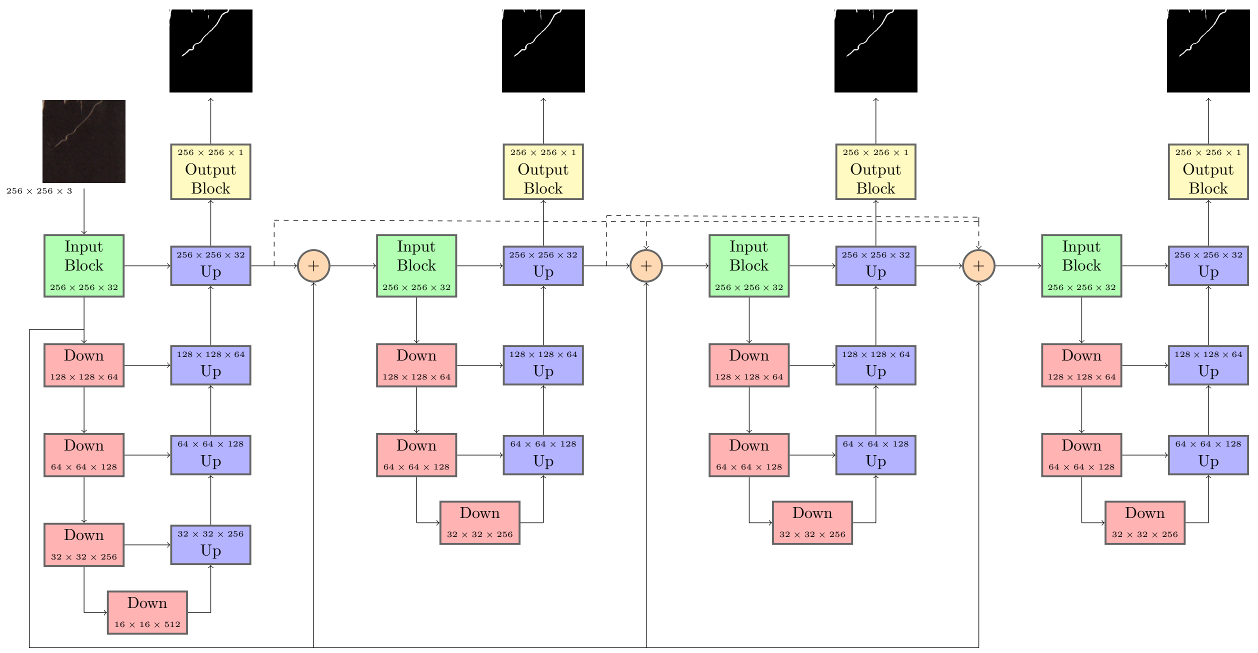
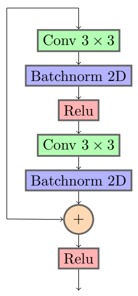


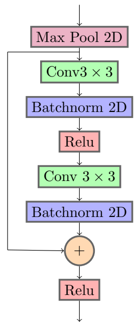
Figure 3.7: Network architecture diagram showing a high level view of the blocks making up the network. One main U-Net structure acts as input to the subsequent smaller U-Net structure. High-level image features are concatenated to the inputs of all networks via skip connections (solid line), and learned high-level features at the output of each iteration are concatenated together with the inputs of each subsequent iteration (dotted lines). Each network iteration has its own segmentation output, and the last network's output is considered to be the final segmentation.

networks with four hidden layers, and is considered to be the input network. All subsequent networks consist of three hidden layers. Each sub-network has its own output segmentation and set of learned weights, and are updated relative to their own outputs during training. The output of the first iteration's input block is passed via skip connection to the input of each subsequent network to preserve the high level features in the input image. The output of each iteration is also passed to the input of each subsequent network, being concatenated together with each-other as well as with the output of the first iteration's input block. Another modification we make in contrast to the model proposed by Li *et al.* [32] is to place the concatenation operation before the input to each network iteration, rather than after the input layer of each subsequent network. This allows the input block of each iteration to process the full input tensor using the residual units. The residual units themselves make use of another type of skip connection within each hidden layer to further preserve high level features at each layer. The architecture is designed to operate on 256×256-pixel sized patches, rather than a full-sized input image. Figure 3.8 details the composition of the blocks which make up the larger network structure.

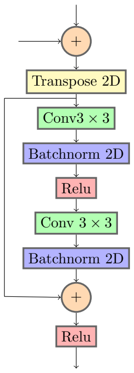
Aside from the input to the first iteration of the network, the inputs to each U-Net structure are not images, rather they are learned feature representations from the previous network. This provides two major benefits in the context of segmentation. First, secondary networks make use of the features learned by previous networks to improve upon the segmentation outputs at each iteration. As training progresses,



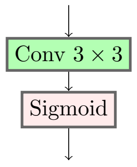
(a) The input block based on the residual unit. A 3×3 convolution is applied, followed by batch normalization, and a ReLU function. The same process is repeated again, but the original input features undergo addition before the last ReLU operation.



(b) The downward residual block is similar to the input block (a) except we apply a 2D max pooling operation to downsample the input features.



(c) The upward residual block is similar to the input block (a) except a transpose is applied to upsample the input features. Features from the corresponding down layer are concatenated before being input to the transpose.



(d) The output block is a simple 3×3 convolution followed by a sigmoid function to produce a final probability map.

Figure 3.8: The blocks that make up the network structure.

each network should become better at producing an accurate segmentation, which is then used as input to subsequent networks to improve their own results. Second, due to each network being trained with respect to its own output, the process of updating weights after each epoch means that each network after the first input network receives a slightly different set of inputs at each epoch, rather than receiving the same training images repeatedly. This acts as a form of data augmentation for each secondary network. As each network's weights are updated, they produce new features as inputs to the following network, which will always be a different representation of some image which has been exposed to the overall system in a previous epoch. This allows training with small datasets, as well as improving the ability of the network to generalize via exposure to varying input features.

In their model, Li *et al.* [32] use sigmoid cross entropy loss during training of the network. In our case, a sigmoid cross entropy loss function may pose problems due to the large number of background pixels compared to the number of root pixels in the input image, particularly in allowing the model to overemphasize the value of background pixel classification. For this reason we opt to use a combination loss function, which makes use of a weighted binary cross entropy loss combined with a weighted Dice loss function defined in Equation 3.1. α and β represent the weight of the binary cross entropy and Dice loss functions, respectively. The binary cross entropy function is represented by $H(y, \hat{y})$ for some predicted matrix y and ground truth segmentation mask $\hat{y}$. $D(y, \hat{y})$ represents the Dice loss function for the same prediction and ground truth segmentation. The binary cross entropy portion of the loss function allows the model to optimize for a more accurate per-pixel classification while the Dice loss portion keeps the segmentation as a whole as accurate as possible. The hyper-parameters α and β allow greater control over the network during training. To train each iteration in the network, this loss function is applied individually to each sub-network and its input and output at each step in the training process. The loss value for the final iteration in the network is considered to be the loss value of the entire network as the output of this iteration will be considered the final segmentation. We make use of early stopping techniques to prevent the model from over-fitting, where the loss value is monitored on the testing set at each epoch. If the loss value degrades by an amount within a specified threshold, the current epoch will be marked as an epoch where degradation has occurred. After a specified number of epochs with degradation of the loss value training will be stopped. Finally, multiplicative learning rate decay is used to help the model converge as training continues for many epochs.

$$\begin{gathered} loss = \alpha H(y, \hat{y}) + \beta D(y, \hat{y}) \\ H(y, \hat{y}) = -\sum_{i}^{N} (y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i)) \\ D(y, \hat{y}) = \frac{2 \sum_i^N y_i \hat{y}_i}{\sum_i^N y_i + \sum_i^N \hat{y}_i} \end{gathered} \tag{3.1}$$

In order to provide the model with localized feature inputs, the network operates on 256 × 256-pixel sized patches. This size provides a view with enough local information to distinguish between roots and background objects, while still providing enough contextual information to ensure that features are identified with respect to the overall RSA. Data augmentation is employed on-the-fly to each patch that is input to the network to further reduce over-fitting of the model to the training set. Images were subjected to random rotation of up to 90 degrees, as well as horizontal or vertical flipping. For evaluation root images are split into 256 × 256-pixel overlapping patches with an overlap of 50 percent. This method of splitting an image into patches ensures that each pixel in the image is examined in four patches. Once segmentation of the individual patches is complete, the patches are put back together using a majority voting system for the final value of each pixel in the full image. A pixel is determined to be foreground if it has been labelled as foreground in the majority of patches in which it appears. In the case of a tie, the pixel is labelled as background. This process helps to reduce errors that may occur due to the way a patch is extracted from the original image.

The model will be governed by a series of hyper-parameters for fine-tuning the training process. The *learning rate* is a standard hyper-parameter which determines the magnitude of steps taken during back-propagation in the network. *Learning rate decay* is a factor used to control multiplicative learning rate decay during training, which is used to decrease the learning rate over the course of many epochs. These hyper-parameters together help to control the convergence rate of the loss function toward its optimal values, allowing rapid changes to network weights at the beginning of training and slower changes toward the end. *Binary cross entropy weight* and *Dice loss weight* are also considered to be hyper-parameters which control the influence of each piece of the loss function as described by α and β in Equation 3.1. The final three hyper-parameters control the early stopping mechanism of the training process. *Stopping Patience* is the minimum number of epochs to wait before considering stopping the training job. *Stopping epochs* is the number of epochs to use to evaluate the stopping criteria. A higher value for *stopping epochs* will allow the model to continue training for longer even if results begin to degrade, though this is not necessarily a bad thing as degradation can occur before improvements in results, highlighting the importance of having fine control over this mechanism. The final hyper-parameter is *stopping tolerance*, a threshold which determines the smallest amount of change in the results in order to stop training. A large *stopping tolerance* will stop the model even if the score improves by a large margin from the last epoch, while a smaller *stopping tolerance* will allow the model to continue training until a small amount of change is detected in the results.

3.3 Experimental Trials

To examine the properties of the proposed model and determine the ideal conditions for achieving high-quality segmentation of RSA we design a set of four trials, followed by a final validation trial. These trials correspond to the different image sets which were outlined in Section 3.1.3. The first trial will make use of hyper-parameter tuning to determine how different model parameters affect the results of segmentations. The second trial compares the best results from the first trial with a model trained with the same input parameters, but with an enhanced set of training patches which include more examples of non-root objects. Both of these trials concern only patch-level results of the segmentations. The third trial will evaluate the models from the first two trials on the full images, and the fourth trial seeks to evaluate the models from the first two trials on images which have been cropped to remove excess background from the image. Validation will perform on the specified validation sets, and the results will be compared to two other recent models which perform segmentation of thin branching structures. All models are trained on 256×256-pixel sized patches, which is a standard practise in training deep segmentation networks and helps with localization of features. In cases when segmentation of a full-sized input image is required, the image will first be split into patches with a 50% overlap. The resulting patches will be segmented and then re-assembled by overlapping the segmented patches into the final binary mask image. Due to the overlapping of patches during the patching process, each pixel in the mask will be represented in four segmented patches, meaning that each pixel in the final segmentation will have four separate classifications as either root or non-root. To consolidate these into a single classification for each pixel, majority voting is applied such that the most-agreed upon probable class for the pixel is accepted as the final class. In the event of a tie, the pixel is classified as background.

3.3.1 Trial 1: Hyper-Parameter Tuning

The first trial we will conduct seeks to evaluate how hyper-parameter configuration affects the model, as well as to determine the optimal set of input parameters for segmentation of root images. The Google

Cloud Platform[2] provides a framework to facilitate simple hyper-parameter tuning via concurrent training of models with varying hyper-parameter inputs. The hyper-parameter tuning tool allows specification of ranges for hyper-parameter inputs to the model. Once the ranges have been specified and the tuning job started, Bayesian optimization is used to search the space of input values for optimal settings. Depending on configured resource constraints many models can be trained simultaneously, with their testing scores being recorded for use in determining improved parameter values. The training score of the current set of models is monitored and used to inform the algorithm on how to update the hyper-parameter values for the next set of models to be trained. The tool also has an early stopping mechanism, whereby training of a single model can be cancelled if the loss values appear to be less successful than other training attempts. This helps to conserve computational resources.

For this trial we will trigger 30 training jobs within the hyper-parameter tuning framework. Each of these 30 models will be trained on the *Hyper-Parameter Patch Training* dataset. Table 3.3 describes the allowed input values for the framework to train models within. The results of the hyper-parameter tuning will be used to analyze the effects of individual hyper-parameters on the training results of the model. The model with the highest final Dice score on the *Testing Patch Set* will be considered the best model and the input hyper-parameter values corresponding to that model will be determined the most optimal for this task.

Table 3.3: Input values for hyper-parameter tuning.

Parameter	**Type**	**Min Value**	**Max Value**
Learning Rate	Double	0.001	0.01
Learning Rate Decay	Double	0.85	0.99
Cross Entropy Weight	Double	0.2	1.0
Stopping Patience	Integer	15	20
Stopping Tolerance	Double	0.0005	0.01
Stopping Epochs	Integer	0	10

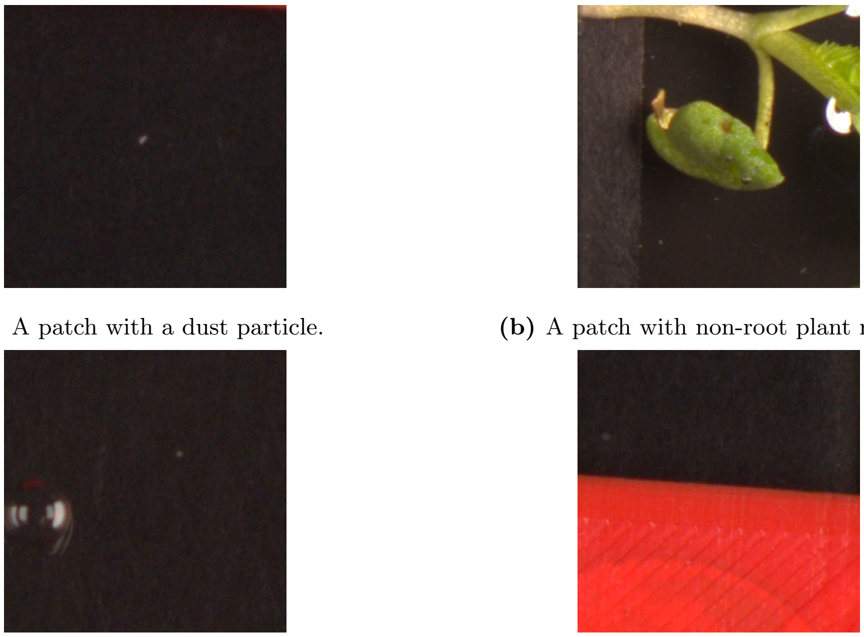
To allow efficient search of the hyper-parameter space we choose a set of input ranges which constrain the search to a controlled region. The learning rate parameter is allowed to fluctuate between 0.001 and 0.01, which are common values used for model training [54, 52, 23]. This range should avoid learning rates which make large steps which may cause poor convergence of the loss function, as well as values which are so low as to increase training time. The learning rate decay is constrained between 0.85 and 0.99 for similar reasons; too much decay will cause poor convergence, while too little will reduce the benefits that come from multiplicative learning rate decay. This specific range of values was chosen due to performance in preliminary tests of the model training process. The cross entropy weight parameter is constrained between 0.2 and 1.0. Due to the fact that testing evaluation is done using Dice score, it is possible that allowing the cross entropy

[2]cloud.google.com

weight to assume a small or zero value will optimize the parameters to make use of only the Dice score portion of the loss function, which may sacrifice the ability of the model to learn from the per-pixel classification results which come from the cross entropy portion of the loss. For this reason the small but non-zero value 0.2 is chosen as the low constraint for this parameter. The value of the cross entropy weight should not be greater than 1.0, as greater values will over-emphasize the weight of the cross entropy loss portion of the function. The input values for stopping patience are constrained between 15 and 20 epochs. This range is chosen to ensure that enough epochs are used to allow the model to begin to learn from the training data without being in danger of over-fitting. The stopping tolerance is chosen to fall between 0.0005 and 0.01. As the Dice score metric is represented on a scale from 0.0 to 1.0, this range allows a maximum tolerance of 0.01 in change to dice score, which is a rather large change at late stages in the training process, while the minimum 0.0005 tolerance is a very small change which still may yield an improvement. Finally, the stopping epochs parameter is constrained between 0 and 10, to allow the early stopping process a reasonable chance of taking effect before over-fitting. The input ranges related to early stopping were chosen based on performance of the model in preliminary testing.

