Computational Science and Engineering

Technical University of Munich

Master's Thesis

Light Microscopy Image Analysis using Neural Networks

Jan Watter

Computational Science and Engineering

Technical University of Munich

Master's Thesis

Light Microscopy Image Analysis using Neural Networks

Author:	Jan Watter
Examiner:	Prof. Dr. Christian Mendl
Assistant advisor:	Dr. Felix Dietrich
Submission Date:	April 15th, 2021

Declaration for the master's thesis

Name:	Jan Watter
Student ID Number (Matrikelnummer):	03665485
Supervisor:	Dr. Felix Dietrich
Examiner:	Prof. Dr. Christian Mendl
Title:	Light Microscopy Image Analysis using Neural Networks

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

[illegible]

April 15th, 2021 Jan Watter

Acknowledgments

I want to thank Prof. Christian Mendl and Dr. Felix Dietrich for the possibility to write this master's thesis at the Chair of Scientific Computing in Computer Science. Felix always helped me out, whenever I had questions about my research or writing. Our discussions helped me to gain a better understanding in image processing, machine learning and 3D modeling. I would also like to thank Prof. Caroline Gutjahr and Dr. Catarina Cardoso from the Gutjahr Lab at the TUM School of Life Sciences for introducing me to their research in plant genetics. Without them, this collaboration would not have been possible. In addition, I want to thank Susi and Anthony for proofreading this thesis. This thesis marks the last step of my master's studies, which complements my academic education started in 2014. This journey was (almost) always supported and encouraged by my parents and grandparents, for which I am deeply grateful for. My fellow students Moritz, Max, Felix, Martin and Mihai were also always supportive. A special thanks to the Newmanhaus and its roommates, my friends from Munich and home.

Abstract

Arbuscular mycorrhizal is a symbiosis of soil fungi and vascular land plants. Current visual analysis methods of arbuscular mycorrhizal fungi (AMF) rely on staining of plant roots, imaging the samples and manual counting of fungal structures in microscopy images.

We developed a software pipeline that automates parts of this image analysis. Therefore, we trained a convolutional neural network (CNN) that generates semantic segmentations: it colors the pixels of the microscopy images according to their corresponding class. These classes represent the fungal structures arbuscules, vesicles and hyphae.

The idea of this thesis is to build a 3D model of a fungal-colonized root section. This allows us to procedurally generate synthetic training data that is used as input for the U-Net deep learning architecture.

Different data augmentations have been applied and several networks have been trained. The networks delivered high-confidence predictions on unseen data. In particular, a F1-score of 84% was achieved for the fungal objects. Future steps for this ongoing research project include to apply the findings of this thesis to real-world microscopy data. Topics like image similarity and transfer learning have been discussed and could be future applications of the trained networks.

Contents

List of Figures

List of Tables

1 Introduction

Arbuscular mycorrhiza (AM) is a mutualistic symbiotic relationship between soil fungi and most families of land plants, occurring in more than 85% of vascular land plants [46]. With regards to geographical distribution, it is probably one of the most common symbiosis on earth [23]. The AM fungus provides the plant with mineral nutrients, while it receives carbohydrates that the plant generates via photosynthesis. Structures similar to arbuscules were discovered in the earliest land plant fossils ($>$ 400 million years ago), indicating that AM have been in existence when plants colonized the land.

Research into cell and developmental biology is driven by the possibility of increased exploitation of the benefits of this symbiosis for sustainable agriculture. Furthermore, it has revealed insights into inter-organismic communication (fungi and land plants belong to different biological kingdoms) and into the plasticity of plant development.

Contemporary approaches of quantifying AM fungal colonization are based on visual diagnosis using microscopes or image scanners to create digital images. These methods rely on staining of the colonized roots, imaging of the samples, followed by manual scoring, done by experimenters. This includes simple and repetitive tasks prone to variation and human errors.

Computer vision is the scientific field that addresses the task of giving computers a high-level understanding of digital images. With the increase of computational power over the recent years, machine learning has enabled immense advances in data analysis and computer vision in particular.

A class of computational models that learn to extract information at several levels of abstractions from input data, is called deep learning. These methods are based on artificial neural network algorithms, which are inspired by the distributed communication nodes and information processing of biological systems [30].

In many computer vision tasks like image classification and image segmentation, convolutional neural networks (CNNs) have outperformed traditional image analysis methods. Typical CNNs consist of three types of processing layers: First, a convolutional layer that uses local receptive fields, called filters, to extract visual features from a group of neighbouring pixels. Following this, the resulting features maps are down-sampled to reduce the size of the features and thus reducing computational complexity, done by the pooling layer.

After several iterations through convolutional and pooling layers, a fully-connected layer follows. This layer maps higher-dimensional feature maps to a one-dimensional vector that allows for high-level decision making, like classification or further processing.

CNNs are responsible for many advances in domains such as diagnostic imaging [55], self-driving cars [5], face recognition [24] and also plant pathology [32].

Despite having performed well in many vision tasks, the neural networks rely heavily on big data. In order to generalize from seen data to unseen data, the neural networks require a lot of high-quality and often labeled training data. In many application domains, this represents a bottleneck and barrier for applying these methods, since there is no access to big data or generating new datasets is infeasible.

The following thesis makes use of a CNN-architecture called U-Net [41] to build a pipeline that detects fungal structures and classifies the corresponding pixels in the images. Due to the limited amount of real-world training data, we built a 3D model of a fungal-colonized plant section, using the computer graphics software blender. This allowed us to procedurally generate synthetic testing data.

The thesis is structured as follows: We start with an introduction to arbuscular mycorrhizal fungi (AMF) and outline how AMF probes are prepared and imaged (cf. 2.1). Subsequently, we describe the task of image segmentation within the context of computer vision and how machine learning techniques deal with this problem (cf. 2.3 and 2.4).

We continue to describe the theoretical background of how CNNs work (cf. 2.4.2) and discuss data augmentation techniques (cf. 2.4.3) and transfer learning (cf. 2.4.4) as methods to deal with poor data availability. Since capturing the similarity and comparing different images is a recurring theme in this thesis, we quickly discuss the topic of image similarity (cf. 2.5) and finish the chapter *State of the Art* (cf. 2) with related software for biological image processing (cf. 2.6).

The body of the thesis (cf. 3) starts with the technical environment of the machines that were used for network training and we list the utilized software (cf. 3.1). Next, the real-world microscopy data (cf. 3.2) and the generation of the synthetic training dataset (cf. 3.3) are discussed. Here, we show the simulation steps, that were implemented to model the biology of a colonized plant root using blender and the following pre-processing steps and data augmentations that were applied.

In the following, we describe the U-Net architecture and the implementation that was used and summarize some of its implementation details like evaluation metrics and loss functions, before showing training and evaluation results.

In the *Conclusion* chapter (cf. 4), we provide a brief summary and outlook about possible next steps for this ongoing research project.

2 State of the Art

This thesis emerged from a collaboration between the Professorship of Plant Genetics (*Gutjahr Lab*) of the TUM School of Life Sciences and the Chair of Scientific Computing in Computer Science at the TUM Department of Informatics. At the Gutjahr Lab interactions of plants with nutrient-delivering arbuscular mycorrhiza fungi are investigated, while the Chair of Scientific Computing uses advanced computing technologies to understand and solve research problems from many different domains.

The goal of this chapter is to explain the biological background of this research project and provide the theoretical background of the computational methods, that have been employed.

2.1 Arbuscular Mycorrhiza Fungi

A mycorrhiza is a mutualistic symbiotic relationship between a fungus and a vascular land plant. Arbuscular mycorrhizal (AM) is a type of this association, where an arbuscular mycorrhizal fungus (AMF) colonizes the host plant's root tissues.

Fungal hyphae, which are long, filamentous branches and are collectively called mycellium, grow inside the root and penetrate its cell walls to form arbuscules that become surrounded by the plasma membrane of the plant cell [6]. This results in a very invasive mutualistic relationship between the plant and the fungus. It is found in more than 85% of vascular plant families and is of huge importance for agriculture, ecosystem management and restoration [46].

The major function of these symbiotic relationships is nutrient acquisition. The arbuscules, which are highly branched, tree-shaped structures, create greater contact surface areas between the fungus and the plant, which facilitate greater nutrient transfer. The fungus aids this nutrient transfer with the uptake of phosphorus, nitrogen, water and amino acids. In exchange for providing these nutrients, the plant provides carbon, which it creates via photosynthesis [27].

The development process of AMF can be separated into distinct steps. In the precontact stage the symbiotic partners exchange signalling molecules to show mutual recognition. Next, the contacted plant cells form an intracellular accommodation structure that guides the fungus into deeper cell layers. Finally, the fungal hyphae progress longitudinally through the spaces between the cells of the root cortex. These hyphae form further branches to initiate arbuscle formations inside plant cells. Post-arbuscular development includes the differentiation of vesicles, which store oil-rich products.

While these differentiation steps follow a precise morphogenetic program, the hyphal network as a whole is not synchronized. Thus, the various types of fungal structures can occur simultaneously inside the plant root. A schematic visualization of AMF is shown in Figure 2.1.

Arbuscules, vesicles and hyphae are the objects, in descending order of importance, that shall be detected by the algorithm later on.

2.2 AMF Preparation and Imaging

Genetic manipulation of symbiotic fungi remains challenging and visual analysis of AMF colonization complements the molecular methods and gene sequencing knowledge.

We continue to summarize how the AMF samples were prepared and imaged: *Lotus japonicus ecotype Gifu wild-type* (plant-name) seeds were scarified and surface sterilized. The imbibed seeds were germinated for 10-14 days and cultivated in quartz sand as substrate. For colonization with *Rhizophagus irregularis* (fungus-name) the roots were inoculated with 500 spores per plant and harvested after five weeks after inoculation. To make the fungal structures visible under the microscope, they were stained according to the ink-vinegar-staining method described in [51].

Colonized root sections were then imaged by a Leica DM6 B wide-field microscope using a 10x magnification objective at 10-14 different focal planes across the root diameter.

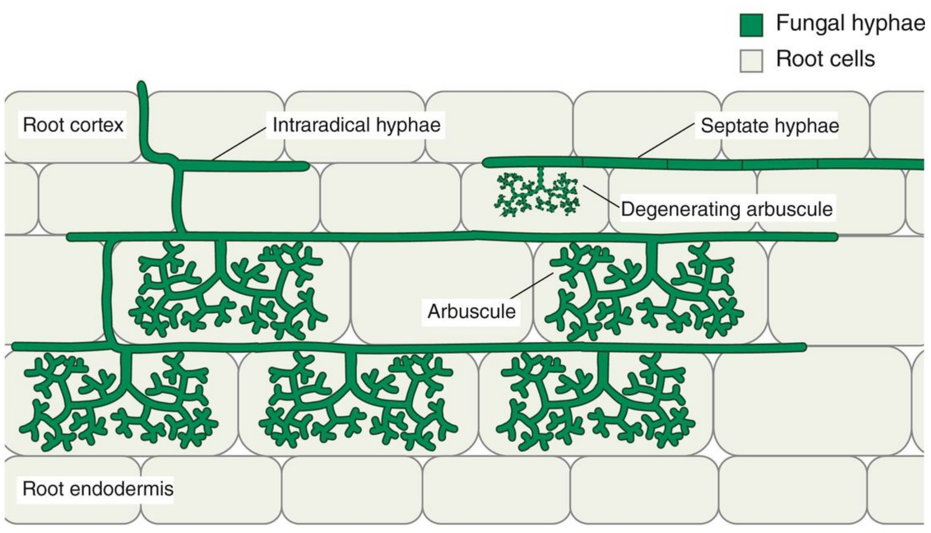


Figure 2.1: Visualization of AMF structures showing intracellular hyphae growing through the root cortex of the host plant. Degenerated and fully developed arbuscules are indicated. With permission from [29].

2.3 Computer Vision

Computer vision can be described as giving computers the ability to *mimic* or *simulate* human vision using sensors and algorithms. To illustrate the complexity of this task, consider Figure 2.2, where two representations of a grayscale image are shown. Note that both of these representations contain the same information but for a human observer it is challenging to find correspondence between them.

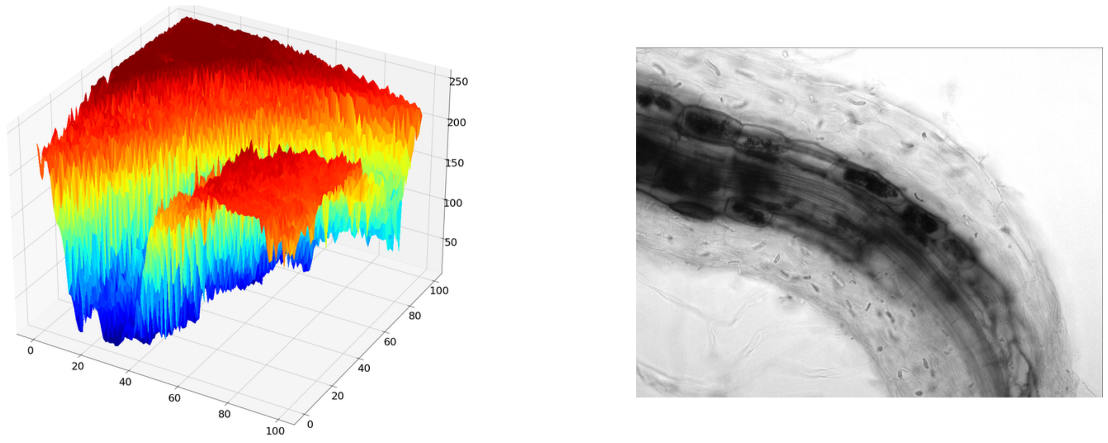


Figure 2.2: Two different representations of the same data. The microscopy image was taken by Catarina Cardoso.

The image representation on the left plots the brightness value of the corresponding pixel on the vertical axis. The image representation on the right is more common and a human observer can quickly identify what is shown.

Humans use a lot of a-priori knowledge to interpret images, while a machine begins with an array of numeric values and algorithms draw conclusions from that alone. In this context, machine learning can be viewed as the attempt to *teach* the machine some a-priori knowledge. Machine learning techniques *feed* the machine with many example data points, so it *learns* the underlying distribution of certain features of the image.

The task in this thesis is to implement an algorithm that finds the pixels that correspond to fungal structures in digital images.

In the field of computer vision this problem is called **semantic segmentation**. It refers to the process of assigning labels to specific regions of an image to simplify image analysis. It can be seen as a pixel-level classification task, where the goal is to partition the image into coherent regions. Applications range from computed tomography, where a tumor has to be located in a volumetric image and isolated from the rest of the image, to face recognition in smartphones [7].

Commonly, the machine learning approaches of object detection and image segmentation are classified into the following categories:

- **Image classification** has the goal of answering the question of what object is depicted in the image. Often, the aim is to find a bounding box that additionally localizes the object. Figure 2.3 visualizes that the algorithm correctly classified and localized the object cat.

- **Object detection** wants to determine where multiple objects are located in given images and how each object can be categorized. It can deal with multiple objects, while image classification deals with a single objects only.

- **Semantic segmentation** is a technique where each pixel shall be labeled (or colored) according to its corresponding class. The goal is to find the boundaries of the objects, in contrast to only draw bounding boxes surrounding the objects. It does not differentiate across instances of the same object.

- **Instance segmentation** works similar to semantic segmentation but classifies each instance as a separate entity. For example if there are two dogs in an image, semantic segmentation only assigns the label *Dog* to the corresponding pixels, while instance segmentation assigns *Dog 1* and *Dog 2* labels to the corresponding image regions.

