University of Bamberg

Faculty of Information Systems and Applied Computer Sciences

Master's Thesis

in

Information Systems

Towards automated segmentation of plants – An exploration of deep neural networks for leaf isolation in agricultural crops

---

Author:
Andreas Foltyn
(Matr.No. 1843301)

Supervisor:
Prof. Dr. Ute Schmid

in cooperation with
Fraunhofer-Institute for Integrated Circuits IIS/EZRT
Department for Contactless Test and Measuring Systems
supervised by Oliver Scholz and Franz Uhrmann

March 31, 2018

UNIVERSITY OF BAMBERG

Abstract

Faculty of Information Systems and Applied Computer Sciences

Information Systems

Master's Degree

Towards automated segmentation of plants – An exploration of deep neural networks for leaf isolation in agricultural crops

by Andreas Foltyn

Plant phenotyping describes the quantitative analysis of observable plant traits and is gaining importance especially for effective plant breeding. Increasingly, the manual examination of plants is replaced by image-based methods, for which raw data often has to be segmented into semantically coherent parts, such as leaves. To this end, this thesis examines how deep neural networks can be used to segment 3D point clouds of plants, semantically as well as instance-base. In semantic segmentation, the plant is divided into distinct classes, while instance segmentation also distinguishes between individual instances of these classes. For this purpose, two approaches were selected for each case, which were compared with each other. The results show that for semantic segmentation existing methods are already performing well. Although satisfactory results have been achieved for instance segmentation, this task still poses a challenge and requires further research.

Contents

List of Figures

List of Tables

List of Abbreviations

ANN	**A**rtifical **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
CVPPP	**C**omputer **V**ision **P**roblems in **P**lant Phenotyping
DiC	**D**ifference **i**n **C**ount
FCN	**F**ully **C**onvolutional **N**etwork
IoU	**I**ntersection **o**ver **U**nion
LSTM	**L**ong **S**hort **T**erm **M**emory
MLP	**M**ulti **L**ayer **P**erceptron
PReLU	**P**arametric **Re**ctifier **L**inear **U**nit
R-CNN	**R**egion-based **C**onvolutional **N**etwork
ReLU	**Re**ctifier **L**inear **U**nit
RNN	**R**ecurrent **N**eural **N**etwork
RoI	**R**egion **o**f **I**nterest
RPN	**R**egion **P**roposal **N**etwork
SBD	**S**ymmetric **B**est **D**ice

Chapter 1

Introduction

1.1 Motivation

In light of an increasing global population and pressures of climate change food security is one of the biggest challenges the world faces. It is estimated that the food production has to increase at least 50 percent until 2050 to meet the global needs (Liu, 2017, p.46). Thus there is a need to cultivate plants that are capable of adapting to the prevailing environmental conditions in order to ensure efficient and sustainable agriculture. An important step to this goal is to find property-determining factors in the genome of plants that predict the development of preferred traits, such as increased resistance to drought. For this purpose, the genetic information and also the plant's phenotype, i.e. its observable characteristics, have to be quantitatively measured in order to correlate the genetic factors with these traits effectively. Both processes are also referred to as genotyping and phenotyping respectively. Even though plant genotyping is well developed, phenotyping is still lacking behind and thus creating a bottleneck (Furbank and Tester, 2011).

In recent years, however, traditional plant phenotyping, which is comprised of manual examination of plants, has increasingly been replaced by non-invasive, image-based methods. These can be based on various modalities leveraging simple 2D imaging techniques, MRIs or even three-dimensional (3D) reconstructions of plants. The 3D representation of a plant is especially useful to extract a broader spectrum of parameters in contrast to simple images. Point clouds are a common form to represent 3D data and described as unordered sets of data points in a three-dimensional space.

Nevertheless raw data representations have to be pre-processed in order to extract meaningful parameters that are useful to quantitively measure the plants phenotypic traits. A common practice in this process is to segment the plant in semantic coherent parts, e.g. segmenting all leaves of a plant to measure the leaf sizes. This task can be described as *semantic segmentation*, if the only goal is to assign class labels to certain regions of a plant. *Instance segmentation* on the other hand also distinguishes between intra-class instances, which is a preferred goal. Closing this semantic gap by using traditional image processing methods is still considered to be a major challenge, as many methods can only be used to a limited extent for automation (Minervini et al., 2015).

Deep learning, however, gained much popularity across many academic disciplines in recent years and has been used in computer vision successfully to produce state-of-the-art results for various tasks. It is described as the application of multiple processing layers to produce multiple levels of representation. It is therefore capable of learning higher level representations of raw data, that can be used for the intended task, e.g. classification of an image. The most common realization are artificial neural networks (ANN) and especially convolutional neural networks (CNN) for processing data with a grid-like topology, e.g. images. Several publications have shown the effectiveness of CNNs for instance segmentation of plant leaves on images (e.g. Ren and Zemel, 2016). But there is a lack of research in applying deep neural networks to 3D representations of plants.

Deep learning in 3D space is currently an emerging topic since there are several limitations compared to images that are currently not solved entirely. This mainly involves the increased computational costs or the need for new approaches to process 3D representations such as point clouds or meshes. As a result, there is a need for research to investigate the current possibilities of deep learning in the 3D domain with respect to plant phenotyping.

1.2 Objectives

This thesis is intended to address the aforementioned research gap and examine deep neural networks that can be used for processing 3D representations of plants. More specifically, it is investigated how to segment leaves of plants that are represented as point clouds. In particular, this work should have an explorative character and thus not focus explicitly on one approach, but rather compare different approaches with each other in terms of defined tasks and performance measures. For this reason, a more in-depth optimization and analysis of individual models is not carried out, since the work should rather clarify the fundamental possibilities and limitations of these approaches.

In deep learning the size of the data set can also be a limiting factor that restricts the generalization of models. Since labelling 3D data is more expensive than simple images, it is also important to investigate possible ways to address this bottleneck. Consequently, the benefits of data augmentation, which can be used as a regularization technique, are to be examined explicitly. As already mentioned applying deep neural networks in 3D space is a non-trival task. Thus, this work should also deal with the current state of research in this area and explore current possibilities and limitations. Therefore this thesis addresses the following research questions:

1. How can deep neural networks be used for the segmentation of point clouds of plants?
2. How do selected approaches perform regarding semantic segmentation of point clouds of plants?

3. How do selected approaches perform regarding instance segmentation of point clouds of plants?
4. How can data augmentation contribute to the improvement of the performance?

1.3 Outline

This thesis is structured as follows:

Chapter 2 - Background and Related Work The thesis first provides a solid foundation of the relevant concepts and describes the current state of research. In doing so the basics of deep neural networks and the currently prevailing methods for semantic segmentation, object detection and instance segmentation using deep neural networks are introduced. Furthermore this chapter goes deeper into current research for deep learning in the 3D domain. Current possibilities and limitations are examined to give a solid foundation to understand the decisions made in course of the thesis.

Chapter 3 - Methods This chapter provides the experimental design, data set and evaluation metrics used in this thesis. Furthermore the selected approaches that are used for semantic and instance segmentation are introduced.

Chapter 4 - Evaluation and Results This chapter describes the specific experiments with their results, building on the general framework previously defined.

Chapter 5 - Discussion The thesis ends with a discussion of the results and the limitations of this work. In addition, an outlook is given on possible research directions.

Chapter 2

Background and Related Work

This chapter provides the necessary background theory for the concepts used in this thesis. Therefore this section first goes into the underlying task of computer vision in phenotyping. Afterwards artificial neural network are introduced. Convolutional neural networks are explained as an important concept for computer vision. Recurrent neural networks will also be explained, as they are used in this work. Also an overview of several CNN architectures that are used for various computer vision tasks is provided. In the last section, different approaches are presented to use deep neural networks for the 3D domain.

2.1 Computer Vision in Plant Phenotyping

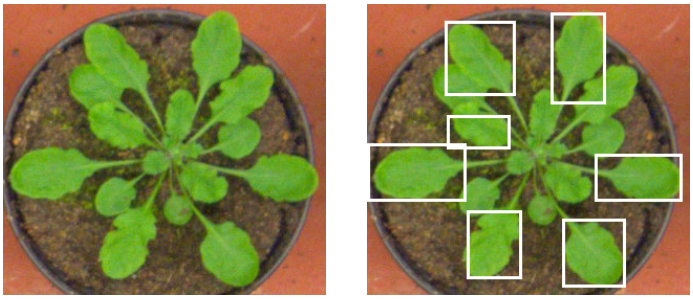
2.1.1 Computer Vision Tasks

At the beginning, the most important tasks of computer vision are introduced in order to provide a concrete understanding of the underlying tasks of this thesis. In figure 3.2, these tasks are illustrated.

Classification The classification of images is understood as assigning a previously defined class Y to an input X. This is one of the core problems on which other computer vision tasks can be build upon.

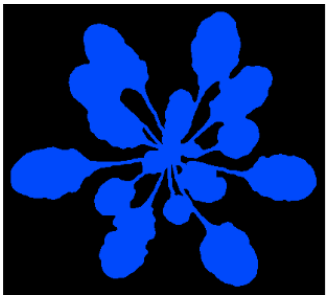
Object Detection This task goes one step further and includes the classification and localization of individual objects. Typically, a bounding box is used that defines the coordinates in which the object is located. The problem is, however, that the object is relatively coarsely isolated. This is not a problem for many tasks, but if you want to determine the area of an object a mere bounding box would be too imprecise.

Semantic segmentation This can be understood as a pixel-precise classification. An image should therefore be divided into individual semantic classes with pixel precision. However, this task does not distinguish between individual instances of a class, as it is the case with object detection. If objects of individual



(A) Original

(B) Object detection



(C) Semantic segmentation

(D) Instance segmentation

FIGURE 2.1: Computer vision tasks illustrated on a plant image from CVPPP leaf segmentation data set (Scharr et al., 2016).

classes are far apart, the process of creating individual instances is relatively straightforward. If many instances of a class are very close to each other, e.g. a crowd of people, it is almost impossible to distinguish between individual objects with a mere semantic segmentation.

Instance Segmentation The two aforementioned problems of object detection and semantic segmentation are now addressed by instance segmentation. This can be understood as combining these two tasks and is at the same time the most difficult task that is presented here. A pixel-accurate segmentation should be made with a distinction between the individual object instances.

2.1.2 Challenges

As already explained, image-based methods are now increasingly being used to measure plants quantitatively. It is currently possible to record plants in a wide variety of ways leveraging different modalities. In the simplest case, pictures can be taken. This is done under laboratory conditions or in the field with hundreds of plants in varying conditions. This means there are requirements for different throughput scales and various conditions under which images are made. However, traditional image analysis can only partially meet these requirements even under laboratory conditions (Minervini et al., 2015).

In this context, the segmentation of individual leaves of a plant plays an important role. This problem can be defined as multi-instance segmentation. It is a relatively difficult task, as leaves may vary in color, shape and size. Often they do not have clear boundaries and therefore occlusions may occur. To solve this problem, the *Leaf Segmentation Challenge* (LSC) of the *Computer Vision Problems in Plant Phenotyping* (CVPPP) workshop was formed in 2014, which also published a benchmark dataset with annotated images for leaf segmentation (Scharr et al., 2016). While semantic segmentation has already yielded good results, instance segmentation still poses a challenge. Traditional image processing techniques rely heavily on complex pipelines that do not provide robustness for the aforementioned requirements. Only in recent years some deep neural network architecture (e.g Ren and Zemel, 2016) have been able to deliver significantly better results on this benchmark data set underlying the effectiveness of deep learning for multi-instance segmentation under these challenging conditions.

In addition to images it is possible to create 3D reconstructions of plants to extract more accurate quantitative measures such as organ-wise traits, e.g. accurate leaf size or branching angle. Currently there are several approaches that allow 3D reconstructions of plants, such as LiDAR, time-of-flight cameras or RGB-D cameras (Liu et al., 2017). While some of these approaches have different areas of application and quality levels, these reconstructions are usually represented as point clouds. The representation of a point cloud is a set of points $\{P_i | i = 1, ..., n\}$, where each point P_i is a vector of its (x, y, z) coordinate and can also contain extra feature channels such as colors, normals or annotations. Thus a plant can be displayed as a set of points in 3D space, possibly with the corresponding color information, if the reconstruction technique allows it. There are some traditional methods for segmenting point clouds that can be applied in this domain. Region-based methods, for example, use neighborhood information from the points to find isolated regions with similar properties (Nguyen and Le, 2013). Model-based methods, on the other hand, use geometric primitive shapes, such as circles or planes, to compare parts of the point cloud and find areas. Graph-based methods, on the other hand, view the point cloud as a graph and can use various algorithms based on it, such as k-nearest neighbors. However, none of these methods are robust enough for real-world applications (Nguyen and Le, 2013, p.229). Therefore, the consideration is to apply deep neural networks in this context as well.

2.2 Artificial Neural Networks

The concept of artificial neural networks (ANN) is used as a collective term for models whose structure is loosely inspired by the neuronal structure of the mammalian cortex. The idea behind it was introduced by McCulloch and Pitts (1943) and was implemented and further refined in the following decades. Typically, these are interconnected processing units that are built up in layers, whereby input data can be transformed into a new representation. By stacking layers, high level representations of data can be generated from simple representations.

Consequently, artificial neural networks represent the canonical realization of deep learning. In this section, in addition to the general concepts of ANNs, *multilayer perceptrons* (MLP) are introduced, which represent the quintessential structure of ANNs, but do not play a major role in the field of computer vision. Convolutional Neural Networks, on the other hand, are widely applied and are also predominantly used in this work. Hence a whole section is dedicated to them.

2.2.1 Deep Learning

The discipline of machine learning generally deals with training models using data in order to generalize to unseen data. A crucial prerequisite for good performance is the data representation. Consequently, the field of representation learning emerged, which extends the training of models to the representations. Deep learning can be understood as a general framework for models that are built up by multiple processing layers that enable them to learn hierarchical representations at different levels of abstraction (LeCun et al., 2015, p.436). The core idea here is to model complex representations from simple ones, which makes it possible to solve correspondingly complex tasks. In this context distributed representations also play an important role (Hinton, 1986). This means that these simple representations can be shared, such as a corner can be part of the image of a face or a car. The realization of this idea is usually implemented by artificial neural networks. Of course, the question arises why deep learning, which in itself is not a new concept, has only experienced such an upswing in recent years. Three major factors can be identified. An important element is the increase in annotated data that is publicly available. It can generally be seen that scale drives deep learning. This is expressed on the one hand in the amount of data, but also in the size of the models, which can be scaled to effectively use this amount of data. Consequently, a lot of computational power is required to process this. Therefore the increasing computational performance and especially the progress of graphic processing units (GPU) is an important second factor in the recent past of deep learning. The last reason is the improvement of certain algorithms. This development will be addressed in the next sections.

2.2.2 Multilayer Perceptron

Multilayer perceptrons, or feedforward neural networks, are used to approximate functions. In case of classification $y = f(x)$ maps the input x to the output label y. MLPs define a mapping $y = f(x, \theta)$ and learn the parameters θ during training. The most essential component is the neuron, as shown in figure 2.2. It is a simple processing unit that has an input x and output y. A dot product is performed with the input values and the corresponding weights w, to which a bias b is added. This value is passed through an activation function σ, which defines the output:

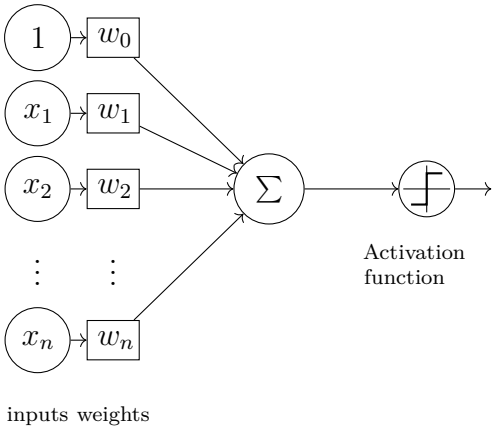


FIGURE 2.2: A single neuron that performs a dot product with weights w_i and the inputs x_i, where $x_0 = 1$ is the bias. The activation function is applied after the dot product.

$$y = \sigma(wx + b). \tag{2.2.2.1}$$

A multilayer perceptron consists of layers composed of such neurons. The number of neurons determines the size of the layers. The outputs of each layer is used as the input of the next one. If each output is now connected to each input of a subsequent layer, it is called a *fully connected layer*. The network consists of a *input layer*, that represents the input data. This data is passed through some *hidden layers* and than a output is created in the *output layer*. A cost function $J(\theta)$ quantifies the prediction error that a network makes. This is typically done by comparing the output of the network with the ground truth of the input data. Thus the goal of training is to minimize the cost function by tweaking the parameters θ of the network.

2.2.3 Activation Functions

Neural networks are especially used to approximate non-linear functions. If individual linear functions are combined, a linear function is obtained again. Therefore it is necessary to use a non-linear activation function. Sigmoid is a classic activation function that was used often in the past. But it is hard to train deeper neural networks using this function.

In the recent past, however, it was shown that the use of the rectified linear unit (ReLU) activation function significantly improves the training of deep neural networks and thus is considered the most popular activation function nowadays (Glorot et al., 2011). However there exists the problem of "dead" ReLUs, when the activation function outputs always the same value for an input. Since its gradient is zero in this case it is unlikely to recover. Therefore leaky rectified units can be used. These allow a small, non-zero gradient when a unit is not active by using a output of $y = 0.01x$ if $x < 0$. Parametric rectified units (PReLU) are similar to leaky ReLUs, but the parameter α is trainable.