3.3.2 Trial 2: Effects of Enhanced Training Dataset

Once the optimal hyper-parameter values have been determined for the model we will conduct a trial to evaluate the effect of non-root patches in the training set. Due to the method used for creating training patches, an emphasis is given on extracting patches with roots in them. The presence of non-root objects, such as water droplets or dust, in the images presents a problem during segmentation, as many of these objects are not annotated in the ground truth and may not appear in the training set. To attempt to enhance the effectiveness of the model in identifying these objects as background we make use of the *Expanded Patch Training Dataset*, which contains 1,069 additional patches which were hand chosen to contain various non-root background objects. Figure 3.9 displays some examples of patches which were chosen for this dataset.



(a) A patch with a dust particle.

(b) A patch with non-root plant matter.

(c) A patch with a water droplet.

(d) A patch with a large non-root object.

Figure 3.9: Examples of patches which were selected for the updated training set.

The model for this trial will be re-trained from scratch on the *Expanded Patch Training Set* using the best performing hyper-parameters as determined in trial 1. The resulting model will be directly compared to the best performing model that was trained in Trial 1 to evaluate how the extra training patches effect the segmentation quality of the model on the image patches from the *Testing Patch Set*.

3.3.3 Trial 3: Patch-Wise Segmentation of Full Images

Once we have determined how the model best performs on the patch datasets we will evaluate model performance on the full-sized images. The images from the *Full Image Dataset* will be used to evaluate the best model from Trial 1 and the model from Trial 2. To evaluate the model on full images, the images are split into 256×256-pixel patches with an overlap of 128 pixels. The individual patches are then segmented by the model and then overlapped together to re-create the original full image. A majority voting system is used for each pixel as they will appear in multiple patches. Should a tie occur, the pixel in question is labelled as background, as it is expected that the model will be more effective at identifying background pixels due to the large class imbalance given in the image by disproportionate amounts of background compared to roots.

The results from this trial, while they should yield similar to the results on the patch datasets, will give insight to how the model is able to segment the entire root system in an image. The model from Trial 1 will be directly compared to Trial 2 to determine how the extra $1,069$ empty patches affect the model's ability to achieve a higher quality segmentation on the full sized images, and whether a significant difference is made.

3.3.4 Trial 4: Patch-Wise Segmentation of Cropped Images

Due to the large amount of background in the images, a significant amount of processing is spent on patches which are generally already known to not contain any roots. Additionally, many of the patches which are extracted from areas of background will contain non-root objects which may be misclassified as root. While it would be ideal for the model to be able to distinguish between these objects and roots (which is explored in Section 3.3.2), another solution is to simply remove these objects from the image before segmentation. If the general region of the image containing roots is known with an estimated bounding box, one could crop the image to remove a large amount of non-root objects, giving the model a more focused image for segmentation. This could help to reduce many errors in segmentation where the model cannot distinguish between non-root objects and true root objects.

For this trial, we will use the *Cropped Image Dataset* to evaluate how the model performs on pre-cropped images. We will directly compare the performance of the top model from Trial 1 and the model from Trial 2 on the cropped images. This should allow us insight on how the model can perform given the presence of non-root objects in the image, and possibly provide a suggested solution for dealing with any errors which these objects may cause.

3.3.5 Validation

Each of the datasets used for the previously discussed trials contains a set of validation images, and in the case of the full and cropped image datasets a hold-out cucumber validation set is provisioned. The validation sets will be used as a final evaluation of the model to ensure that results are not biased by images which have previously been processed by the model. Additionally, the hold-out cucumber sets will be used to evaluate the ability of the model to segment images of roots from a species which it has no prior exposure to. The best performing model as determined by Trials 1-4 will be evaluated on the *Validation Patch Set*, the *Full Image Validation* and *Cucumber* sets, and the *Cropped Image Validation* and *Cucumber* sets. This will give a finalized set of results for the model, which can be directly compared to other models for the task of root segmentation.

3.3.6 Comparison with Recent Models

The results from our model will be compared to two recent models, *SegRoot* designed by Wang *et al.* [54] and *IterNet* as proposed by Li *et al.* [32], on each of the datasets. To allow direct comparison, we will re-train each of these models from scratch with the recommended input hyper-parameters by their respective authors. For *IterNet* the number of iterations is set to 3, *dropout* was set to 0.1, *batch size* was set to 32, *epochs* was set to 200, and *learning rate* was set to 0.001. For *SegRoot* the *weight decay* is set to $5e-4$, the *batch size* is set to 64, *epochs* was set to 100, and *learning rate* was set to 0.01. Models will be compared by mean Dice similarity coefficient, mean IoU, sensitivity, and specificity scores, which are further described in Section 3.4.

These two approaches have been chosen for comparison due to their use of deep learning for segmentation of thin structures. *SegRoot* [54] makes use of a CNN architecture, *SegNet* [5], which is designed for semantic segmentation of images. *SegRoot* has previously done well to segment images of roots in soil, removing much of the noise caused by the soil in the images. This makes it an ideal comparator for our approach. *IterNet* [32] is the architecture that our approach is based on, and so it makes sense to include a direct comparison to identify how the changes we have made affect the segmentation quality. *IterNet* has been previously applied to segmentation of retinal vessels, which have a similar structure to root systems, suggesting that it should apply well to the segmentation of roots. The comparator approaches chosen here should provide insight to how non-root objects can affect segmentation quality, while providing baseline segmentation results which make use of similar deep learning techniques.

It may seem useful to compare these deep learning approaches with a classical image processing approach, such as Otsu's method[39]. Deep learning approaches require significant computational resources and time for training and tuning, while a simple Otsu threshold value could be computed quickly for each image and without the need for extensive training. While this would be an ideal approach for streamlining the RSA segmentation process, these types of threshold based segmentation approaches consider properties of the image histogram, rather than the properties of the objects within the image. By design, threshold based approaches use pixel intensity values to separate all pixels in the image into two classes, foreground and background. This does not allow us to distinguish between root and non-root objects. Even an adaptive thresholding approach, whereby patches of the input image are given a threshold individually, would identify noise in the image as part of the foreground class, whether or not they are part of the root system. This makes these types of approaches inherently bad for the problem of identifying RSA in the presence of non-root objects. As such, we do not further consider these types of approaches in this work.

3.4 Evaluation Metrics

To provide reliable evaluation and facilitate standardized comparison of results we will make use of four evaluation metrics which are common to image segmentation and deep learning. The model will be evaluated using Dice similarity coefficient, pixel-wise accuracy, sensitivity, and specificity scores. Each of these metrics can be defined in terms of True Negatives (TNs), True Positives (TPs), False Negatives (FNs), and False Positives (FPs), given a ground truth segmentation $\hat{y}$ and a predicted segmentation y. A True Positive occurs when a pixel y_{ij} and the same pixel in the ground truth $\hat{y}_{ij}$ are both marked as foreground. A True Negative is the opposite, when a pixel y_{ij} and the pixel $\hat{y}_{ij}$ are both marked as background. False Positives occur when a predicted pixel y_{ij} is marked as foreground, but the same pixel $\hat{y}_{ij}$ in the ground truth is marked as background. False Negatives occur when a predicted pixel y_{ij} is marked as background and the same pixel $\hat{y}_{ij}$ is marked as foreground. These definitions help us to further define the evaluation metrics to be used for the model. The evaluation metrics used in this work are used in the context of pixel-wise classification into root (the “positive” class) and non-root (the “negative” class) classes.

Dice similarity coefficient is a metric which characterizes the similarity of two regions on a scale from 0.0 to 1.0, with 1.0 being perfect similarity. Equation 3.2 defines the Dice similarity coefficient in terms of TP, FP, and FN. The Dice similarity coefficient puts focus on the foreground region, meaning that we get a normalized score regardless of the size of the segmented region. This allows us to see how well the root system is segmented from the background even when there is a large imbalance in favour of background pixels, rather than seeing how well the model correctly classifies pixels in the image.

$$DSC = \frac{2TP}{2TP + FP + FN} \tag{3.2}$$

IoU describes the area of intersection of the predicted foreground with the ground truth foreground, normalized by the union of both regions. Equation 3.3 defines IoU in terms of TP, FP, and FN. Similarly to Dice coefficient, an IoU score of 0.0 indicates that the segmented region is completely disjoint from the ground truth region. Higher values indicate a higher level of intersection between the predicted and ground truth regions, thus indicating a better segmentation. The IoU metric is positively correlated with Dice similarity coefficient, though it differs in the magnitude of penalty given for missed TP identification.

$$IoU = \frac{TP}{TP + FP + FN} \tag{3.3}$$

Finally, Sensitivity and Specificity score represent the True Positive Rate and True Negative Rate of the model, respectively. The Sensitivity score, defined in Equation 3.4, tells us how well the model can identify foreground pixels, or in our case, what percentage of the root system is correctly identified as root. Similarly, the Specificity score, defined in Equation 3.5, tells us how well the model can identify background pixels, or what percentage of the background is correctly identified as such. It is expected that for images with a large number of background pixels the specificity score will be quite high compared to the sensitivity score, as the model is exposed to more background pixels during training.

$$SENS = \frac{TP}{TP + FN} \tag{3.4}$$

$$SPEC = \frac{TN}{TN + FP} \tag{3.5}$$

These four metrics together describe important aspects of the model’s ability to segment RSA from images. Direct comparison between models can be done with these metrics, with higher scoring models in all four categories being considered as giving a better quality segmentation. With this in mind, it is important to consider what each metric describes and how that relates to the problem of root segmentation. The Dice similarity coefficient appears to give the most directly meaningful description of the ability of the model to produce a segmentation, being directly related to the foreground region of the image, regardless of size. IoU score gives a secondary insight into the segmentation quality of the model. Following this, the Sensitivity metric is useful in detailing the amount of the root system that is captured by the model, with Specificity detailing the same information about the classification of background.

Pixel-wise accuracy is another common metric used in evaluating deep learning models. Pixel-wise accuracy is simply the percent of correctly identified pixels in the prediction versus the ground truth. As defined in Equation 3.6, this metric gives an overall sense of how well the model can identify both foreground and background pixels. While this information can be useful in evaluating deep learning systems, it is important to note that in the context of images with a disproportionately large number of background pixels to foreground pixels (such as in root images) the accuracy can be quite high while yielding a low quality segmentation. As an example, if the ground truth segmentation consists 90% of background pixels, a model which randomly identifies 100% of pixels as background would have a pixel-wise accuracy of 90%, which is quite high considering none of the desired foreground was identified. This highlights that, while it can give useful insights, a high pixel-wise accuracy is not necessarily representative of a better model. For this reason, accuracy score is omitted during evaluation of the proposed model in favour of the aforementioned evaluation metrics.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.6}$$

4 Results and Discussion

Here we detail the performance results of the model across four trials as described in Section 3.3. The results presented are designed to evaluate the ability of the proposed model to provide accurate automatic segmentation, as well as to generalize to other species of plant roots. The first trial discusses the effects of hyper-parameter selection on the final segmentation produced by the model, detailing the best choices for parameter inputs. The second trial outlines the effects of augmenting the training set with additional non-root objects for improving identification of background. The third and fourth trials explore how the model performs on full sized images and cropped images respectively. These trials represent two possible approaches which would be used in a full RSA pipeline. Finally, the model is evaluated on a set of validation images as well as a set of cucumber root images to test for generalizability.

Models have been trained using the Google Cloud Platform in a virtual environment with four NVIDIA-Tesla-T4 GPUs. While some trials vary the input hyper-parameters, all training jobs were initialized with a maximum number of 150 epochs with a batch size of 32. Additionally, the Dice weight hyper-parameter in the loss function was set to a constant 1.0 to provide a constraint on the network to improve the Dice score of final output segmentations. The number of iterations used for the model was three.

4.1 Trial 1: Hyper-Parameter Tuning

We used the hyper-parameter tuning framework in the Google Cloud Platform with the specifications in Section 3.3.1. Due to resource and budgetary constraints, only 30 models were able to be trained with hyper-parameter tuning. Table 4.1 summarizes the results of the hyper-parameter search, representing the chosen hyper-parameter values, the elapsed training time, final training step which activated early stopping, and the resulting Dice score on the testing set. Entries are sorted in descending order of Dice score. The varying amount of training time for models is reflective of the early stopping mechanism, which outlines the importance of correctly choosing hyper-parameter values for optimal performance.

Table 4.1: Performance of the models found via hyper-parameter tuning, in decreasing order of Dice score.