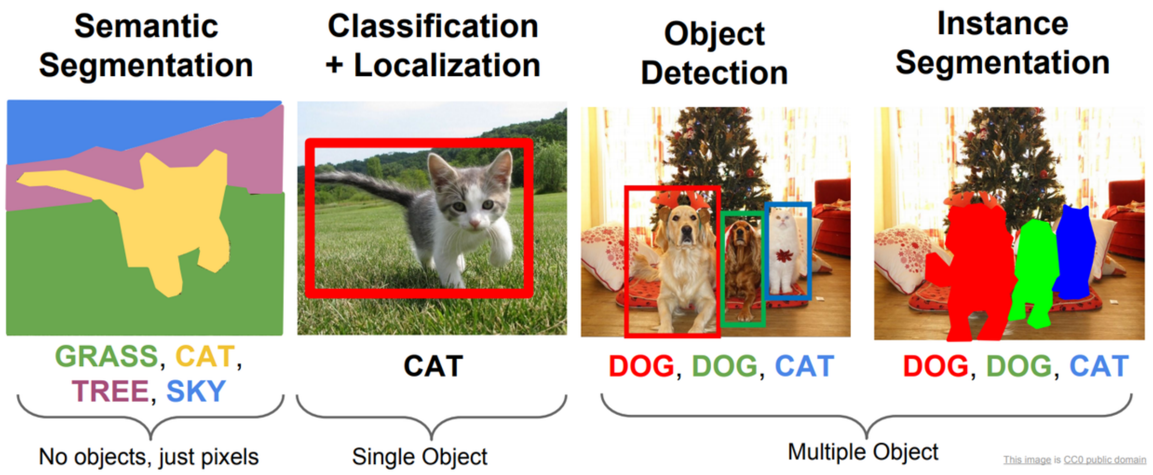


Figure 2.3: Computer vision: semantic segmentation, classification, object detection and instance segmentation. Taken from [19].

Machine learning techniques, especially deep learning and convolutional neural networks have superseded early image processing methods that rely on low-level vision cues in automated image processing [3].

2.4 Machine Learning

Machine learning can be defined in many different ways. We propose to use the following definition from [33]:

> *"Machine learning can be broadly defined as computational methods using experience to improve performance or to make predictions. Here, experience refers to the past information available to the learner, which typically takes the form of electronic data collected and made available for analysis. This data could be in the form of digitized human-labeled training sets, or other types of information obtained via interaction with the environment."*

Artificial neural networks (ANNs) are a class of machine learning algorithms that are inspired by the structure and function of biological nervous systems like the human brain. They consist of a high number of linked computational nodes, referred to as neurons, which change their weights in order to optimize a certain cost function.

The basic structure of ANNs is shown in 2.4. Commonly, a multidimensional vector is loaded as input, which then gets distributed to the hidden layers. The process of adjusting the weights, in order to optimize a cost function, is referred to as learning. A system compromised of multiple, stacked hidden layers falls in the category of **deep learning** networks.

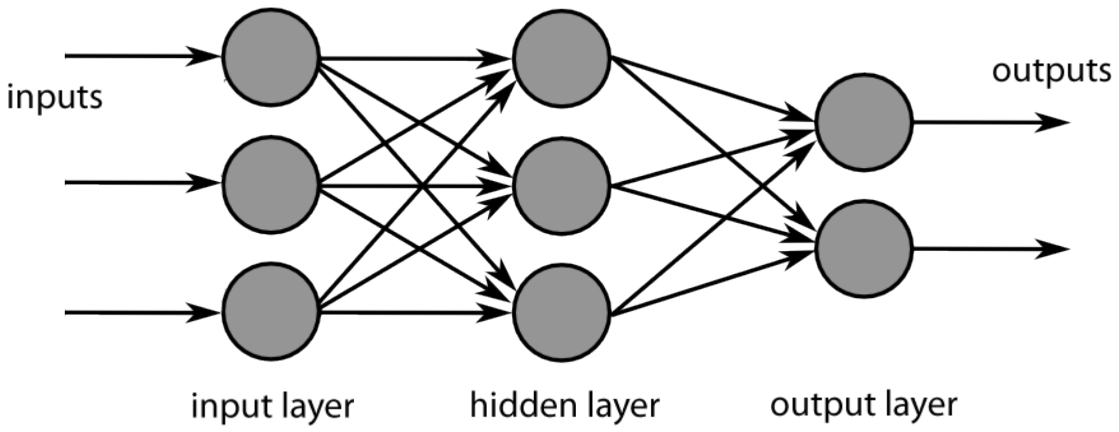


Figure 2.4: A three-layered feedforward neural network composed of an input, a hidden and an output layer. The information moves only in one direction from the input nodes through the hidden nodes to the output nodes. Taken from [9], licensed under CC BY-SA 3.0, author: Christoph Burgmer.

2.4.1 Supervised Learning

In the supervised learning paradigm, the goal is to approximate a function called classifier, $f : X \to Y$, from a training dataset T_n, which is composed of input-output pairs (x_i, y_i), where $x_i \in X$ and $y_i \in Y$:

$$T_n = \{(x_1, y_1), \ldots, (x_n, y_n)\} \in (X \times Y)^n.$$

X describes some feature set, usually $X \subset \mathbb{R}^d$ and $Y \subset \mathbb{N}$ is discrete for classification problems.

In semantic segmentation, the training dataset T_n consists of RGB-images and Y is the set of labeled (or annotated or output) images. To be more precise, the goal of the neural network (NN) is to map each pixel from the RGB-color space, which usually contains 256 values in the Red, Green and Blue channel, to the *label-space*, which is made up of consecutive integers, depending on how many objects shall be classified.

In this framework of statistical learning, there are two fundamental concepts:

1. The training data is identically and independently generated from a fixed but unknown joint probability distribution function $P(x, y)$. The goal of the learning process is to find the function f that attempts to model the dependency encoded in $P(x, y)$ between the input x and the output y.

2. To evaluate the agreement between the prediction $f(x)$ and the ground-truth y, a loss or cost function $L(f(x), y)$ is introduced. Loss functions are classified according to their regularity or singularity properties and according to their ability to produce convex or non-convex criteria for optimization [16]. The loss functions that are commonly used in semantic segmentation tasks are explained in section 3.4.5.

2.4.2 Convolutional Neural Networks (CNNs)

The implementation we utilized to perform semantic segmentation is based on convolutional neural networks (CNNs), which are powerful image classifiers. The main difference to ANNs is that CNNs try to encode image specific features into the architecture, making the network more suited for image-focused tasks.

The input vector consists of the pixel intensity values of an image. Consequently, a series of hidden layers is used to extract features from the input image. Each layer in a CNN applies a different set of filters by convolution and then combines the result by addition, feeding the output into the next layer of the network. During training, the CNN automatically learns the values of these filters. The last layer in a CNN uses these higher-level features to make predictions, regarding the contents of the image.

The CNN structure we used in this thesis, is called U-Net [41] and consists of convolutional, activation, max-pooling and fully-connected layers. We continue to explain the functionality of these layers.

- The **input layer** represents the image data itself, specifically the pixel values of the image. When dealing with RGB images the number of channels is three, one for each color. The dimension of channels is often referred to as depth (for RGB images: depth = 3, for grayscale images: depth = 1).

- **Convolutional layers** are the core building block of CNNs. They consist of a set of k filters (also called kernels), which extract features from the input image. When the data is fed into a convolutional layer, each filter is *slid* across the image to produce a 2D *activation* or *feature map*.

  In mathematical terms, an element-wise multiplication (dot product) between a filter-sized patch of the input image and the filter is applied. By doing this, the convolutional layers activate different, local parts of the image and map certain features of the image to a new layer.

  Each kernel produces an activation map which will be stacked along the depth dimension and is the input for the next layer. This concatenation in conjunction with the following layers results in a hierarchical representation of the image. As [28] points out:

  > *"Pixels are assembled into edglets, edglets into motifs, motifs into parts, parts into objects, and objects into scenes."*

  The following mathematical expressions for this chapter are taken from [37]: the output $Y_i^{(l)}$ of layer l consists of $m_1^{(l)}$ feature maps of size $m_2^{(l)} \times m_3^{(l)}$. A feature map Y_i is computed as

  $$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)} \tag{2.1}$$

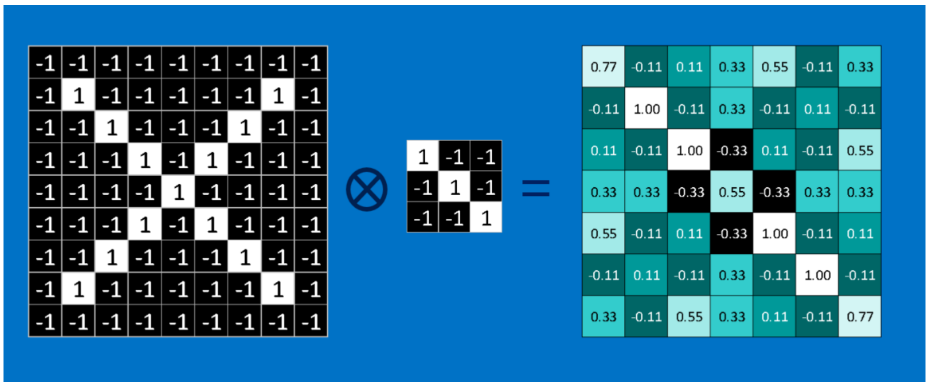
  where $B_i^{(l)}$ is a bias matrix and $K_{i,j}^{(l)}$ is the filter with a specified size connecting the j^{th} feature map in layer $(l-1)$ with the i^{th} feature map in layer (l). The operation $(I * K)$ describes a discrete 2D convolution. For a given filter $K \in \mathbb{R}^{2h_1+1\times 2h_2+1}$ the discrete 2D convolution of Image I with kernel K is given by

  $$(I * K)_{r,s} := \sum_{u=-h_1}^{h_1} \sum_{u=-h_2}^{h_2} K_{u,v} I_{r-u,s-v} \tag{2.2}$$

  with filter K defined as

  $$K = \begin{bmatrix} K_{-h_1,-h_2} & \cdots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \cdots & K_{h_1,h_2} \end{bmatrix}.$$

The resulting feature map provides information on how well the local filter fits the patch of the image. Figure 2.5 shows an image which gets convolved with a filter from top left to bottom right and the resulting feature map.



$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \\ -0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\ 0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\ 0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\ 0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\ -0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\ 0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77 \end{bmatrix}$$

Figure 2.5: Discrete 2D convolution. For simplification the image pixel-values and filter weights are chosen with values of $-1, 1$. Taken from [40], licensed under CC0 1.0 Universal (CC0 1.0).

- An **activation layer** introduces a non-linearity to the network, so that it is able to learn more non-linear patterns from the data. It consists of an activation function that takes a feature map, generated by the convolutional layer, and creates an activation map as its output.
  Since the activation function is an element-wise operation over the input volume, the dimensions of the input and the output are identical. Expressed in mathematical notation: the activation function is a function of the feature data $Y_i^{(l-1)}$ from the convolutional layer $l-1$ and generates the activation volume $Y_i^{(l)}$:

$$Y_i^{(l)} = f(Y_i^{(l-1)}) \tag{2.3}$$

where,

$$\begin{gathered} Y_i^{(l)} \in \mathbb{R}^{m_1^{(l)} \times m_2^{(l)} \times m_3^{(l)}} \\ Y_i^{(l-1)} \in \mathbb{R}^{m_1^{(l-1)} \times m_2^{(l-1)} \times m_3^{(l-1)}} \\ m_1^{(l)} = m_1^{(l-1)} \wedge m_2^{(l)} = m_2^{(l-1)} \wedge m_3^{(l)} = m_3^{(l-1)} \end{gathered}$$

with $m_1^{(l-1)}$ 2D feature maps generated by the same number of filters in convolutional layer $(l-1)$, each with size $m_2^{(l-1)} \times m_3^{(l-1)}$. Common activation functions are sigmoid, hyperbolic tangent functions or (leaky) rectified linear units (ReLUs).

- The **pooling layer** performs downsampling along the spatial dimensionality of the input data. It has two hyperparameters:[1] the spatial extent of the filter $F^{(l)}$ and the stride $S^{(l)}$.
  The input of a pooling layer is a feature volume of size $m_1^{(l-1)} \times m_2^{(l-1)} \times m_3^{(l-1)}$ and the output is a volume of size $m_1^{(l)} \times m_2^{(l)} \times m_3^{(l)}$ with

$$\begin{aligned} m_1^{(l)} &= m_1^{(l-1)} \\ m_2^{(l)} &= \left(m_2^{(l-1)} - F^{(l)}\right)/S^{(l)} + 1 \\ m_3^{(l)} &= \left(m_3^{(l-1)} - F^{(l)}\right)/S^{(l)} + 1 \end{aligned}$$

  Since feature detection is more relevant, compared to the features exact location, the pooling operation aims to preserve detected features as more compact representations. It discards less significant data at the cost of spatial resolution. Put differently, it reduces the number of parameters and the computational complexity of the network.

- The **fully-connected layer** in a CNN maps the resulting activation map from the previous layers to a class probability distribution. Mathematically this can be expressed as

$$y_i^{(l)} = f\left(z_i^{(l)}\right) \quad \text{with} \quad z_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} w_{i,j}^{(l)} y_i^{(l-1)} \tag{2.4}$$

  The goal of this layer is to train the weight parameters $w_{i,j}^{(l)}$ to create a stochastic likelihood representation of each class, based on the concatenation of previous layers. Similar to the activation layer, the function f is a non-linear function.

To sum up: CNNs use weighted, sparsely connected layers, which preserve the spatial characteristics of the images. Convolutional and pooling layers downsample the resolution of images while expanding the depth of their feature maps. This can generate lower-dimensional and more useful representations of the high-dimensional inputs. The success of CNNs has led to high interest and optimism in implementing deep learning architectures to computer vision tasks.

[1]Hyperparameters are parameters which are set before training and specify some aspect of the network architecture or training process.

2.4.3 Data Augmentation

Many scientific fields hope to improve current benchmarks by using deep CNNs for computer vision tasks. One of the most challenging aspects regarding the training of NNs, is to improve their ability to generalize from seen data (training data) to unseen data (test data).

Overfitting the training data is a main reason for poor generalization ability. It can be identified when the validation error starts to increase, while the training error continues to decrease during network training. Data augmentation is a powerful technique to reduce overfitting. Others include using more sophisticated NN-architectures, batch-normalization and transfer learning.

Data augmentation works under the assumption that the training dataset can be artificially increased and thus more information can be extracted by the NN.

Before discussing augmentation techniques it might be useful to mention some of the challenges that the network encounters in image segmentation tasks: different lightning conditions, noisy data, rotations, shape deformations, background objects and different scales. The task of data augmentation is to capture these variances of the data and apply them to the training dataset, so that the network generalizes better.

Data augmentation techniques can be classified into the following two, not mutually exclusive, categories:

1. **Data warping** augmentations transform the input images such that their label is preserved. Examples are color transformations, geometric transformations, erasing parts of the image, adversarial training and neural style transfer.

2. **Oversampling** techniques create new, synthetic image instances and add them to the training dataset. Examples include mixing images, feature space augmentations and generative adversarial networks.

In this thesis only data warping augmentations have been used, as presented in chapter 3.3.2. For a comprehensive survey on data augmentations in deep learning, it is referred to [44].

2.4.4 Transfer Learning

Many ML algorithms perform well under the assumption that training and test data are drawn from the same distribution. However, if the distribution changes they need to be rebuilt from scratch. This entails the need for collecting new training data and in many real-world applications this can be expensive or impossible. In such cases it would be useful to reduce the need and effort to recollect training data and have a knowledge transfer between different domains.

Semi-supervised classification can be seen as a kind of precursor of transfer learning. It is used if there is too few labeled data, to build a good classifier, available by utilizing a large amount of unlabeled data and a small amount of labeled data. Variations of supervised and semi-supervised learning paradigms have been studied [57] but still assume that the underlying distribution of labeled and unlabeled data is the same.