Sigmoid

$$y = \sigma(x) = \frac{1}{1 + exp^{-x}} \tag{2.2.3.1}$$

Rectified Linear Unit (ReLU)

$$y = R(x) = max(0, x) \tag{2.2.3.2}$$

Leaky Rectified Linear Unit

$$f(x) = \begin{cases} x, & \text{if } x > 0. \\ 0.01x, & \text{otherwise.} \end{cases} \tag{2.2.3.3}$$

Parametric Rectified Linear Unit (PReLU)

$$f(x) = \begin{cases} x, & \text{if } x > 0. \\ \alpha x, & \text{otherwise.} \end{cases} \tag{2.2.3.4}$$

2.2.4 Optimization

Optimization is the process of changing the parameters θ of a network in order to minimize the cost function $J(\theta)$ (Goodfellow et al., 2016, p.82). Thus the performance measure, that is used to evaluate the model, is only optimized indirectly. It is important to note the difference in literature regarding a cost and a loss function. Generally the loss function L only describes the prediction error a network makes per example. The cost function $J(\theta)$ on the other hand describes the average over the training set:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}, \theta), y^{(i)}) \tag{2.2.4.1}$$

However, most authors use both terms interchangeably and therefore this work will also use the loss function to describe the average error over the training set. The most common strategy to optimize neural networks is to calculate the gradient of the cost function $\bigtriangledown_\theta J(\theta)$ and use its opposite direction to update θ in order to converge to a (local) minimum. The learning rate η determines the step size to reach this minimum. A gradient is defined as as vector of partial derivatives of parameters in respect to the cost.

This process is also known as *gradient descent*. The algorithm to compute the gradient efficiently is called *backpropagation* (Rumelhart et al., 1986). After the information has been propagated through the network in the first step and a scalar $J(\theta)$ has been calculated, this error can flow back in the backward pass by using backpropagation in the network to determine the gradient. It is important to mention that the cost function of a deep neural network is highly non-convex. Therefore it is not guaranteed to converge to a global minimum. However, it

was shown that saddle points are a much more profound problem since they slow down training drastically and give the illusion of the existence of a local minimum (Dauphin et al., 2014). Moreover converging to local minima in deep neural networks often leads to a satisfying parameter optimization, whereas reaching a global minimum could cause overfitting (Choromanska et al., 2014). Various types of optimization algorithms are used in practice. Two common algorithms used in modern deep neural networks are as follows.

Stochastic Gradient Descent (SGD) Calculating the gradient of a full training data set can be in practice not feasible because of computing limitations. Alternatively it is possible to use only one sample to compute the gradient. This method is called Stochastic Gradient Descent (Robbins and Monro, 1951):

$$\theta = \theta - \eta * \bigtriangledown_\theta J((x^{(i)}; \theta), y^{(i)}) \tag{2.2.4.2}$$

where η describes the learning rate that is used to update the parameters. In practice SGB is also used to describe the concept of *mini-batch gradient descent* that uses small subsets $(i : i + n)$ of the training data instead of a single sample i. This approach is one of the most common one used nowadays to train deep neural networks. Nevertheless vanilla SGD has a couple of challenges, such as determining the right learning rate.

ADAM Adaptive Moment Estimation is able to compute an adaptive learning rate and thus tweaks the learning rate for each parameter during training time. It is only required to set the initial learning rate (Kingma and Ba, 2014).

2.2.5 Regularization

An essential aspect of machine learning and neural networks is to achieve good generalization of models by avoiding overfitting. Especially models with many parameters are prone to just "memorize" the training data and thus achieve considerably worse results with previously unseen data. In order to reduce this problem, different strategies can be used, which are described as regularization. In the following some prominent techniques are explained.

Early Stopping A validation dataset is used to approximate the generalizability of the network during training time. The model parameters are saved every time the validation error improves. This approach thus uses a model with the least validation error and not the latest parameters, that could be significantly overfitted.

Dropout A common regularization technique is dropout (Srivastava et al., 2014). A predefined amount of neurons are dropped per layer during training time at each gradient descent step. That means that these neurons do not contribute in the forward pass of the network and are not updated in the backward pass. In this way the network has to learn more robust features. While testing, however, dropout is not applied.

L2 Parameter Regularization This approach is also known as weight decay and penalizes large weights in the network by adding a regularization term $\frac{1}{2}\lambda w^2$ to the cost function for every weight w, where λ is the regularization strength. Therefore the previously defined cost function would be modified in this way:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)}, \theta), y^{(i)}) + \sum_{w} \frac{1}{2}\lambda w^2 \tag{2.2.5.1}$$

Data Augmentation Probably the most common way to improve the ability to generalize is to use more data, so the network can't memorize all training data and is forced to create generalizable features. The way data can be augmented depends heavily on the underlying data. Images can simply be scaled or rotated. However, it should be noted that it may lead to poorer performance if the data augmentation produces undesired side effects, such as rotation giving an image a different meaning than the label suggests.

2.2.6 Batch Normalization

Batch normalization (Ioffe and Szegedy, 2015) is used to normalize activations in deep neural networks. As the name suggests, this is done per mini-batch. In order to explain its use, the authors introduce the term *Internal Covariate Shift* and define it as "the change in the distribution of network activations due to the change in network parameters during training" (Ioffe and Szegedy, 2015, p.2). This phenomenon can be described as follows. A layer L is dependent on its previous layer and takes the output of $L-1$ to model it accordingly. After some training iterations, the parameters in the previous layer change and so does the statistical distribution of the input x for layer L. Now L is forced to adapt to this new distribution. This process can significantly slow convergence. Batch normalization is now placed between the layers to normalize the input for L. This can improve the convergence rate of the network. Higher learning rates can also be used and parameter initialization is no longer as important. This normalization step can be described as follows:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}, \tag{2.2.6.1}$$

where $x^{(k)}$ is the output of layer $L-1$, which gets normalized by the mean and variance of its mini-batch and thus creating with $\hat{x}^{(k)}$ the input for layer L

2.3 Convolutional Neural Networks

Convolutional neural networks (CNN) are a special kind of neural network, that are used to process data with grid-like topology, e.g. images. This network was

first introduced by LeCun (1989) and was inspired by the way the animal visual cortex processes information. Currently CNNs are the most important networks for computer vision tasks. In the following the building blocks are introduced. The general concepts of neural networks, that were introduced in section 2.2 also apply to CNNs. Figure 2.3 shows the building blocks of a typical CNN.

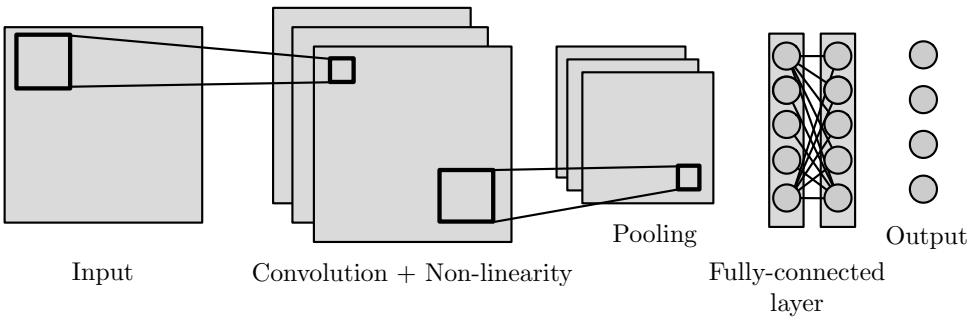


FIGURE 2.3: This illustration shows building blocks of a CNN used to classify images.

2.3.1 Convolution

The most important part of a CNN is the convolutional layer. The major difference to the fully-connected network introduced in section 2.2.2 is the use of local connection. Instead of connecting each input to each output, a *kernel* or *filter* is used to convolve with the input to create a *feature map*. Formally this can be expressed as

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-m), \tag{2.3.1.1}$$

where I is the image with dimensions m and n, K represents the kernel and S is output at location i and j.

There are three important key characteristics that come with the use of convolutions (Goodfellow et al., 2016, p.324) .

Sparse Interaction Multilayer perceptrons have all input units interact with all output units, whereas CNNs use a kernel that is smaller than the input. Therefore it only interacts with a small part of the input at a given time. This has consequences regarding the memory and computational complexity. Hence, the kernel only has to store a small set of weights compared to MLPs.

Parameter Sharing MLPs have one weight for one specific input. It is only used at one place in the networks. CNNs on the other hand have a kernel with k weights that are used across one input.

Equivariance The output of the convolution operator changes with certain transformations to the input in the same way. That means if an image is

translated by one pixel, the output is also translated by one pixel. Therefore it does not matter if we apply the translation to the input or output.

The main parameters of the convolution operator are as follows:

Stride It describes the size of the convolution step. If a convolution operation has a stride of 1, the kernel moves one unit at a time. Other sizes can also be used to downsample the feature maps faster.

Padding To prevent downsampling, however, padding can be used. This adds values to the feature map's borders. These can be zeros or values of the border.

Dilation In order to increase the receptive fields dilated convolutions can be applied (Yu and Koltun, 2015). It is a technique to exponentially increase the receptive field of convolutions without increasing the number of parameters by introducing gaps. Some publications also call this operation *à-trous* convolution.

2.3.2 Pooling

The pooling layer is used to downsample a feature map by using summary statistics in order to obtain an invariance for small translations of the input. By reducing the spatial size one can also reduce the amount of parameters in the network and hence increase the ability to generalize. Max pooling is an often used pooling operation, that obtains the maximum value of rectangular non-overlapping neighborhoods of an input (Zhou and Chellappa, 1988). However, the benefits of using a pooling layer in a CNN have been increasingly questioned in recent years. Springenberg et al. (2014) showed that max-pooling can be replaced by simply using an increased stride in the convolutional layer without loss of accuracy. Also Sabour et al. (2017) suggested the replacement of the pooling layer by using a novel architecture called *Capsule Network*.

2.3.3 Transposed Convolution

This operation can be thought of as the opposite the convolution operation and hence it is also called *deconvolution*. But it is important to note, that the later term is not entirely accurate but it is often used in practice. Transposed convolution can be considered as a trainable upsampling operation. Instead of mapping a region of the input to a single value, this operations maps a single value to a region of the output by reversing the convolution operation. This operation is often used in encoder-decoder structure in order to upsample the downsampled feature map to the dimensions of the input.

2.3.4 Transfer Learning

The weight initialization of a network plays a critical role for the training. In the simplest case one would use a distribution, e.g. normal distribution, to initialize it. Transfer learning on the other hand describes the use of a pre-trained network, that was trained for one setting and is exploited to improve generalization in another setting (Goodfellow et al., 2016, p.526). The concept is based on the idea that the first layers of a CNN usually learn very generic features that can also be used in a new context. This usually works in such a way that you initialize layers 1 to l with these pre-trained weights and then fix them for training. Then the remaining deeper layers are trained to convergence. In a fine-tuning step all layers are trained again, usually with a lower learning rate. Transfer learning often plays a critical role if there are not enough training data.

2.4 Recurrent Neural Networks

In the following, recurrent neural networks (RNN) will be presented as the third important group of artificial neural networks. Although the main field of application is the processing of sequential data, RNNs have also been used more frequently in computer vision in recent years. This work also relies on the application of a recurrent network, which is why the background is explained as well.

2.4.1 Recurrent Neurons

Feedforward networks assume that each input is independent of each other. However, if a network is fed with sequential data, e.g. a time series, one can assume that the prediction at time t should also depend on the previous inputs. Recurrent neural networks address precisely this issue. They have a feedback connection which connects a neuron to itself or a previous layer. Thus neurons can be thought of having a "memory", which enables them to use information from previous time steps. RNNs are therefore particularly useful to predict a next data point, e.g. time series forecasting. This memory or hidden state h is therefore depended on the previous state h_{t-1} and the current input x_t:

$$h_t = f(h_{t-1}, x_t) \tag{2.4.1.1}$$

Figure 2.4 shows an unfolded RNN. However, a RNN can take different input-output configurations. So for example, there is only one input at t, but the network generates multiple outputs for the next steps by its feedback connections.

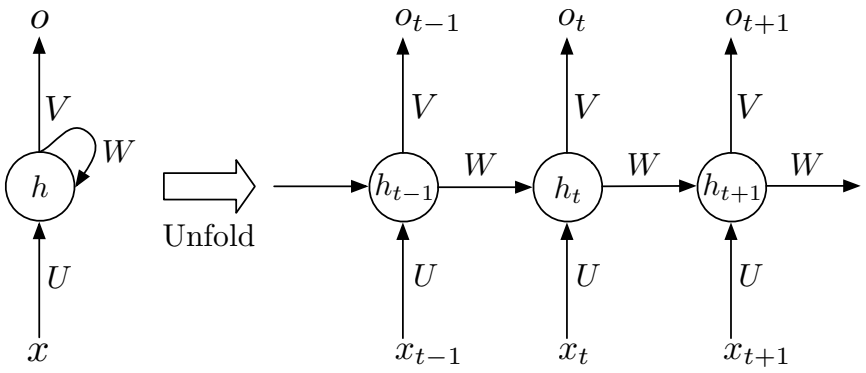


FIGURE 2.4: An unfolded recurrent network is shown. U, V, W are weights used for the inputs x, outputs o and hidden states h, respectively.

2.4.2 Long Short-Term Memory

However, the vanilla recurrent neural network presented previously is relatively rarely used in practice, as it has the problem that information is already forgotten after a few time steps and the problem of vanishing gradient can occur during training. Hochreiter and Schmidhuber (1997) tried to solve this problem by introducing Long Short-Term Memory (LSTM). The classical RNNs are extended by modifying the repeating cells. The individual components are briefly described in the following. A LSTM network consists of individual LSTM cells. Figure 2.5 shows the structure of a cell at time t. Each unit has an input and an output and is connected to itself. The basic concept used in LSTM are gates, which are used to decide which information should be kept or forgotten. The LSTM unit receives the current input x_t, the previous cell output h_{t-1} and the cell state C_{t-1} as input. However, not all inputs are used for each gate.

The forget gate f_t is a sigmoid layer that looks at the previous output h_{t-1} and input x_t in order to output a number between 0 and 1 for the previous cell's state C_{t-1}. The input gate i_t is used to decide which values are to be updated. Now a new potential update value g_t for $C_t - 1$ is to be generated in the *tanh* gate. To update the old cell state C_{t-1}, it is first multiplied by f_t. This decides what information should be forgotten. On the other hand, i_t is multiplied by the new cell candidates g_t to scale how much each state value is to be updated. The new cell state C_t now results from the addition of both operations. The output h_t is created by multiplying the output gate o_t by the current cell state squashed by the *tanh* operation. For all gates a bias b is added. The LSTM cell can therefore be described as:

$$
\begin{aligned}
f_t &= \sigma(b_f + x_t U_f + h_{t-1} W_f) \\
i_t &= \sigma(b_i + x_t U_i + h_{t-1} W_i) \\
g_t &= tanh(b_g + x_t U_g + h_{t-1} W_g) \\
o_t &= \sigma(b_o + x_t U_o + h_{t-1} W_o) \\
C_t &= f_t * C_{t-1} + i_t * g_t \\
h_t &= o_t * tanh(C_t).
\end{aligned}
\tag{2.4.2.1}
$$

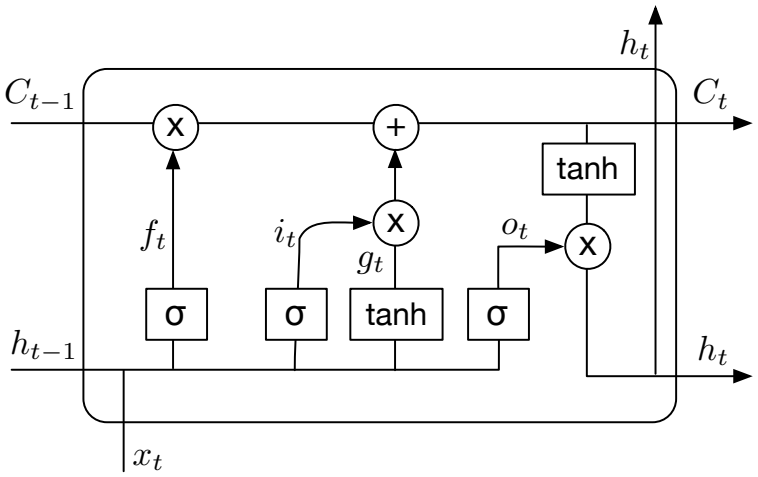


FIGURE 2.5: This illustration shows the components of a single LSTM cell.

In addition to this classic structure, there are many other variants. Also the question arises whether there are combinations of neural network types that can benefit from each other's properties. In fact, there are combinations of CNNs and RNNs. The simplest approach would be to use a CNN for the feature extraction, followed by a RNN. e.g. for image caption. ConvLSTM (Shi et al., 2015), on the other hand, is a variant of the LSTM that has been optimized for processing images, since the fully-connected layers of the LSTM are not useful for this task. The idea behind this is that all fully-connected layers of the LSTM gates are replaced by convolution operations.

2.5 Architectures

Now that various types of neural networks have been explained, the architectures which can be realized using them are described. First, the base architectures for CNNs are addressed. These can be used as a basis for more complex architectures or for classification of images. Afterwards the architectures that can be used for the already defined computer vision tasks are examined. Consequently, this section aims to give an understanding of fundamental architectures and how

they can be modified or combined for different tasks. To this end, more detailed explanations of any approaches that may be necessary for this work or for general understanding are presented. Otherwise, any other approaches are only briefly mentioned in order to show the range of possibilities. Architectures that are implemented and evaluated as part of this work are mentioned in this section, but ultimately described in more detail in chapter 3.