Mean DSC	**Training Step**	**Train Time**	**Learning Rate**	**Learning Rate Decay**	**Crossentropy Weight**	**Stopping Patience**	**Stopping Tolerance**	**Stopping Epochs**
0.936	149	9 hr 28 min	0.00332	0.89661	0.42715	17	0.01000	7
0.934	49	2 hr 29 min	0.00373	0.90361	0.43432	15	0.00995	7
0.926	199	9 hr 39 min	0.00573	0.92774	0.24987	15	0.00932	5
0.925	21	1 hr 14 min	0.00373	0.89465	0.48001	20	0.00995	7
0.924	64	9 hr 33 min	0.00547	0.91167	0.77971	19	0.00533	6
0.922	69	9 hr 40 min	0.00364	0.89527	0.43544	16	0.00711	8
0.921	34	1 hr 51 min	0.00550	0.92000	0.60000	18	0.00525	5
0.919	22	1 hr 18 min	0.00340	0.88736	0.41711	16	0.00614	8
0.918	22	1 hr 15 min	0.00318	0.89646	0.37651	17	0.01000	8
0.918	22	1 hr 15 min	0.00745	0.95032	0.49987	20	0.00319	2
0.917	21	1 hr 13 min	0.00422	0.89580	0.44161	15	0.00738	10
0.916	22	1 hr 13 min	0.00358	0.89545	0.43265	17	0.00702	7
0.916	19	1 hr 6 min	0.00524	0.89901	0.83991	20	0.00456	7
0.911	56	2 hr 53 min	0.00383	0.89582	0.48209	19	0.00995	5
0.911	26	1 hr 26 min	0.00346	0.89518	0.45709	18	0.01000	7
0.908	25	1 hr 24 min	0.00485	0.89783	0.86183	20	0.00375	7
0.906	26	1 hr 25 min	0.00370	0.88605	0.43497	17	0.00715	7
0.905	27	1 hr 30 min	0.00371	0.89533	0.43649	15	0.00710	9
0.905	22	1 hr 14 min	0.00142	0.86066	0.23803	15	0.00946	8
0.904	50	2 hr 34 min	0.00322	0.89422	0.42500	17	0.00691	7
0.902	65	6 hr 42 min	0.00574	0.86168	0.62611	18	0.00483	10
0.899	37	1 hr 57 min	0.00323	0.89977	0.38399	16	0.01000	9
0.898	25	1 hr 25 min	0.00408	0.89623	0.54761	18	0.00992	5
0.898	25	1 hr 25 min	0.00352	0.88279	0.38143	16	0.00607	10
0.898	12	45 min 57 sec	0.00371	0.89213	0.49687	16	0.00663	10
0.895	21	1 hr 10 min	0.00349	0.89031	0.36946	16	0.00654	9
0.894	38	2 hr 2 sec	0.00298	0.89994	0.28756	17	0.01000	8
0.885	11	47 min 29 sec	0.00599	0.93184	0.64064	17	0.00679	7
0.866	7	33 min 24 sec	0.00443	0.91120	0.44106	15	0.00690	2
0.808	4	22 min 43 sec	0.00357	0.88582	0.38293	15	0.00671	8

Figure 4.1 depicts the distribution of Dice scores on each image in the testing set for the top ten models. We can see in this chart that on average the model outputs a segmentation with a high Dice score in the 0.90 to 1.0 range, though there are many outliers. Many of the outliers in this case are patches which have non-root objects in them, as can be seen in Figure 4.2. Figure 4.2a and Figure 4.2b are examples of successful segmentations of root systems with relatively high Dice scores. We can see that in these patches there are clear areas in the image which are root. In Figure 4.2e however, there is a large amount of plant shoot which is not considered part of the RSA that is segmented by the model. Additionally in Figure 4.2f we can see that water droplets along the root are segmented as part of the root system, which reduces the Dice score of the segmentation. This model produces a mean Dice score of 0.936 with standard deviation 0.162 on the *Testing Patch Set*.

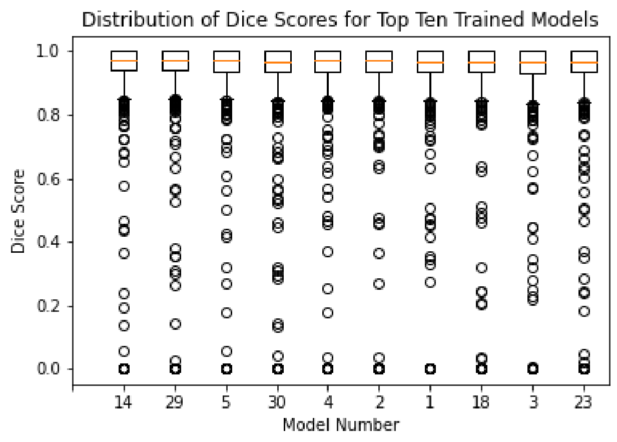
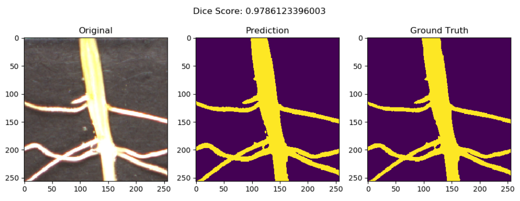
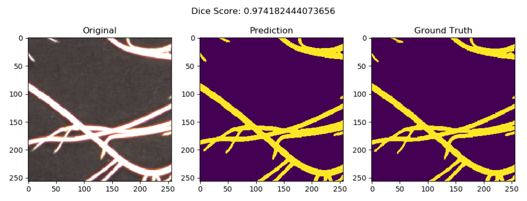


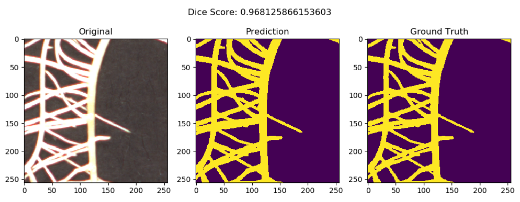
Figure 4.1: Box plot of the Dice score results for the top ten trained models on the *Testing Patch Set*, where the models on the x-axis are in decreasing order of their mean Dice score. We can see many outliers, most of which are caused by non-root objects within the image.



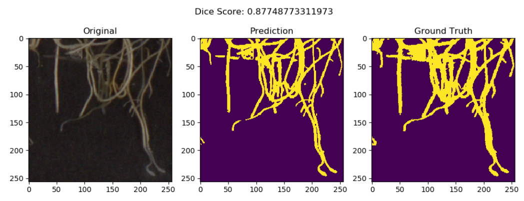
(a) A patch with a successful segmentation.



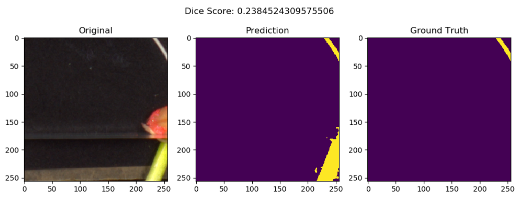
(b) A more complex segmentation.



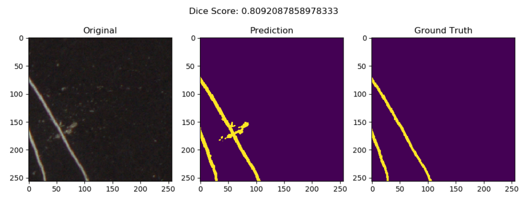
(c) Roots with many gaps and thin areas that are segmented.



(d) A patch with a very complex structure.



(e) A patch containing stem which is misclassified as root.



(f) Water droplets cause reduced segmentation quality.

Figure 4.2: Input, prediction, and ground truth patches for the *Testing Patch Set* on the best model found with hyper-parameter tuning.

To assess the effects of different hyper-parameters on the resulting segmentation quality of the model, a parallel coordinate plot is provided in Figure 4.3. We can see from this plot that a smaller learning rate (around 0.0035) combined with a learning rate decay of around 0.895 seem to be distinctive of a well performing model. Additionally it appears that the network prefers a lower cross entropy weight, though it is less indicative of high Dice score as some models with larger cross entropy weight did perform well. It would seem that stopping patience, stopping epochs, and stopping tolerance have a smaller impact on the overall performance, so long as they are chosen within the ranges implied by the analysis. Stopping patience appears to prefer to be closer to the range of 15 to 17, a number of stopping epochs between 5 and 10 are prevalent in most of the models, and stopping tolerance in the range of 0.004 to 0.010 appear to have small impact on the final Dice score.

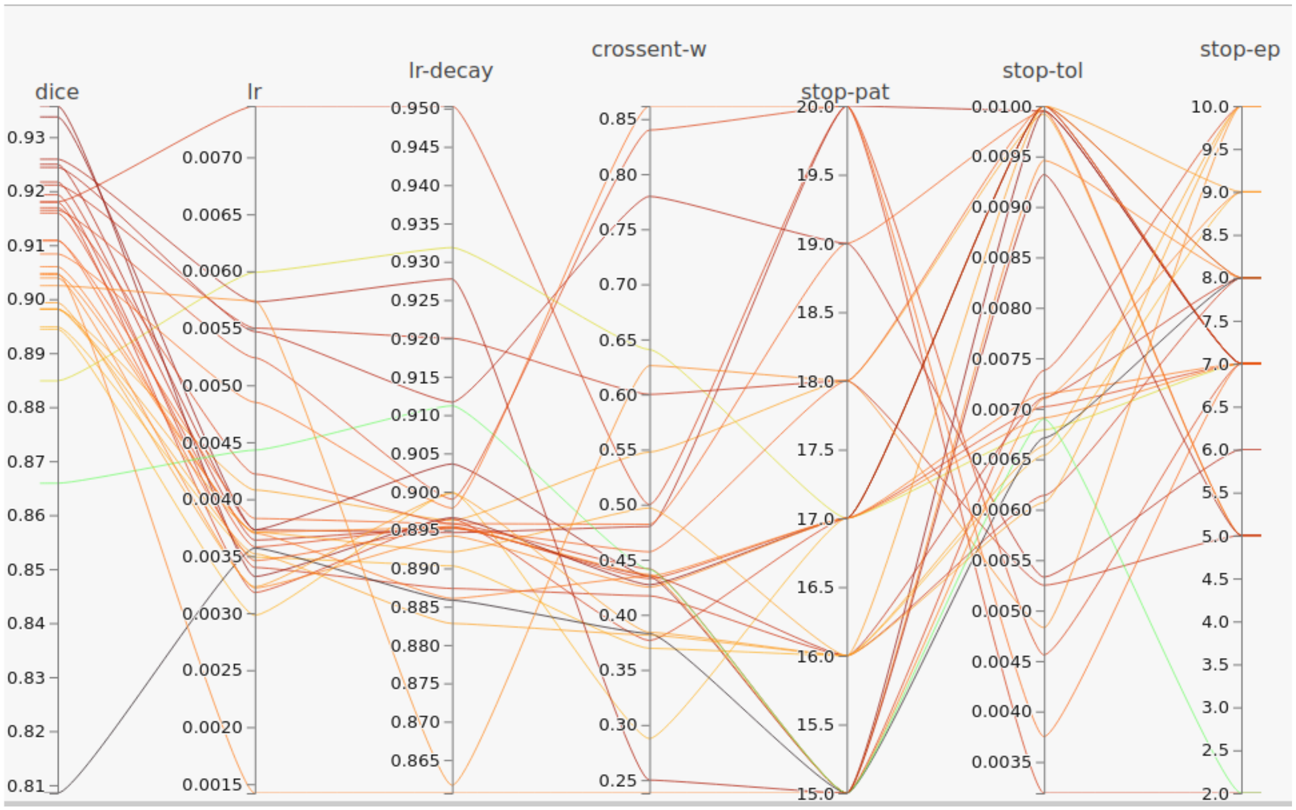


Figure 4.3: Parallel Coordinate plot describing the relationship between the hyper-parameters and the resulting Dice score for all 30 trials. We can see from this plot that the model prefers a smaller learning rate as well as a larger stopping patience. Other parameters seem to have a smaller effect on the overall results.

4.2 Trial 2: Effects of Enhanced Training Dataset

The second trial directly addresses some of the shortcomings of the model that are identified in the results of Trial 1. Using the hyper-parameters of the highest performing of the 30 models in Trial 1, determined by Dice score, we re-train the same model using the *Expanded Patch Training Set* as described in Section 3.3.2.

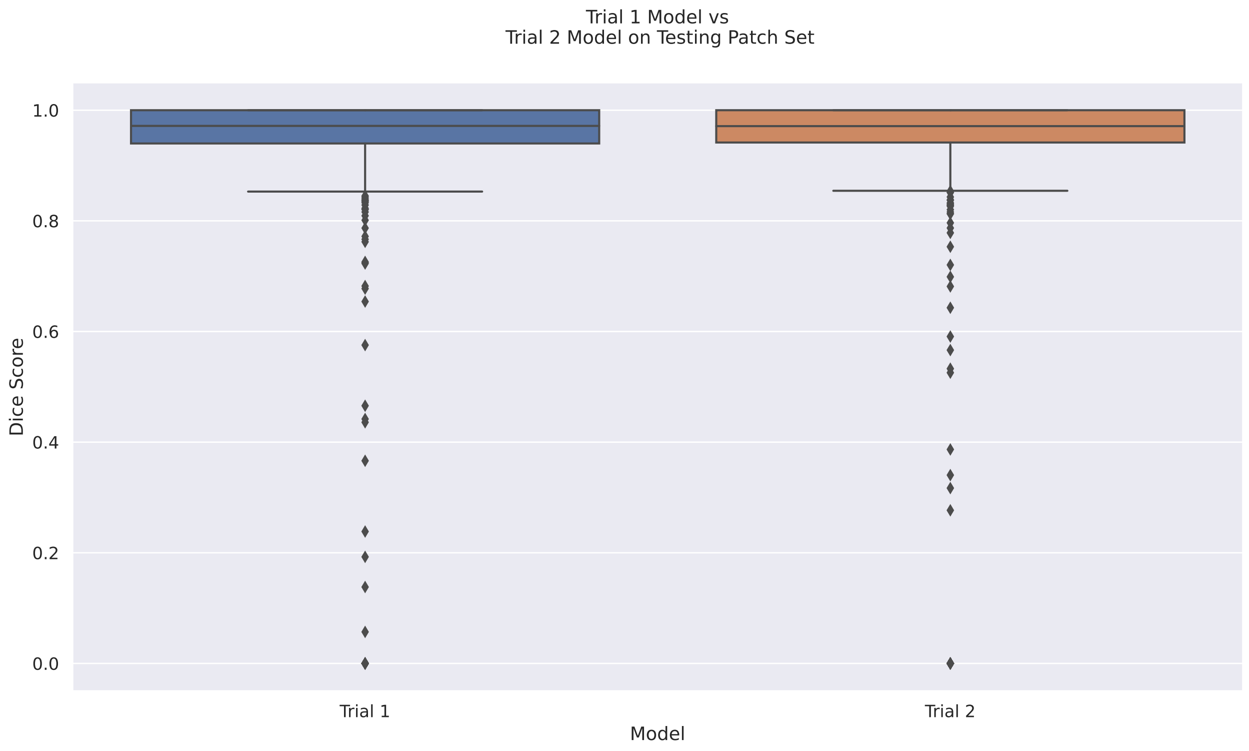


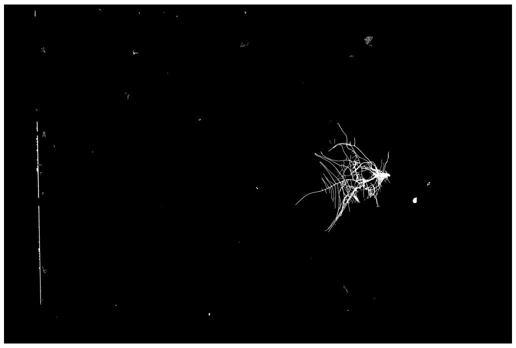
Figure 4.4: A box plot comparing the results of the models from Trial 1 and Trial 2. We can see that the distributions of dice score for the two models are very similar, indicating that both models will perform similarly to each other on the patched datasets.