In contrast to this, transfer learning allows the distributions, tasks and domains used in training and test data to be different. It aims to extract knowledge from one or more source tasks and applies this knowledge to a novel target task. This is different to **multitask learning**, where all of the source and target tasks are learned simultaneously, since transfer learning cares more about the target task.

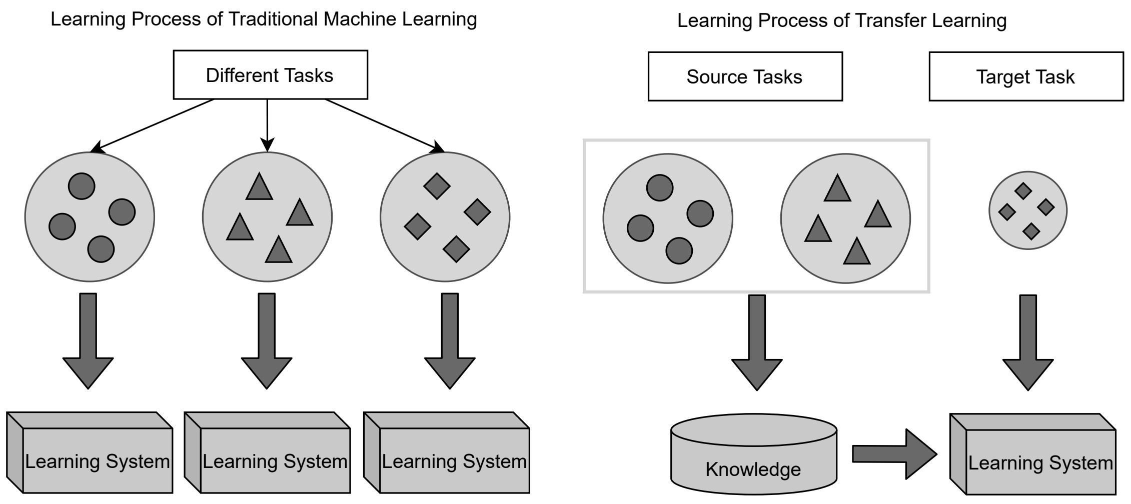


Figure 2.6: Different ML learning techniques. On the left-hand side: traditional ML, on the right-hand side: transfer learning. Based on [35] Figure 1.

The different learning processes of traditional ML algorithms and transfer learning is seen in Figure 2.6. While the traditional ML algorithms learn each new task from scratch, transfer learning algorithms transfer knowledge from previous tasks to a target task, which has fewer training data available.

For a mathematical definition, we use following definitions similar to 2.4.1 but differentiate between feature space $\mathcal{X}$ and feature set X:

Domain $\mathcal{D}$ consists of feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, with $X = x_1, ..., x_n \in \mathcal{X}$. Generally speaking, two domains are different, when they have different feature spaces or marginal probability distributions.

For a specific domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task consists of label space $\mathcal{Y}$ and a classifier $f(\cdot)$, which can be learned from training data consisting of input-output pairs x_i, y_i with $x_i \in X$ and $y_i \in Y$. This task is denoted by $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$. Following this, a unified **definition of transfer learning** reads as follows [44]:

> *"Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$."*

We did not employ transfer learning in this work but a potential application could be to use our model as a source task and apply transfer learning to a target task, like the segmentation of a real-world AMF light-microscopy image dataset.

2.5 Image Similarity

Image similarity occurs in different contexts in this thesis:

- The NN evaluates the similarity between its prediction and the groundtruth and adjusts its weights to maximize similarity (or minimize its loss function) accordingly. Evaluation metrics and loss functions for semantic segmentation are discussed in detail in section 3.4.

- The computer generated training data should resemble the real-world microscopy images as close as possible. The microscopy images show a lot of variability as seen and discussed in section 3.2. We do not know the underlying distribution of this data. One could argue that the NN tries to approximate this distribution during its training.
  In section 3.3.1 we explain the modeling of the 3D root model, and show the measures that were applied to generate realistic renderings.

In this section, we want to outline some theoretical aspects of image similarity.

It is a non-trivial task and active research field in computer vision. Here, the term similarity measure is used as general term both for similarity measures which reach their maximum when $A = B$ and dissimilarity or distance measures which reach their minimum when $A = B$, for images A and B.

A quantitative image similarity measure between two images A and B has two components: Firstly, a transformation T which extracts the characteristics of an input and converts it to a multi-dimensional feature vector. Secondly, a distance measure D which quantifies the similarity between two images, where D is defined in the multi-dimensional features space. [31].

$$S(A, B) = D(T(A), T(B)) \tag{2.5}$$

A similarity measure or distance is considered a metric when it satisfies the following:

1. Nonnegativity: $S(A, B) \geq 0$,
2. Reflexivity: $S(A, B) = 0$,
3. Symmetry: $S(A, B) = S(B, A)$,
4. Triangle Inequality: $S(A, C) + S(C, B) \leq S(A, B)$.

Although the properties of having a metric are desirable, a similarity measure can be effective without being a metric. Similarity or distance measures that are invariant to sensor parameters or insensitive to radiometric changes are often not metrics [22]. For example measures that are formulated as joint probability distribution of image intensities are not metrics but can be effective in comparing images captured by different sensors.

Similarity measures can be broadly divided into two, non-mutually exclusive, groups:

- **Global Measures:** Return a single value describing the overall similarity between the two input images. Furthermore, they can be divided into measures which require spatially registration of the input images and those which do not require the images to be spatially registered.
- **Local Measures:** Return a similarity image or map which characterizes the local similarity between the two input images. These measures need the input images to be spatially registered.

Spatial image registration describes the process of transforming different types of image data into the same coordinate system.

2.6 Related Software for Biological Image Processing

Recent progress in biological imaging has resulted in an increase of quantity and quality of digital biological images [39]. With larger datasets, automated image analysis and high-throughput methods become necessary, resulting in new tools developed in collaborations between computer scientists, biologists, mathematicians and physicists. In this section we briefly layout some of these tools.

- **ImageJ (Fiji)** is an open-source software, designed for analyzing scientific multidimensional images [2]. Fiji refers to a distribution of ImageJ, where many plugins are pre-installed. It remains a widely used application for image processing and data analysis when dealing with biological image data.

- **Ilastik** is an open-source tool for interactive image classification, segmentation and analysis leveraging machine learning algorithms [47]. The labels of the objects of interest have to be assigned manually. This process slows down an automated pipeline but is very user-friendly as no machine learning expertise or previous image processing experience is required.

- **Cellprofiler** is an open-source software designed to help scientists identify and measure biological entities, process images and export data for further analysis [10]. The users can create their own pipeline and use a number of algorithms to perform cell segmentation.

- **PlantSeg** is an open-source pipeline for volumetric segmentation of plant tissues. It employs a CNN to predict cell boundaries and graph partitioning to segment cells based on the neural network predictions [53]. Since we used the NN implementation of PlantSeg, it will be discussed in section 3.5.

- **AMFinder** is an open-source software which was released during the time of the research of this thesis and is described as

  > "*AMFinder allows for automatic computer vision-based identification and quantification of AM fungal colonization and intraradical hyphal structures on ink-stained root images using convolutional neural networks* [18].[2]"

  Evidently, it tackles the same task while using similar techniques when compared to our approach. However, there are a few differences, for example the prediction pipeline is built hierarchical and semi-supervised.
  During the pre-processing stage the images are divided into tiles before the first round of prediction follows. A CNN classifies the tiles into the mutually exclusive classes *'colonized root section'*, *'non-colonized root section'*, and *'background / not a root / other'*. Upon user request AMFinder can proceed with a more refined analysis of the tiles classified as colonized root section, where a second CNN predicts the presence of arbuscules, vesicles, hyphopodia, and intraradical hyphae. The user can specifically review tiles with low-confidence predictions.
  In contrast, our network assigns each pixel in the image to a class resembling arbuscules, vesicles and (depending on the trained network) hyphae. Therefore, a more refined per-image analysis is possible. As a further research project, a comparison between the two tools could be carried out.

[2] *ClearSee*, which is an optical clearing reagent, was used during the fungal staining process as a contrast enhancer for the training dataset. High contrasted fungal structures mean a high signal for the NN thus ClearSee turned out to be very useful for the following image analysis and network training.

3 Light Microscopy Image Analysis using Neural Networks

This chapter describes the development of the render pipeline, the training of the neural network and the following results.

We start by outlining the technical environment of the machines used for network training, utilized software and details of the microscopy data. Following this, we discuss the simulation pipeline, that models the biology and renders the computer-generated training data in detail.

Before the data can be fed into the network, pre-processing steps had to be implemented and data augmentations were applied, which are discussed. Next, we present the network architecture and some of the implementation details like evaluation metrics and loss functions of the software. Finally, we show training results and evaluate the performance of the network.

3.1 Technical Environment and Utilized Software

Runtime results have been gathered on a local machine and on the *atsccs68* compute node in Garching. The render pipeline, as well as the pre-processing pipeline have been employed on the local machine while the training of the NN was done on the atsccs68. The technical properties of the machines are listed in table 3.1.

The data pre-processing pipeline, as well as post-processing and evaluation were implemented in Python3 [50] using libraries OpenCV [8], h5py [12] and scikit-image [49] among others. Image processing software ImageJ (Fiji), mentioned in 2.6, was used for initial image analysis and the NN was trained using the U-Net implementation of PlantSeg, (cf 2.6). The U-Net is implemented using the Pytorch library [36]. The conda package manager [1] was used to create separate Python environments to resolve dependency issues between different libraries and library versions. For enabling network training on the GPUs, the CUDA toolkit [34] was utilized.

For building the 3D model of the fungus-colonized root section, the 3D computer graphics software blender [13] was used.

	atsccs68-machine	local machine
CPU	i7-3770 @ 3.40GHz	i5-7500 @ 3.40GHz
Threads/core	2	1
Cores/socket	4	4
Socket	1	1
RAM	16 GB	16 GB
GPU	GP102 TITAN Xp	GeForce GTX 1660 SUPER
OS	Ubuntu 16.04.7 LTS	Manjaro Linux
Kernel	Linux 4.15.0-139-generic x86-64	Linux 5.4.89-1-MANJARO x86-64

Table 3.1: Technical properties of the atsccs68 compute node, which was used to train the networks and the local machine, which was used to generate the synthetic image training data using blender.

3.2 Microscopy Data

We continue to discuss some of the technical details of the microscopy data, as well as various aspects of the data variability:

- Different phenotypes of arbuscules (wild-type and mutant type)
- Different lightning levels
- Different magnification levels
- Possibly different types of staining and thus different contrast and saturation levels of fungal structures

For this thesis, the distinction between wild-type and mutant-type is not of relevance, since the 3D model only renders one type of arbuscules, resembling wild-type (or mature) arbuscules. However, the mutant-type is relevant for AMF-research and will be of importance in future research projects. The ink-staining gives the fungal structures their bluish appearance. Depending on this staining and on the person that performs the imaging, the lightning levels can differ. Furthermore, the magnification levels can differ, although only a 10x magnification objective has been used so far.

Current research like PlantSeg [53] shows that the performance of CNNs in segmentation tasks is sensitive to changes in pixel (or voxel) size. This implies that using different levels of scaling in the training data can be helpful. Finally, different staining methods could be used in the future, for example ClearSee as contrast enhancer as described in 2.6.

The relevant image properties that are constant across the different imaging rounds are listed in table 3.4. A complete list of the metadata, including the detailed microscope settings, can be exported using ImageJ.

File format	File size	Pixel type	Bits per pixel	Dimensions
.tif or .lif (Leica Image File)	~100MB	uint8	8	2048x1536 10-14 Focal Planes 3 Channel (RGB)

Table 3.2: Image properties of the microscopy data taken by a Leica DM6 B wide-field microscope using a 10x magnification objective

Additionally, the plant biologists provide annotations of arbuscules (for future research also a distinction between wild-type and mutant-type arbuscules is designated), vesicles and hyphae on the corresponding z-stack. As annotating the complete hyphal network would be infeasible and hyphae are of less relevance, they are marked by coarser annotations. Figure 3.1 shows an image of a colonized root section together with the annotations from the plant biologists. Furthermore, we show some more plots to illustrate the variability of the data in figure 3.2.

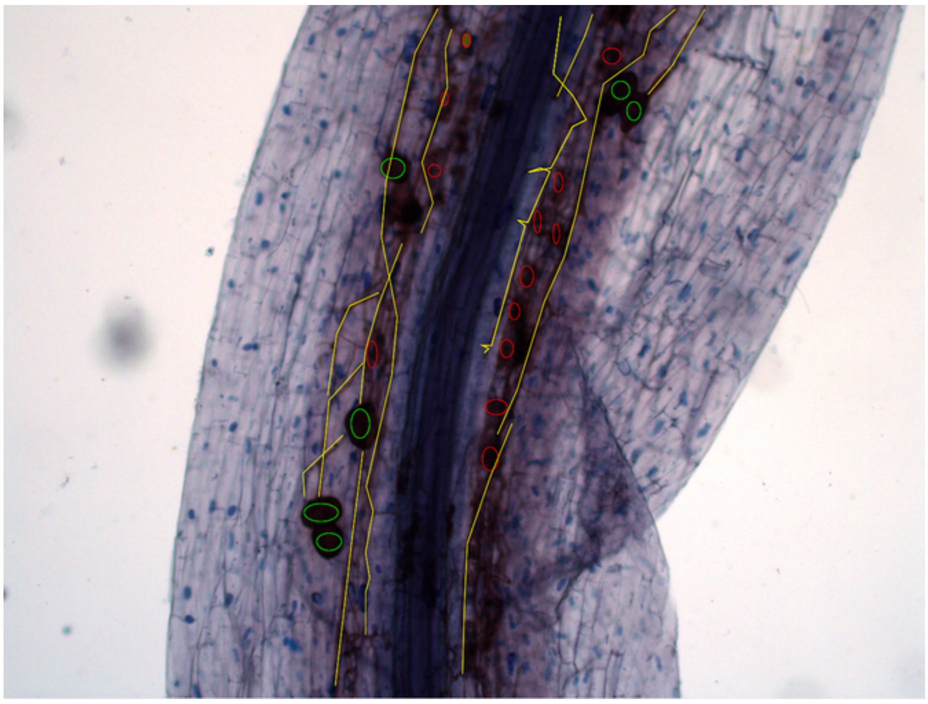


Figure 3.1: Annotated colonized root section. The red circles represent arbuscules, the green circles represent vesicles and some of the hyphal structures are annotated as yellow lines.