2.5.1 Base Architectures

Base architectures of CNNs are commonly known architectures that are used in practice as feature extractors and then are followed by a separate logic for corresponding tasks. In the simplest case, a fully connected layer is used to classify images. In addition, for these architectures it is often possible to get pre-trained weights, which is very useful if you have small data sets or want to speed up the training. Three well-known architectures are presented below. However, there are many more, so this is only meant to be an excerpt.

AlexNet AlexNet (Krizhevsky et al., 2012) is the deep CNN that won the ILSVRC-2012 with a top-5 accuracy of 84.6% and outperformed the best traditional image processing technique by a margin of over 10%. While this architecture is relatively simple, it popularized some changes to the classic LeNet CNN. Above all, the advantages of ReLU and max pooling were an important change. It consists of five convolutional layers, three max-pooling, three fully-connected layers and dropout.

VGG-16 This model was presented by the Visual Geometry Group (VGG) of Oxford University (Simonyan and Zisserman, 2015). There are different variants of VGG, each consisting of 5 blocks. After each block, the size of the feature map is cut in half by max pooling. Depending on the variant, up to 5 convolutional kernels can be connected in series in the blocks. The most popular variant is the VGG-16 with 16 layers. However, the network contains many parameters and is relatively slow to train.

ResNet While it is possible to train relatively deep networks, the problem of vanishing gradient exists with a normal architecture, e.g. VGG, if you put more and more convolutional layers in series. To work around this problem, He et al. (2016) use residual blocks to build the first deep neural network with 152 layers that was successfully trained. Not only is the network deeper, it also has considerably fewer parameters than a comparable VGG network. Another popular variation is ResNet-101 which has 101 layers. The residual function $F(x)$ is a skip connection that combines the input of a block with its output. Consequently, an unchanged representation of the previous block can be included. Thus, multiple blocks can be skipped during backpropagation, which makes it easy for the signal to flow through very deep networks. An interesting observation

is the comparison of the additive component with the functionality of LSTM. For CNNs and RNNs this seems to have a significant positive influence on the learning process.

2.5.2 Object Detection

Object detection makes a detailed statement about the localization of objects in images, e.g. by using a bounding box. Although this work is mainly concerned with segmentation, object detection is an essential part of many algorithms for instance segmentation. Therefore, possible architectures should be covered here. Current architectures used for object detection can be split into two broad categories, two-stage and unified architectures.

The two-stage architecture was first introduced by Girshick et al. (2014) and is also known as region-based CNN (R-CNN). First object proposals are hypothesized that identify possible regions of interest (RoI). These regions are then resampled and classified using fully-connected layers. Since then, this approach has been refined several times in order to increase speed and accuracy. Faster R-CNN (Ren et al., 2017) is the current canonical architecture for object detection and used as meta-architecture for many follow-up publications (e.g. He et al., 2017). Therefore, it is appropriate to address this architecture in more detail, since it involves a lot more parts than networks used for image classification.

Faster R-CNN As already explained, the first stage of many networks usually consists of the use of a base architecture. Also in this case ResNet or VGG-16 could be used to create a feature map of an image as shown in figure 2.6 This feature map is used as the basis for all further operations. First it is used in the region proposal network (RPN) to generate object proposals. Two separate maps are created by convolution operations. One is used to determine during inference whether an object is at a position (x, y) on the feature map. The other is used to refine the default bounding box coordinates that should surround the object. The logic to train this prediction is based on sliding a window with fixed dimensions over the feature map and comparing each position with the ground truth of the image. If the window at the position (x, y) has an intersection with an object that exceeds a threshold value, then this position is considered to surround an object. Using this information and the ground truth bounding box coordinates the loss for both feature maps can be calculated and thus be trained. From this stage the object proposals, i.e. bounding box coordinators are taken and passed to the second stage of the network. These object proposals are extracted from the original feature map of the base architecture and reduced to a fixed size in the RoI pooling layer, since they serve as an input for a fully-connected layer that needs a fixed input sizes. Therefore RoI pooling can be considered as adaptive max pooling, where the output has fixed dimensions, which means that the individual pooling regions must have flexible dimensions. Subsequently, these object proposals are classified and refined using fully connected layers.

Using this two stage approach Faster R-CNN's main purpose is to provide good accuracy to the disadvantage of inference speed.

Thus several other approaches try to improve the inference speed in order to be used for real-time applications. Basically, they reduce computing time by avoiding to resample proposals and removing the fully-connected layer of an R-CNN architecture and thus only using one stage. Liu et al. (2016) introduced an architecture that creates many object detections for different scaled feature maps using a similar mechanism as RPN.

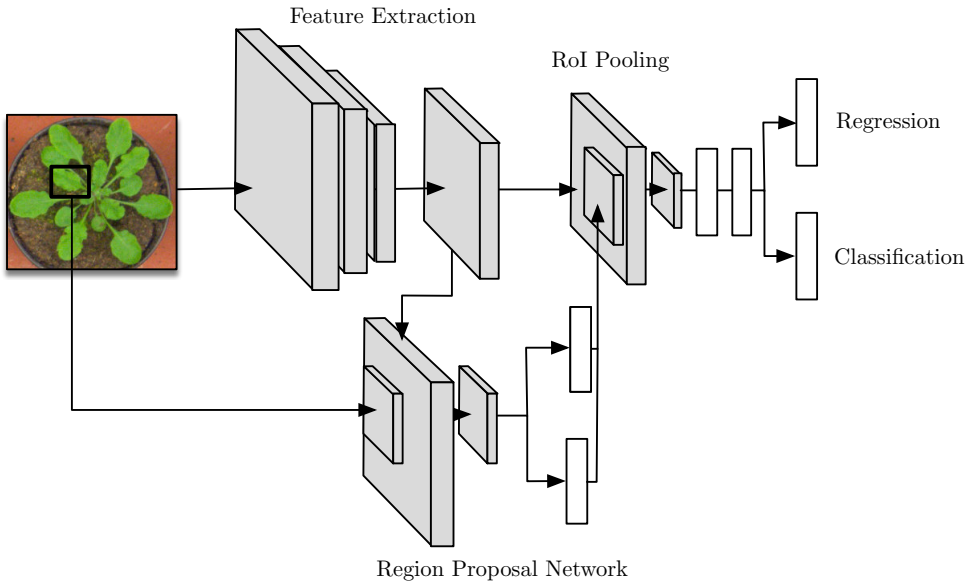


FIGURE 2.6: Faster R-CNN architecture.

2.5.3 Semantic Segmentation

Many of the architectures currently used for semantic segmentation can be attributed to the fully convolutional neural network (FCN) introduced by Long et al. (2015). The basic idea is to take a base architecture and scale the resulting downsampled feature map to the original size of the input using upsampling operators, such as deconvolution kernels. This type of architecture can generally be called an *encoder-decoder*. The bottleneck of this architecture is to scale up the low-resolution feature map, since spatial information is lost in the encoder part. The subsequent publications have therefore been mostly concerned with optimizing the decoder. U-Net (Ronneberger et al., 2015) is a popular architecture, that uses skip connections between each encoder and decoder level to retrieve lost spatial information. Semantic segmentation generally requires a balance between global and local information, which can often be lost in CNNs (Garcia-Garcia et al., 2017, p.10). Consequently, there are several methods to counteract this. Krähenbühl and Koltun (2012) used a *Conditional Random Field* (CRF) to refine segmentation in a post-processing step. This allows low-level image information, i.e. pixels, to be combined with segmentation output score to detect fine details as well as long-range dependencies.

In order to increase the receptive field of convolutions without using pooling dilated convolutions can be used, which was introduced by Yu and Koltun, 2015, who also proposed the use of a multi-scale module to aggregate context of multiple scales. As it already becomes clear, a way to increase the performance for semantic segmentation is to support the aggregation of global and local features. Zhao et al. (2016) have therefore introduced PSPNet, which applies dilation in a ResNet architecture and pools its feature map into four different resolutions by four pooling operations. These are upsampled and then concatenated to create the final segmentation mask. This architecture and DeepLab (Chen et al., 2018) is considered as the current state-of-the-art.

While these publications are mainly concerned with optimization by using various modifications, there are also approaches that take a different approach. Kendall and Gal (2017) have shown that the application of *bayesian deep learning* can be used to optimize semantic segmentation. Bayesian deep learning describes the intersection between deep learning and the bayesian probability theory to model the uncertainty of deep neural networks. In this context, a distinction is made between two uncertainties. *Aleatoric uncertainty* is caused by noise, which makes a model unable to explain something and it is unlikely that it can be reduced by more data. This uncertainty can be caused by object boundaries or objects that are difficult to recognize. This is modeled by modifying the loss function in Kendall and Gal (2017). *Epistemic uncertainty* arises from limited data or knowledge and is modeled by leveraging drop out. Consequently, the possibility to model both uncertainties could offer several possibilities to improve existing models.

2.5.4 Instance Segmentation

Instance segmentation can be considered as a combination of object detection and semantic segmentation. Since instance segmentation is the preferred goal in plant phenotyping and also one of the more difficult computer vision tasks it is important to give a detailed assessment of the current state-of-the-art architectures and how they differ from each other. Therefore categories are used to create a better differentiation. The relevance of the methods is also compared by comparing their performance on benchmark datasets. Microsoft Common Object in Context (MS COCO) is a popular dataset used for evaluating object detection and instance segmentation (Lin et al., 2014). In addition there is the aforementioned CVPPP leaf segmentation benchmark that is perhaps more relevant for this work.

Proposal-based Many current instance segmentation approaches rely on object proposals, e.g. bounding boxes. Therefore their foundations are often grounded in object detection approaches. Dai et al. (2015) proposed a three stage cascade architecture. First objects are proposed in the form of class-agnostic bounding boxes using a RPN. These are than segmented and classified in the following stages. This approach was the winner of the MS COCO 2015 challenge.

He et al. (2017) on the other hand designed an architecture that works in parallel instead of relying on a cascading stages. It can be considered as a Faster R-CNN architecture with an additional segmentation head that works in parallel to the classification head. This architecture is also known as Mask-RCNN and is considered to be the current state-of-the-art. This is emphasized by the fact that the top placed architectures of the MS COCO 2017 challenge are based on the Mask R-CNN structure. However, these approaches could have problems with several instances of the same class that are located within one bounding box.

Proposal-free On the other hand, there are some architectures that are not explicitly based on object detection pipelines. This includes, for example, the use of recurrent neural network. Romera-Paredes and Torr (2016) used a Convolutional LSTM that produces binary segmentation mask for each instance sequentially and thus does not require the use of object proposals. Due to its spatial memory the architecture keeps track of already segmented pixels and thus is able to handle occlusions. This approach is comparable to attention based models, that are commonly used for image caption generation. It performs reasonably well on the CVPPP leaf segmentation dataset and is on par with non-deep-learning approaches, if Conditional Random Fields is used to refine the prediction in a post-processing step. However, the authors suspect that the small data set has a strong impact on the performance. Ren and Zemel (2016) proposed a rather complex model with four major components. A box network consisting of a CNN and a LSTM is used to propose bounding box proposals. These proposals are than segmented in a segmentation network using a semantic segmentation architecture with skip connections. In the last step a scoring network checks if an instance is found and when to stop. An external memory keeps track of segmented objects. This architecture is currently the best performing model on the CVPPP benchmark. Salvador et al. (2017) introduced a simple encoder-decoder architecture that resembles Romera-Paredes and Torr (2016). The main difference is the use of skip connections and ConvLSTMs at each decoder level in order to create higher resolution feature maps and thus not requiring any post-processing steps. The performance can be ranked between the two other recurrent approaches. De Brabandere et al. (2017), on the other hand, proposed a rather different approach, which is particularly focused on dealing with occlusions. The idea behind this is to display each pixel in a n-dimensional feature space, where the goal of training is to bring pixels of an object instance closer together in this feature space and to create distance to pixels of other instances. This feature space can be divided into clusters during testing and thus individual instances can be segmented. To achieve this, the authors used a discriminative loss function similar to the approaches used for face recognition. The performance is on par with Ren and Zemel (2016). While all of these models perform well on the one-class CVPPP dataset they get outperformed on complex datasets by proposal-based approaches like Mask-RCNN.

2.6 3D Deep Learning

This section shows in more detail how deep neural networks can be applied in the 3D domain. In general, 3D representations can be divided into two broad categories: regular and irregular forms. The former has a rasterized, discrete structure, such as images. The latter, on the other hand, can be found in geometric shapes such as point clouds or polygonal meshes. Typical CNNs can be applied to regular structures, whereas point clouds require alternative models, since they have an unordered data representation. A point cloud with N points can have $N!$ permutations of these points without changing their shape. A CNN relies on local structures of data, such as pixels in a neighborhood. Hence, it is not straightforward to apply a convolution operator to a unordered set of points. However, point clouds can also be transformed into other forms of representation, which opens up further possibilities. In the following, three common approaches for applying deep neural networks to three-dimensional data are presented.

2.6.1 View-based Approach

Instead of working directly with three-dimensional data, 2D projections of the model can be created. These views can then be used to train a common 2D CNN. This method has already been used in various publications for classification and segmentation of 3D models. Figure 2.7 shows a multi-view 3D CNN proposed by Su et al. (2015) that uses an end-to-end architecture with view pooling to aggregate all views to one output prediction. An improved version of this approach is currently the best performing model on the ModelNet benchmark dataset for classification of 3D models (Kanezaki et al., 2016). Boulch et al. (2017) showed the effectiveness of this approach regarding semantic segmentation of large scale point clouds of outdoor scenes on the Semantic3D benchmark dataset.

The advantage of this approach is that one can use state-of-the-art CNNs for images. Also there are already huge amount of labelled 2D data such as ImageNet that can be leveraged, e.g. by using a pre-trained model. This enables you to train a model much faster and requires much less annotated training data. However, this approach also has some disadvantages. On the one hand information from a 3D model is lost by using projections. However, this loss of information can be reduced by giving the views additional information such as depth information (RGB-D) or tilt angle in addition to the color channels. There is also the problem of choosing the views, i.e. how many and which view angles. In particular, objects with strong occlusion may not be detected.

Ultimately, the fusion of information plays a decisive role. This is relatively simple when classifying an object or for semantic segmentation. Su et al. (2015) implemented the classification in a end-to-end manner. For semantic segmentation Boulch et al. (2017) had to aggregate all view segmentations and use a per point majority vote. Nevertheless the fusion of instance segmentations is a much more challenging task.

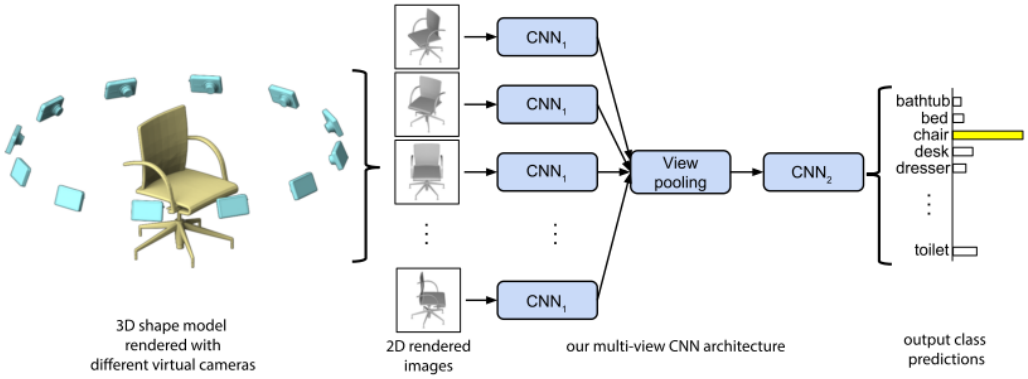


FIGURE 2.7: Multi-view CNN used for 3D shape recognition. Figure extracted from Su et al. (2015).

2.6.2 Volumetric Methods

The volumetric data representation can be considered as a counterpart of images. It is represented in the simpelst case as data points in a three-dimensional grid and is also known as voxel grid. Consequently, a convolutional neural network can be applied directly to this data structure. Popular applications of this structure can be found in medical technology, e.g. MRIs or CTs. Since this approach works in the three-dimensional space directly, it preserves information otherwise lost with a view-based approach. Nonetheless, the transformation of a point cloud into a voxel grid also results in a loss of information, since it involves discretization. Maturana and Scherer (2015) proposed with VoxNet an architecture for classifying 3D objects using a 3D CNN. However, the performance cannot compete with multi-view approaches. This may be due to the rather simple architecture used in VoxNet. For semantic segmentation Milletari et al. (2016) used the aforementioned U-Net architecture to construct a similar architecture for processing MRIs. With VoxelNet Zhou and Tuzel (2017) proposed an architecture for object detection with very good performance on the KITTI benchmark dataset (Geiger et al., 2012) for pedestrian and car detection using LiDAR point clouds. The general structure is based on the one-stage object detection pipeline. But it also leverages PointNets (Qi et al., 2016) to compensate for the information loss during the discretization of the point clouds.

An inherent problem using voxel grids is the computational and memory costs. Therefore many publications only use a rather small resolution. In addition, a voxel grid is a very inefficient form of representation, since many voxels are ultimately unoccupied. Consequently, an attempt was made to reduce the complexity by using sparse representations. Riegler et al. (2016) created a 3D CNN using an unbalanced octree. This results in a highly memory and computational efficient representation. They show that semantic segmentation in particular can benefit from using a higher resolution.