The hyper-parameter values are as follows: *Learning Rate*: 0.00332, *Learning Rate Decay*: 0.89661, *Binary Cross Entropy Weight*: 0.42715, *Stopping Tolerance*: 0.01, *Stopping Epochs*: 7, *Stopping Patience*: 17. All other parameters and model settings are unchanged.

The mean Dice score on the testing set after re-training was 0.937 with standard deviation 0.160, which is only a marginal difference from the results of the model trained in Trial 1. Results of a two sample t-test indicate that there is no significant difference between the results of the two models on the patch testing set ($p = 0.892, \alpha = 0.05$). In Figure 4.4 we can see that the distributions of the dice score outputs are very similar. The model from Trial 2 does appear to have fewer low outliers than the model from Trial 1, though the average case for both models suggests that they are functionally similar on the patched datasets.

4.3 Trial 3: Patch-Wise Segmentation of Full Images

The model from *Trial 1* and the model from *Trial 2* were both evaluated on the *Full Image Testing Set*. As specified in Section 3.3.3 the full sized images are first split into 256×256-pixel sized patches with an overlap of 128 pixels. After prediction, the patches are stitched back together using a majority vote system. The model from Trial 1 resulted in a mean Dice score of 0.867 with standard deviation 0.065, while the model from Trial 2 had a mean Dice score of 0.889 with standard deviation 0.070. This is not a significant increase in segmentation quality ($p = 0.377, \alpha = 0.05$). Figure 4.5a shows an example segmentation from the Trial 1 model, which has many areas marked as foreground which are not part of the root system. We can see in Figure 4.5b that some of these errors have been mitigated by the model in Trial 2, however the more of the above ground plant matter has been incorrectly segmented as root.


(a) The output on a full image using the model from Trial 1.

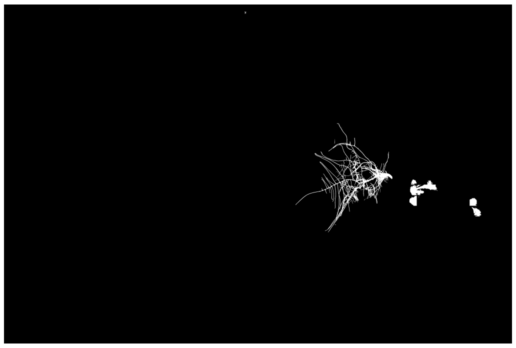

(b) The output on a full image using the model from Trial 2.

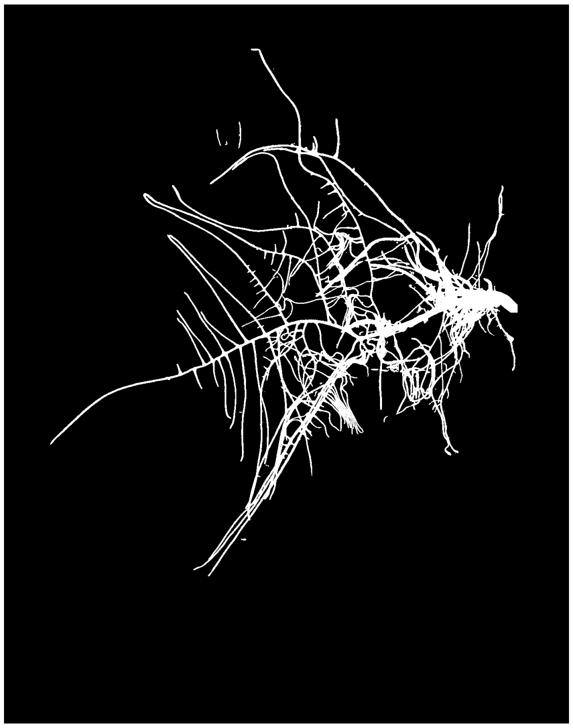
Figure 4.5: Full image output examples from a) the model trained in Trial 1 and b) the model trained in Trial 2. The model in Trial 2 is more robust to non-root objects in the image, however above ground plant matter is incorrectly identified as root.

4.4 Trial 4: Patch-Wise Segmentation of Cropped Images

The models from Trial 1 and Trial 2 were also evaluated on the *Cropped Image Testing Set.* This set of images reduces the amount of non-root structures in the image, meaning that the deficiencies of the model from Trial 1 are less likely to affect the overall quality of the segmentation. The model from Trial 1 gave a mean Dice score of 0.938 with standard deviation 0.029 on these images, while the model from Trial 2 gave a mean Dice score of 0.936 with standard deviation 0.029. A two sample t-test shows that there is no significant difference in the results of the two models on the cropped image set ($p = 0.872, \alpha = 0.05$). Figure 4.6 displays the segmentation results of the two models on the cropped images.

The model from Trial 1 performs significantly better on the *Cropped Image Testing Set* than the *Full Image Testing Set*, as shown by a two sample t-test ($p = 0.001, \alpha = 0.05$). There is also a significant difference in the results of the model from Trial 2 on the *Full Image Testing Set* versus the *Cropped Image Testing Set* ($p = 0.028, \alpha = 0.05$). Table 4.2 gives a summary of the results of each model on the various testing sets.

Comparing the results of the model from Trial 1 and the model from Trial 2 on the *Testing Patch* set, the *Full Image Testing* set, and the *Cropped Image Testing* set we can see that both models perform better on cropped images than on the full images. Figure 4.7 illustrates this. We can additionally see that there are fewer outliers from both models on the *Full Image* and *Cropped Image* sets compared to the *Testing Patch* set.


(a) The output on a cropped full image using the model from Trial 1.

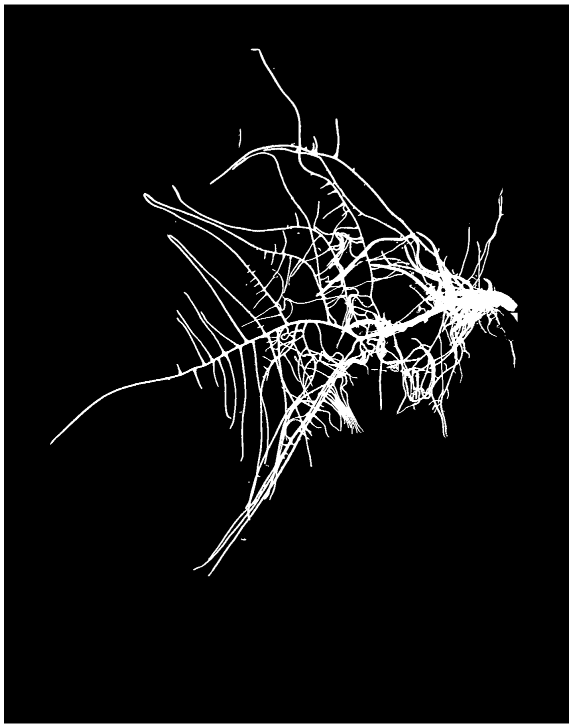

(b) The output on a cropped full image using the model from Trial 2.

Figure 4.6: Cropped image output examples from a) the model trained in Trial 1 and b) the model trained in Trial 2.

Table 4.2: Results for each of the models on different testing sets.

Training Set	**Testing Set**	**Mean DSC**	**Mean IoU**	**Mean Sensitivity**	**Mean Specificity**
Hyper-Parameter Patch Training Set	Testing Patch Set	0.936	0.905	0.963	0.995
	Full Image Set	0.867	0.771	0.941	0.997
	Cropped Image Set	0.938	0.884	0.936	0.997
Expanded Patch Training Set	Testing Patch Set	0.937	0.906	0.950	0.995
	Full Image Set	0.889	0.807	0.930	0.998
	Cropped Image Set	0.936	0.881	0.927	0.997

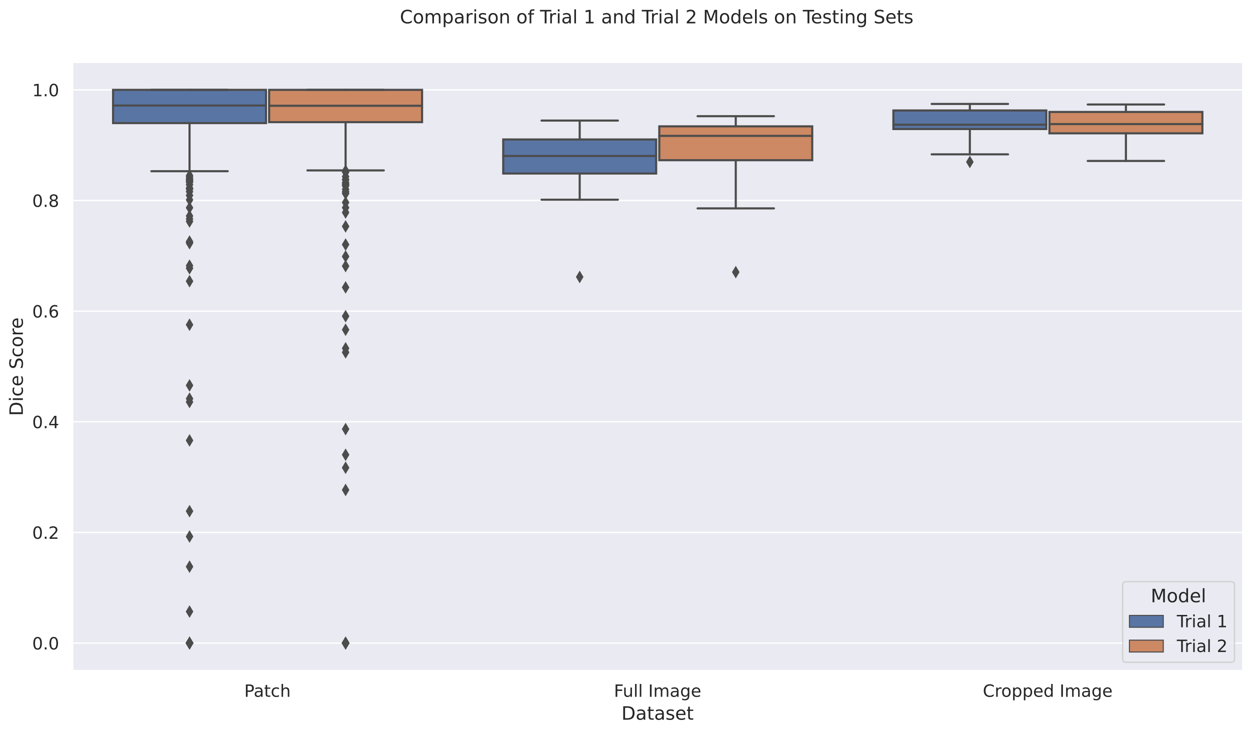


Figure 4.7: A box plot comparing the results of the models from Trial 1 and Trial 2 on each of the *Testing Patch* set, *Full Image Testing* set, and *Cropped Image Testing* set. We can see that there are significantly more poorly performing outliers on the patched dataset for both models. We can also see that both models have higher Dice scores on the *Cropped Image Testing* set than on the *Full Image Testing* set, because cropping the image discards much of the non-root objects which cause errors.

4.5 Validation

At this point we adopt the model trained in Trial 2 as the best performing model, due to its slightly larger training set. While the evidence suggests that both models will perform similarly, in some cases (such as in Figure 4.5) this model has the ability to provide a cleaner segmentation when given images with non-root structures in them. For validation of model results, we test the model using the validation image sets within each of the *Validation Patch Dataset*, *Full Image Validation Dataset*, and the *Cropped Image Validation Dataset*. These are images which have never been segmented by any of the models, giving an unbiased view of how the model performs on new data.

On the *Validation Patch Set* the model produces a mean Dice score of 0.955 with standard deviation 0.149, on the *Full Image Validation Set* the model gives a mean Dice score of 0.838 with standard deviation 0.235, and on the *Cropped Image Validation Set* the model gives a mean Dice score of 0.910 with standard deviation 0.047. This performs as we would expect given the results on the testing sets. For further insight into the performance of the model, we also provide a per-species breakdown of these results. For the *Full Image Validation Set* the model produces a mean Dice score of 0.542 with standard deviation 0.369 on images of canola roots, mean a Dice score of 0.932 with standard deviation 0.025 on images of soybean roots, and

Table 4.3: Per-species breakdown of results on the validation sets.

Dataset	Species	Mean DSC	Mean IoU	Mean Sensitivity	Mean Specificity
Full Image Set	Canola	0.542	0.465	0.813	0.995
	Soybean	0.933	0.875	0.942	0.999
	Wheat	0.916	0.846	0.918	0.995
Cropped Image Set	Canola	0.830	0.710	0.768	0.993
	Soybean	0.953	0.910	0.950	0.998
	Wheat	0.922	0.855	0.909	0.998

a mean Dice score of 0.916 with standard deviation 0.015 on images of wheat roots. On the *Cropped Image Validation Set* the model gives a mean Dice score of 0.830 with standard deviation 0.033 on images of canola, a mean Dice score of 0.953 with standard deviation 0.012 on images of soybean, and a mean Dice score of 0.923 with standard deviation 0.012 on images of wheat. These per-species results are summarized in Table 4.3.

Finally, we evaluate the ability of the model to generalize to a species of plant which was not present in the training set. The model was evaluated on the *Full Image Hold-Out Cucumber Set* and the *Cropped Image Hold-Out Cucumber Set*. A mean Dice score of 0.834 with standard deviation 0.123 was given by the model on the *Full Image Hold-Out Cucumber Set*, and a mean Dice score of 0.844 with standard deviation 0.131 was produced on the *Cropped Image Hold-Out Cucumber Set*. We can see in Figure 4.8 how the model performs on each of the testing, validation, and hold-out cucumber sets. Results on the validation images are consistent with expected results as seen on the testing sets. The *Full Image Validation* set does see two low dice score outliers which are considerably worse than the outlier seen in the *Full Image Testing* set. These two images had particularly small root systems, with many roots clumped closely together. It also appears that two square markers in the image were partially identified as root, causing further error. When we consider the results on the *Full Image Cucumber* and *Cropped Image Cucumber* sets, we see a reduction in performance, as well as a slight widening of the distribution. This is expected, as the model was not trained on any images of Cucumber roots. Example segmentations of cucumber roots are shown in Figure 4.9.