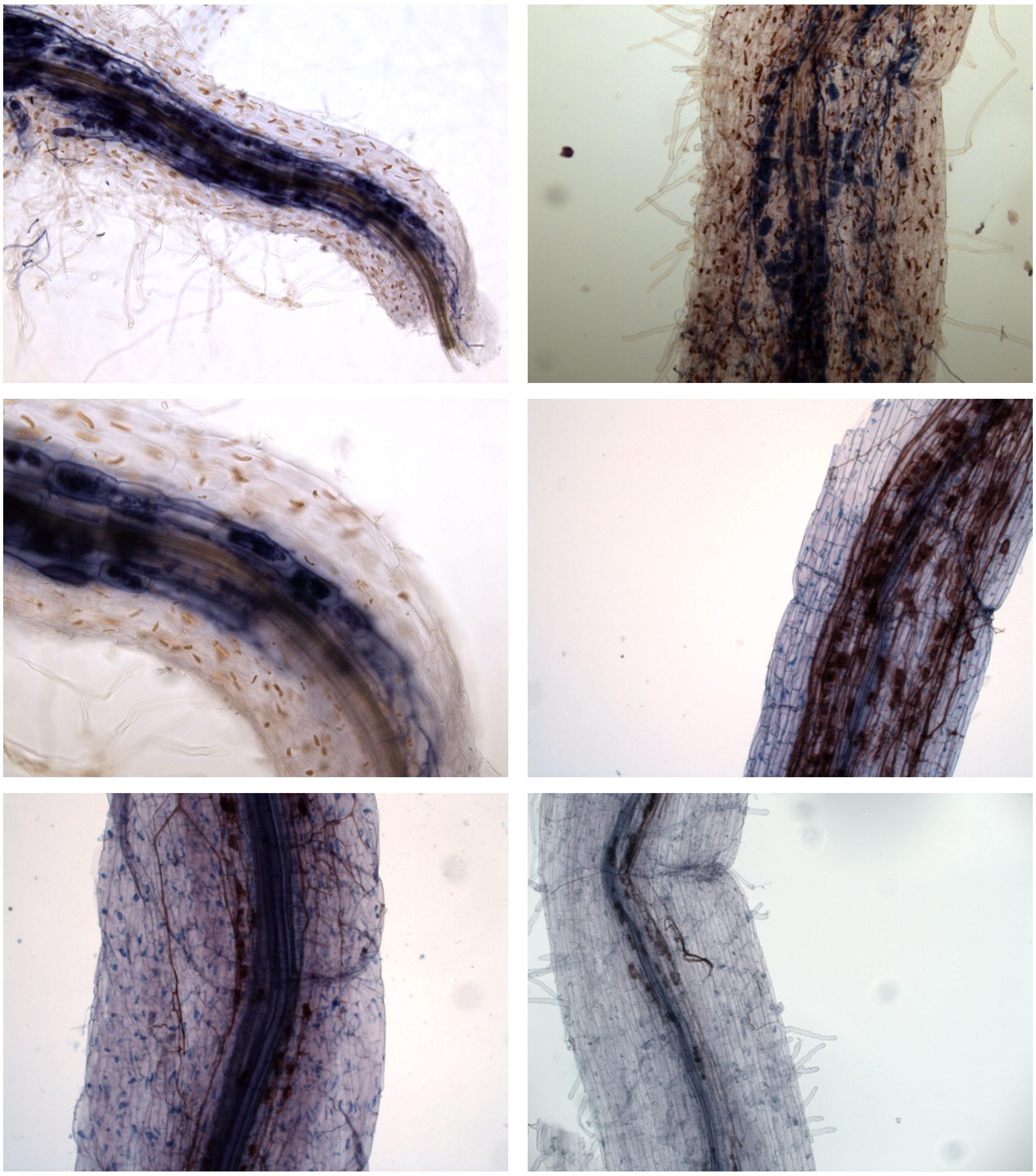


Figure 3.2: Six example microscopy images from three different imaging rounds, showing the variability of the data.

3.3 Data Pipeline

The training data is generated using the 3D computer graphics software blender [13]. We built a 3D model of a fungus-colonized root section, which allows us to render computer generated training data.

Generally speaking, modelling in 3D is a more complex task but has several advantages compared to modeling in 2D. Often in 2D, modeling several models are needed, for example for different views. In 3D, one scene can be sufficient to represent the complete model. It allows for more flexibility in the rendering process when changing camera perspectives and lightning conditions.

In our context, 3D modeling has the advantage of directly mapping the plant anatomy, like the proportions of the internal structures (root, arbuscules and vesicles and the surrounding cell morphology), to a geometrical model. In addition, this allows to render the model at different focal points (or *depths*).

In the microscopy images, fungal or plant objects can occlude each other when they are located at the same X- and Y-positions but different Z-positions. This is why the roots are imaged at different focal points. In a 3D model a rendering at different depths is possible, whereas in a 2D model there is no depth by definition.

Another general, non-technical advantage of 3D models over 2D models might be better communication of complex concepts between different research fields and maybe even to broader audiences.

3.3.1 Simulation Pipeline Using Blender

The generation of the synthetic training data follows the common 3D simulation pipeline: object modeling, rigging, shading, compositing and rendering. Furthermore, the blender Python API was used to procedurally create new poses of the plant root and generate renders automatically.

Object modeling

The main idea behind the modeling of the plant root anatomy is to build a polygon representing the shape or geometry of a single cell, stacking these polygons on top of each other and adding modifiers to attach them to Bezier curves, which correspond to the pose of the root as a whole. Figures 3.4a and 3.4b show this process.

At this point, it seems useful to quickly outline the basic plant root anatomy that shall be modeled. Figure 3.3 schematically shows the cross-section of a dicot root. We divided the model into the groups (called collections in blender) **root core**, **root shell**, **outer cell layer** and **root hairs**.

The **root core** collection consists of 3 different cell types and corresponds to the xlem and phloem of the plant, which make up the stele and the central part of the root. Outside the stele lies the endodermis, which is the innermost cell layer of the cortex of the plant. In our 3D model this, together with the pericycle, which is the outermost layer of the stele, is represented by the **root shell** collection.

The **outer cell layer** collection represents the epidermis, which is a single layer of cells that forms the boundary between plant and the external environment. A visualization of blender is shown in 3.4d. The **root hairs** collection represents the root hairs of the plant evidently.

The fungal structures, which are the elements that shall be detected and segmented by the NN in a later stage, are modeled using a similar technique. A screenshot of a blender 3D viewport is shown in 3.4c.

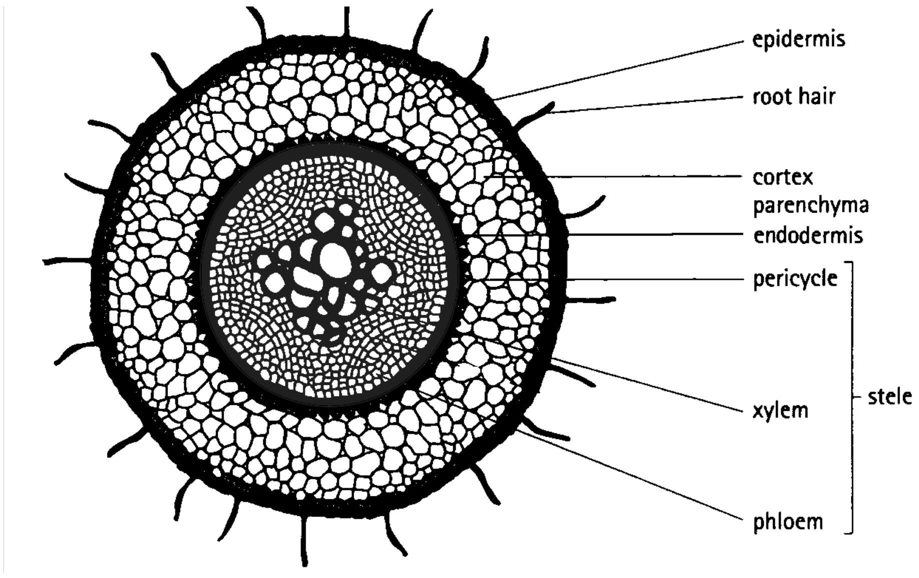
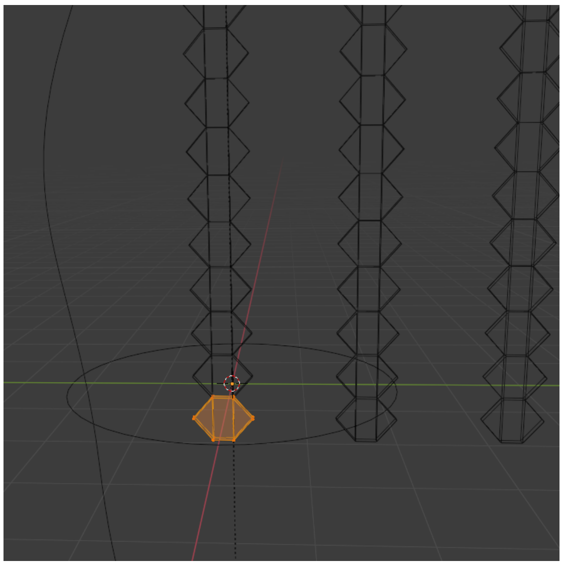
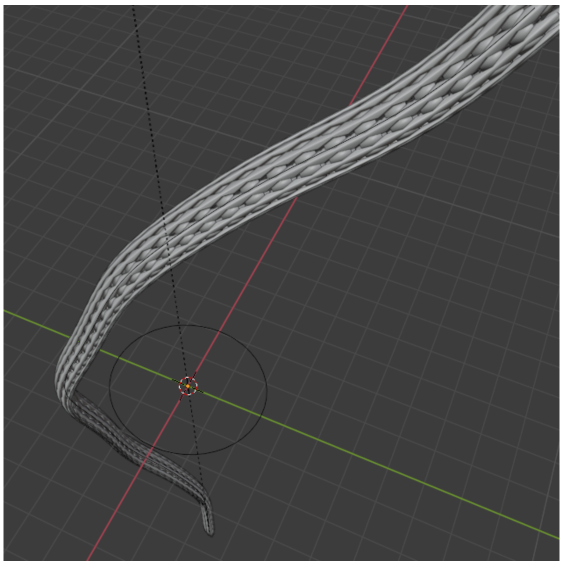


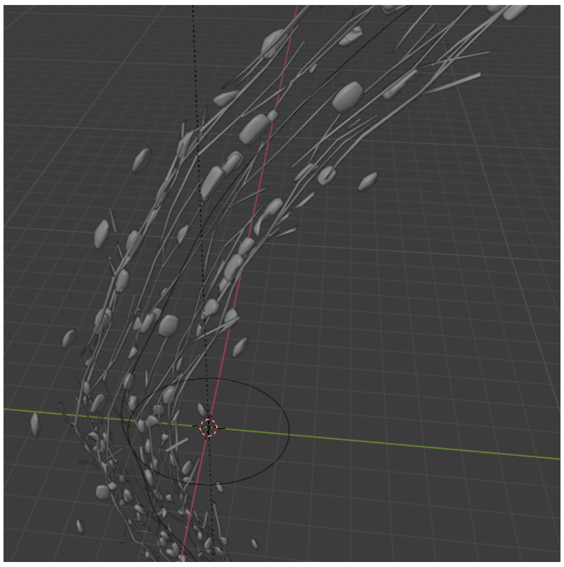
Figure 3.3: Modified schematic visualization of a cross-section of a dicot root taken from [45], licensed under CC BY 3.0.



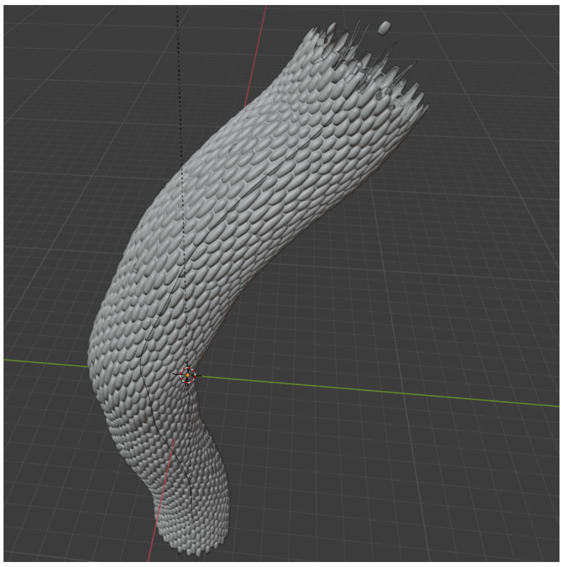
(a) A single cell of the root core is selected (in orange) and then duplicated. A circular and a longitudinal Bezier curve which gets *wrapped around* by these cells is shown as well.



(b) The resulting geometric structure, consisting of three different cell types, representing the root core.



(c) Fungal structures: arbuscules, vesicles and hyphae modeled in blender, displayed in object mode.



(d) Modeling of the epidermis, the outermost cell layer.

Figure 3.4: Screenshots of the blender 3D Viewport showing different objects from the 3D scene. (a) is rendered in wireframe shading, while (b)-(d) are rendered in solid-view.

Rigging and Scripting

Rigging or skeletal animation in computer graphics can be described as assigning relationships between objects so that applying a transformation to one object will induce a transformation in the other object. So far, the model consists of a surface representation, often called mesh, only.

Using rigging techniques we add a set of (interconnected) parts, called bones (or collectively: skeleton or rig), which can animate or deform the 3D mesh. This virtual armature is needed so we can generate new poses, meaning new mesh deformations.

This procedural generation of new configurations is done using the blender Python API. We wrote a Python script that will create a number of configuration files and run the blender rendering on each of them. Through a random number generator, the bones and in turn the whole mesh of the model, will be slightly translated, rotated and scaled for each rendering process. Furthermore, file-export options can be set.

In the future, functions that allow to manually adjust the number of arbuscules, vesicles and hyphae could be implemented. The armature is shown in Figure 3.5.

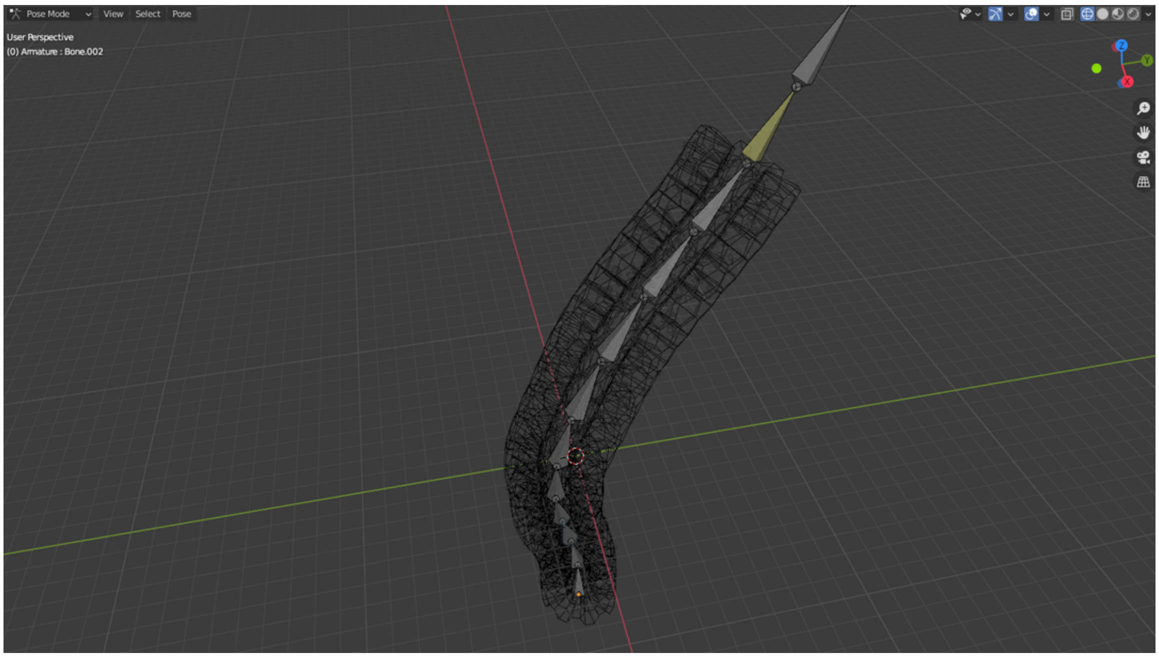


Figure 3.5: Armature and part of the mesh displayed as wireframe.

Shading

Shading is the process of computing the intensities and colors of a model in a 3D scene, based on the surface and illumination properties. This could be the surface's distance to the light sources, its angle to the lights and its angle to the camera.

In blender so-called shading nodes, which define materials, lights and background of the scene can be used to build a network of connected nodes and their resulting output is be used by the renderer. In our case we defined materials, which resemble the biological structures like cortex material, cell hull material, cell nucleus material (cell nuclei are not used so far), fungus material etc. and assigned them to their corresponding objects.