2.6.3 Approaches for Irregular Forms

In contrast to the two approaches mentioned above, this section deals with neural networks capable of processing point clouds directly. As mentioned above, it is not possible to use a common CNN for this case and therefore alternative networks have to be used. Thus, Qi et al. (2016) proposed a novel architecture based on multilayer perceptrons. The key idea is to learn a spatial encoding per point and than aggregate all points using a symmetric function to create a global feature vector. The symmetric function is used to create a permutation invariance since there is no inherent order of the points in a point cloud. PointNet's current architecture is only capable of classification and semantic segmentation of objects. PointNet considers each point individually. This means that local geometric structures are not used. With PointNet++ Qi et al. (2017b) introduced a hierarchical network to include the local neighborhood. Qi et al. (2017a) proposed a proposal-based instance segmentation architecture for the 3D domain based on PointNet. However, the object proposals are based on 2D images, so this can only be used in special cases, e.g. RGB-D images.

Wang et al. (2018) extended the PointNet approach by using special convolutions that take local structures into account by using k-nearest neighbor graphs. With PointCNN, Li et al. (2018) also proposed a generalization of the convolution operator for point clouds.

In recent years, there has been a emerging trend towards applying deep neural networks to the non-Euclidean domains, such as graphs and manifolds. This is also referred to as *geometric deep learning* (Bronstein et al., 2017). Various approaches exist for this. Some try to transform 3D surfaces to 2D patches and then use them in a normal 2D CNN. Others try to extend the convolution operation for 3D meshes. However, these are limited to smooth manifold meshes. While many of these approaches show potential, none can really be used effectively for the underlying problem at the moment, which rules out further discussion and consideration of these methods in the context of this work.

Chapter 3

Methods

This chapter presents the methodology used in this work. First, the general experimental design is explained. The individual components are then described in more detail in this section. The data set is described, then the evaluation measures and finally four architectures that are evaluated in this work.

3.1 Experimental Design

This general experimental design serves as a framework for the individual experiments. For these experiments, the corresponding factors of this design can be adapted. At this point it is important to bring the objectives of this work back into focus. In the previous chapter, possible architectures for the segmentation of point clouds were identified. Now selected architectures are to be compared with each other. In addition, the impact of data augmentation on performance is to be addressed.

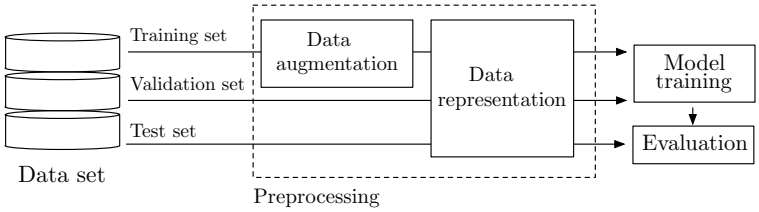


FIGURE 3.1: Design of general experimental framework.

Thus, two major *controllable factors* can be identified: the learning algorithm and data augmentation. In Figure 3.1, these two components can be viewed in the overall picture. In addition to these main objectives, the experiments will also look at the input representation as a controllable factor to determine possible intra-model variance in order to allow a deeper analysis and comparison of the models. Thus, the preprocessing step includes data augmentation and modification of the input representation. At the moment, however, the individual levels of the factors of the pre-processing step should not be dealt with, as these are discussed in the following chapter. It is important to note that the hyperparameters of the selected algorithm can also be considered as controllable

factors. However, these are fixed in this experiment because they were optimized in advance or predefined by the individual author's recommendations.

The *uncontrollable factors* here are in particular the non-deterministic nature of the training process of neural networks and the data set split. The *response variable* represents a performance measure P defined for the task of the model. If the goal would be to find the optimal factor combination to optimize P, a factorial design would have to be chosen, where the optimal configuration is found by grid search. This approach is only partially implemented in this paper, as factor combinations are omitted in some places. However, such decisions are explained in the corresponding part. Therefore the experimental design follows a *fractional factorial design*.

The data set remains the same for all test series. It should be noted that the goal is not to optimize the performance measure based on the training error, but on the generalization error. Consequently, in addition to the training data set, a test set is also required to test for this error. A validation set is used to optimize the algorithm, since the hyperparameters cannot be adjusted to the generalization error of the test set. This ultimately refers to the fact that early stopping is used as regularization for all experiments for which a validation set is needed.

In order to reduce the influence of uncontrollable factors, repeated tests are usually carried out. In this way, the non-deterministic process of training can be reduces by averaging repeated tests with fixed factors. Also *k-fold cross validation* can reduce the impact of the training/test split or the uncertainty of the result for small test sets. It is usually used for small to moderate data sets and not necessary for large data sets. Although only a moderately sized data set is used in this work, k-fold cross validation is not used. The reason is that a large number of models are trained in this work. This takes a lot of time, which is not uncommon for training deep neural networks. This decision corresponds to the requirements of various benchmark data sets, which also have moderately sized data sets of a few hundred samples (e.g. CVPPP).

3.2 Dataset

3.2.1 Criteria

In the context of this work 3D point clouds of plants should serve as data set. These should be annotated accordingly for instance-based segmentation of leaves. The conversion into an annotation for semantic segmentation is trivial and does not have to be separate. Looking at the currently available data sets, however, it is noticeable that no publicly accessible data set with these requirements exists to date. If one relaxes the condition that it must contain plants, then some data sets can come into question. However, a lot have application-specific characteristics, such as multi-modal data sources for the KITTI benchmark data set. Furthermore, they are often perfect 3D models, e.g. ModelNet, which

means that the models tested for them are not transferable to problems faced by 3D reconstructions of plants. Also, many do not have annotation for instance segmentation but only for object detection and often contain many classes. However, the problem of one-class multi-instance segmentation has also motivated the release of the CVPPP benchmark dataset, although there already are many other benchmark data sets for instance segmentation of images. This is a specific problem where many algorithms may perform worse on the other benchmark data sets with many different classes.

Therefore it was decided to create a synthetic data set with the above mentioned conditions. In order to learn features from data that allow a good generalizability, a certain quantity and quality of the data must exist. On the one hand, the data set must be large enough so that the network does not "memorize" the data set. On the other hand, a certain variability in the data set must exist and the annotation of the data should be correct. For example, if a large number of synthetic models are created that differ only slightly from each other, a *data leakage* could occur in which the test data set contains models that are almost identical to models in the training data set.

Nevertheless, there is no general guideline for the quantity of data, as this is highly dependent on the complexity of the underlying problem. For example, you probably need less data to train a model to detect simple geometric shapes than for the detection of highly deformable objects, such as humans. In addition, the quality of the data set is difficult to quantify, so that it should be assessed visually.

Nicotiana tabacum or also known as a tobacco plant was chosen since the plant's structure is simple and the leaves are easily distinguishable from each other visually. It is important to emphasize once again that the results obtained in this work should not be directly transferable to real 3D reconstructions of plants. The focus is on the comparison of models regarding certain computer vision tasks and how these models deal with the corresponding data structure and possible artifacts present in 3D reconstructions. Consequently the focus was on creating a moderately large number of tobacco plants that reflect different possible quality levels of 3D reconstructions. To make this dataset easier to reference, it is referred to in the following as `Tobacco` dataset.

3.2.2 Generation Process

For the technical implementation the free open-source software *Blender* was used, in which the plants were modeled by hand. With the L-system (Lindenmayer, 1975) exists a formalism that is already successfully used to model complex plants on different abstraction levels. However, this work did not make use of this, as the tobacco plant is relatively easy to model. In Blender, simple tobacco plants were initially modeled in pots using real tobacco textures. In the next step, these models were provided with a skeleton to animate the plants according to various parameters. During the animation the size, position, shape, angle of inclination and surface of the leaves could be changed. The length and width of

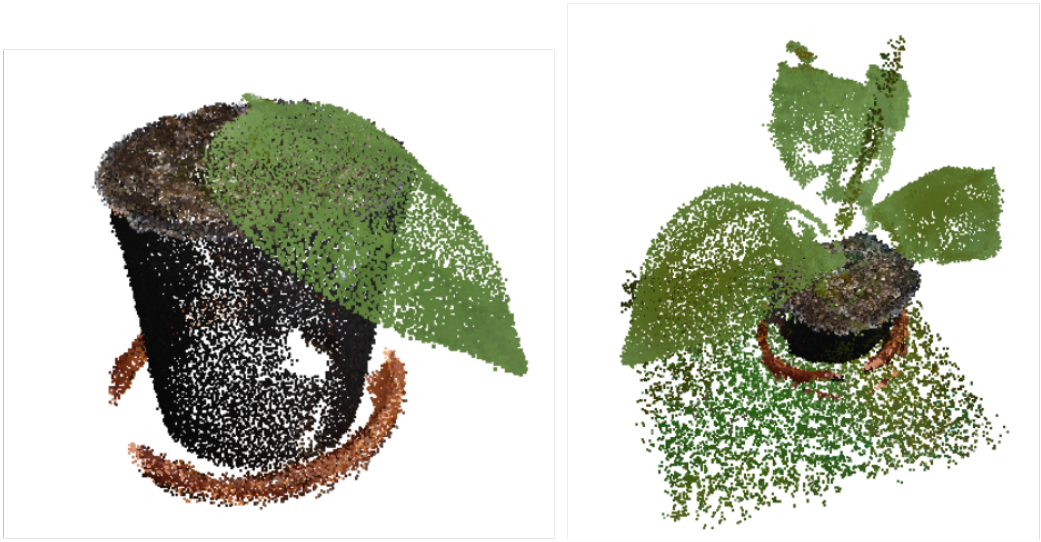


FIGURE 3.2: Visualization of synthetic plant models of the `Tobacco data set`. The left illustration shows a plant with few artifacts while the right plant has more pronounced artifacts.

the stem and pot could also be modified. Artifacts were added to the plant during the animation, such as distorting or removing certain vertices of the model. The animations were then exported from Blender as annotated point clouds (*.ply* format). Attention was paid to ensure the individual exported models differ in an acceptable way from each other, so that the data set contains enough variability. As a last step, gaussian noise added even more variability.

It was also very important to ensure that the test data set does not include any plants that are too similar to plants in the training set. The naive approach of creating a test set would be to randomly split the data set. However, since only one split is used in the experiments a unfavorable split could occur.

3.2.3 Description

A total of 327 models were produced. Table 3.1 shows the detailed information regarding the individual data set splits. Besides the point coordinates (x, y, z), the point clouds also contain RGB values and the corresponding label for instance segmentation. Figure 3.2 shows plants with few and with more pronounced artifacts. In addition, the degree of variability becomes clear in the pictures. The left picture shows a tobacco plant with only one leaf and quite generic structure, whereas the right one shows a plant with four leaves and squeezed pot. Additionally the soil is shown in the model and the stem is bigger but not very visible due to the strong artifacts.

TABLE 3.1: Statistical information about the `Tobacco` dataset.

	Training Set	Validation Set	Test Set
Number of point clouds	272	25	30
Average number of points	97 621	121 570	132 235
Average number of leaves	3.8	3.8	4.14
Maximum points per leaf	19 365	19 365	19 365
Minimum points per leaf	7 114	5 597	4 833

3.3 Evaluation Metrics

The choice of effective evaluation measures is an important part to evaluate the performance of models. This section will introduce metrics and the reasoning behind using these. Many evaluation metrics were proposed regarding object detection and semantic segmentation but only a few are consistently used by many research papers and benchmark datasets. Pascal VOC and MS COCO are one of the most popular benchmark datasets for object detection and segmentation. Both use *Average Precision* for object detection and instance segmentation, since both tasks can be evaluated in the same way. *Intersection over Union* is used for semantic segmentation. For the CVPPP leaf segmentation Scharr et al. (2016) propose the use of *Symmetric Best Dice (SBD)* instead of using Average Precision. A possible reason could be that this measure does not need confidence scores provided by the model. Another metric used is the *Difference in Count (DiC)*, that measures the difference in leaf counts between ground truth and prediction. In this work the Intersection over Union (equation 3.3.0.2) is used for semantic segmentation. The evaluation measures of the CVPPP leaf segmentation data set are applied for instance segmentation in order to ensure greater comparability to the corresponding 2D counterpart. Consequently, Symmetric Best Dice (equation 3.3.0.4) and Difference in Count (equation 3.3.0.5) are used to measure the results of this task.

Intersection over Union (IoU) It is a popular metric for segmentation and computes the ratio between the intersection and union of the ground truth A and prediction B. It is also referred to as Jaccard index and can be described as

$$IoU(A, B) = \frac{A \cap B}{A \cup B}. \tag{3.3.0.1}$$

This metric can also be described by true positive (TP), false negatives (FN) and false positives (FP).

$$IoU = \frac{TP}{TP + FN + FP} \tag{3.3.0.2}$$

Symmetric Best Dice (SBD) The SBD can be divided into two parts. First, the Best Dice (BD) is defined as

$$BD(L^a, L^b) = \frac{1}{M} \sum_{M}^{i=1} \max_{1 \leq j \leq N} \frac{2|L_i^a \cap L_i^a|}{|L_i^a| + |L_i^a|} \tag{3.3.0.3}$$

where L^a and L^b are leaf segmentations with their corresponding leaf objects $1 \leq i \leq M$ for L^a and $1 \leq j \leq N$ for L^b. Leaf areas (number of points) are denoted as $|\cdot|$. At first, intersections are sought for the leaves of L^a in L^b, which have a maximum dice coefficient. The average is calculated from this. Now the ground truth and prediction are chosen for L^a in the Symmetric Best Dice and the smallest one is chosen.The SBD can be defined as

$$SBD(L^{pred}, L^{gt}) = min\left\{BD(L^{gt}, L^{pred}),), BD(L^{pred}, L^{gt})\right\}, \tag{3.3.0.4}$$

where L^{pred} and L^{gt} stand for the predicted and ground truth leaf segmentations, respectively.

Difference in Count (DiC) It measures the absolute difference in leaf count between the ground truth $Leafs_{gt}$ and prediction $Leafs_{pred}$:

$$DiC = |Leafs_{pred} - Leafs_{gt}|. \tag{3.3.0.5}$$

3.4 Semantic Segmentation

3.4.1 Selection Criteria

This section briefly explains how models used for semantic segmentation were selected. Suitable models have already been identified in chapter 2 which are capable of semantically segmenting 3D data. Consequently, at this stage the search was limited to these models. Furthermore, three general approaches how to process point clouds have been identified. First of all, it would be desirable to test a model that is able to handle point clouds directly. The decision was made in favor of PointNet (Qi et al., 2016), as it is a novel architecture that delivers good results with existing benchmark data sets. At the time of implementation there was no public implementation of PointNet++ (Qi et al., 2017b), so the basic version was chosen.

In addition, the multi-view approach and voxel-based models were described. Suitable implementations for semantic segmentation exists for both approaches. However, a multi-view approach has already been selected for instance-based segmentation. Therefore voxel-based models were shortlisted. V-Net was chosen because this architecture has already been applied with good results to volumetric MRI data and has a U-architecture that is already successfully used with images.

3.4.2 Voxel-based Network

The first architecture used for semantic segmentation is V-Net which was introduced by Milletari et al. (2016). It was developed especially for the segmentation of 3D MRI images, which are available as voxel grid. It is a fully convolutional network with encoder-decoder architecture and corresponding 3D operators. The special aspect of the network are the incorporated skip connections in the V-architecture. The challenge of FCNs is to project the heavily compressed feature maps onto the input dimensions. In this architecture, as shown in Figure 3.3, features from each level of downsampling are connected to feature maps in the corresponding level of upsampling. In this way, spatial information that has been lost in the downsampling process can be recovered.

Each step in downsampling consists of one to three convolutional layers. In addition, a residual function is learned, as it is also used in ResNets. In each stage the input is processed by the convolutional layer and the non-linearity function. The input is then added to the output of this stage. PRelu is used as the activation function. In contrast to the official architecture, this work uses batch normalization and dropout after each convolutional layer. In total the input is downsampled four times before it is upsampled in the encoder part of the network by transposed convolution. Each upsampling stage consists of the same residual blocks as mentioned above, but with transposed convolutions instead.

The result is a feature map with the same dimensions as the input. Two channels are used to segment the background and the foreground - in this case the leaves. A log-softmax function is used to get the log-probabilites per voxel. During the training these serve as input for the loss function. During testing, however, these are used directly and voxels with a foreground probability > 0.5 are segmented as leaves.

One underlying problem using voxel grids for semantic segmentation is a possible class imbalance. Voxel grids of objects contain mostly empty voxels. Thus leaves could only occupy a small fraction of all voxels. As a result, the network can simply classify all voxels as background during training, as this will quickly reduce the loss. As a result, the network could get stuck in a local minimum and cannot form meaningful generalizable features. One possibility would be the weighting of the classes, but this introduces another hyper-parameter that complicates the training process. The authors of V-Net have therefore presented the dice loss D as a new type of loss function, which is supposed to avoid this problem:

$$D = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2} \tag{3.4.2.1}$$

where N is the number of voxels in the grid, $p_i \in P$ the predicted binary segmentation and the ground truth binary volume $g_i \in G$.