We compare our models to the *SegRoot* model proposed by Wang *et al.* [54] and the *IterNet* model proposed by Li *et al.* [32]. Both models were re-trained from scratch on the *Expanded Patch Training Set* and evaluated on the same sets as our model. Table 4.4 summarizes the results of our adopted model along with the results of *SegRoot* and *IterNet*. We can see in Figure 4.10 that *ITErRoot* outperforms the comparator models, particularly on the *Full Image Set*. This visualization shows us that not only the mean segmentation dice score is improved, but the distribution of expected dice scores is improved as well. There are however a few outliers that do perform approximately as poorly as the minimum dice score observed by the comparators in the *Full Image Validation* set and *Full Image Cucumber* set. It is unclear exactly what has caused these

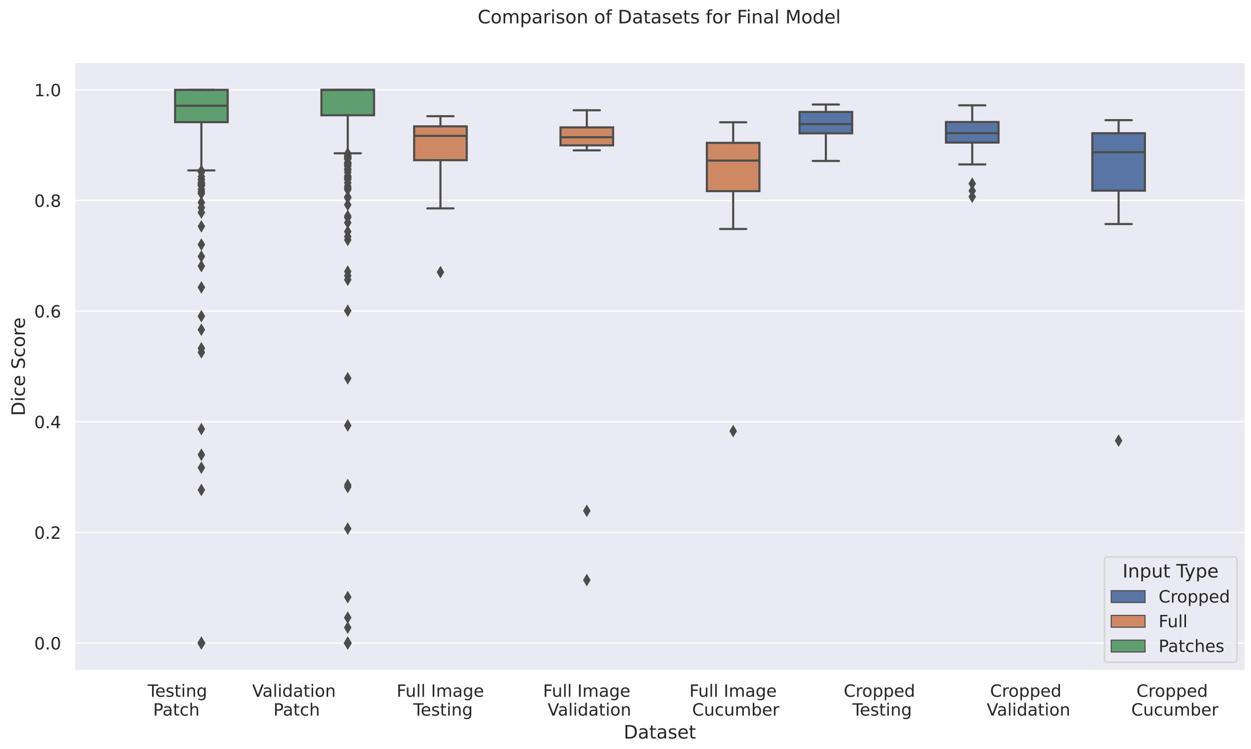


Figure 4.8: A box plot illustrating the results of the final model (the model trained in Trial 2) on each of the testing, validation, and hold-out cucumber validation sets. We can see that the results on each of the validation sets are consistent with what we expect given the results on the testing sets. We can also see a decrease in performance on the *cucumber* images. This is expected, due to the training sets not containing any images of cucumber roots.

particular outputs. Figure 4.11 shows example segmentations from each model.

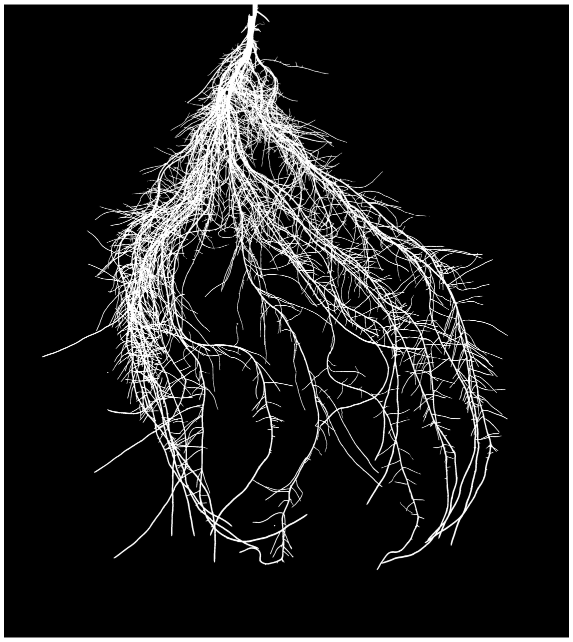
4.6 Discussion

The results we have seen from each of the experimental trials give much insight into the properties of the *ITErRoot* architecture. From these results we are able to better understand how the properties of the dataset, the different hyper-parameter input choices, and different experimental choices affect the output segmentation quality. The remainder of this section will discuss the benefits and drawbacks of our approach, and bring to light some considerations for future improvements on this work. We begin by considering how our data was annotated.

While the *Friendly Ground Truth* annotation tool provides a streamlined process for annotation of root systems, our dataset was created by non-experts in the field of RSA, and plant phenotyping in general. One shortcoming of this work is the lack of expert verification on the ground truth segmentations produced with the tool. It would be ideal to have a plant science expert who is familiar with RSA involved in further annotation processes with the tool in order to ensure that important details are in fact being annotated, and that the resulting data is useful to plant breeders later on in the RSA phenotyping pipeline. It may be

Table 4.4: Comparison With Other Models On Each Dataset. All models trained on the *Expanded Patch Training Set*.

Dataset	**Model**	**Mean DSC**	**Mean IoU**	**Mean Sensitivity**	**Mean Specificity**
Testing Patch Set	ITErRoot	**0.937**	**0.906**	**0.950**	**0.995**
	SegRoot	0.774	0.697	0.789	0.990
	IterNet	0.809	0.760	0.867	0.991
Validation Patch Set	ITErRoot	**0.955**	**0.936**	**0.963**	**0.997**
	SegRoot	0.836	0.792	0.869	0.992
	IterNet	0.846	0.814	0.903	0.996
Full Image Testing Set	ITErRoot	**0.889**	**0.807**	**0.930**	**0.998**
	SegRoot	0.313	0.194	0.637	0.972
	IterNet	0.399	0.256	0.703	0.972
Full Image Validation Set	ITErRoot	**0.838**	**0.769**	**0.901**	**0.998**
	SegRoot	0.234	0.137	0.640	0.969
	IterNet	0.350	0.222	0.688	0.975
Full Image Cucumber Set	ITErRoot	**0.834**	**0.731**	**0.779**	**0.998**
	SegRoot	0.426	0.274	0.582	0.970
	IterNet	0.618	0.463	0.703	0.986
Cropped Testing Set	ITErRoot	**0.936**	**0.881**	**0.927**	0.997
	SegRoot	0.703	0.563	0.647	0.986
	IterNet	0.791	0.687	0.705	**0.999**
Cropped Validation Set	ITErRoot	**0.910**	**0.838**	**0.889**	0.997
	SegRoot	0.716	0.572	0.639	0.992
	IterNet	0.791	0.674	0.684	**0.999**
Cropped Cucumber Set	ITErRoot	**0.844**	**0.748**	**0.779**	0.998
	SegRoot	0.720	0.564	0.583	0.998
	IterNet	0.814	0.693	0.708	**0.999**


(a) An example of a predicted segmentation with Dice score 0.9189

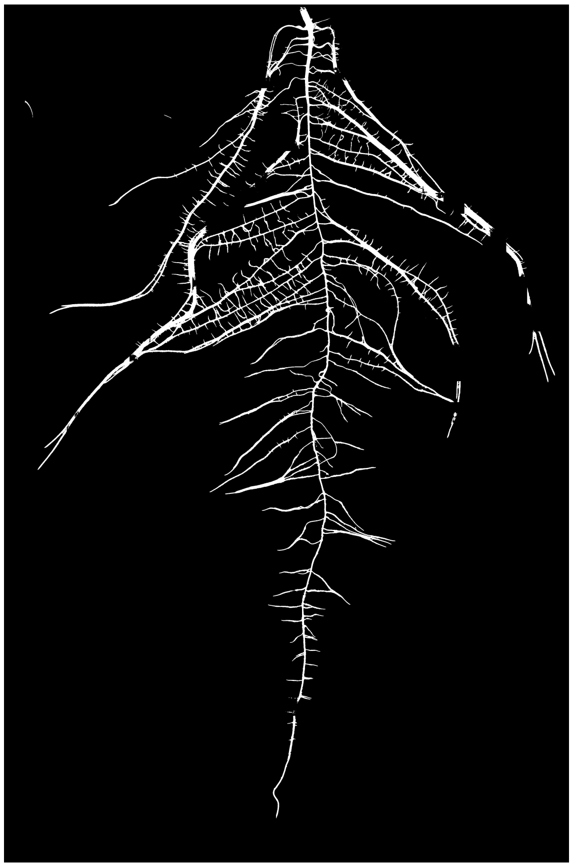

(b) An example of a predicted segmentation with Dice score 0.909

Figure 4.9: Output segmentation masks from the hold out cucumber set showing a) a segmentation with Dice score 0.9189 and b) a segmentation with Dice score 0.909.

beneficial to perform inter-rater agreement studies with the dataset, comparing how an expert's annotations may differ from a non-expert's, as well as the variation that may occur among many different annotators. This type of study would give further insight into the quality and usefulness of the dataset we have created, as well as the *Friendly Ground Truth* annotation tool.

Our hyper-parameter tuning trial provides some insight into the effects of hyper-parameter choice, however it does not give strong evidence for a particular set of input values to achieve maximum performance. Many of the models were trained with a *learning rate* in the 0.30 to 0.45 range, though some outliers did give similar or slightly better performance. Similarly, the *learning rate decay* parameter prefers to be in the 0.89 to 0.90 range. The *binary cross entropy weight* parameter has less influence on the Dice score, as values within the entire input range appear to give high Dice score performance with no obvious pattern. Additionally, it appears that while setting the early stopping related parameters, namely *stopping patience*, *stopping tolerance*, and *stopping epochs*, to conservative values will allow the model more time to train while still not overfitting to the training data. It would seem that hyper-parameter values chosen in these ranges will provide a more stable training process, though are not guaranteed to give a specific performance output.

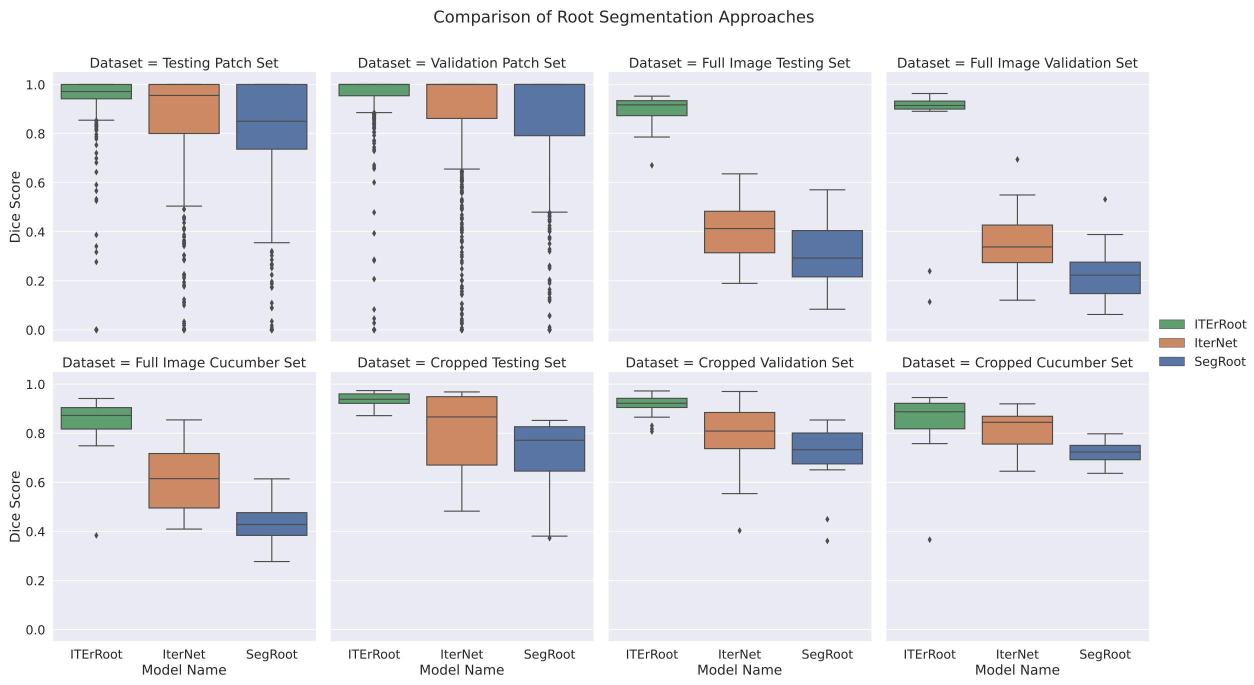
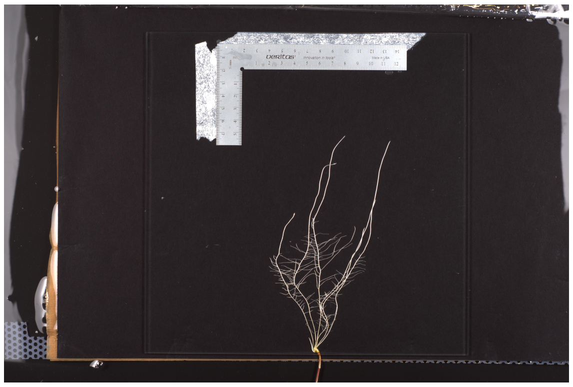


Figure 4.10: A multi facet plot comparing *SegRoot*, *IterNet*, and *ITErRoot* on each of the datasets. We can see that *ITErRoot* outperforms the comparator models, particularly in the case of the *Full Image Set*, where non-root objects are present.