To build the material, mainly the *Principled BDSF* shading node was utilized. This shader combines multiple layers like the specular layer, roughness layer, sheen tint layer, transmission layer, alpha layer and many more to create a wide variety of materials. These layers were also the ones used most often for creating the different materials.

As main reference microscopy image for shading and modeling the plant root Figure 3.6 was used.

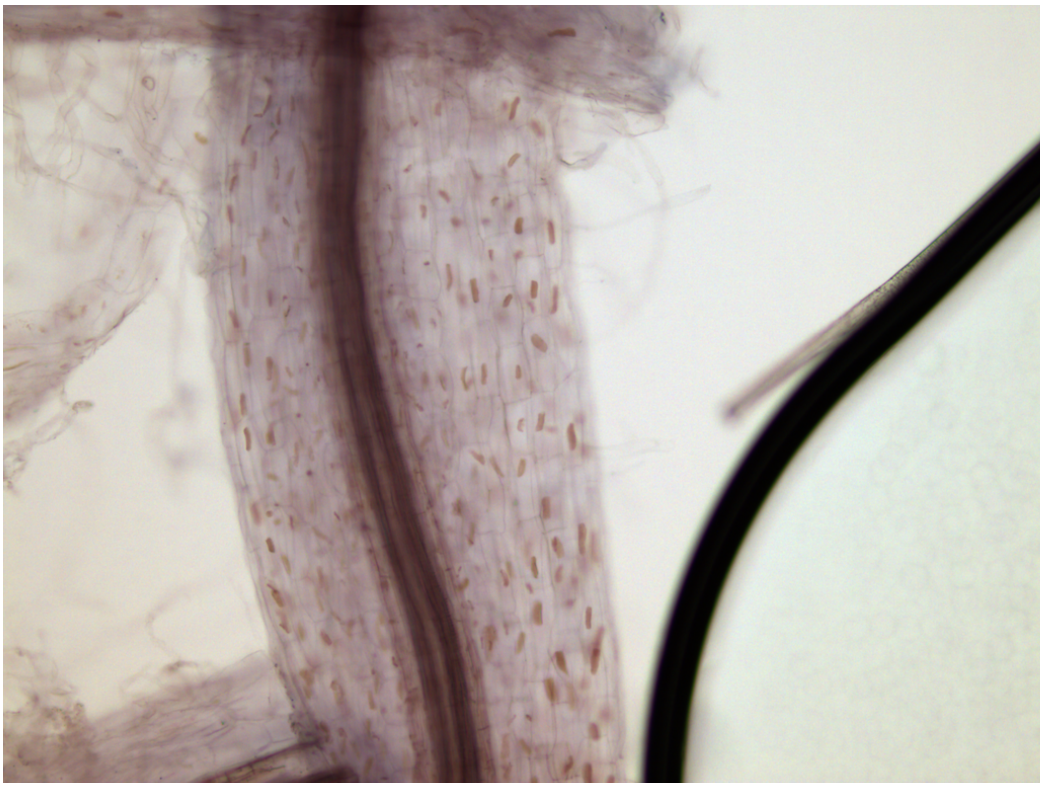


Figure 3.6: Microscopy image used as main reference for creating the shaders and modeling the plant root. An air bubble is seen on the right-hand side. Image taken by Catarina Cardoso.

Compositing and Rendering

There are sometimes cotton fibers or air bubbles visible in the images as seen in Figure 3.6. The cotton fibers were present in the substrate to hold the sand to the pots. Objects resembling these thin, dark fibers as well as air bubbles were added as background objects in the 3D model.

To mimic the effect of chromatic abberation, the compositing node *Lens Distortion* was used. Chromatic abberation is a color distortion that results in blurring or unwanted colored edges along objects in an image. It is due to a failure of a camera lens to focus different wavelengths onto the same focal points.

The NN needs two types of outputs: Firstly, an image that resembles the microscopy image and secondly, a target image that shows the segmented parts only. Using the blender compositor one can specify how these different outputs should be exported.

Figure 3.7 shows an example render and the corresponding target data and Figure 3.8 shows more example configurations that the pipeline generates procedurally. Observing the target data, the different distribution patterns of the fungal objects can be noticed. So far, the fungal objects are arranged manually and may need more elaborate distributions.

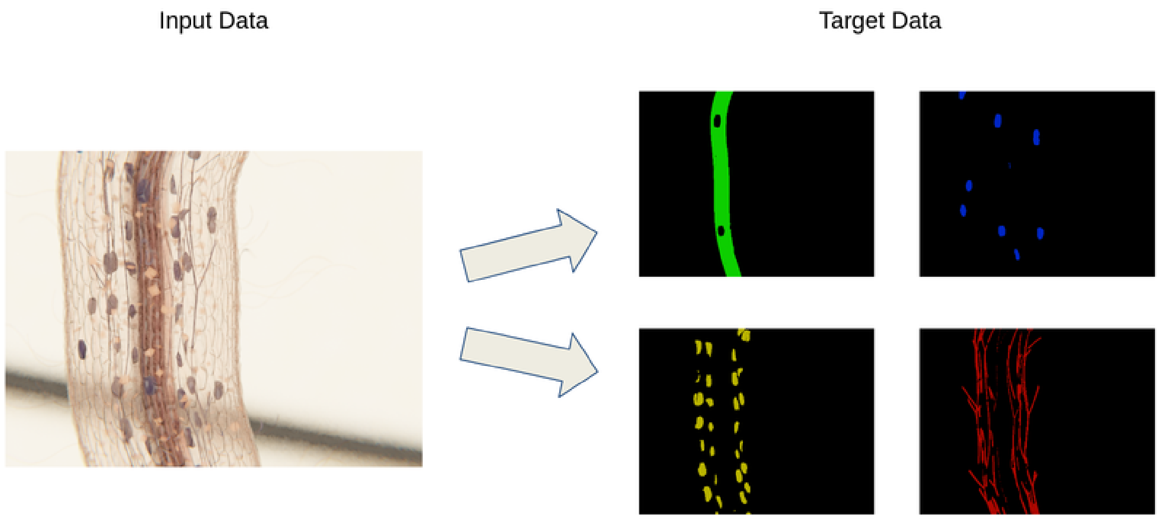


Figure 3.7: Realistic render on the left and the corresponding segmented target images (segmentation maps) on the right. The segmented objects shown here are root cortex (green), vesicles (blue), arbuscules (yellow) and hyphae (red).

3.3.2 Data Pre-processing

Figure 3.9 shows the pre-processing steps of the rendered image data, that need to be applied, before the training data can be used as input for the NN.

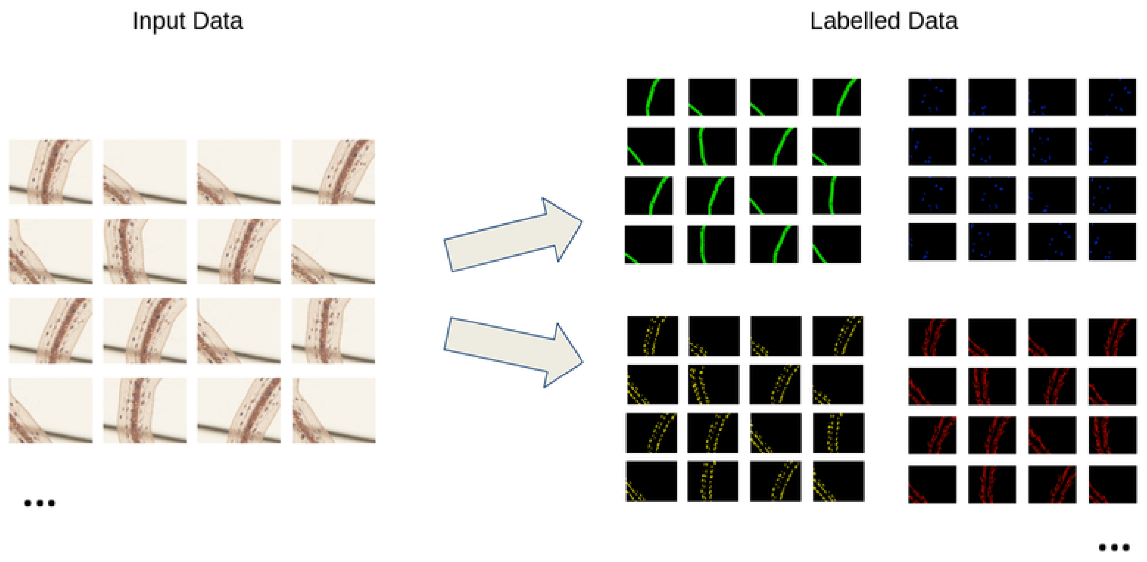


Figure 3.8: Example poses of the root procedurally generated by the render pipeline.

The left path of the flow-diagram shows the processing steps for the segmentation maps, while the right path shows the transformations for the input (or realistically rendered) images, which need to be resized only. The PyTorch framework requires the additional singleton dimension $D = 1$ for all input data.

Depending on the loss function, the input data and network need to be adjusted: Networks using BCE- or Dice-loss or their combination need their data to be one-hot encoded, as well as a Sigmoid layer as output layer. A network using cross-entropy loss requires the segmentation maps as categorical variables. Here, consecutive integers (often 0 is used as the background class) are used. Additionally, a Softmax layer as output layer is required by networks using cross-entropy loss.

One-hot encoding, sometimes called dummy coding in statistics, can be defined as follows: a dummy variable y_i, indicates when pixel p_i is a member of set A. In our case, p_i can be seen as a pixel value and A as the arbuscule label, then:

$$y_i = \mathbf{1}_{\mathrm{I}}(p_i) = \begin{cases} 1, & \text{if } p_i \in A \\ 0, & \text{if } p_i \notin A \end{cases} \tag{3.1}$$

with $\mathbf{1}_{\mathrm{I}}(x)$ as the indicator function of the membership of A.

In order to train the network, the data needs to be converted to the `.hdf5` File format. Each hdf5 file consists of two datasets, one representing the segmentation maps and one representing the input image. With these hdf5 files, the data is ready to be processed by the U-Net implementation.

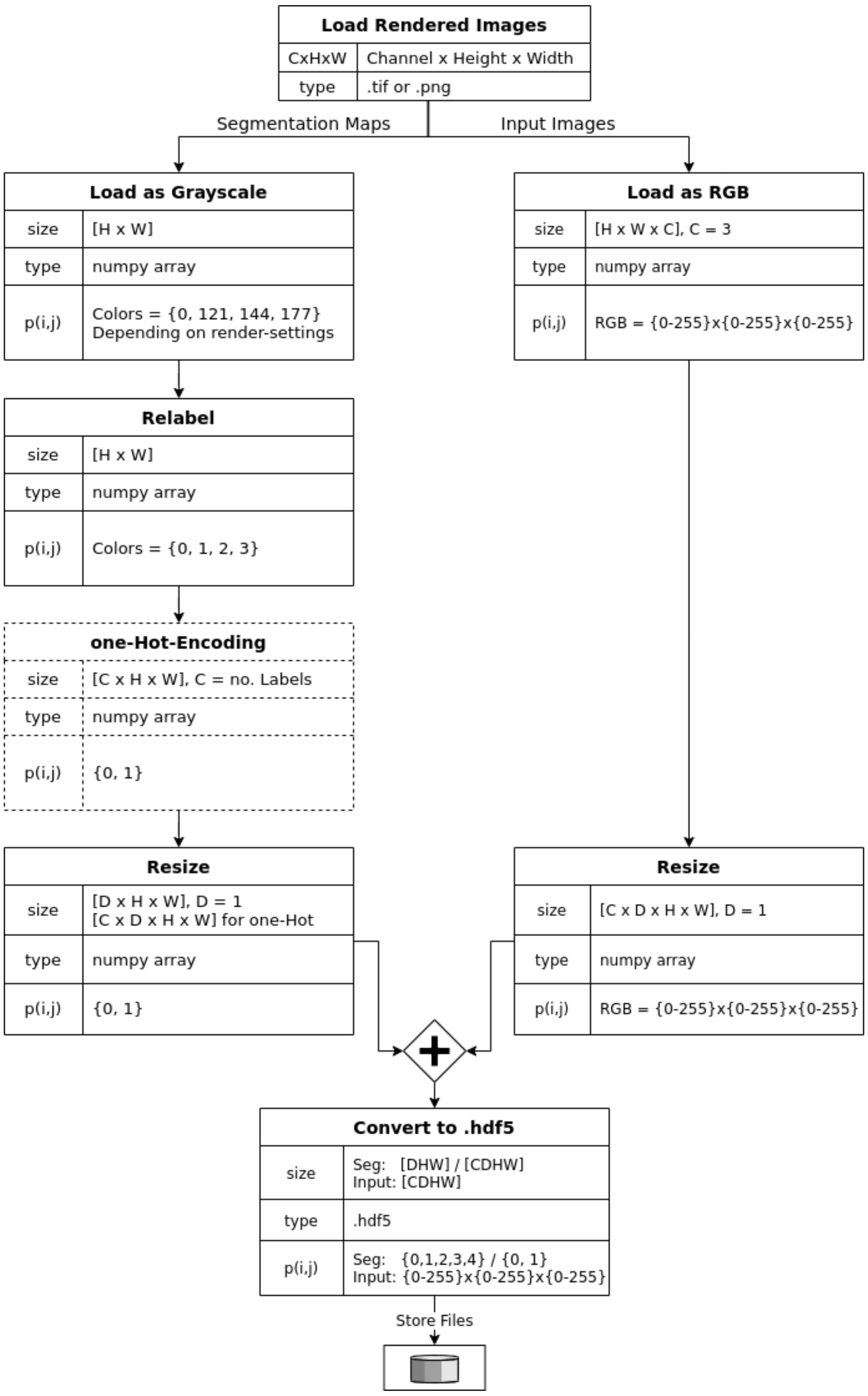

Load Rendered Images
CxHxW | Channel x Height x Width
type | .tif or .png
Segmentation Maps
Input Images
Load as Grayscale
size | [H x W]
type | numpy array
p(i,j) | Colors = {0, 121, 144, 177} Depending on render-settings
Load as RGB
size | [H x W x C], C = 3
type | numpy array
p(i,j) | RGB = {0-255}x{0-255}x{0-255}
Relabel
size | [H x W]
type | numpy array
p(i,j) | Colors = {0, 1, 2, 3}
one-Hot-Encoding
size | [C x H x W], C = no. Labels
type | numpy array
p(i,j) | {0, 1}
Resize
size | [D x H x W], D = 1 [C x D x H x W] for one-Hot
type | numpy array
p(i,j) | {0, 1}
Resize
size | [C x D x H x W], D = 1
type | numpy array
p(i,j) | RGB = {0-255}x{0-255}x{0-255}
Convert to .hdf5
size | Seg: [DHW] / [CDHW] Input: [CDHW]
type | .hdf5
p(i,j) | Seg: {0,1,2,3,4} / {0, 1} Input: {0-255}x{0-255}x{0-255}
Store Files

Figure 3.9: Pre-processing of blender-output, exemplary with 4 classes to segment.

3.3.3 Data Augmentation

This chapter lists the data augmentations, that have been employed before training the model. Before augmenting the data, each image X_i gets normalized to mean $\mu = 0$ and standard deviation $\sigma = 1$. This is sometimes referred to as *z-score normalization* and defined as

$$Z(X_i) = \frac{X_i - \bar{X}_i}{\sigma_i} \tag{3.2}$$

with $\bar{X}_i$ as the mean of the pixel values of image X_i and σ_i as the standard deviation across pixel values [1].

This normalization is necessary for some of the augmentation techniques and important for ML tasks in general, since it improves convergence speed and accuracy and combats vanishing and exploding gradients during network training [52].