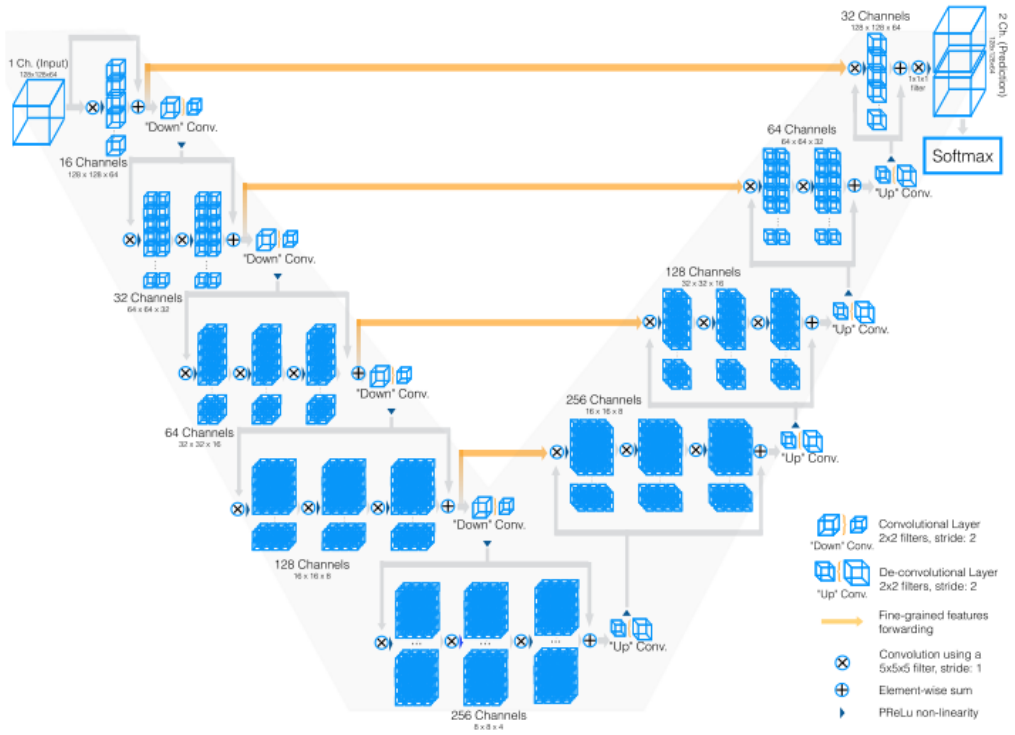


FIGURE 3.3: V-Net Architecture. Figure extracted from Milletari et al. (2016).

However, only insufficient convergence could be achieved with this proposed loss function. Therefore, this function has been replaced by the *negative loss likelihood function*. For this work an implementation from Macy (2017) was used as a basis and modified to the needs accordingly. It is implemented in Python 3 and PyTorch is used as a deep learning library.

3.4.3 PointNet

While this architecture was already mentioned in chapter 2, it will be explained in detail here. PointNet is one of the few currently released models that uses deep neural networks to deal directly with point clouds in order to be used for computer vision tasks such as classification and semantic segmentation. The procedure is relatively simple. Spatial encodings are learned from each point and then aggregated into a global feature vector. Figure 3.4 shows the complete network. In this work, the original architecture of the authors is adopted without change. The central challenge to process point clouds is their unordered representation. To be invariant against permutation of the input, PointNet uses a symmetric function. This refers to a function whose output is independent to the order of the input, as it is the case with the + operator. For this purpose max-pooling is used in the network. The output of the max pooling function is a global feature vector. This could be used directly for classification of the point cloud. For segmentation, however, this vector is now appended to each point feature vector, as shown in Figure 3.4. This allows the segmentation

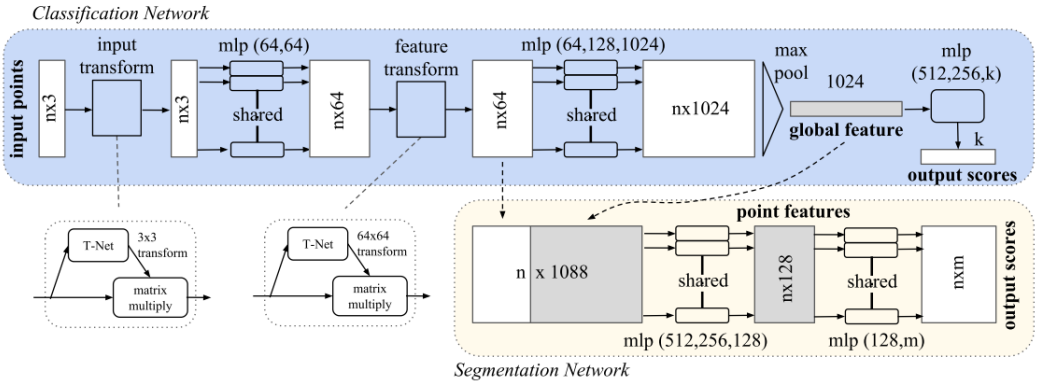


FIGURE 3.4: PointNet Architecture. Figure extracted from Qi et al. (2016)

.

subnetwork to use local and global information per point. This network uses multilayer perceptron with ReLU activations to process points. However, the weights are shared. Consequently, these operations can rather be regarded as 1D convolutions. In addition, batch normalization is applied. This network also uses a joint alignment network to make point clouds invariant to certain geometric transformations. This is a subnetwork that resembles the entire network. Here an affine transformation is learned, which is applied to the input in order to align them into a canonical space. An implementation, which has converted the original TensorFlow implementation of the PointNet authors into PyTorch, was adopted for this work (He, 2017).

3.5 Instance Segmentation

3.5.1 Selection Criteria

Chapter 2 presented some approaches for instance-based segmentation of images. For the 3D domain, only architectures capable of detecting objects using bounding boxes were identified, such as VoxelNet. For many applications, such as object detection for autonomous vehicles, bounding boxes are sufficient. However, these models are not recommended for leaf segmentation since bounding boxes may cause insufficient separation. Consequently, no published model can be used without modifications for the underlying task. Therefore, in this work two approaches are developed that can largely build on existing methods.

On the one hand, it was decided to use the multi-view approach, as many of the existing approaches can be used for semantic segmentation and changes only have to be made to some parts. For the instance segmentation of view projections, the proposal-based network Mask-RCNN (He et al., 2017) was used since it is regarded as a state-of-the-art architecture.

To compare this approach, a voxel-based model was implemented, since the transformation of a 2D CNN into a 3D CNN is much easier than, for example,

restructuring PointNet for object detection. Therefore the 2D CNN of Salvador et al. (2017) was chosen. One reason for this is that it provides good performance on the CVPPP dataset and also has an official public implementation. The model by Ren and Zemel (2016) performs better on the CVPPP data set, but the architecture is much more complex and computational expensive. Mask-RCNN was not used since GPU optimized 3D operators, which are necessary for the development, do not currently exist in relevant deep learning libraries and the development would have caused unnecessary overhead.

3.5.2 Multi-View Approach

This section now explains the selected multi-view approach. The general pipeline is straightforward to explain. Different views of the point clouds are generated, segmented with a 2D CNN and then projected onto the point cloud. The details are explained below. This approach is loosely adopted and modified from Boulch et al. (2017) and its official public code, who uses the multi-view approach for semantic segmentation.

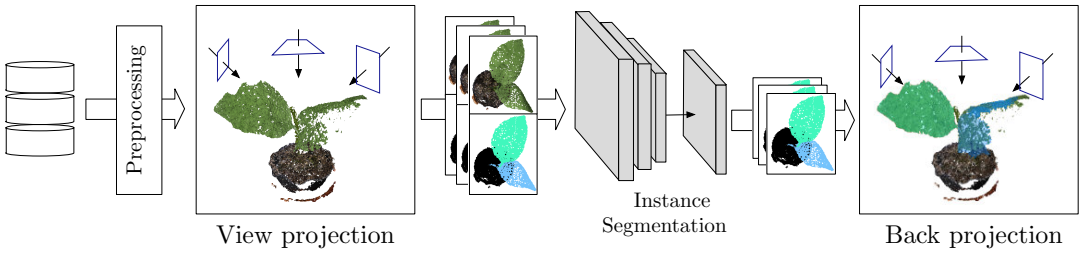


FIGURE 3.5: Multi-view approach workflow.

Preprocessing First the point clouds have to be processed to generate a satisfactory quality of the projections. In the raw form, a visualization would have many gaps, since there are no connections between the individual points. Consequently, a mesh of the point cloud is generated. However, before constructing a mesh, the *poisson disc sampling* algorithm (Bridson, 2007) is used to create a subsampling and to adjust the distances between the individual points. The reason for this is that point clouds of real scans show varying local densities. Poisson disc sampling creates a more natural point pattern in which the points are close to each other but maintain a minimal distance. The next step is to create a mesh using the *ball-pivoting* algorithm (Bernardini et al., 1999). This was chosen because the result shows relatively few additional distortions. Other algorithms which create a waterproof mesh could cause too much distortion of the mesh.

View projection In this step the meshes are used to create 2D projections. These are first loaded into a 3D viewer, which was implemented using the Python library *PyQtGraph*. In advance, 16 camera positions were determined, which cover the plant models as comprehensively as possible. These have a fixed

distance to the plants. Three projections are generated for each of these camera positions:

- Normal RGB images
- Labelled images
- Unique-face-color

The RGB images are being used as training or test data. On the annotated images the individual leaf instances are represented by a color code. The unique-face-color images are used to project the images onto the point clouds in the last step. On them each face of a mesh has a unique color. This allows an efficient matching of visible faces and pixels. All images are created with a resolution of 640x640 pixels. It should be noted that a shader was used for the RGB images, whereas this was not used for the other images in order to avoid distortions. Since information is inevitably lost during projections, some approaches add additional information, such as depth information or normals to the data (Boulch et al., 2017). However, this is not the case as a pre-trained weights are used for the network, which allows only an input with three color channels.

Instance segmentation The projections of the 3D models now serve as the data input for the 2D CNN. Mask-RCNN (He et al., 2017) is used for this purpose. This network is based on Faster R-CNN, which was explained in chapter 2 and uses a ResNet-101 (He et al., 2016) as a feature extractor. There are generally two main differences. For one a segmentation head creates a class-specific segmentation mask in parallel to the classification of the object proposals. The same object proposals are used for this as for classification. In addition, the RoI pooling layer is replaced by *RoI Align*, which interpolates the feature map bilinear instead of adaptive max pooling. The reason for this is that too much spatial information is lost by RoI pooling. It is sufficient for classification, but not for creating a fine segmentation mask. A simple fully convolutional subnetwork with four layers is used for segmenting the RoIs. For the implementation a TensorFlow version of Mask-RCNN was used from Waleed (2017).

Back projection Now that the predictions of the 2D CNN are available, they first have be projected onto the point cloud meshes in order to be transferred to the original data set in the last step, which can then be evaluated. For back projection, the already mentioned unique-face-color images are used to assign the predictions to the faces or vertices of a mesh. This means that there are a maximum of 16 possible assignments per vertex, one per image. In the case of semantic segmentation, one could now simply find the corresponding class per vertex by a majority vote. However, there are individual instances that have different labels and therefore no obvious assignment. For example, a leaf can be represented on image I_1 as instance number 2 and on image I_2 as instance number 6. Some authors (Ren and Zemel, 2016) use the Hungarian algorithm to create an efficient assignment between instance prediction and ground truth labels. A bipartite graph is used, with the intersection of union representing

the weights of this graph. So it would be thinkable to apply this approach for matching the individual instances. However, the decision was made to take a simpler approach. Algorithm 1 shows the procedure that is used. First the labels of all images are matched, and then the instances are assigned to the vertices by a majority vote. However, this approach has the problem that instances may not match because there is no overlap. In this case, there may be too many instances on the back projection. One way to avoid this would be to use a threshold condition for voting, so that at least a number of images with the same instance number must vote for a vertex, otherwise the vertex would not receive an assignment. This has also been implemented and the optimal voting threshold is determined in the experiments by using the validation set. K-nearest neighbors with $k = 3$ is used to classify vertices without assignment.

Algorithm 1: Back projection of images to point cloud

Input: Sequence of labelled images $[I_1,...,I_j]$
Unique-face-color images $[F_1,...,F_j]$
Point cloud P with v vertices and f faces
Voting threshold T

Output: Labelled point cloud P_l

Use F_i to get associated vertices of each pixel of a labelled image I_i
Assign associated labels to vertices v_i
Initialize image I_{init} with highest segmentation coverage
Function :
  Select most frequent leaf label L_{init} of I_{init}
  Iterate through all images I
  Find image I_{match} with highest leaf intersection L_{match} with L_{init}
  Replace L_{match} with L_{init} if label not changed before
  Repeat for all leaves of I_{init} and I_{match}
  Append I_{match} to I_{init}
end
Repeat function to merge and match leaf label one image per time
Use a majority vote per vertex with a minimum of T votes to assign a leaf label
return P_l

3.5.3 Voxel-based Network

This section introduces the voxel-based 3D CNN. This is an adaptation of the architecture of Salvador et al. (2017). For the implementation the publicly accessible code of the authors was used and modified accordingly. Consequently, the authors' 2D CNN approach will be explained and the modifications are explained at the corresponding points. To make this architecture easier to name the abbreviation of the paper *Recurrent Semantic Instance Segmentation* (RSIS) is used.

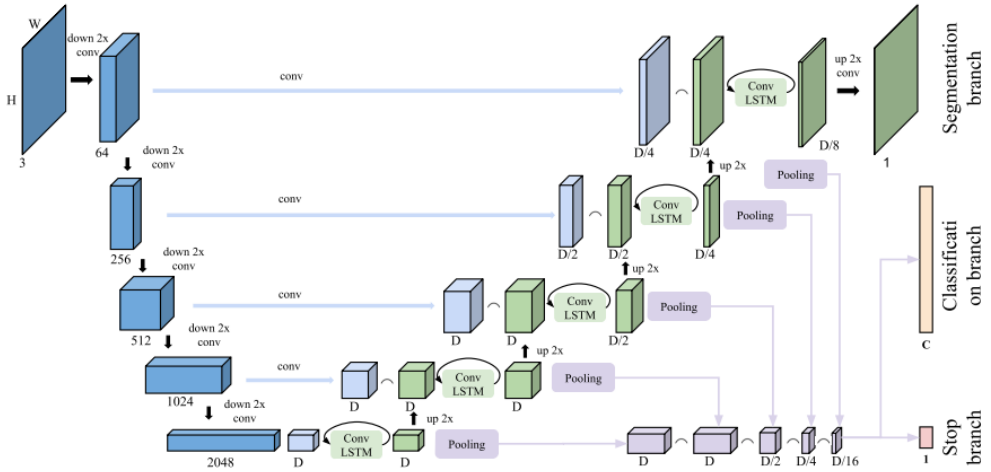


FIGURE 3.6: Architecture of the 2D RSIS architecture. Figure extracted from Salvador et al. (2017).

The general network structure resembles an encoder-decoder architecture as already introduced with V-Net. The input is first sampled down and then scaled back to the original input dimension by the decoder. Instance segmentation in this network is not done by bounding box proposals, as it is the case with Mask-RCNN, for example. The recurrent decoder enables the network to segment one object after another instead of directly producing an entire semantic segmentation map.

In the original architecture a ResNet-101 is used for the encoder part. For this implementation, however, a modified VGG-16 is used. The original VGG-16 structure consists of 5 downsampling blocks. However, the last two downsampling steps were discarded because the inputs in the experiments have a relatively low resolution. Otherwise, the feature map would be too small. Also all 2D operators have been replaced with 3D operators. As a result, the encoder can take a voxel grid x as input. The encoder is now used to extract the features $F = encoder(x)$, where $F = [f_0, f_1, f_2]$ contains all outputs of the respective downsampling blocks.

The decoder is structured parallel to the encoder and takes f_i as input at upsampling level i and thus behaves like V-Net. But instead of normal convolution operators, ConvLSTM units are used. For each time step the ConvLSTM gets a bilinear upsampled hidden state $h_{i-t,t}$ of level $i-1$ and the corresponding feature map f_i through the skip connection. The network now outputs three values for each time step. At the highest level in the network a final binary segmentation mask of the object is created. Furthermore, the class probability and objectness score are generated. The objectness score serves as a stop criterion during the test period.

The network is trained by a multi-task loss, which consists of three components. Segmentation loss is a Intersection over Union that expresses how good the segmentation between prediction and ground truth is. The Hungarian algorithm is used to assign the individual instances. The classification loss describes the

deviation from the ground truth classification of the found objects. Finally, the stop loss is used to decide when the model should stop searching for new objects.

Chapter 4

Evaluation and Results

In this chapter the results of the experiments are presented. To illustrate these in a structured way, the first section will deal with semantic segmentation and then the results of instance segmentation are shown. The experiments are evaluated according to the previously defined performance measures. In addition, extended results are presented to give the reader deeper insight into the limitations of the methods.

All experiments were ran on a Ubuntu server equipped with 64GB of memory, an Intel^{R} Core^{TM} i7-7700 CPU working at 3.60GHz, and a NVidia GTX 1080 with 8 GB of video memory. Training and testing was performed using GPU support with CUDA 8.0 for PyTorch and TensorFlow. Python 3.5 was used for the entire implementation.

4.1 Semantic Segmentation

4.1.1 Experiments

In the following, the series of experiments for semantic segmentation are presented. These are based on the general experimental design discussed in chapter 3.1. V-Net and PointNet have been adopted according to the description in chapter 3. For all experiments, the architecture and hyperparameters were not changed to guarantee comparability. In addition to the general comparison between the models, the intra-model variation with respect to three parameters should also be determined:

- Input size
- Channel
- Data augmentation

For both models, all three factors could be altered, whereby the precise implementation regarding the input size differs for both. Since PointNet works directly with points, this model was first trained with 1024 points, as it was also carried out by the authors. Furthermore it was checked whether the performance increases by doubling the points, so that the model was also trained with 2048.

The point clouds were reduced from its original size using uniformly sampling by a voxel-grid of size 128^3 and then randomly sampling it down to the required size. Since both models should be compared in terms of performance on the original point cloud, the point cloud was left unchanged during testing. The possibility of using PointNet on the original size during testing also corresponds to the authors' statement.