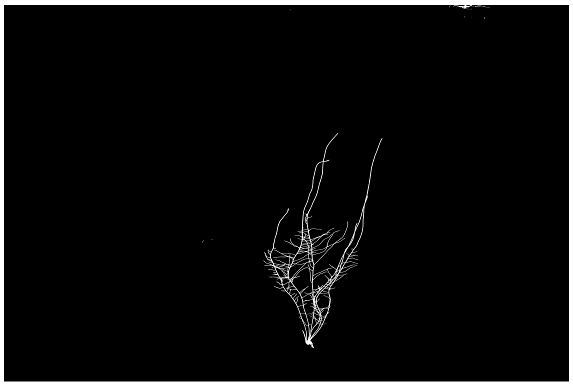
Another consideration is that we were only able to evaluate 30 models, with a rather large hyper-parameter input space. It is possible that further analysis of the impact of specific hyper-parameters may reveal insight on exactly how each hyper-parameter affects the model and point toward better optimization of the network architecture for producing high quality segmentation output.

The best performing model from *Trial 1* resulted in a mean Dice score of 0.936 on the *Testing Patch Set*, while achieving a mean Dice score of 0.867 on the *Full Image Set*. We can see from the examples in Figure 4.2 and Figure 4.5a that the model does a reasonably good job in segmenting the root system. Many of the complex gaps and root tip details are preserved, giving a good quality segmentation of some of the more complicated root systems. We can see, however, that the model picks up on many non-root objects in the image, which in turn reduces the overall Dice score of the segmentation even though much of the root system is correctly identified.

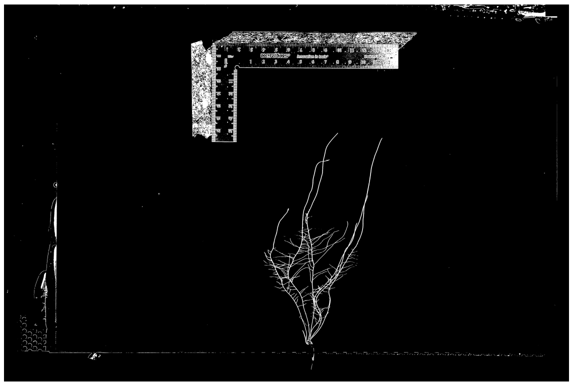
From the *Trial 2* model we can directly see the impact that the additional empty patches make on the model's ability to identify roots. In Figure 4.5b we can see that almost none of the non-root objects identified by the model in *Trial 1* are present, however the distributions of the Dice score results from the two models show that they perform very similarly. This would suggest that the simple addition of non-root objects in the training set is not enough to provide a higher quality segmentation output. In some cases, such as the image in Figure 4.5b we do see a visually different segmentation, though with the evaluation metrics available to us there is no simple way to determine if it truly has a better quality. In this work, no exploration was done to



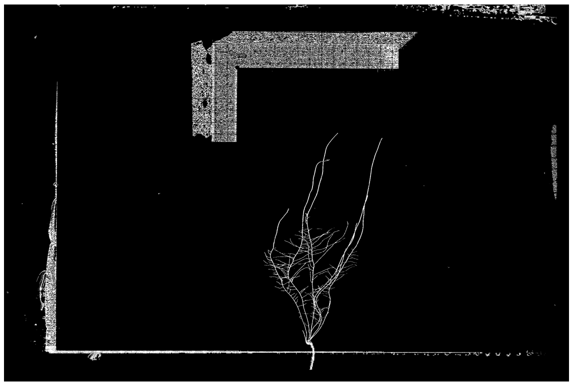
(a) An input validation image.



(b) The predicted validation image from ITErRoot.



(c) The predicted validation image from IterNet.



(d) The predicted validation image from SegRoot.

Figure 4.11: Output segmentations for an input image (a) from b) ITErRoot c) IterNet and d) SegRoot

determine the effects of the number of iterations used in the model. It is possible that improved segmentation quality may be achieved with this architecture by making use of a different number of iterations.

In *Trial 4* we explored the effects of cropping the images before segmentation, which is another approach to mitigating the effects of non-root objects which are present in the image. Interestingly, the results of both models on the cropped images are very similar, with no statistically significant difference in the results. This implies that, while it may be possible to produce a model which can consistently identify non-root objects to improve the segmentation quality, one could achieve a similar performance by performing a pre-cropping operation on the images before segmentation. While it is important to design models which can accurately segment RSA from an image, an equally important problem could be the accurate removal of non-root objects in images via pre-processing or post-processing methods. It is quite reasonable to apply these methods in conjunction with our model to achieve the best possible results.

Another consideration for the results on the *Full Image Set* and the *Cropped Image Set* is the effect of the majority voting system that was used when re-assembling the patched segmentation outputs. In the event of a tie, when two patches have identified a pixel as root, and two patches have identified the same pixel as background, we default to classifying that pixel as background. This is under the assumption that due to the large class imbalance of background pixels versus root pixels the model will naturally be better at identifying background than it is at identifying roots. This voting approach may be affecting the resulting full image segmentation quality, positively or negatively. It may be worthwhile to consider how reversing this system, classifying the pixel as root in the event of a tie, may effect the results, or perhaps using a different re-assembly system altogether. Further understanding of how the re-assembly process affects the final segmentation results may help to identify other shortcomings or advantages of the network architecture.

When comparing the validation results on a per-species basis, we can see that the model performs quite poorly on full images of canola roots compared to soybean and wheat roots. These results are improved dramatically when evaluating on the cropped images, though canola roots still have the lowest Dice score performance. The differences in the RSA of canola roots from soybean and wheat give a partial explanation for these results. The canola roots in our dataset tend to have more roots which grow together, appearing thicker and much harder to identify than in the wheat and soybean images. For this reason we would expect a small decrease in segmentation quality when evaluating canola roots, though not to the degree that we see when evaluating the full sized images. The results on the cropped images provide insight to the second part of the explanation, which is simply that the canola images which are present in the validation set have prominent non-root objects in them which are removed when cropping is done. It is possible that these non-root objects do not share properties of the $1,069$ additional patches which we added to the training set in *Trial 2*, which would explain why they are not properly identified by the model. This further enforces the idea that much work needs to be done in pre-processing or post-processing root images to remove undesirable non-root objects.

Figure 4.9 displays two segmented cucumber root systems. We can see from Figure 4.9a that even a complex root system can be successfully segmented, even though this species of plant has never before been seen by the model. This shows that the model can effectively identify roots in a general sense, regardless of species. The example in Figure 4.9b has roots which have grown together and are much harder to identify, causing gaps in the segmentation. These are properties which are similar to the canola examples described previously. We can see from this information that the model can identify thin root structures, however it is identifying some parts of the root system as non-root objects, which is the main point of weakness in our system. In fact, the mean Dice score of the full sized cucumber images was 0.834, compared to the mean Dice score of the full sized canola images which was 0.542. The model actually performs worse on the canola images, of which many examples are present in the training set, than it does on the cucumber roots which are not present in the training set. This shows that the model can generalize well to the segmentation of thin root structures.

In Table 4.4 we detail a direct comparison between the results of *ITErRoot*, *SegRoot* [54], and *IterNet* [32] on each of the testing and validation sets which we have used. *ITErRoot* outperforms the other models in almost every metric, in every dataset, often by quite a large margin in terms of Dice score. *IterNet* gives a higher specificity score on the cropped image sets, though it is important to keep in mind that the specificity score is the model's ability to identify background pixels, rather than foreground (root) pixels. While it is ideal to maximize the sensitivity and specificity scores, for this particular task sacrificing sensitivity score for specificity score is not indicative of a better quality segmentation. Notably, *ITErRoot* performs more consistently on the full sized images, and provides a much better segmentation. All of these models were trained on the *Expanded Patch Training Set*, which contain the additional empty patches, implying that our network architecture is much better suited to identifying non-root objects, as well as producing a higher quality segmentation of the RSA. It is important to consider that we did not perform hyper-parameter tuning on the comparator networks. As such, it is possible that expending more resources to perform such tuning with these architectures may yield results that are consistent with those that were achieved with *ITErRoot*.

One aspect of our architecture that we did not explore was the intermediate outputs of the individual iterations within the network. Further analysis of these outputs may indicate the added value of extra iterations in the network and whether they are worth the additional computational resources required to train and perform prediction with them. A direct comparison of the segmentation quality at each iteration in the network may reveal improved methods for developing iterative network architectures and allow for further improvements of segmentation quality in root images. While there is a significant resource requirement for increased numbers of iterations, it would be valuable to have an understanding of the value of each iteration in order to make decisions about accuracy/resource trade-offs. Analysis of these individual outputs could also be compared directly to other segmentation approaches to determine if the network is truly learning to identify root objects, or if other properties of the image are being learned by the iterations. Further exploration of the individual outputs of the sub-networks may help to identify exactly what the individual networks are doing and expose improved methods for training deep segmentation networks.

Our model provides an accurate system for automatic segmentation of RSA which has been proven to generalize to other species of plant roots. The model achieves higher segmentation quality over comparison models by almost every metric. Rigorous analysis has detailed the strengths and weaknesses of this network architecture for the task of segmentation of RSA, which are important for future deployment in a larger RSA pipeline, as well as outlining areas of the RSA pipeline which could be improved to increase performance of automated root phenotyping approaches.

5 Conclusion

The proposed neural network architecture has been evaluated on an extensive set of trials designed to identify strengths and weaknesses for the task of segmenting root systems. Overall our model provides segmentation results which outperform other prominent approaches to the issue of segmenting thin branching structures. The model is well suited for the segmentation of RSA as part of a larger automatic phenotyping pipeline.

5.1 Contributions

This work has produced the following contributions to the field of high-throughput RSA phenotyping:

- A novel tool for annotation of root systems was designed and used to create training data
- A novel dataset of 2D images of root systems of differing species was curated and used for training
- A neural network architecture ITErRoot, which is well suited to segmentation of root systems

The main objective of this work was to design and evaluate a generalizable and accurate automated approach to root segmentation. To that end we have developed a neural network architecture which is capable of producing high-quality segmentations of RSA, outperforming the segmentation quality of recent approaches to this task. We have provided rigorous analysis of the proposed model which outline the strengths and weaknesses which affect segmentation quality. Our evaluation shows that the proposed architecture is well suited for segmentation of RSA and can generalize to new species of plant roots.

We have also designed a novel tool for annotation of root systems which can be used to produce ground truth data for training future RSA segmentation models. This tool helps to streamline the difficult task of manual root annotation while providing high-quality ground truth data. This tool was used to produce a novel image dataset of root images containing differing species of plant roots, along with high-quality ground truth segmentations which can be used to train any segmentation approach for RSA segmentation.

Our approach to the problem of high-throughput root segmentation has performed well in cases of complex root structure, as well as mitigating the errors which occur from misidentified non-root objects. Overall our results show that this is a promising method for RSA segmentation which could be used as part of a larger automated pipeline for the phenotyping of plant roots.

5.2 Future Work

Our analysis outlines a few areas where further research may help to improve segmentation of RSA. These areas are outlined by the weaknesses found in our analysis, where segmentation accuracy has suffered due to various properties of roots. These areas are as follows:

- A solution for more accurate segmentation of roots which grow together. As was seen in the results for canola RSA, roots can grow very closely together becoming very difficult to distinguish between. A solution to this problem would further increase the segmentation quality and generalizability of the model.

- A method for removing non-root objects within the input image. While our model does perform quite well in the presence of non-root objects, it is difficult to say how the model may perform if more complex background objects are present. Some form of automatic cropping could help to mitigate these errors, as well as other image processing methods for removing unwanted objects in the image.

References

[1] R. Adams and L. Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, June 1994.

[2] Md Zahangir Alom, Mahmudul Hasan, Chris Yakopcic, Tarek M Taha, and Vijayan K Asari. Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation. *arXiv preprint arXiv:1802.06955*, 2018.

[3] Jose M. Alvarez, Theo Gevers, Yann LeCun, and Antonio M. Lopez. Road scene segmentation from a single image. In *Computer Vision – ECCV 2012*, volume 7578, pages 376–389, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[4] Patrick Armengaud, Kevin Zambaux, Adrian Hills, Ronan Sulpice, Richard J. Pattison, Michael R. Blatt, and Anna Amtmann. EZ-Rhizo: integrated software for the fast and accurate measurement of root system architecture. *The Plant Journal*, 57(5):945–956, December 2009.

[5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, December 2017.

[6] William A. Barrett and Eric N. Mortensen. Interactive live-wire boundary extraction. *Medical Image Analysis*, 1(4):331–341, February 1997.

[7] Paramita Basu and Anupam Pal. A new tool for analysis of root growth in the spatio-temporal continuum. *New Phytologist*, 195(1):264–274, March 2012.

[8] Philippe Borianne, Gérard Subsol, Franz Fallavier, Audrey Dardou, and Alain Audebert. GT-RootS: An integrated software for automated root system measurement from high-throughput phenotyping platform images. *Computers and Electronics in Agriculture*, 150:328–342, May 2018.

[9] Alexander Bucksch, James Burridge, Larry M. York, Abhiram Das, Eric Nord, Joshua S. Weitz, and Jonathan P. Lynch. Image-Based High-Throughput Field Phenotyping of Crop Roots. *Plant Physiology*, 166(2):470–486, October 2014.

[10] Jinhai Cai and Stan Miklavcic. Surface fitting for individual image thresholding and beyond. *IET Image Processing*, 7(6):596–605, February 2013.

[11] Jinhai Cai, Zhanghui Zeng, Jason N. Connor, Chun Yuan Huang, Vanessa Melino, Pankaj Kumar, and Stanley J. Miklavcic. RootGraph: a graphic optimization tool for automated image analysis of plant roots. *Journal of Experimental Botany*, 66(21):6551–6562, June 2015.

[12] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986.

[13] Hao Chen, Mario Valerio Giuffrida, Peter Doerner, and Sotiros A. Tsaftaris. Adversarial Large-scale Root Gap Inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2619–2628. IEEE, June 2019.

[14] Venkateswararao Cherukuri, Vijay Kumar B G, Raja Bala, and Vishal Monga. Multi-Scale Regularized Deep Network For Retinal Vessel Segmentation. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 824–828. IEEE, 2019.

[15] Kashyap Chitta, Jose M. Alvarez, and Martial Hebert. Quadtree Generating Networks: Efficient Hierarchical Scene Parsing with Sparse Convolutions. In *The IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

[16] Tino Colombi, Norbert Kirchgessner, Chantal Andrée Le Marié, Larry Matthew York, Jonathan P. Lynch, and Andreas Hund. Next generation shovelomics: set up a tent and REST. *Plant and Soil*, 388:1–20, February 2015.

[17] Clément Douarre, Richard Schielein, Carole Frindel, Stefan Gerth, and David Rousseau. Transfer Learning from Synthetic Data Applied to Soil–Root Segmentation in X-Ray Tomography Images. *Journal of Imaging*, 4(5):65i–79, May 2018.