The data augmentations, that were employed, are data warping techniques (cf. 2.4.3). We present the plots of the data augmentations as grayscale images, using the *magma* colormap from matplotlib [25] in a non-normalized form, otherwise the plots would appear meaningless. Furthermore, note that the augmentations are applied in a randomized fashion by the U-Net implementation.

- **Flip image** across vertical or horizontal axis as seen in 3.10

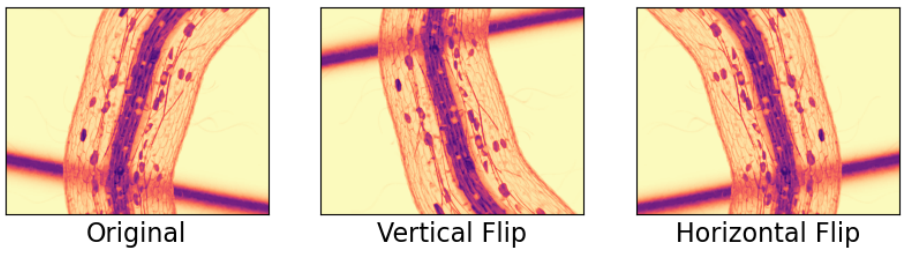


Figure 3.10: Data augmentation: Flip image

- **Rotate image 90 degrees** clockwise or counter-clockwise as seen in 3.11

[1]For RGB-images this normalization can be done channel-wise or per whole image.

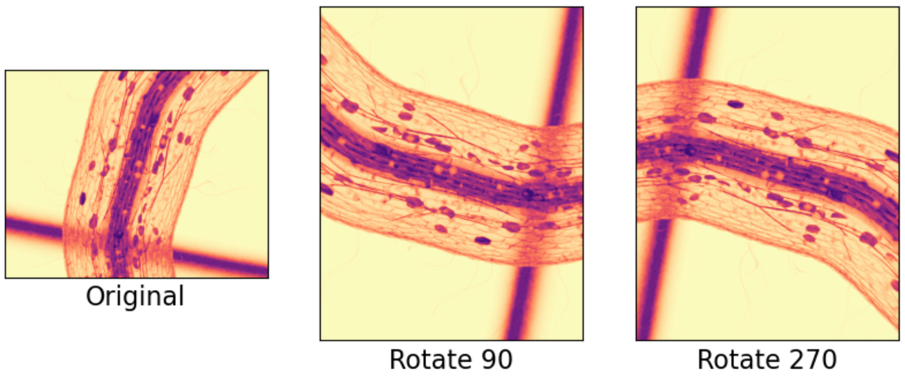


Figure 3.11: Data augmentation: Rotate 90 degrees

- **Random rotate image** by user-defined angular spectrum. In Figure 3.12 this is set to 45 degrees.

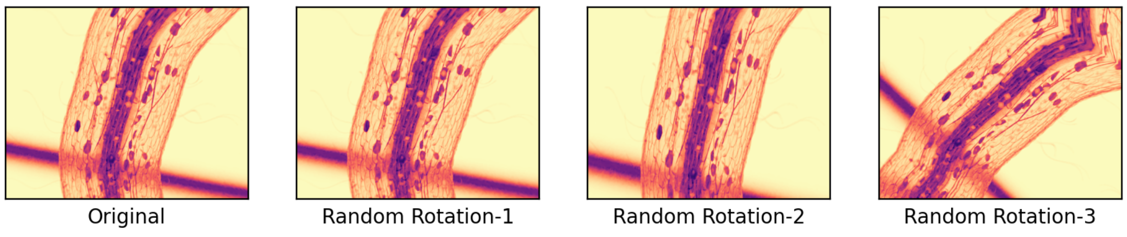


Figure 3.12: Data augmentation: Rotation with user-defined angle

- Apply **elastic deformation** of images as done in original U-Net paper [41] and 3D U-Net paper [11], shown in 3.13.

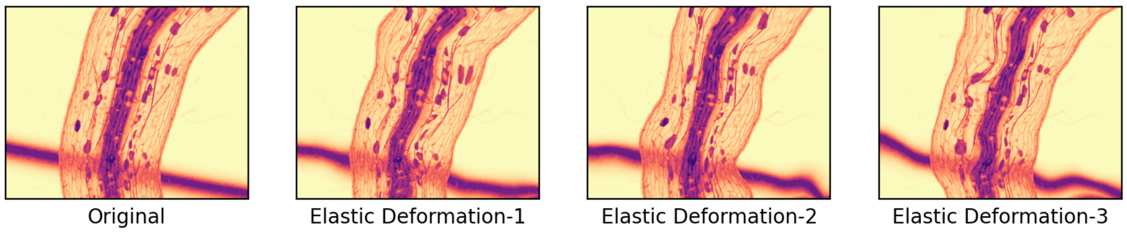


Figure 3.13: Data augmentation: Elastic deformation

- Apply **random contrast** by using transformation $T(X_i) = \bar{X}_i + \alpha(X_i - \bar{X}_i)$, for image X_i and its mean $\bar{X}_i$. Here $\alpha = [0.3, 1.5]$, where a value is chosen randomly from this interval for the standardised grayscale image. The transformation is shown in Figure 3.14.

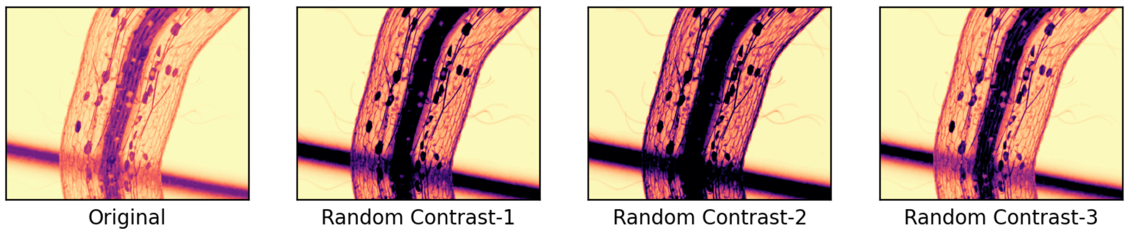


Figure 3.14: Data augmentation: Random contrast

- **Additive Gaussian noise:** The noise, that gets added to the original intensity values, follows a Gaussian distribution. Put differently, the statistical noise has a probability density function equal to that of the Gaussian distribution. The probability density function p of a Gaussian random variable z is given as

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{3.3}$$

with gray-level z, mean μ and standard deviation σ of a user-defined interval. The transformation is shown in Figure 3.15.

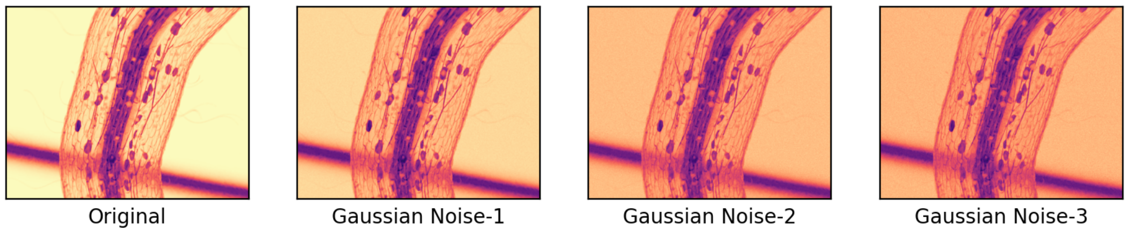


Figure 3.15: Data augmentation: Gaussian noise

- **Poisson noise:** The noise, that gets added to the original intensity values, is drawn from a Poisson distribution with probability mass function:

$$p_P(k) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{3.4}$$

with k as the number of samples (here number of grayscale values of the image) and a user-defined range of $\lambda > 0$, which gets sampled randomly. The transformation is shown in Figure 3.16

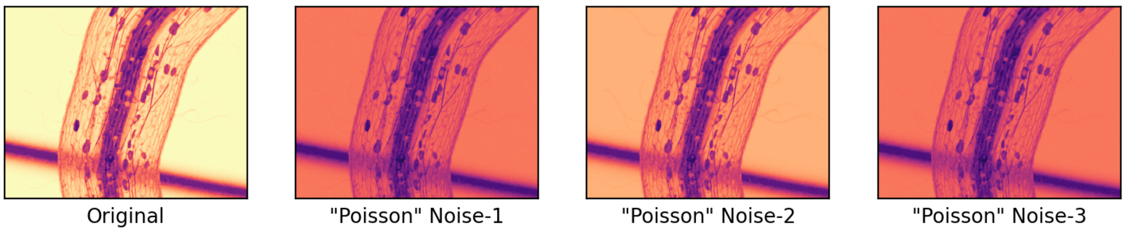


Figure 3.16: Data augmentation: Poisson noise

3.3.4 Data Post-processing

When running predictions, the U-Net generates probability maps as its output. For each class, a probability map, that displays the probabilities of all pixels belonging to this class, is predicted. For example, if the network was trained on 4 classes with images of resolution `1600x1200`, the output is a tensor of [4 x 1 x 1600 x 1200]. The singleton dimension is due to PyTorch but has no meaning regarding the images.

To evaluate this segmentation and produce meaningful plots, we applied different thresholds on the probability maps. A threshold of 0.5 produced the best evaluation metrics. Details on the training results and network predictions are shown in section 3.6.

As a sidenote: the `skimage.measure.label` algorithm labels connected regions on an array. By running this, one can simulate instance segmentation and quickly count the number of objects in an image. Figure 3.17 shows the predictions of vesicles and arbuscules of the image on the left, generated by the network. The mentioned connected-components algorithm was applied on the probability maps. Given that the single fungal objects are separated, one can use this to let the machine count the number of objects. This might be useful for future research on this project.

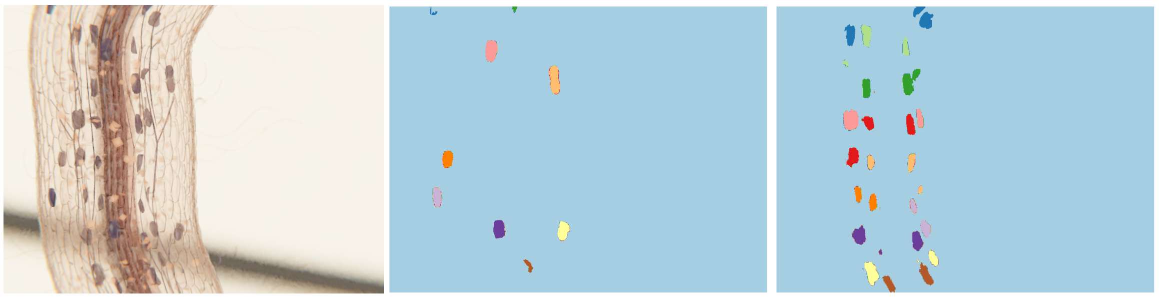


Figure 3.17: Connected components algorithm of skimage.

3.4 Evaluation Metrics and Loss Functions

Closely related to the mage similarity chapter (cf. 2.5) and loss functions are evaluation metrics, since they try to evaluate a distance between two corresponding images as well.

In the context of NNs, this image pair consists of the predicted data and the ground-truth. The following section discusses the different metrics, that are used to evaluate the accuracy of the model and how well the model performs on unseen data.

The metrics used for semantic segmentation tasks are mostly variations on Pixel Accuracy and Intersection over Union (IoU), which are based on pixel-wise similarity between ground-truth and predicted segmentation masks. The supported semantic segmentation metrics of the PyTorch U-Net implementation are: Average Precision, Mean Intersection over Union, Dice Coefficient and Adapted Rand Error. The following paragraphs will explain the basic ideas behind these metrics and point out potential shortcomings.

For the following explanations we denote these notation details: there are $k + 1$ total classes, including a background class; p_{ij} is the number of pixels of class i classified as class j.

Put differently, p_{ii} refers to the true positives (TP), while p_{ij} and p_{ji} represent the number of false positives (FP) and false negatives (FN) respectively. Lastly, p_{jj} can be interpreted as true negatives (TN).

3.4.1 Pixel Accuracies

The simplest metric **Pixel Accuracy** is computed by the ratio between the number of correct classified pixels and the total number of pixels [21]:

$$PixelAccuracy = \frac{\sum_{i=0}^{k} p_{ii}}{\sum_{i=0}^{k}\sum_{j=0}^{k} p_{ij}} \tag{3.5}$$

In **Mean Pixel Accuracy** the ratio of correct classified pixels is calculated in a per-class basis before averaged over the total number of classes:

$$MeanPixelAccuracy = \frac{1}{k+1}\sum_{i=0}^{k}\frac{p_{ii}}{\sum_{j=0}^{k} p_{ij}} \tag{3.6}$$

A significant limitation of these scores is their bias for datasets with high class imbalances.

In many real-world datasets the background-class can be large and one can artificially improve the object class accuracies: By always predicting the classes of objects and never predicting the background-class, all object class accuracies get improved at the expense of lowering a single scalar in the average (the score of the background class) [14].

The following metrics try to counteract this limitation.

3.4.2 Intersection over Union or Jaccard Index

Initially we define the **confusion matrix** for binary classification: it depicts the predicted instances of a predicted class in the rows and the instances of an actual class in the columns as shown in table 3.3.

		Ground-truth	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN

Table 3.3: Confusion matrix

With this, the following metrics can be easily understood. **Intersection over Union** (IoU) or **Jaccard Index** is the standard metric for segmentation purposes. It is computed by taking the ratio between the intersection and the union of two sets. Here, these are the ground-truth X and the predicted segmentation $\hat{X}$:

$$IoU(X,\hat{X}) = \frac{\left|X \cap \hat{X}\right|}{\left|X \cup \hat{X}\right|} \rightarrow \frac{TP}{TP + FN + FP} \tag{3.7}$$

The IoU score can range from 0 to 1. A score of 1 means that the predicted pixels exactly match the ground-truth pixels, while a score of 0 means none of the predicted pixels match the ground-truth class. A common visualization is shown in Figure 3.18.

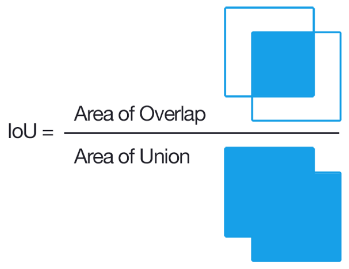


Figure 3.18: Visual equation of IoU. Author: Adrian Rosebrock [42]. Licensed under CC BY-SA 4.0.

A pixel-wise **Mean Intersection over Union (MIoU)** is computed on a per-class basis and then averaged over the classes. It is calculated the following way:

$$MIoU = \frac{1}{k+1}\sum_{i=0}^{k} \frac{p_{ii}}{\sum_{j=0}^{k} p_{ij} + \sum_{j=0}^{k} p_{ji} - p_{ii}} \tag{3.8}$$

One limitation of this method is that it evaluates the amount of correct segmented regions but not necessarily how accurate the segmentation boundaries are [15].

The next metric and a further improvement is **Average Precision**: For each object in the prediction, the IoU is calculated with the corresponding ground-truth mask in the image. If it exceeds a certain threshold, it is counted as TP, otherwise as FP. At each threshold the precision is then calculated across all masks before it is averaged across multiple IoU values. Refer to [4] for a step-by-step explanation.

3.4.3 Dice-Coefficient or F1-score

Similar to the Jaccard Index is the **Dice-Coefficient** or **F1-Score**, which gets calculated the following way:

$$Dice(X, \hat{X}) = \frac{2\left|X \cap \hat{X}\right|}{\left|X + \hat{X}\right|} \rightarrow \frac{2TP}{2TP + FN + FP} \tag{3.9}$$

Its range is between 0 and 1. In contrast to IoU, it does not satisfy the triangle inequality and can be considered a semi-metric version of the Jaccard Index. Which of these measures works best is dependent on the application.

Recent research [56] points out, that the metrics mentioned above account for the correctness of predictions on a global level but fail to capture region-based agreement between predicted regions and ground-truth.