V-Net, on the other hand, works with the discretization of point clouds into voxels. Consequently, resolutions of 32^3 and 64^3 were chosen as input sizes, whereby 64^3 was set due to the technical limitation of time and memory complexity. The discretization of the point cloud results in a voting within each voxel regarding the label and the color information. While testing, all points inside the voxel are assigned to the corresponding predicted class of the voxel. Therefore it has to be emphasized that the input sizes for V-Net were used for training as well as for testing. With PointNet the input sizes were only used for training and tested on the original point cloud.

To increase the data set using data augmentation, the point clouds were randomly rotated around the vertical axis before any input transformation. For data augmentation $4x$ and $8x$ size of the original training set were created. Consequently, the training set in the 4x variant consists of 1 088 samples and in the 8x variant of 2 176 samples. The factor *Channel* describes if color information is used or not. To simplify the description in the further process, the individual configurations are written in brackets. So V-Net$[32^3, binary, -]$ would describe a V-Net model that was trained with an input size of 32^3, had no color information and was trained on the original data set.

The maximum training duration for all models was set to 1 000 epochs. After each epoch the model was tested on the validation set. If the validation loss was lower than the best so far, the weights of the model were saved. The training ended automatically after 1000 epochs or when the validation loss has not improved for 100 epochs. The stored weights of the best validation loss were used for evaluation. Thus early stopping is applied as a regularization technique. Both also use a weight decay with $\lambda = 1e^{-4}$ for PointNet and $\lambda = 1e^{-8}$ for V-Net. ADAM is used as optimizer for both with a learning rate of $\eta = 0.001$. Both networks were trained with 10 samples per mini-batch.

4.1.2 Results

The results of the experiments are described in the following. Table 4.1 shows the results of the evaluation of V-Net. Here the different Intersection of Union values are shown regarding the predefined parameters. It can be seen that the values of the input size 32^3 improve slightly with increasing data augmentation. Whereas with an input size of 64^3 it is not straightforward. It can generally be seen that 64^3 performs better in every respect than the lower resolution. Table 4.2 shows the results of PointNet. Here it can initially be observed that the results are within the range of V-Net, but are surpassed by the best performing

V-Net. Moreover, there does not seem to be a consistent pattern of influence by the factors besides the addition of color.

Figure 4.1 shows the robustness of the V-Net architecture in relation to the input size. The best binary models of 32^3 and 64^3 were selected and evaluated using test data of different quality. The left graph (a) shows the IoU in relation to the number of points in the test set. The right graph (b) shows the relation between gaussian noise added to test set and the performance. It can be seen that the 32^3 representation is significantly more stable in both tests. Figure 4.2 shows the robustness test for PointNet, which shows that it is much more stable than V-Net. Furthermore, table 4.3 shows a comparison of the number of parameters and the occupied memory of the GPU during training. PointNet seems to be much more memory efficient in regard to its performance.

Figure 4.3 shows a very simple example that both methods can segment very well. Only subtle differences can be detected, especially on the stem. Figure 4.4, on the other hand, shows a sample with very strong artifacts where large parts of the pot are not present. PointNet seems to have more problems with this than V-Net and segments a large part of the area as a leaf. In the figure 4.5 a constellation is shown which did not appear in the training data. This means that it displays a leaf lying on the ground and therefore not on the pot. V-Net was able to partially recognize the leaf, but the model tends to over-segmentation. PointNet, on the other hand, could not recognize this new constellation.

To go beyond point estimates, the models with the best performances of both architectures, which were achieved without color information, should now be compared with each other using a paired t-test. First, the difference of the individual IoU values of the test samples is calculated and checked for normality using the Shapiro-Wilk test. The null hypothesis for this test states that there is a normal distribution. The alternative hypothesis is that there is none. With this test $p = 0.002$ is reached and since a significance at $p < 0.05$ is given, no normal distribution is present. Therefore, a paired t-test should not be performed. However, in such a case the Wilcoxon signed rank can be used to determine whether the median of both performances differ significantly with the absence of a normal distribution of differences. This test results in $p = 0.02$, which is less than the threshold $p < 0.05$. Consequently, it can be assumed that the median IoU of both models differ significantly for the underlying test samples.

TABLE 4.1: Comparison of Intersection over Union (IoU) on the test data set using trained V-Net models with various parameter combinations.

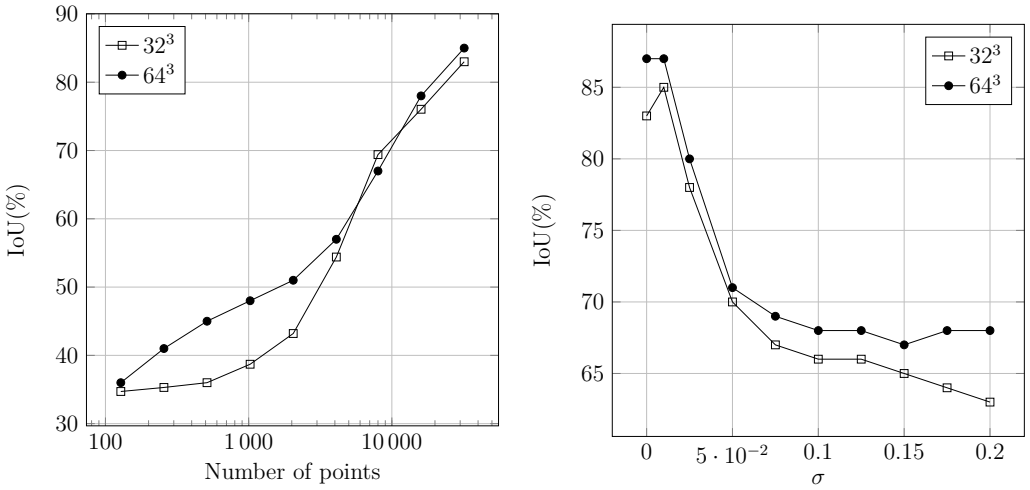
Input Size	Channel	Data augmentation	IoU(%) ↑
32^3	binary	-	81.2
32^3	binary	4x	82.4
32^3	binary	8x	83.3
64^3	binary	-	87.3
64^3	binary	4x	83.9
64^3	binary	8x	86.4
32^3	rgb	-	89.9
32^3	rgb	4x	89.6
32^3	rgb	8x	90.5
64^3	rgb	-	93.4
64^3	rgb	4x	**93.9**
64^3	rgb	8x	91.6

TABLE 4.2: Comparison of Intersection over Union (IoU) on the test data set using trained PointNet models with various parameter combinations.

Points	Channel	Data augmentation	IoU(%) ↑
1024	binary	-	81.2
1024	binary	4x	78.4
1024	binary	8x	78.3
2048	binary	-	83.6
2048	binary	4x	81.5
2048	binary	8x	81.4
1024	rgb	-	90.5
1024	rgb	4x	90.8
1024	rgb	8x	91.5
2048	rgb	-	89.5
2048	rgb	4x	91.2
2048	rgb	8x	**91.7**

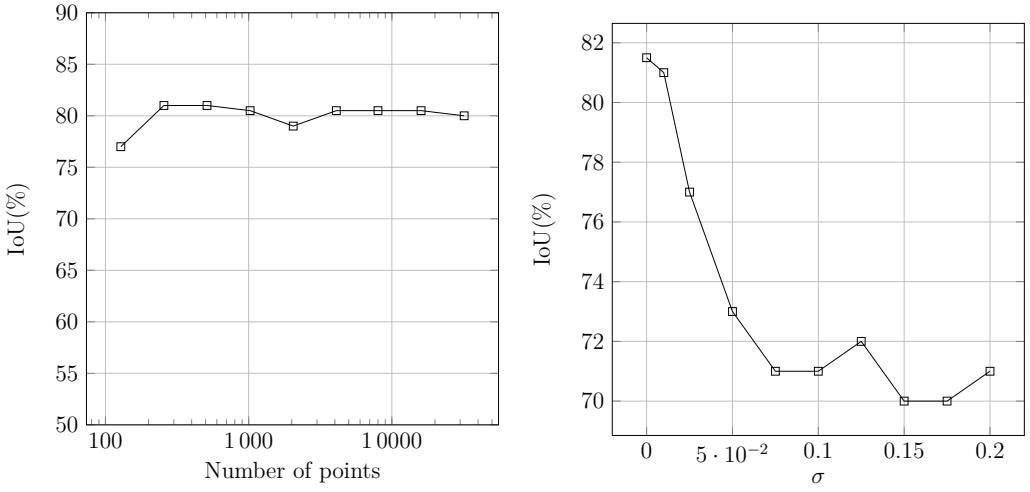
TABLE 4.3: GPU memory consumption comparison between V-Net and PointNet during training. Both use a batch size of 10 samples.

Model	Parameters	GPU memory
PointNet (1024)	8 005 199	1.6 Gb
PointNet (2048)	8 005 199	2.7 Gb
V-Net(32)	45 607 934	2.2 Gb
V-Net(64)	45 607 934	5.7 Gb



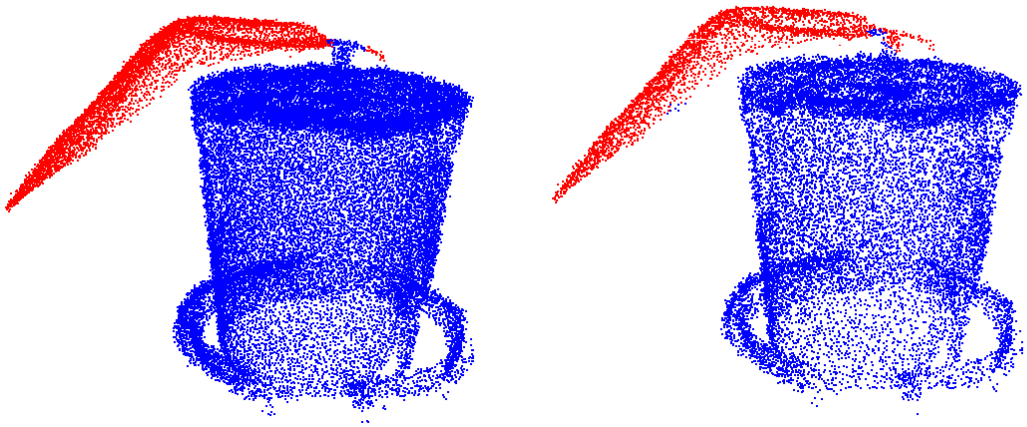
(A) Point cloud size alteration (B) Addition of gaussian noise

FIGURE 4.1: Robustness test for V-Net[64^3,binary,-] with different perturbations while testing.



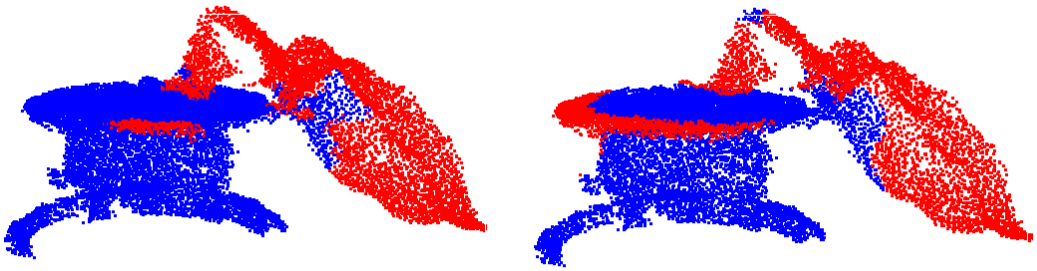
(A) Point cloud size alteration (B) Addition of gaussian noise

FIGURE 4.2: Robustness test of PointNet[2048,binary,-] with different perturbations while testing.



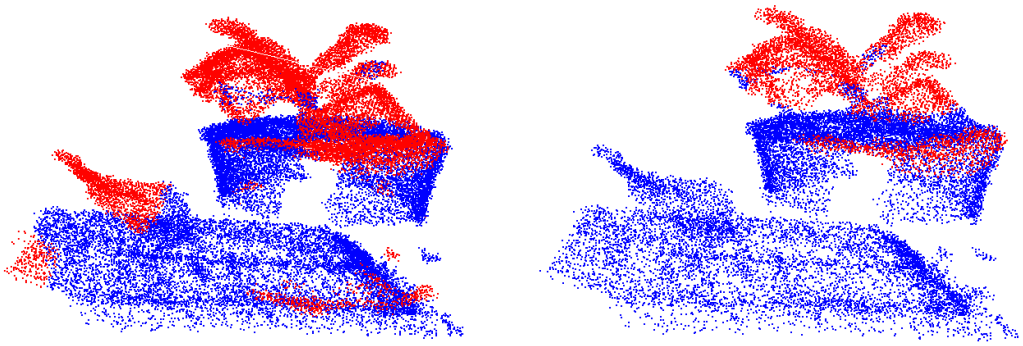
(A) V-Net with $IoU = 98.5\%$ (B) PointNet with $IoU = 98.2\%$

FIGURE 4.3: Comparison between V-Net [64^3,binary,-] and PointNet [2048,binary,-]. The leaf class is visualized with the color red.



(A) V-Net with $IoU = 83\%$ (B) PointNet with $IoU = 62\%$

FIGURE 4.4: Comparison between V-Net [64^3,binary,-] and PointNet [2048,binary,-]. The leaf class is visualized with the color red.



(A) V-Net with $IoU = 63\%$ (B) PointNet with $IoU = 57\%$

FIGURE 4.5: Comparison between V-Net [64^3,binary,-] and PointNet [2048,binary,-]. The leaf class is visualized with the color red.

4.2 Instance Segmentation

4.2.1 Experiments

In the following, the series of experiments for instance segmentation are presented. The multi-view and RSIS models have been adopted according to the description in chapter 3. For all experiments, the architecture and hyperparameters were not changed to guarantee comparability. In addition to the general comparison between the models, the intra-model variation with respect to three parameters should also be determined:

- Input size
- Channel
- Data augmentation

For RSIS, all three parameters could be modified. This model has the same parameter requirements as the previously evaluated V-Net. Since the input is in the form of voxel grids, two input sizes are tested again with 32^3 and 64^3, which was set due to the technical limitation of time and memory complexity. The discretization of the point cloud results in a voting within each voxel regarding the label and the color information. For testing, all points inside the voxel are assigned to the corresponding class. To increase the data set using data augmentation, the point clouds were randomly rotated around the vertical axis before any input transformation. For the data augmentation $4x$ and $8x$ size of the training set were created. The factor *Channel* describes if color information is used or not.

However, no factors have been changed for the multi-view approach. Data augmentation should not be used for the multi-view approach, as this approach generates images from different angles. This would only result in more views, which may have a less favorable angle than the previously defined camera positions in the multi-view pipeline. The input size is fixed at $640x640$ pixels for all views. For the multi-view approach, only images with RGB values were used.

The maximum training duration for RSIS was set to 1000 epochs. After each epoch the model was tested on the validation set. If the validation loss was lower than the best so far, the weights of the model were saved. The training ended automatically after 1000 epochs or when the validation loss has not improved for 60 epochs. The stored weights of the best validation loss were used for evaluation. For the training of RSIS also a weight decay of $\lambda = 1e^{-6}$ is used. As optimizer ADAM is used with a initial learning rate of $\eta = 0.0007$ for the encoder and $\eta = 0.001$ for the decoder. It was trained with 6 samples per mini-batch.

Transfer learning was used for the training of Mask R-CNN. This implies that the weights of a ResNet101 were used that had been pre-trained with the MS COCO data set. The training was divided into two phases. First, all views were used, i.e. a total of 4 352 images, to train the uppermost layers of the network with a learning rate of $\eta = 0.001$, whereby the lower layers were fixed. This was

done for 25 epochs. Afterwards the weights with the best validation loss were used to fine tune them further. The learning rate was set to $\eta = 0.0001$ and the entire network was trained for 25 epochs. For the training of Mask-RCNN a weight decay of $\lambda = 1e^{-4}$ was used. Mini-batch stochastic gradient descent was used as optimizer.

4.2.2 Results

Now the individual results of the experiments are described in more detail below. Table 4.4 shows the results of the RSIS models with different input sizes, data augmentation and color information. It is evident that data augmentation can significantly improve the performance. However, there are two outliers here. Both show a drop while increasing the input size from 4x to 8x. The best result was achieved with 8x data augmentation, RGB values and a size of 32^3. However, the difference to the best result of binary voxel grids seems to be only marginal.

Table 4.5 shows the results of the multi-view approach. This shows that with a SBG of 63.2% at 16 views the multi-view approach scores significantly worse than the best result of RSIS with 75.8%. In addition, the results of the individual views for the back projection are also shown. At 63.5 %, this is very close to the reproduced point cloud. To evaluate the back projection algorithm, it was performed with the ground truth labels. One can see that the performance in this case is very good, but still not perfect.

Figure 4.6 visually illustrates the difference in segmentation between the two methods. As you can see, RSIS has a good performance except for small wrong segmented regions, which could possibly still be eliminated by post-processing. The multi-view approach, on the other hand, shows a under-segmentation of a leaf. The next figure 4.7 shows a case where the multi-view approach has better performance. What is striking about RSIS is that it assigns remote regions to one leaf.