[18] Taras Galkovskyi, Yuriy Mileyko, Alexander Bucksch, Brad Moore, Olga Symonova, Charles A. Price, Christopher N. Topp, Anjali S. Iyer-Pascuzzi, Paul R. Zurek, Suqin Fang, John Harer, Philip N. Benfey, and Joshua S. Weitz. GiA Roots: software for the high throughput analysis of plant root system architecture. *BMC Plant Biology*, 12(1):116–128, 2012.

[19] Steven Guan, Amiri A. Khan, Siddhartha Sikdar, and Parag V. Chitnis. Fully Dense UNet for 2D Sparse Photoacoustic Tomography Artifact Removal. *IEEE Journal of Biomedical and Health Informatics*, 24(2):235–576, 2020.

[20] Shuai Guo, Songyuan Tang, Jianjun Zhu, Jingfan Fan, Danni Ai, Hong Song, Ping Liang, and Yang Jian. Improved U-Net for Guidewire Tip Segmentation in X-ray Fluoroscopy Images. In *Proceedings of the 2019 3rd International Conference on Advances in Image Processing*, pages 55–59, November 2019.

[21] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[22] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31, 2017.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[24] Anjali S. Iyer-Pascuzzi, Olga Symonova, Yuriy Mileyko, Yueling Hao, Heather Belcher, John Harer, Joshua S. Weitz, and Philip N. Benfey. Imaging and Analysis Platform for Automatic Phenotyping and Trait Ranking of Plant Root Systems. *Plant Physiology*, 152(3):1148–1157, March 2010.

[25] Qiangguo Jin, Zhaopeng Meng, Tuan D Pham, Qi Chen, Leyi Wei, and Ran Su. DUNet: A deformable network for retinal vessel segmentation. *Knowledge-Based Systems*, 178:149–162, 2019.

[26] Konstantinos Kamnitsas, Christian Ledig, Virginia F J Newcombe, Joanna P Simpson, Andrew D Kane, David K Menon, Daniel Rueckert, and Ben Glocker. Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78, 2017.

[27] Michael Kampffmeyer, Nanqing Dong, Xiaodan Liang, Yujia Zhang, and Eric P. Xing. ConnNet: A Long-Range Relation-Aware Pixel-Connectivity Network for Salient Segmentation. *IEEE Transactions on Image Processing*, 28(5):2518–2529, May 2019.

[28] Yasmin M Kassim, O V Glinskii, V V Glinsky, V H Huxley, G Guidoboni, and K Palaniappan. Deep U-Net Regression and Hand-Crafted Feature Fusion for Accurate Blood Vessel Segmentation. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1445–1449. IEEE, IEEE, September 2019.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 60(6):84–90, June 2017.

[30] Pankaj Kumar, Chunyuan Huang, Jinhai Cai, and Stanley J. Miklavcic. Root phenotyping by root tip detection and classification through statistical learning. *Plant and Soil*, 380(1):193–209, March 2014.

[31] Jacques Le Bot, Valérie Serra, José Fabre, Xavier Draye, Stéphane Adamowicz, and Loïc Pagès. DART: A software to analyse root system architecture and development from captured images. *Plant and Soil*, 326(1):261–273, 2010.

[32] Liangzhi Li, Manisha Verma, Yuta Nakashima, Hajime Nagahara, and Ryo Kawasaki. IterNet: Retinal Image Segmentation Utilizing Structural Redundancy in Vessel Networks. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 3656–3665, 2020.

[33] Wenqi Li, Guotai Wang, Lucas Fidon, Sebastien Ourselin, M Jorge Cardoso, and Tom Vercauteren. On the Compactness, Efficiency, and Representation of 3D Convolutional Networks: Brain Parcellation as a Pretext Task. In *International conference on information processing in medical imaging*, pages 348–360. Springer, 2017.

[34] Guillaume Lobet, Loïc Pagès, and Xavier Draye. A Novel Image-Analysis Toolbox Enabling Quantitative Analysis of Root System Architecture. *Plant Physiology*, 157(1):29–39, September 2011.

[35] Guillaume Lobet, Michael P. Pound, Julien Diener, Christophe Pradal, Xavier Draye, Christophe Godin, Mathieu Javaux, Daniel Leitner, Félicien Meunier, Philippe Nacry, Tony P. Pridmore, and Andrea Schnepf. Root System Markup Language: Toward a Unified Root Architecture Description Language. *Plant Physiology*, 167(3):617–627, March 2015.

[36] Birgit Möller, Hongmei Chen, Tino Schmidt, Axel Zieschank, Roman Patzak, Manfred Türke, Alexandra Weigelt, and Stefan Posch. rhizoTrak: A flexible open source Fiji plugin for user-friendly manual annotation of time-series images from minirhizotrons. *Plant and Soil*, 444(1-2):519–534, February 2019.

[37] Kerstin A. Nagel, Alexander Putz, Frank Gilmer, Kathrin Heinz, Andreas Fischbach, Johannes Pfeifer, Marc Faget, Stephan Blossfeld, Michaela Ernst, Chryssa Dimaki, Bernd Kastenholz, Ann-Katrin Kleinert, Anna Galinski, Hanno Scharr, Fabio Fiorani, and Ulrich Schurr. GROWSCREEN-Rhizo is a novel phenotyping robot enabling simultaneous measurements of root and shoot growth for plants grown in soil-filled rhizotrons. *Functional Plant Biology*, 39(11):891–904, June 2012.

[38] Katayoun Neshatpour, Houman Homayoun, and Avesta Sasan. ICNN: The Iterative Convolutional Neural Network. *ACM Transactions on Embedded Computing Systems*, 18(6):1–27, December 2020.

[39] Nobuyuki Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, January 1979.

[40] Jordon Pace, Nigel Lee, Hsiang Sing Naik, Baskar Ganapathysubramanian, and Thomas Lübberstedt. Analysis of maize (Zea mays L.) Seedling Roots with the High-Throughput Image Analysis Tool ARIA (Automatic Root Image Analysis). *PLoS ONE*, 9(9):1–10, September 2014.

[41] Alain Pierret, Santimaitree Gonkhamdee, Christophe Jourdan, and Jean Luc Maeght. IJ_Rhizo: An open-source software to measure scanned images of root samples. *Plant and Soil*, 373(1):531–539, July 2013.

[42] Michael P. Pound, Andrew P. French, Jonathan A. Atkinson, Darren M. Wells, Malcolm J. Bennett, and Tony Pridmore. RootNav: Navigating Images of Complex Root Architectures. *Plant Physiology*, 162(4):1802–1814, August 2013.

[43] Catherine Reeb, Jaap Kaandorp, Fredrik Jansson, Nicolas Puillandre, Jean Yves Dubuisson, Raphaël Cornette, Florian Jabbour, Yoan Coudert, Jairo Patiño, Jean François Flot, and Alain Vanderpoorten. Quantification of complex modular architecture in plants. *New Phytologist*, 218(2):859–872, January 2018.

[44] Rubén Rellán-Álvarez, Guillaume Lobet, Heike Lindner, Pierre-Luc Pradier, Jose Sebastian, Muh-Ching Yee, Yu Geng, Charlotte Trontin, Therese LaRue, Amanda Schrager-Lavelle, Cara H Haney, Rita Nieu, Julin Maloof, John P Vogel, and José R Dinneny. GLO-Roots: an imaging platform enabling multidimensional characterization of soil-grown root systems. *eLife*, 4:1–26, August 2015.

[45] Daniela Ristova, Ulises Rosas, Gabriel Krouk, Sandrine Ruffel, Kenneth D. Birnbaum, and Gloria M. Coruzzi. RootScape: A Landmark-Based System for Rapid Screening of Root Architecture in Arabidopsis. *Plant Physiology*, 161(3):1086–1096, March 2013.

[46] Eric D. Rogers and Philip N. Benfey. Regulation of plant root system architecture: implications for crop advancement. *Current Opinion in Biotechnology*, 32:93–98, 2015.

[47] Weibin Rong, Zhanjing Li, Wei Zhang, and Lining Sun. An improved canny edge detection algorithm. In *2014 IEEE International Conference on Mechatronics and Automation*, pages 577–582, August 2014.

[48] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer International Publishing, 2015.

[49] Sohini Roychowdhury, Dara D Koozekanani, and Keshab K Parhi. Iterative Vessel Segmentation of Fundus Images. *IEEE Transactions on Biomedical Engineering*, 62(7):1738–1749, February 2015.

[50] R. Slovak, C. Goschl, X. Su, K. Shimotani, T. Shiina, and W. Busch. A Scalable Open-Source Pipeline for Large-Scale Root Phenotyping of Arabidopsis. *The Plant Cell*, 26(6):2390–2403, 2014.

[51] Abraham George Smith, Jens Petersen, Raghavendra Selvan, and Camilla Ruø Rasmussen. Segmentation of roots in soil with U-Net. *Plant Methods*, 16(1):1–15, 2020.

[52] Carole H. Sudre, , Wenqi Li, , Tom Vercauteren, , Sebastien Ourselin, and M. and Jorge Cardoso. Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations. In M. Jorge Cardoso, Tal Arbel, Gustavo Carneiro, Tanveer Syeda-Mahmood, João Manuel R.S. Tavares, Mehdi Moradi, Andrew Bradley, Hayit Greenspan, João Paulo Papa, Anant Madabhushi, Jacinto C. Nascimento, Jaime S. Cardoso, Vasileios Belagiannis, and Zhi Lu, editors, *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 240–248. Springer International Publishing, 2017.

[53] Olga Symonova, Christopher N. Topp, and Herbert Edelsbrunner. DynamicRoots: A Software Platform for the Reconstruction and Analysis of Growing Plant Roots. *PLoS ONE*, 10(6):1–15, June 2015.

[54] Tao Wang, Mina Rostamza, Zhihang Song, Liangju Wang, G. McNickle, Anjali S. Iyer-Pascuzzi, Zhengjun Qiu, and Jian Jin. SegRoot: A high throughput segmentation method for root image analysis. *Computers and Electronics in Agriculture*, 162(May):845–854, May 2019.

[55] Anton Paul Wasson, RA Richards, R Chatrath, SC Misra, SV Sai Prasad, GJ Rebetzke, JA Kirkegaard, J Christopher, and M Watt. Traits and selection strategies to improve root systems and water uptake in water-limited wheat crops. *Journal of experimental botany*, 63(9):3485–3498, May 2012.

[56] L. C. Williamson, S. P.C.P. Ribrioux, A. H. Fitter, and H. M. Ottoline Leyser. Phosphate Availability Regulates Root System Architecture in Arabidopsis. *Plant Physiology*, 126(2):875–882, June 2001.

[57] Feixiang Yan, Hong Zhang, and C. Ronald Kube. A multistage adaptive thresholding method. *Pattern Recognition Letters*, 26(8):1183–1191, June 2005.

[58] Robail Yasrab, Jonathan A Atkinson, Darren M Wells, Andrew P French, Tony P Pridmore, and Michael P Pound. RootNav 2.0 : Deep Learning for Automatic Navigation of Complex Plant Root Architectures. *GigaScience*, 8(11), July 2019.

[59] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. Road Extraction by Deep Residual U-Net. *IEEE Geoscience and Remote Sensing Letters*, 15(5):749–753, March 2018.

[60] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. UNet++: Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation. *IEEE Transactions on Medical Imaging*, 39(6):1856–1867, December 2019.

Appendix

Annotation Manual Provided To Volunteers

Annotation Manual

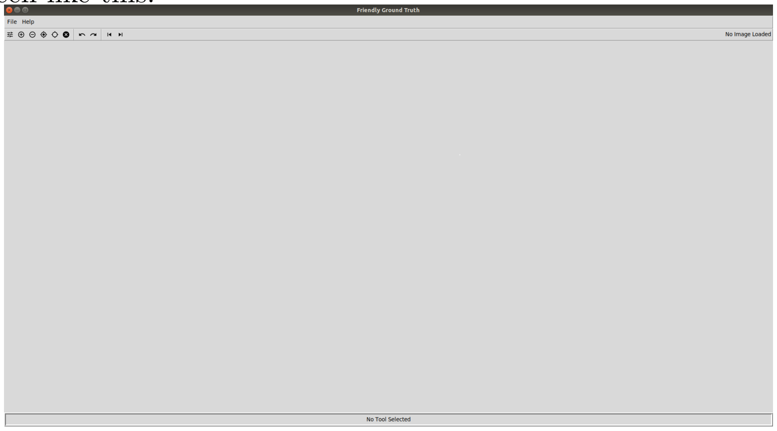
Kyle Seidenthal

April 2020

Contents

1 Getting Started

First, you'll need to download Friendly Ground Truth, an annotation tool for root images, here. Once you have done that, you should be looking at a screen like this:



A detailed user manual can be found here. It outlines how all of the tools work, and it would be a good idea to have that open while you are going through this manual. Take a moment to look at the keyboard shortcuts window in the about menu.

When you first start the program, it will prompt for an output directory. Please choose the annoations/ directory in the folder you were given. This will be the default location for saving output files.

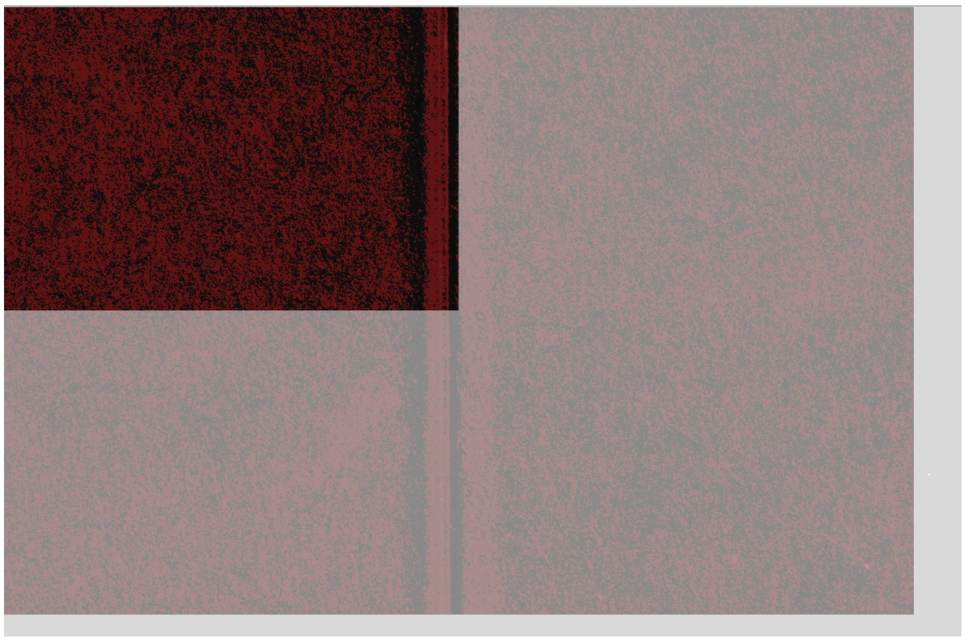
The preview window is docked in the main window by default. If you like, you can use the "View" menu to change the positioning of the preview window to "floating" or "hidden" to create more space for working in. You can drag and scroll inside the preview window to examine the whole image and its current mask. Clicking inside any patch in the preview window will move the patch view to that patch for annotating.