3.4.4 Rand Error

The last evaluation metric, that is implemented in PlantSeg, is a variant of the **Rand Error** (RE) [38]. We first need to define the Rand index (RI), which is a measure of the similarity between data clusterings.

Since a segmentation can be viewed as a clustering of pixels, it has been proposed as an evaluation metric: Given two segmentations X and $\hat{X}$ (in our case ground-truth and predicted segmentation) of an Image I with n pixels, we define the following:

- a as the number of pairs of pixels that are part of the same object in X and in $\hat{X}$ (TP). Meaning they are assigned the same label.
- b as the number of pairs of pixels that are in different objects in X and in different objects in $\hat{X}$. They belong to different classes.

Then the RI is defined as:

$$RI = \frac{a + b}{\binom{n}{2}} \rightarrow \frac{TP + TN}{TP + TN + FP + FN} \tag{3.10}$$

Similar to the previous scores, the RI ranges from 0 to 1, where 0 indicates that the two data clusterings do not agree on any pair of pixels and 1 indicates, that the clusterings are the same. Finally the Rand Error is defined as:

$$RE = 1 - RI \tag{3.11}$$

PlantSeg comes with a adjusted version of the Rand Error, an explanation is given in [43].

Note how related these metrics are to the topics of Image similarity (cf. 2.5) and loss functions (cf. 3.4.5). Loss functions like the Dice-loss can be directly calculated from the Dice-coefficient by Dice Loss $= 1-$Dice Coefficient, since the goal of the NN is to minimize the loss function.

3.4.5 Loss Functions in PlantSeg

The implemented **loss functions** in PlantSeg are:

1. **Binary cross-entropy** (BCE)
2. **Cross-entropy**, where class weights can be specified
3. **Pixel-wise cross-entropy**, where additionally per pixel weights can be classified to assign more weight to important or under-represented regions in the ground-truth
4. **Dice-Loss** defined as $1 - DiceCoefficient$ (cf. 3.8) for binary segmentation; for more than two classes it computes the Dice-loss over each channel and averages the results
5. **Generalized Dice-Loss**, where additionally class weights can be specified
6. **BCE-Dice-Loss**, a linear combination of BCE and Dice loss, calculated as $\alpha * BCE + \beta * Dice$, where α, β can be specified

Some of these supported loss functions, as well as the issue of class-imbalance, is explained in [48]. Generally speaking, cross-entropy loss functions are susceptible to class imbalance, while Dice-losses tend to perform better for data imbalance.

3.5 U-Net Architecture and PyTorch Implementation

The CNN that was used for this thesis is a PyTorch implementation [54] of the 3D U-Net architecture described in [11]. The original network structure is shown in image 3.19.

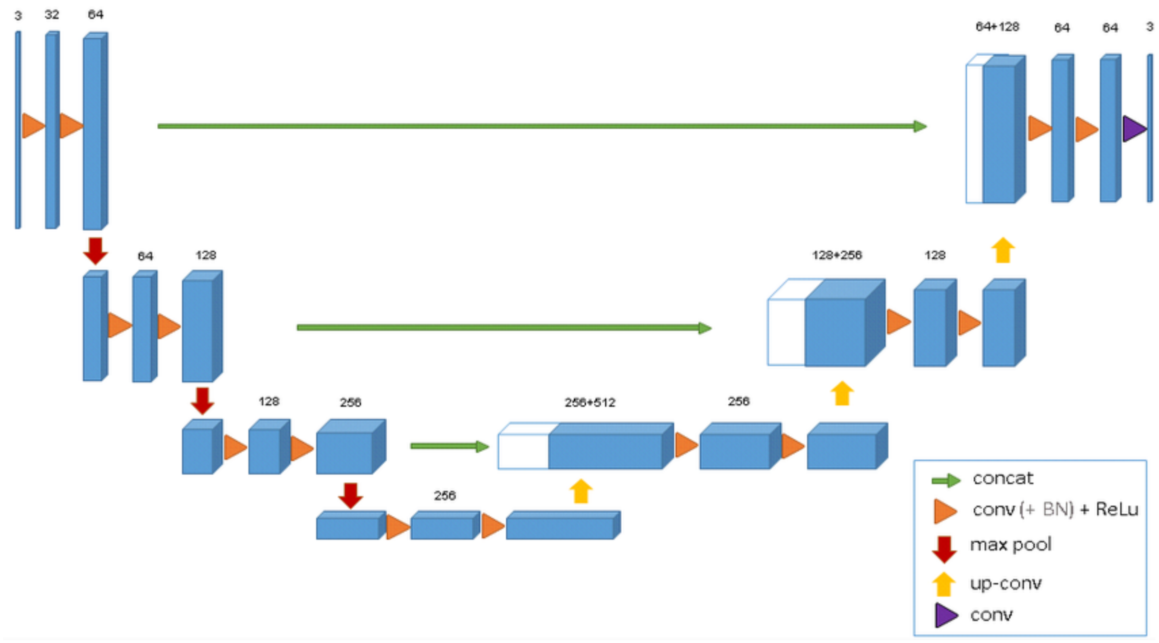


Figure 3.19: The original 3D U-Net architecture. Blue boxes represent feature maps. The number of channels is denoted above each feature map. With permission from Özgün Çiçek [11] and Springer International Publishing AG.

It consists of a contracting path on the left side and an expansive path on the right side, resembling a U-shape.

The contracting path resembles the typical architecture of a CNN. Each layer consists of two repeated $3x3x3$ convolutions each followed by a rectified linear unit (ReLU) and a $2x2x2$ max-pooling operation with stride 2 for downsampling. Thus, the feature channels get doubled in each downsampling step. For the sake of completion, we show the definition of the ReLU function:

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{3.12}$$

The levels in the expansive path consist of $2x2x2$ transpose convolutions (or upconvolutions), using strides of two in each dimension, followed by two $3x3x3$ convolutions each followed by a ReLU.

Skip-connections between the layers of equal resolution of the two paths, represented as green horizontal arrows in the diagram, provide high-resolution features to the expansive path. Meaning the input of an expansive block is a concatenation of the output from the previous expansive block and the output from the corresponding contractive block before the max-pooling layer. Thus, the spatial information lost during downsampling is recovered. These symmetrical skip connections work very effectively in dense prediction tasks [17].

At the final layer, a $1x1x1$ convolution is used to map each 64-component feature vector to the desired number of classes, which is 3 in the original implementation. Since the network does not contain any dense layers, it can process images of any size.

Also batch normalization [26] is used before each ReLU.

The **U-Net implementation** used in this thesis is written in Python and uses the machine learning library PyTorch [36]. This implementation is also used in the segmentation tool PlantSeg [53] and highly modular. It can be used for semantic segmentation (binary and multi-class) and regression problems. Both a 3D U-Net architecture based on [11] and the standard 2D U-Net architecture, as well as a Residual U-Net, can be trained.

Data parallelism using the parallel computing platform and application programming interface CUDA can be utilized. The data has to be provided in the HDF5 file format [20].

A configuration file for training of a network, that was used in this thesis, is listed in the appendix (cf. 4.1).

3.6 Training Results

In this section, we present results of the network training. Starting with differences and similarities of the models, we continue to show detailed information about a particular 4-class model.

Several models with different hyperparameters have been trained:

- No. of classes: 4 classes vs 3 classes (background, arbuscules, vesicles and hyphae); Hyphae were omitted in the 3-class models
- Dataset size: ranging from 50 images to 500 images
- Different learning-rates and learning-rate schedulers
- Different number of feature maps at each level in the U-Net
- Different number of epochs and iterations
- Different weight decays and data augmentations

The loss function (cross-entropy-loss) and evaluation metric (MeanIoU) were constant over all models. Changing them would require different pre-processing of the input data.

An interesting observation for all models, is the behaviour of the evaluation score on the training and validation set. The evaluation score converges towards 0.5. This can be seen in Figure 3.23, which shows the evaluation score of a training set. When the large background class was omitted, the networks converge towards even lower evaluation scores. However, on the test set, the evaluation scores are much better, as shown below (cf. 3.5).

Another observation is that the image data generated by the blender-pipeline is quite similar. This could explain why the loss function decreases rapidly during the beginning of the network training, while further iterations show less improvements. This can be seen in Figure 3.22, which shows the loss during training.

The networks always stopped their training, due to the stopping criterion *minimum learning rate reached*. Learning rate schedulers were responsible for reducing the learning rate when certain criteria are fulfilled. Different schedulers have been employed: *Reduce-On-Plateu*, which reduces the learning rate after a metric has stopped improving and *Multi-Step*, which reduces the learning rate after a certain number of epochs.

The following statistics and predictions show the details of a 4-class model. Although this model performed very well, other models using considerably less training data (for example only 50 images) performed similar. This shows again, that the training images were quite similar to each other.

Table 3.4 shows information about the dataset and network configurations.

Data				
Data split	No. images	File size [MB]	Classes	Image Resolution
			Background	
Training:	350	1.200	Vesicles	
Validation:	70	240	Arbuscules	973 x 736
Test:	50	160	Hyphae	
Total:	**470**	**1.600**	**Total: 4**	

Table 3.4: Data properties of a 4-class model.

Figures 3.22, 3.23 show the network loss and evaluation score for the training set. Figures 3.22, 3.23 show the network loss and evaluation score for the validation set. Interesting is the outlier near iteration 5k. Similar to other models, the evaluation score on the validation set is more smooth, compared to the training set.

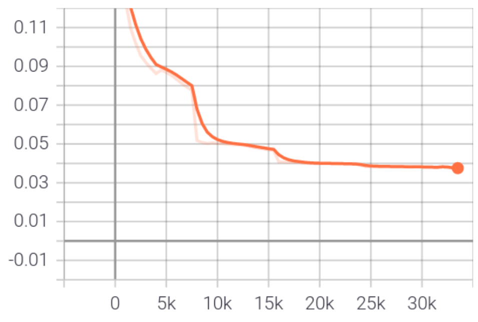


Figure 3.20: Network loss for a 4-class model on the training set. The diagram shows the loss on the y-axis and no. of iterations on the x-axis.

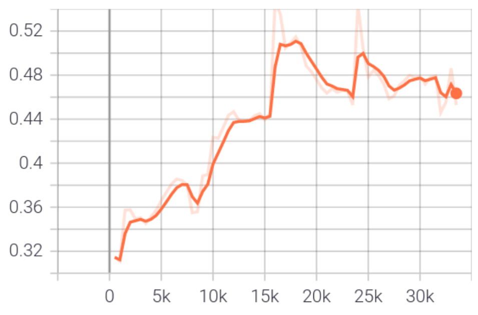


Figure 3.21: Evaluation score for a 4-class model on the training set.

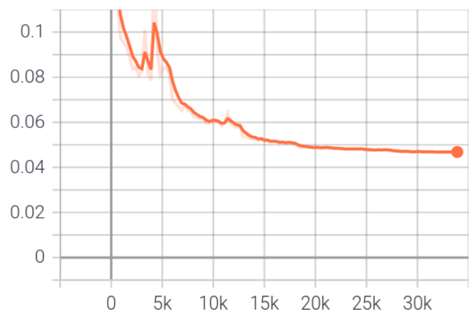


Figure 3.22: Network loss for a 4-class model on the validation set.

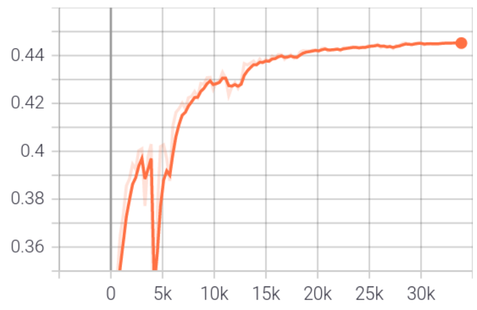


Figure 3.23: Evaluation score for a 4-class model on the validation set.

Evaluation metrics are shown in table 3.5. Firstly, Precision, Recall, F1-score and the Jaccard-index are calculated on a per-class basis. Interestingly, vesicles perform consistently better than arbuscules and hyphae. A possible explanation for this behaviour could be the higher contrast of vesicles with regard to their surroundings.

The accuracy metric computes the fraction of correct predictions. Since the background class dominates the other classes, this leads to a misguided high score. Therefore, we included the F1-score and Jaccard-index, averaged over the fungal objects, excluding the background class. The Adapted Rand error (in the table AR) measures similarity of data clusterings.

A detailed explanation of the evaluation metrics and the problem of class-imbalance is explained in section 3.4.

Label-wise					Averaged		Averaged w/o background	
Classes	Precision	Recall	F1	Jacc.	Acc.	AR	F1	Jacc.
Background:	98.9	99.6	99.2	98.5	98.2	88.8	84.4	73.2
Hyphae:	89.5	77.0	82.7	70.6				
Arbuscules:	84.5	85.0	84.7	73.6				
Vesicles:	91.8	87.9	89.6	81.4				

Table 3.5: Evaluation metrics of a 4-class model on the test set. Precision, Recall, F1-score and Jaccard-index are first calculated for each label. Accuracy and the Adjusted Rand-error are metrics that take all classes into account. Because the background class is very large, compared to the fungal object classes, we included an averaged version of the F1-score and Jaccard-index, that excludes the background class. The results are shown in per cent.

Finally, we show a few predictions of the U-Net, compared to the groundtruth data for the test set. The colormap of the groundtruth and prediction images shows hyphae in red, arbuscules in yellow and vesicles in blue. Clearly, the network has the most problems when arbuscules and hyphae overlay. Occasionally, it confuses arbuscules with vesicles. This happens only when vesicles and arbuscules appear very similar. Viewing the original render, this difference can be difficult to detect for a human observer as well. Otherwise, the network produces satisfying predictions.

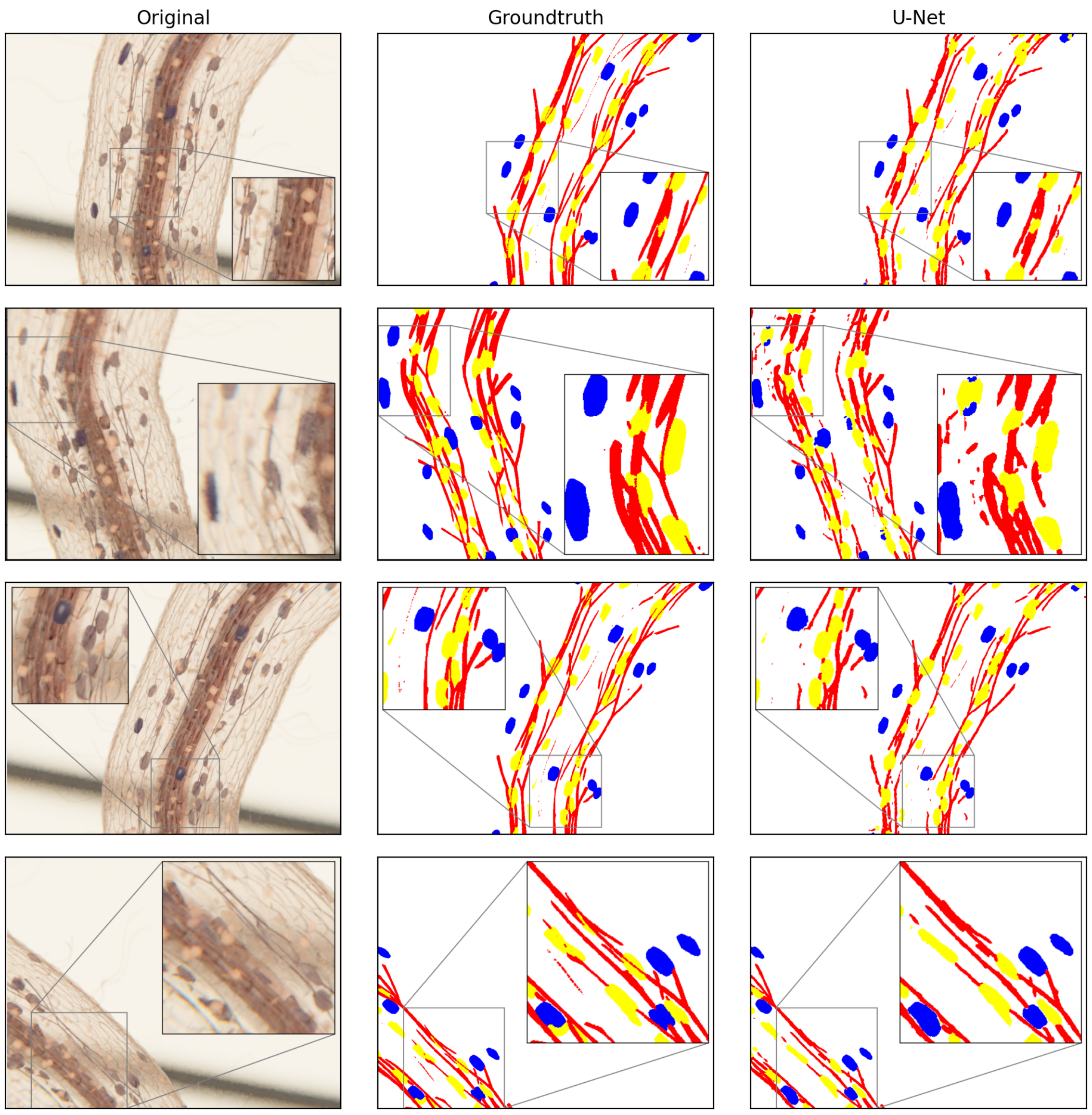


Figure 3.24: Predictions of the U-Net, compared to the groundtruth and the corresponding rendered data. The images are taken from the test set.

4 Conclusion

We presented an image processing pipeline that assists with the visual analysis of quantifying AMF colonization by segmenting images into regions that correspond to fungal structures like arbuscules, vesicles and hyphae.

Due to the lack of labeled microscopy image data, we built a 3D model of a colonized root section, that allowed us to procedurally generate training data. This data was further processed and augmented, before fed into a CNN. The utilized CNN-architecture is a PyTorch implementation of the U-Net architecture and part of the PlantSeg segmentation pipeline.

We presented evaluation results and showed a few predictions of the network for unseen data. The evaluation results for the fungal structures show a F1-score of 84%. As the prediction images show, the network produces satisfying results. In conclusion, the overall aim of the thesis, meaning the creation of an automated 3D rendering pipeline to generate training data, and subsequent training of a deep learning neural network, has been reached.

As this thesis is an ongoing research project, we outline some suggestions for improvements and an outlook. There are several ideas to train more robust networks by modifying the render pipeline: use different camera angles, use different zoom levels for better scale invariance, render at different focal-planes to create a z-stack, alter lightning conditions of the scene, alter hue and saturation of plant and fungal objects and change distribution patterns of arbuscules, vesicles and hyphae. These modification ideas show again how powerful, modular and extensible 3D models can be.

The next step for this ongoing research project is to apply this segmentation pipeline to real-world microscopy data. In this thesis several suggestions have been made, regarding future work: Use the models, trained in this thesis, as pre-trained models for transfer learning; extend the 3D model to mutant arbuscules and generate more variety in the renderings; apply more hyperparameter tuning with respect to the network training.

During development of this thesis, a paper called *Artificial intelligence enables the identification and quantification of arbuscular mycorrhizal fungi in plant roots* and the corresponding software *AMFinder* [18] were published, after two years of development. This shows the scientific interest in AMF image analysis research.

AMFinder also uses CNNs and predicts the presence of fungal structures in image-tiles. With these current research developments in mind, future projects could focus on classifying different developmental stages in the growth process of AMF. Here, a network would not only detect the presence of arbuscules but further distinct into fully-developed or mutant arbuscules.

Appendix

Code 4.1 shows an example configuration file for network training.

```yaml
manual_seed: 0
model:
  name: UNet2D
  # number of input channels (for RGB images: 3)
  in_channels: 3
  # number of output channels = number of semantic classes
  out_channels: 4
  # layer ordering: groupnorm + conv + relu
  layer_order: gcr
  # fix number of groups for the groupnorm
  num_groups: 8
  # number of feature maps at each level of the U-Net
  f_maps: [32, 64, 128, 256]
  # Softmax is used instead of Sigmoid
  final_sigmoid: false
  # for segmentation problems: true
  is_segmentation: true
trainer:
  # path the model and tensorboard logs are saved
  checkpoint_dir: "/path/to/model"
  # set to 'last_checkpoint.pytorch' if traning shall be resumed
  resume: null
  validate_after_iters: 100
  log_after_iters: 100
  epochs: 30
  iters: 100000
  # hint to the trainer if lower or higher validation scores are better
  eval_score_higher_is_better: True
# Adam optimizer config
optimizer:
  # initial learning rate
  learning_rate: 0.002
  # weight L2 regularization weight
  weight_decay: 0.00001
# loss configuration
loss:
  # for multi-class semantic segmentation problem: standard cross-entropy
  name: CrossEntropyLoss
# configuration of the evaluation metric
eval_metric:
  name: MeanIoU
  # skip background class so that it doesn't dominate remaining semantic classes
  skip_channels: [0]
# configure the learning rate scheduler
lr_scheduler:
  name: ReduceLROnPlateau
  mode: max
  factor: 0.02
  patience: 5
loaders:
  # save training data as .hdf5 / .h5 files:
  # input should be saved as CDHW format; for example 4 classes: 4x1xHxW
  # if cross entropy is used, target should be saved in DHW
  dataset: StandardHDF5Dataset
  # name of the target dataset inside the .h5 file
  label_internal_path: 'label'
  # name of the input dataset inside the .h5 file
  raw_internal_path: 'raw'
  # batch size
  batch_size: 16
  # number of worker processes in the loader
  num_workers: 8
  # training data loader config
  train:
    file_paths:
      - '/path/to/training_files/'
    # building of patches for training, "sliding window"
    # with the size of 256x256 and stride of 32x32
    slice_builder:
      name: SliceBuilder
      patch_shape: [1, 256, 256]
      stride_shape: [1, 32, 32]
      skip_shape_check: True
    # training transformation / data augmentations
    transformer:
```

```yaml
      raw:
        - name: Standardize
        - name: RandomFlip
        - name: RandomRotate90
        - name: RandomRotate
          axes: [[2, 1]]
          angle_spectrum: 30
          mode: reflect
        - name: ElasticDeformation
          spline_order: 3
          execution_probability: 0.1
        - name: RandomContrast
          execution_probability: 0.1
        - name: AdditiveGaussianNoise
          scale: [0.0, 0.5]
          execution_probability: 0.1
        - name: AdditivePoissonNoise
          lam: [0.0, 0.01]
          execution_probability: 0.1
        - name: ToTensor
          expand_dims: true
      label:
        - name: RandomFlip
        - name: RandomRotate90
        - name: RandomRotate
          axes: [[2, 1]]
          angle_spectrum: 30
          mode: reflect
        - name: ElasticDeformation
          spline_order: 0
          execution_probability: 0.1
        - name: ToTensor
          expand_dims: false
          dtype: 'int64'
  # validation data loader config
  val:
    file_paths:
      - '/path/to/validation_files/'
    slice_builder:
      name: SliceBuilder
      patch_shape: [1, 256, 256]
      stride_shape: [1, 256, 256]
      skip_shape_check: True
    transformer:
      raw:
        - name: Standardize
        - name: ToTensor
          expand_dims: true
      label:
        - name: ToTensor
          expand_dims: false
          dtype: 'int64'
```

Source Code 4.1: Example configuration file for training semantic segmentation problem with standard 2D U-Net using the PyTorch implementation described in section 3.5.

Bibliography

[1] Anaconda software distribution, 2020.

[2] Michael D Abràmoff, Paulo J Magalhães, and Sunanda J Ram. Image processing with imagej. *Biophotonics international*, 11(7):36–42, 2004.

[3] 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, 2017.

[4] Stephen Bailey. Step-by-step explanation of scoring metric. https://www.kaggle.com/stkbailey/step-by-step-explanation-of-scoring-metric, 2018. Accessed: 2021-03-09, Licensed under Apache License, Version 2.0: http://www.apache.org/licenses/LICENSE-2.0.

[5] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017.

[6] Paola Bonfante and Andrea Genre. Mechanisms underlying beneficial plant–fungus interactions in mycorrhizal symbiosis. *Nature communications*, 1(1):1–11, 2010.

[7] Alan C Bovik. *Handbook of image and video processing*. Academic press, 2010.

[8] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[9] Christoph Burgmer. Diagram of a multi-layer feedforward arificial neural network. https://commons.wikimedia.org/wiki/File:MultiLayerNeuralNetworkBigger_english.png, 2010. Accessed: 2021-03-28, Licensed under CC BY-SA 3.0: https://creativecommons.org/licenses/by-sa/3.0/.

[10] Anne E Carpenter, Thouis R Jones, Michael R Lamprecht, Colin Clarke, In Han Kang, Ola Friman, David A Guertin, Joo Han Chang, Robert A Lindquist, Jason Moffat, et al. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology*, 7(10):R100, 2006.

[11] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation.

In *International conference on medical image computing and computer-assisted intervention*, pages 424–432. Springer, 2016.

[12] Andrew Collette. *Python and HDF5*. O'Reilly, 2013.

[13] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[14] Gabriela Csurka, Diane Larlus, Florent Perronnin, and F Meylan. What is a good evaluation measure for semantic segmentation? *IEEE PAMI*, 26(1), 2004.

[15] Gabriela Csurka, Diane Larlus, Florent Perronnin, and France Meylan. What is a good evaluation measure for semantic segmentation?. In *BMVC*, volume 27, pages 10–5244, 2013.

[16] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

[17] Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The importance of skip connections in biomedical image segmentation. In *Deep learning and data labeling for medical applications*, pages 179–187. Springer, 2016.

[18] Edouard Evangelisti, Carl Turner, Alice McDowell, Liron Shenhav, Temur Yunusov, Aleksandr Gavrin, Emily K Servante, Clement Quan, and Sebastian Schornack. Artificial intelligence enables the identification and quantification of arbuscular mycorrhizal fungi in plant roots. *bioRxiv*, 2021.

[19] Serena Yeung Fei-Fei Li, Justin Johnson. Lecture 11: Detection and segmentation. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf, 2017. Accessed: 2021-01-02, Licensed under CC0 1.0, https://creativecommons.org/publicdomain/zero/1.0/deed.en.

[20] Mike Folk, Gerd Heber, Quincey Koziol, Elena Pourmal, and Dana Robinson. An overview of the hdf5 technology suite and its applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pages 36–47, 2011.

[21] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.

[22] A Ardeshir Goshtasby. *Image registration: Principles, tools and methods*. Springer Science & Business Media, 2012.

[23] Caroline Gutjahr and Martin Parniske. Cell and developmental biology of arbuscular mycorrhiza symbiosis. *Annual review of cell and developmental biology*, 29:593–617, 2013.

[24] Guosheng Hu, Yongxin Yang, Dong Yi, Josef Kittler, William Christmas, Stan Z Li, and Timothy Hospedales. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 142–150, 2015.

[25] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[27] Arshad Javaid. Arbuscular mycorrhizal mediated nutrition in plants. *Journal of Plant Nutrition*, 32(10):1595–1618, 2009.

[28] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional neural networks and applications in vision. In *ISCAS 2010, IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, 2010.

[29] Leonie H. Luginbuehl and Giles E.D. Oldroyd. Understanding the arbuscule at the heart of endomycorrhizal symbioses in plants. *Current Biology*, 27(17):R952 – R963, 2017.

[30] Adam H Marblestone, Greg Wayne, and Konrad P Kording. Toward an integration of deep learning and neuroscience. *Frontiers in computational neuroscience*, 10:94, 2016.

[31] H. B. Mitchell. *Image Similarity Measures*, pages 167–185. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[32] Sharada P Mohanty, David P Hughes, and Marcel Salathé. Using deep learning for image-based plant disease detection. *Frontiers in plant science*, 7:1419, 2016.

[33] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[34] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020.

[35] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[37] Simon Pöcheim. Convolutional neural networks. https://wiki.tum.de/display/lfdv/Layers+of+a+Convolutional+Neural+Network. Accessed: 2021-01-03.

[38] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[39] Adrienne HK Roeder, Alexandre Cunha, Michael C Burl, and Elliot M Meyerowitz. A computational image analysis glossary for biologists. *Development*, 139(17):3071–3080, 2012.

[40] Brandon Rohrer. How do convolutional neural networks work. https://e2eml.school/how_convolutional_neural_networks_work.html, 2016-08-18. Accessed: 2021-03-15; Licensed under CC0 1.0 Universal (CC0 1.0): https://creativecommons.org/publicdomain/zero/1.0/.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[42] Adrian Rosebrock. A visual equation for intersection over union. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, 2016. Accessed: 2021-03-08, Licensed under CC BY-SA 4.0: https://creativecommons.org/licenses/by-sa/4.0/.

[43] Nader Shaar. Snemi3d 3d segmentation. http://brainiac2.mit.edu/SNEMI3D/evaluation, 2013. Accessed: 2021-03-09.

[44] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[45] Siyavula. Dicotyledonous root profile showing the major tissues found in the root system which also aid in transport. https://intl.siyavula.com/read/science/grade-10-lifesciences/support-and-transport-systems-in-plants, 2015. Accessed: 2021-03-15, Licensed under CC BY 3.0: https://creativecommons.org/licenses/by/3.0/.

[46] Sally E Smith and David J Read. *Mycorrhizal symbiosis*. Academic press, 2010.

[47] Christoph Sommer, Christoph Straehle, Ullrich Koethe, and Fred A Hamprecht. Ilastik: Interactive learning and segmentation toolkit. In *2011 IEEE international symposium on biomedical imaging: From nano to macro*, pages 230–233. IEEE, 2011.

[48] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 240–248. Springer, 2017.

[49] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[50] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[51] Horst Vierheilig, Andrew P Coughlan, URS Wyss, and Yves Piché. Ink and vinegar, a simple staining technique for arbuscular-mycorrhizal fungi. *Applied and environmental microbiology*, 64(12):5004–5007, 1998.

[52] Xing Wan. Influence of feature scaling on convergence of gradient iterative algorithm. In *Journal of Physics: Conference Series*, volume 1213, page 032021. IOP Publishing, 2019.

[53] Adrian Wolny, Lorenzo Cerrone, Athul Vijayan, Rachele Tofanelli, Amaya Vilches Barro, Marion Louveaux, Christian Wenzl, Susanne Steigleder, Constantin Pape, Alberto Bailoni, et al. Accurate and versatile 3d segmentation of plant tissues at cellular resolution. *BioRxiv*, 2020.

[54] Adrian Wolny, Lorenzo Cerrone, Athul Vijayan, Rachele Tofanelli, Amaya Vilches Barro, Marion Louveaux, Christian Wenzl, Sören Strauss, David Wilson-Sánchez, Rena Lymbouridou, Susanne S Steigleder, Constantin Pape, Alberto Bailoni, Salva Duran-Nebreda, George W Bassel, Jan U Lohmann, Miltos Tsiantis, Fred A Hamprecht, Kay Schneitz, Alexis Maizel, and Anna Kreshuk. Accurate and versatile 3d segmentation of plant tissues at cellular resolution. *eLife*, 9:e57613, jul 2020.

[55] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

[56] Yuxiang Zhang, Sachin Mehta, and Anat Caspi. Rethinking semantic segmentation evaluation for explainability and model selection. *arXiv preprint arXiv:2101.08418*, 2021.

[57] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. 2005.