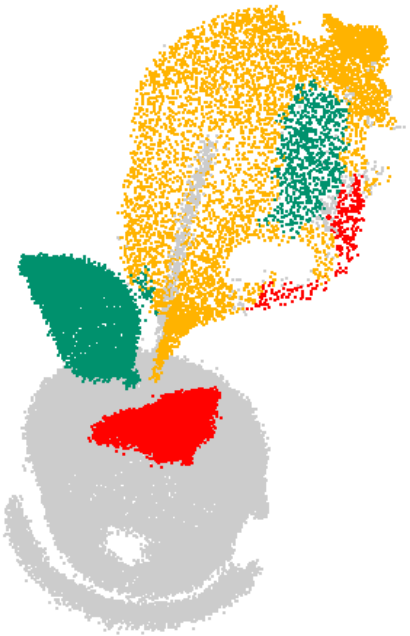
Although there is a clear difference between the two methods, a statistical test should still be carried out to maintain consistency. The best performing models of both methods are used for this test. The Sharpio Wilk test is also performed in this case to verify whether the differences of the SBD values of both models are normally distributed. The test yields $p = 0.63$, which means that the null hypothesis cannot be rejected for the alternative hypothesis and thus a normal distribution can be assumed. Therefore, a paired t-test is performed. Here $p = 0.017$ is reached with a mean of differences of 12.34%. Consequently, it can be assumed that the results of both models differ significantly.

TABLE 4.4: Comparison of Symmetric Best Dice (SBD) and Difference in Count (DiC) on the test data set using trained RSIS models with various parameter combinations.

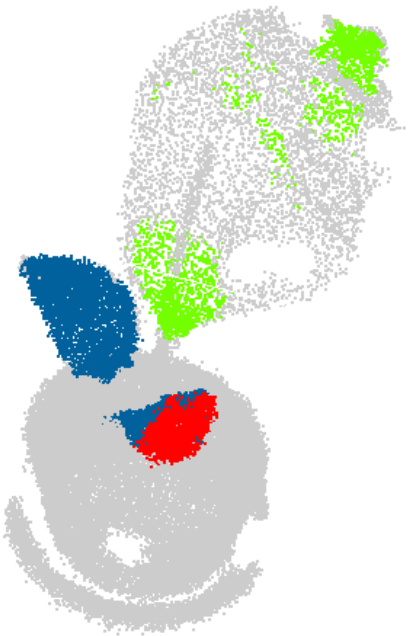
Input size	Channel	Data Augmentation	SBD (%) ↑	|DiC| ↓
32^3	binary	-	43.8	2.2
32^3	binary	4x	70.3	0.5
32^3	binary	8x	69.4	0.8
64^3	binary	-	59.9	1.3
64^3	binary	4x	66.5	0.9
64^3	binary	8x	74.5	0.7
32^3	rgb	-	60.4	0.9
32^3	rgb	4x	69.3	0.9
32^3	rgb	8x	**75.8**	**0.5**
64^3	rgb	1x	58.4	1.4
64^3	rgb	4x	71.2	0.9
64^3	rgb	8x	73.3	0.8

TABLE 4.5: Symmetric Best Dice (SBD) and Difference in Count(DiC) of the multi-view approach using Mask-RCNN as 2D CNN. Results are also shown for the images and back projection with ground truth labels. Voting threshold is 4 and was obtained by the validation set.

Data Representation	Views	SBD (%) ↑	|DiC| ↓
Point cloud	16	63.2	0.8
Images	-	65.5	0.95
Point cloud (ground truth)	-	94.8	0.3

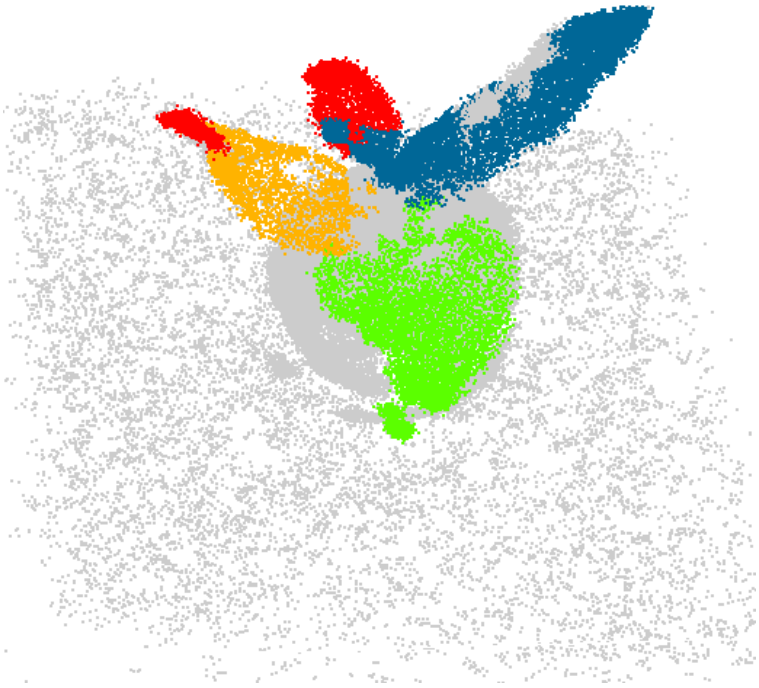


(A) RSIS with $SBD = 93.2\%$

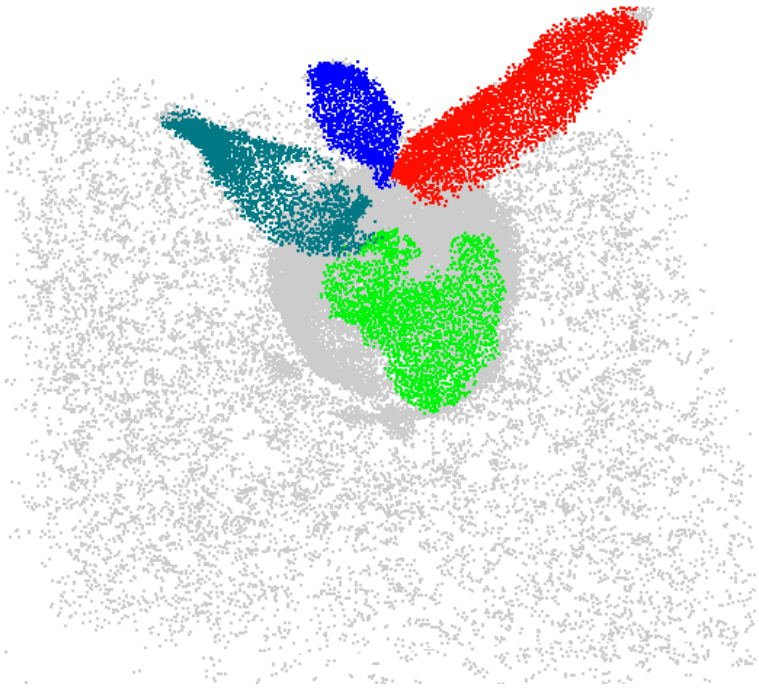


(B) Multi-view approach with $SBD = 61.2\%$

FIGURE 4.6: Comparison between multi-view and RSIS [64^3, binary, 8x]. Each color except gray represents a leaf.



(A) RSIS with $SBD = 78.3\%$



(B) Multi-view approach with $SBD = 90.5\%$

FIGURE 4.7: Comparison between multi-view and RSIS [64^3, binary, 8x]. Each color except gray represents a leaf.

Chapter 5

Discussion

This chapter presents the discussion. First the results are discussed in detail. The aim is to analyze them and to explore possible interpretations. In the next step, a retrospective look at the work is taken to critically determine various limitations and suggest possible solutions. Building on the results, but also on the limitations, a outlook for further research is presented. This chapter and thus also the thesis ends with a final conclusion.

5.1 Results

5.1.1 Semantic Segmentation

The experiments compared two fundamentally different approaches. V-Net was able to deliver relatively solid results. A statistical test also revealed a significant difference to PointNet in terms of performance on the test set. The results indicate that V-Net is able to improve its performance by increasing the resolution and using color information. But data augmentation did not provide clear results. In some cases the performance could be improved. Whereas in others it got worse. PointNet did not really improved with a bigger input size. Of course PointNet was only trained on two different point cloud sizes and was tested on the entire point cloud. Therefore, these factors cannot be compared with each other directly, since V-Net can naturally make more precise predictions due to a higher resolution while testing. The fact that rotation as data augmentation has not a great impact on PointNet can be explained by the joint alignment network in PointNet, which is used to make PointNet invariant to rotations. However, the addition of color information lead to an improvement of the results. This data set has no large color variation. Consequently, it is natural that the results are so good using color.

A visual comparison of both methods reveals various details of the errors. The first is that PointNet is not able to segment a previously unknown constellation, whereas V-Net segments the leaf relatively well. Consequently, the question arises as to how PointNet's performance reacts to changeable circumstances. The most significant problem might be that PointNet has a lack of local context, i.e. it only uses point-wise and global features, but neglects the neighborhood.

Although PointNet has weaknesses in this respect, it can outperform V-Net in the robustness test. This property was explicitly underlined by the authors of PointNet and was confirmed here even with a more sophisticated voxel-based model than Qi et al. (2016) used in their comparison. Looking at the results of V-Net's robustness tests, it is noticeable that V-Net(64^3) is more robust than V-Net(32^3).

It could have been assumed that V-Net(32^3) would be more robust due to the coarser discretization. It is therefore possible that V-Net(64^3) has encountered more artifacts of the point cloud due to the higher resolution and thus became more robust during training.

Nevertheless, any perturbations of the point cloud seem to affect the convolutional layers accordingly. Perturbations of this kind are relatively rare in images. Compression artifacts can be processed well by a CNN, whereas gaussian noise reduces the performance of images accordingly (Dodge and Karam, 2016). 3D reconstructions may contain strong artifacts, which are expressed by noise but also by missing parts, which were also tried to be included in the data set. The visualization of the results showed that plant models with correspondingly low perturbations could achieve better results. This can be explained by the small data set, that does not let the model generalize well for these kind of artifacts. However, at this point one can ask the question how to deal better with naturally occurring perturbations. In this work the rotation around the horizontal axis of the point cloud was used as data augmentation technique. Is it conceivable that perturbations as data augmentation could improve the stability of V-Net?

In some cases, data augmentation may cause a decrease in performance. There is, of course, the question as to why. On the one hand, it is imaginable that only the non-deterministic nature of the learning process is to blame. But it could also be due to the fact that data augmentation enlarges the data set and thus also the duration of an epoch. Based on the assumption that the additional data brings little or no variability into the data set, the training might converge in fewer epochs, since the batch size remains the same and thus more weight updates take place in one epoch. Consequently, there is a possibility that the smaller data set could get a better weight configuration due to a higher validation frequency.

5.1.2 Instance Segmentation

Two different approaches were also compared for instance segmentation. The experiments have shown that RSIS can achieve better results than the multi-view approach. While in semantic segmentation the voxel-based model only achieved marginal improvements in performance by data augmentation, the performance increase by RSIS can be seen more clearly. Even if both performance measures are not directly comparable, one can still ask oneself why RSIS is experiencing strong performance improvements. A significant difference between the two models in terms of training is that RSIS uses multi-task learning. Consequently, the possibility exists that V-Net can achieve relatively little from extending the data set through its single-task learning, since the task during training is relatively

simple and simple rotations do not force it to make features more generalizable. On the other hand, it would be conceivable that RSIS is massively overfitted without data augmentation. V-Net, however, could possibly already achieve very good results without data augmentation due to stronger regularization by using dropout. By looking at the visualizations, it can be seen that RSIS sometimes assigns widely spaced areas to one leaf. This probably shows the influence of the LSTM. In a proposal-based model, a leaf would be in a predefined area, which means that segmentation can only take place in this bounding box. Therefore, if the bounding boxes are well positioned, this type of error would probably not occur in proposal-based models.

The multi-view approach does not perform that well. The images deliver a similar result as the back projection on the point cloud. The test of the back projection algorithm with the ground truth gives a very good but not perfect result. The question is whether this is solely due to the back projection algorithm or whether the number of views was not sufficient to obtain a perfect result.

5.2 Limitations

Like many other works, this work also contains limitations that should be mentioned in order to point out possible problems. In this context, however, it is not only intended to refer to the limitations, but also to make suggestions for certain points in order to overcome them.

As already mentioned at the beginning of this work, it is an explorative work whose focus is to go into the breadth instead of into the depth of individual approaches. Therefore, no exhaustive hyperparameter search was performed to find the most optimal setting for each network. Many of the recommendations suggested by the respective authors were adopted and modified by trials using the training or validation set. However, some methods still offer room for optimization, especially with regard to regularization. Furthermore, the data set represents a limitation. Although it provides an acceptable basis for the comparison of the methods, its synthetic nature, small size and variability make it less generalizable in respect to the performance on real 3D reconstructions.

Another point concerns the multi-view approach. The final performance of this approach depends on countless factors along the pipeline. The meshing process, the settings of the 3D viewer, the definition of the views and the back projection algorithm are all factors that can influence the performance regardless of the CNN used. Especially the algorithm used in this work is relatively simple and offers room for improvement.

An important problem that should be addressed is the choice of a performance measure that explicitly states the quality of segmentation that is best suited for leaf segmentation in plant phenotyping. The IoU and SBD were used in this work. The question now is whether both measures are really sufficient to evaluate the problem described? Both measures can generally be used and especially the IoU is used for semantic segmentation in many publications. Both

can therefore provide a statement on the general quality of the segmentation. However, segmentations with the same IoU or SBD can look very different. In the experiments the difference between PointNet and V-Net is not very dramatic and there could theoretically be a possibility that with a small difference between two models, the model with the worse performance on the measures used in this work is better suited for further use in the phenotyping pipeline. The use of Difference in Count is therefore a step in the right direction to support the SBD in this respect. At this point, the question now arises as to how far it is possible to define better performance measures for the needs in the subsequent processing of point clouds, e.g. by using more precise measures like precision and recall to determine over- and under-segmentation. Also distance measures between ground truth and prediction, such as the modified Hausdorff distance, could refine the performance measurement. Also it seems interesting to address if it is possible to nudge deep neural networks to behave in a certain way to address certain performance metrics requirements. In this respect, it would be conceivable to adapt the loss function to penalize certain behavior more.Furthermore, it would also be possible to use multi-task learning to nudge the network in a desired direction. One could therefore define *auxiliary tasks*, such as key point detections of leaves, where certain areas of leaves are marked similar to facial landmarks, in order to obtain possibly more robust predictions and to force the network to represent these areas particularly well. A side effect of course would be a potentially better generalizability, which has already been achieved by multi-task learning with auxiliary tasks (e.g. Zhang et al., 2014).

5.3 Outlook

The performance of the entire system depends on the underlying data and the methods used. Therefore, possible future research for both points is presented in the following.

The `tobacco` data set was appropriate for the underlying explorative task of this thesis, but an appropriate benchmark data set is required for further progress in this research area. Ideally, it should consist of 3D reconstructions and have high quality annotations. However, the annotation of point clouds is a laborious task. Consequently, it would be possible to improve the synthetic generation of data, so that it strongly approximates the conditions and limitations of 3D reconstructions. In recent years, the use of Generative Adversarial Networks (GAN) for the creation of synthetic data has been highlighted in particular. There are already approaches that successfully generate point clouds. From this point of view, the benefit of GANs would be twofold. On the one hand, it would be conceivable to use GANs to generate annotated synthetic data directly. Another interesting approach of using GANs would be to make simulated synthetic point clouds more realistic by approximating naturally occurring artifacts. This approach was implemented by Shrivastava et al. (2017) for the refinement of synthetic faces on pictures. It would therefore be worthwhile to see to what extent this approach can be applied to point clouds.

The second crucial aspect concerns the models that are used. For semantic segmentation of 3D data, good architectures already exist with PointNet and V-Net. These can be further optimized for this task, either by using dilated convolutions or memory optimization using octree-based architectures. Another possible approach would be to merge PointNet with a 3D CNN. This could work by feature end-to-end fusion to possibly combine the advantages of PointNet with the advantages of a CNN. In this context it would also be interesting to explore to what extent bayesian deep learning can be used to model the uncertainty caused by naturally occurring perturbations of a 3D reconstruction, and thus to possibly improve predictions.

For instance segmentation, however, it looks like there is no canonical architecture for the 3D domain yet. This work transformed the RSIS architecture from the 2D to the 3D domain and optimized it for this purpose. However, it uses a recurrent network. The question of how proposal-based models perform for the underlying problem remains open. It would therefore be interesting to investigate Mask-RCNN in this regard. In general, this subdomain offers a lot of possibilities to combine different ways of instance segmentation with various approaches of 3D deep learning. Especially the combination of point cloud based methods with voxel-based methods seems to be interesting. As the performance also depends on the quality of the reconstructions, it would be interesting to see to what extent multiple modalities can be merged in order to improve the stability and performance of the results. This would be analogous to the KITTI dataset for self-driving cars, which contains images and point clouds. It would be conceivable to use both modalities to merge them into an end-to-end network, since by using images one can bypass corresponding reconstruction artifacts.

5.4 Conclusion

At the beginning of this work, the current research gap regarding the use of deep neural networks in the 3D domain for the segmentation of plants was addressed. The aim of the work was to explore potential approaches and to evaluate and compare selected ones. In addition, the question was asked to what extent data augmentation can improve the performance. The focus was placed on semantic and instance segmentation, which are important for the underlying problem of plant phenotyping. Three general approaches have been identified to use deep neural networks for processing 3D point clouds which cover the use of volumetric representations, 2D view-based representations and architectures that can handle irregular forms directly. In order to evaluate selected approaches, a synthetic data set had to be generated as part of this work. Two approaches were compared for semantic segmentation. On the one hand PointNet, an architecture that can work directly with point clouds, was used. On the other hand, with V-Net a 3D CNN was selected that works with voxelized point clouds. It was shown that the CNN can deliver satisfactory results. PointNet performs slightly worse, but is more robust against perturbations of point clouds. However, it seems to have

difficulties to address unknown situational context. Data augmentation gives inconclusive results in both cases.