The folder you were given contains two folders and a file. The file is this document. The two folders are labelled annotations, and images. The images folder contains a number of .tif images which need to be annotated, and the annotations folder is the folder where the annotations will be stored.

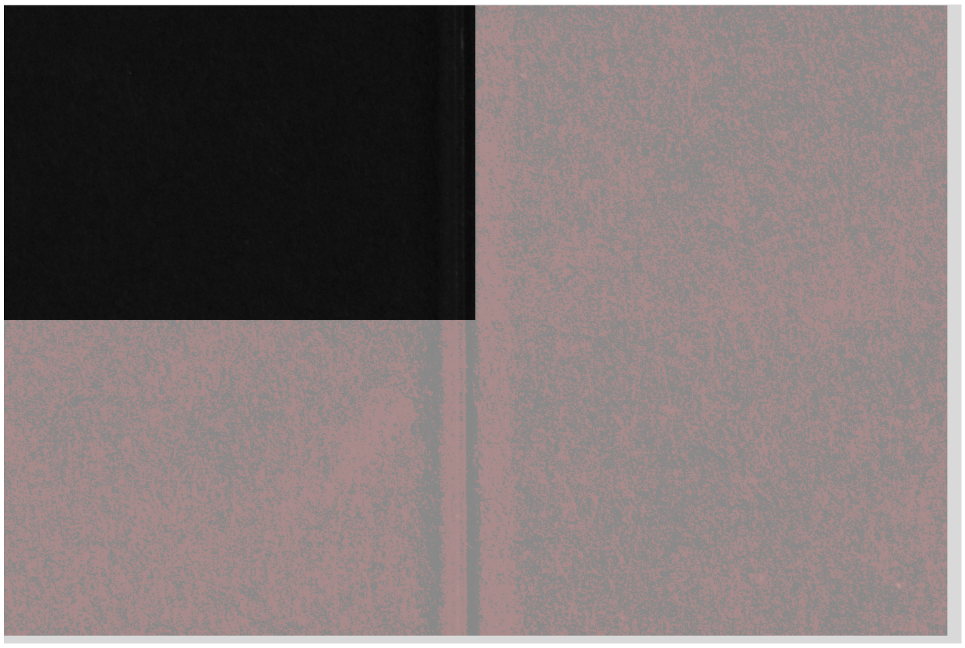
2 Steps For Annotation

The goal for this annotation process is to mark all pixels in each image as either root (foreground) or not root (background). The Friendly Ground Truth tool displays current foreground pixels in red, so the goal is for all the parts of the image that are root to be marked red, while everything else is not red.

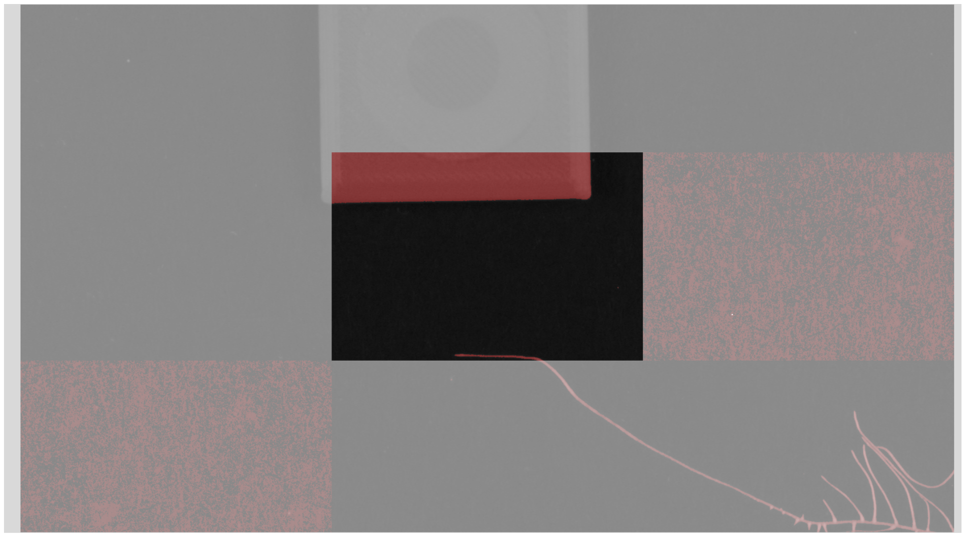
The first step is to open an image using the file menu. The image will be loaded and split into patches, which will look something like this:



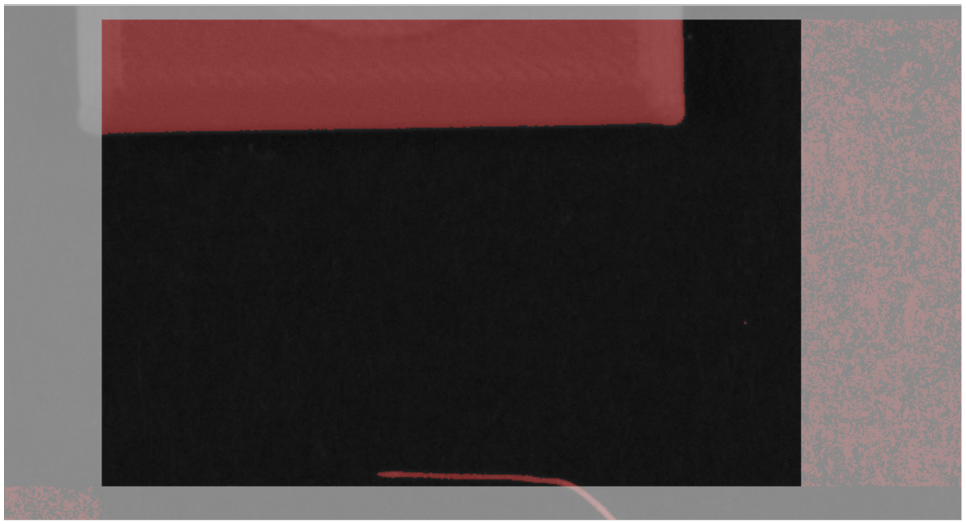
The greyed out parts are neighbouring patches to the one you will be annotating. The current patch (the not-greyed out one) is full of red, but there is no root visible in this patch. For this case, we use the 'No Root Tool' by pressing 'X' on the keyboard:



Now we use the right arrow key to move to the next patch. The first few patches will likely all be the same, with no roots in them, so we repeat this process until we arrive at a patch with something a bit more interesting:

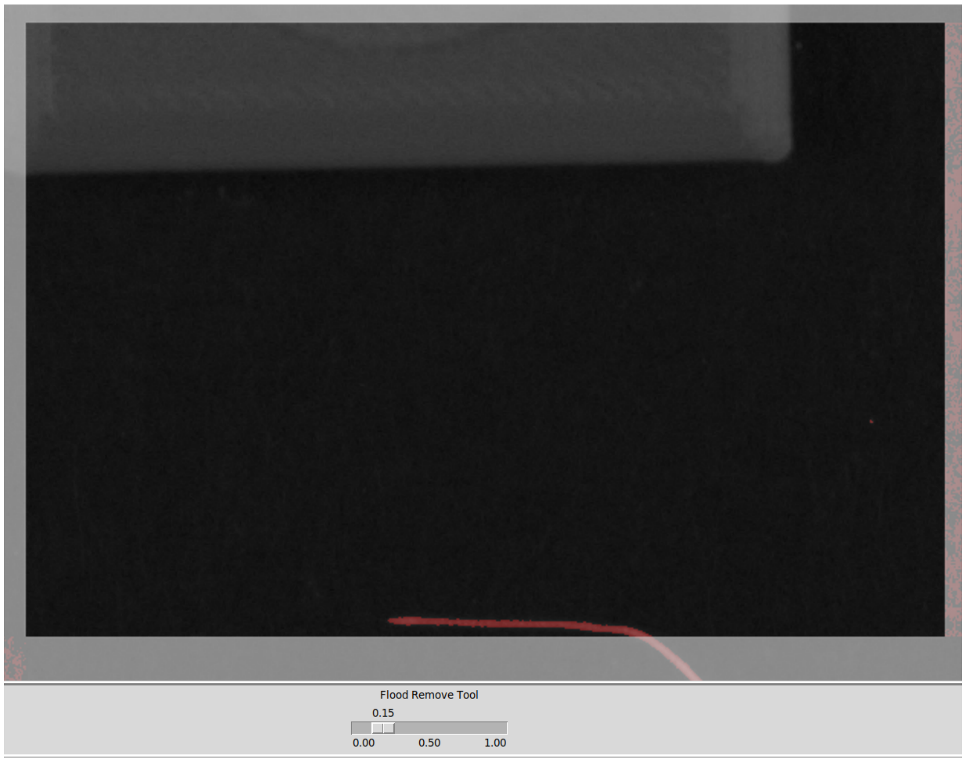


This patch has a bit of root in it, and we can see from a neighbouring patch that there are even more to come. We can scroll the mouse-wheel to zoom in a bit more, and see that the automatic threshold has done a pretty good job getting the roots in this patch, but there is a big red area at the top:

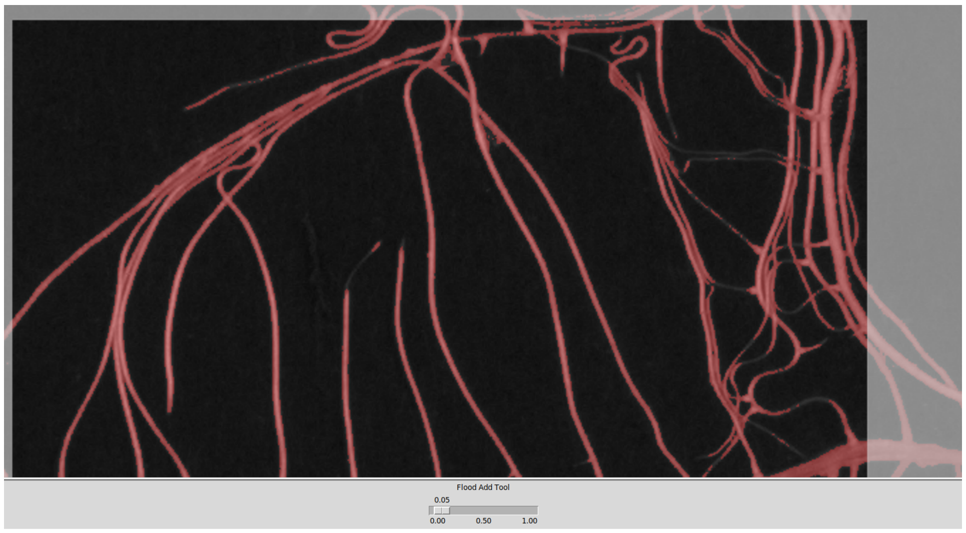


To remove the red area at the top, we could use the remove brush ('r'), or we can use the flood remove tool ('l') and click on the area to remove. After adjusting the tolerance (shown at the bottom) the only red part of the image is the root, and we can move on to the next patch.

5



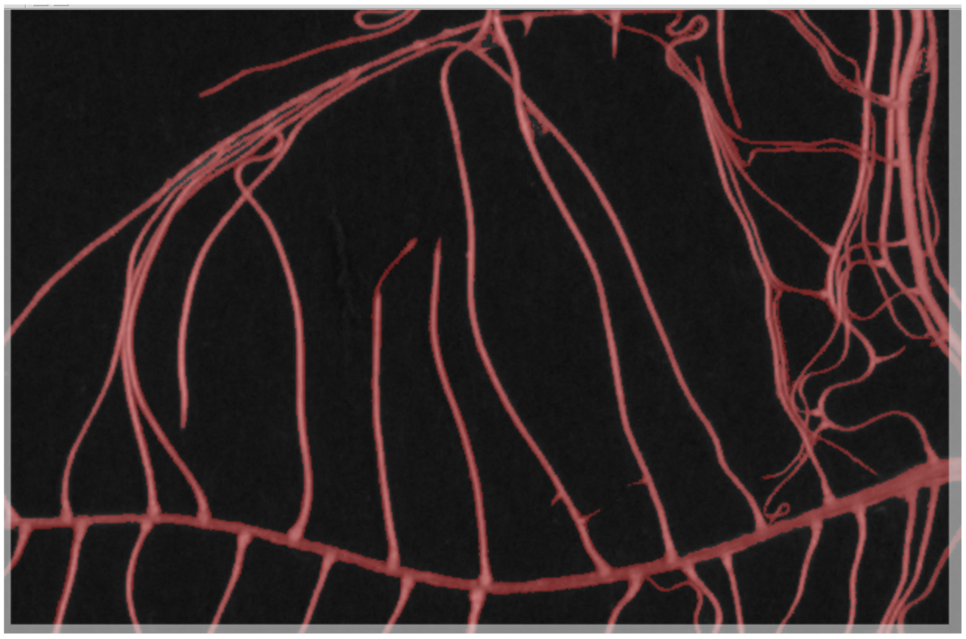
This next patch is a bit complicated:



In this case, we use the flood add tool ('f') to paint in the missing parts

6

of root, and then use the flood remove tool ('l') to remove the background parts from between roots that are very close together.



Sometimes, you can adjust the overall threshold ('t') to get all the root pixels, and then go in with the flood remove tool ('l') to cut out bits of background that were included by accident. Depending on the patch this might save you a bit of time, but if there are a lot of small spaces in the background, it is better to use the flood add tool ('f') to paint in the missing root parts.

3 When You're Done

When you hit the last patch in the image, a pop-up window will inform you that you have finished all the patches. You can use the preview window to review the whole image mask and navigate to any patches that might need additional annotation. If there are any parts of the root missing from the mask, or any large non-root objects that have been outlined, you can go back and fix those patches. Otherwise, select save and select the annotations folder in the package that you downloaded from OneDrive. The mask images will be saved there.

If necessary, you can use the 'Load Existing Mask' option in the file menu to load in a saved mask at a later time. First, load the original .tif image that you wish to edit the mask for, then select 'Load Existing Mask' from the file menu and select the .png mask file that corresponds with the image you chose. You can then edit and save the mask as you would normally.

Once you have completed all the images in the folder you were given, make a zip file from the annotations-xxx-x folder (where xxx-x are some numbers), and name it annotations-xxx-x_complete.zip. For example, if the folder was called annotations-002-3, I would create a zip file called annotations-002-3_complete.zip, containing both the images and the annotations folder. Upload this file to the OneDrive folder that you got the original files from.