For instance segmentation a multi-view approach was chosen, which segments 2D projections of point clouds using Mask R-CNN, a state-of-the-art 2D CNN, and projects them onto the point clouds again. In addition, a 2D CNN was adapted to handle 3D input in form of voxel grids. It also distinguishes itself by the use of a recurrent component. It was shown that the voxel-based model performs better than the multi-view approach, which has many factors in the data pipeline that influence its performance. Data augmentation achieved good performance improvements with the CNN. Consequently, this work showed that deep neural networks can be used in the 3D domain for the segmentation of plant point clouds. However, further research is needed in some areas in order to expand and optimize these approaches. In this context, instance segmentation still poses a particular challenge.

Bibliography

Bernardini, Fausto et al. (1999). "The ball-pivoting algorithm for surface reconstruction". In: *IEEE Transactions on Visualization and Computer Graphics* 5.4, pp. 349–359. ISSN: 10772626. DOI: 10.1109/2945.817351. URL: http://www.research.ibm.com/vistechnology/pdf/bpa{_}tvcg.pdf.

Boulch, Alexandre et al. (2017). "SnapNet: 3D point cloud semantic labeling with 2D deep segmentation networks". In: *Computers & Graphics*. ISSN: 00978493. DOI: 10.1016/j.cag.2017.11.010. URL: http://linkinghub.elsevier.com/retrieve/pii/S0097849317301942.

Bridson, Robert (2007). "Fast Poisson disk sampling in arbitrary dimensions". In: *ACM SIGGRAPH 2007 sketches on - SIGGRAPH '07*, 22–es. ISBN: 9781450347266. DOI: 10.1145/1278780.1278807. URL: http://www.cs.ubc.ca/{~}rbridson/docs/bridson-siggraph07-poissondisk.pdfhttp://dl.acm.org/citation.cfm?doid=1278780.1278807.

Bronstein, Michael M et al. (2017). *Geometric Deep Learning: Going beyond Euclidean data*. DOI: 10.1109/MSP.2017.2693418. arXiv: 1611.08097. URL: https://arxiv.org/pdf/1611.08097.pdf.

Chen, Liang-Chieh et al. (2018). "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". In: arXiv: 1802.02611. URL: https://arxiv.org/pdf/1802.02611.pdfhttp://arxiv.org/abs/1802.02611.

Choromanska, Anna et al. (2014). "The Loss Surfaces of Multilayer Networks". In: ISSN: 15337928. arXiv: 1412.0233. URL: http://arxiv.org/abs/1412.0233.

Dai, Jifeng, Kaiming He, and Jian Sun (2015). "Instance-aware Semantic Segmentation via Multi-task Network Cascades". In: ISSN: 10636919. DOI: 10.1109/CVPR.2016.343. arXiv: 1512.04412. URL: http://arxiv.org/abs/1512.04412.

Dauphin, Yann et al. (2014). "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: ISSN: 10495258. arXiv: 1406.2572. URL: http://arxiv.org/abs/1406.2572.

De Brabandere, Bert, Davy Neven, and Luc Van Gool (2017). "Semantic Instance Segmentation with a Discriminative Loss Function". In: ISSN: 21607516. DOI: 10.1109/CVPRW.2017.66. arXiv: 1708.02551. URL: https://arxiv.org/pdf/1708.02551.pdfhttp://arxiv.org/abs/1708.02551.

Dodge, Samuel and Lina Karam (2016). "Understanding how image quality affects deep neural networks". In: *2016 8th International Conference on Quality of Multimedia Experience, QoMEX 2016*. ISBN: 9781509003549. DOI: 10.1109/QoMEX.2016.7498955. arXiv: 1604.04004. URL: https://arxiv.org/pdf/1604.04004.pdf.

Furbank, Robert T. and Mark Tester (2011). *Phenomics - technologies to relieve the phenotyping bottleneck*. DOI: 10.1016/j.tplants.2011.09.005. URL: http://www.ncbi.nlm.nih.gov/pubmed/22074787http://linkinghub.elsevier.com/retrieve/pii/S1360138511002093.

Garcia-Garcia, Alberto et al. (2017). "A Review on Deep Learning Techniques Applied to Semantic Segmentation". In: *arXiv preprint*, pp. 1–23. DOI: 10.1007/978-1-4471-4640-7. arXiv: 1704.06857. URL: https://arxiv.org/pdf/1704.06857.pdfhttp://arxiv.org/abs/1704.06857.

Geiger, Andreas, Philip Lenz, and Raquel Urtasun (2012). "Are we ready for autonomous driving? the KITTI vision benchmark suite". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361. ISBN: 9781467312264. DOI: 10.1109/CVPR.2012.6248074. arXiv: 1612.07695. URL: http://www.cvlibs.net/publications/Geiger2012CVPR.pdf.

Girshick, Ross et al. (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 580–587. ISBN: 9781479951178. DOI: 10.1109/CVPR.2014.81. arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep sparse rectifier neural networks". In: *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* 15, pp. 315–323. ISSN: 15324435. DOI: 10.1.1.208.6449. arXiv: 1502.03167. URL: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. URL: http://www.deeplearningbook.org/.

He, Kaiming et al. (2016). "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/.

He, Kaiming et al. (2017). "Mask R-CNN". In: ISSN: 0006-291X. DOI: 10.1109/ICCV.2017.322. arXiv: 1703.06870. URL: https://arxiv.org/pdf/1703.06870.pdfhttp://arxiv.org/abs/1703.06870.

He, Xinwei (2017). *PointNet-Pytorch*. URL: https://github.com/eriche2016/pointnet2.pytorch.

Hinton, Geoffrey E (1986). "Learning Distributed Representations of Concepts". In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* 1.3 i, pp. 46–61. ISSN: 10414347. DOI: 10.1109/69.917563. arXiv: 1312.3005. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.408.7684{\&}rep=rep1{\&}type=pdf.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "LONG SHORT-TERM MEMORY". In: *Neural Computation* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. arXiv: 1206.2944. URL: http://www7.informatik.tu-muenchen.de/{~}hochreithttp://www.idsia.ch/{~}juergenhttp://www7.informatik.tu-muenchen.de/{~}hochreit{\%}5Cnhttp://www.idsia.ch/{~}juergen.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: ISSN: 0717-6163. DOI: 10.1007/s13398-014-0173-7.2. arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

Kanezaki, Asako, Yasuyuki Matsushita, and Yoshifumi Nishida (2016). "RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints". In: arXiv: 1603.06208. URL: https://arxiv.org/pdf/1603.06208.pdfhttp://arxiv.org/abs/1603.06208.

Kendall, Alex and Yarin Gal (2017). "What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?" In: ISSN: 10709878. DOI: 10.1109/TDEI.2009.5211872. arXiv: 1703.04977. URL: https://arxiv.org/pdf/1703.04977.pdfhttp://arxiv.org/abs/1703.04977.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

Krähenbühl, Philipp and Vladlen Koltun (2012). "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: ISSN: 9781618395993. arXiv: 1210.5644. URL: https://arxiv.org/pdf/1210.5644.pdfhttp://arxiv.org/abs/1210.5644.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances In Neural Information Processing Systems*, pp. 1–9. ISSN: 10495258. DOI: http://dx.doi.org/10.1016/j.protcy.2014.09.007. arXiv: 1102.0183. URL: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

LeCun, Yann (1989). "Generalization and network design strategies". In: *Connectionism in perspective*, pp. 143–155. ISBN: 978-0444880611.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444. ISSN: 0028-0836. URL: http://dx.doi.org/10.1038/nature14539http://10.0.4.14/nature14539.

Li, Yangyan et al. (2018). "PointCNN". In: 2. arXiv: 1801.07791. URL: https://arxiv.org/pdf/1801.07791.pdfhttp://arxiv.org/abs/1801.07791.

Lin, Tsung Yi et al. (2014). "Microsoft COCO: Common objects in context". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8693 LNCS. PART 5, pp. 740–755. ISBN: 978-3-319-10601-4. DOI: 10.1007/978-3-319-10602-1_48. arXiv: 1405.0312. URL: https://arxiv.org/pdf/1405.0312.pdf.

Lindenmayer, Aristid (1975). "Developmental algorithms for multicellular organisms: A survey of L-systems". In: *Journal of Theoretical Biology* 54.1, pp. 3–22. ISSN: 10958541. DOI: 10.1016/S0022-5193(75)80051-8. URL: https://www.sciencedirect.com/science/article/pii/S0022519375800518.

Liu, Pascal (2017). *The future of food and agriculture: Trends and challenges*. P. 180. ISBN: 9789251095515. DOI: ISBN978-92-5-109551-5. URL: http://www.fao.org/3/a-i6583e.pdf.

Liu, Suxing et al. (2017). "Novel Low Cost 3D Surface Model Reconstruction System for Plant Phenotyping". In: *Journal of Imaging* 3.3, p. 39. ISSN: 2313-433X. DOI: 10.3390/jimaging3030039. URL: http://www.mdpi.com/2313-433X/3/3/39.

Liu, Wei et al. (2016). "SSD: Single shot multibox detector". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9905 LNCS, pp. 21–37. ISBN: 9783319464473. DOI: 10.1007/978-3-319-46448-0_2. arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325http://dx.doi.org/10.1007/978-3-319-46448-0{_}2.

Long, Jonathan, Evan Shelhamer, and Trevor Darrell (2015). "Fully Convolutional Networks for Semantic Segmentation preprint". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298965. arXiv: 1411.4038.

Macy, Matt (2017). *V-Net - PyTorch*. URL: github.com/mattmacy/vnet.pytorch.

Maturana, Daniel and Sebastian Scherer (2015). "VoxNet: A 3D convolutional neural network for real-time object recognition". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 922–928. ISBN: 9781479999934. DOI: 10.1109/IROS.2015.7353481. URL: https://www.ri.cmu.edu/pub{_}files/2015/9/voxnet{_}maturana{_}scherer{_}iros15.pdf.

McCulloch, Warren S. and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133. ISSN: 00074985. DOI: 10.1007/BF02478259. arXiv: arXiv:1011.1669v3. URL: https://pdfs.semanticscholar.org/5272/8a99829792c3272043842455f3a110e841b1.pdf.

Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi (2016). "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". In: *arXiv preprint arXiv:1606.04797*, pp. 1–11. DOI: 10.1109/3DV.2016.79. arXiv: 1606.04797. URL: http://arxiv.org/abs/1606.04797.

Minervini, Massimo, Hanno Scharr, and Sotirios A. Tsaftaris (2015). "Image analysis: The new bottleneck in plant phenotyping [applications corner]". In: *IEEE Signal Processing Magazine* 32.4, pp. 126–131. ISSN: 10535888. DOI: 10.1109/MSP.2015.2405111. URL: http://ieeexplore.ieee.org/document/7123050/.

Nguyen, Anh and Bac Le (2013). "3D point cloud segmentation: A survey". In: *IEEE Conference on Robotics, Automation and Mechatronics, RAM - Proceedings*. IEEE, pp. 225–230. ISBN: 9781479911998. DOI: 10.1109/RAM.2013.6758588. arXiv: 15334406. URL: http://ieeexplore.ieee.org/document/6758588/.

Qi, Charles R. et al. (2016). "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: DOI: 10.1109/3DV.2016.68. arXiv: 1612.00593. URL: http://arxiv.org/abs/1612.00593.

Qi, Charles R et al. (2017a). "Frustum PointNets for 3D Object Detection from RGB-D Data". In: arXiv: 1711.08488. URL: https://arxiv.org/pdf/1711.08488.pdfhttp://arxiv.org/abs/1711.08488.

Qi, Charles R. et al. (2017b). "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: arXiv: 1706.02413. URL: http://arxiv.org/abs/1706.02413.

Ren, Mengye and Richard S Zemel (2016). "End-to-End Instance Segmentation with Recurrent Attention". In: ISSN: 1063-6919. DOI: 10.1109/CVPR.2017.39. arXiv: 1605.09410. URL: https://arxiv.org/pdf/1605.09410.pdfhttp://arxiv.org/abs/1605.09410.

Ren, Shaoqing et al. (2017). "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6, pp. 1137–1149. ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2577031. arXiv: 1506.01497. URL: https://arxiv.org/pdf/1506.01497.pdf.

Riegler, Gernot, Ali Osman Ulusoy, and Andreas Geiger (2016). "OctNet: Learning Deep 3D Representations at High Resolutions". In: ISSN: 1063-6919. DOI: 10.1109/CVPR.2017.701. arXiv: 1611.05009. URL: https://arxiv.org/pdf/1611.05009.pdfhttp://arxiv.org/abs/1611.05009.

Robbins, Herbert and Sutton Monro (1951). "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3, pp. 400–407. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729586. URL: http://projecteuclid.org/euclid.aoms/1177729586.

Romera-Paredes, Bernardino and Philip Hilaire Sean Torr (2016). "Recurrent instance segmentation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9910 LNCS, pp. 312–329. ISSN: 16113349. DOI: 10.1007/978-3-319-46466-4_19. arXiv: 1511.08250. URL: https://arxiv.org/pdf/1511.08250.pdf.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536. ISSN: 00280836. DOI: 10.1038/323533a0. arXiv: arXiv:1011.1669v3. URL: http://www.nature.com/doifinder/10.1038/323533a0.

Sabour, Sara, Nicholas Frosst, and Geoffrey E Hinton (2017). "Dynamic Routing Between Capsules". In: arXiv: 1710.09829. URL: http://arxiv.org/abs/1710.09829.

Salvador, Amaia et al. (2017). "Recurrent Neural Networks for Semantic Instance Segmentation". In: arXiv: 1712.00617. URL: http://arxiv.org/abs/1712.00617.

Scharr, Hanno et al. (2016). "Leaf segmentation in plant phenotyping: a collation study". In: *Machine Vision and Applications* 27.4, pp. 585–606. ISSN: 14321769. DOI: 10.1007/s00138-015-0737-3. URL: https://www.cse.msu.edu/{~}yinxi1/docs/MVA15.pdf.

Shi, Xingjian et al. (2015). "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: ISSN: 10495258. arXiv: 1506.04214. URL: https://arxiv.org/pdf/1506.04214.pdfhttp://arxiv.org/abs/1506.04214.

Shrivastava, Ashish et al. (2017). "Learning from Simulated and Unsupervised Images through Adversarial Training". In: *arXiv*, pp. 1–16. ISSN: 1063-6919. DOI: 10.1109/CVPR.2017.241. arXiv: arXiv:1612.07828v2. URL: https://arxiv.org/pdf/1612.07828.pdf.

Simonyan, Karen and Andrew Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations (ICRL)*, pp. 1–14. ISSN: 09505849. DOI: 10.1016/j.infsof.2008.09.005. arXiv: 1409.1556. URL: https://arxiv.org/pdf/1409.1556.pdfhttp://arxiv.org/abs/1409.1556.

Springenberg, Jost Tobias et al. (2014). "Striving for Simplicity: The All Convolutional Net". In: ISSN: 02548704. DOI: 10.1163/_q3_SIM_00374. arXiv: 1412.6806. URL: https://arxiv.org/pdf/1412.6806.pdfhttp://arxiv.org/abs/1412.6806.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15, pp. 1929–1958. ISSN: 15337928. DOI: 10.1214/12-AOS1000. arXiv: 1102.4807. URL: http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm{_}content=buffer79b43{\&}utm{_}medium=social{\&}utm{_}source=twitter.com{\&}utm{_}campaign=buffer.

Su, Hang et al. (2015). "Multi-view Convolutional Neural Networks for 3D Shape Recognition". In: *Ieee Iccv*, pp. 945–953. ISSN: 15505499. DOI: 10.1109/ICCV.2015.114. arXiv: 1505.00880. URL: https://arxiv.org/pdf/1505.00880.pdfhttp://vis-www.cs.umass.edu/mvcnn/docs/su15mvcnn.pdf.

Waleed, Abdulla (2017). *Mask-RCNN - TensorFlow*. URL: https://github.com/matterport/Mask{_}RCNN.

Wang, Yue et al. (2018). "Dynamic Graph CNN for Learning on Point Clouds". In: arXiv: 1801.07829. URL: https://arxiv.org/pdf/1801.07829.pdfhttp://arxiv.org/abs/1801.07829.

Yu, Fisher and Vladlen Koltun (2015). "Multi-Scale Context Aggregation by Dilated Convolutions". In: ISSN: 00237205. DOI: 10.16373/j.cnki.ahr.150049. arXiv: 1511.07122. URL: https://arxiv.org/pdf/1511.07122.pdfhttp://arxiv.org/abs/1511.07122.

Zhang, Zhanpeng et al. (2014). "Facial landmark detection by deep multi-task learning". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8694 LNCS. PART 6. Springer, Cham, pp. 94–108. ISBN: 9783319105987. DOI: 10.1007/978-3-319-10599-4_7. arXiv: 1604.02878. URL: http://link.springer.com/10.1007/978-3-319-10599-4{_}7.

Zhao, Hengshuang et al. (2016). "Pyramid Scene Parsing Network". In: DOI: 10.1109/CVPR.2017.660. arXiv: 1612.01105. URL: http://arxiv.org/abs/1612.01105.

Zhou and Chellappa (1988). "Computation of optical flow using a neural network". In: *IEEE International Conference on Neural Networks* 86, 71–78 vol.2. DOI: 10.1109/ICNN.1988.23914. URL: http://ieeexplore.ieee.org/document/23914/.

Zhou, Yin and Oncel Tuzel (2017). "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection". In: *arXiv*. arXiv: 1711.06396. URL:

https://arxiv.org/pdf/1711.06396.pdfhttp://arxiv.org/abs/1711.06396.

Declaration of Authorship

Ich erkläre hiermit gemäß § 17 Abs. 2 APO, dass ich die vorstehende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Unterschrift:

Datum:
