Rheinische
Friedrich-Wilhelms-Universität Bonn

Bachelor thesis

Superresolution 3D Image Segmentation for Plant Root MRI

Author:
Jannis Horn

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Prof. Dr. Joachim K. Anlauf

Submitted: August 27, 2018

Declaration of Authorship

I declare that the work presented here is original and the result of my own investigations. Formulations and ideas taken from other sources are cited as such. It has not been submitted, either in part or whole, for a degree at this or any other university.

Location, Date

Signature

Abstract

Magnet Resonance Imaging (MRI) can be used to depict plant root architecture through opaque soil without the use of invasive markers. To automatically extract the root architecture reliably from the sampled MRI images the resolution and the signal-to-noise ratio have to be enhanced. This is a super resolution (SR) problem. Increasing the signal-to-noise-ratio can be done by image segmentation. Both problems can be solved using convolutional neural networks. This thesis applies shallow convolutional neural networks, trained on synthetic plant root data, to the problem of 3D image segmentation and the joint problem of image segmentation and super resolution for plant root MRI. These networks are analyzed based on the used architecture and their parameter and guidelines for these parameters are extracted.

Contents

1 Introduction

1.1 Motivation

Plant root architecture is a crucial part of soil exploration. To analyze the root architecture without altering or destroying it, non-invasive imaging methods are needed. Magnet Resonance Imaging (MRI) is one such imaging method. MRI creates a volumetric image of the scanned area by primarily measuring water content.

Root architecture has to be extracted from the scan. This is mostly done by hand, utilizing a thresholded MRI scan. The thresholding is done to reduce most of the background noise. Doing so is time-consuming and can result in losing root tips or thinner roots due to the threshold being to great, see e.g. Figure 1.1 .

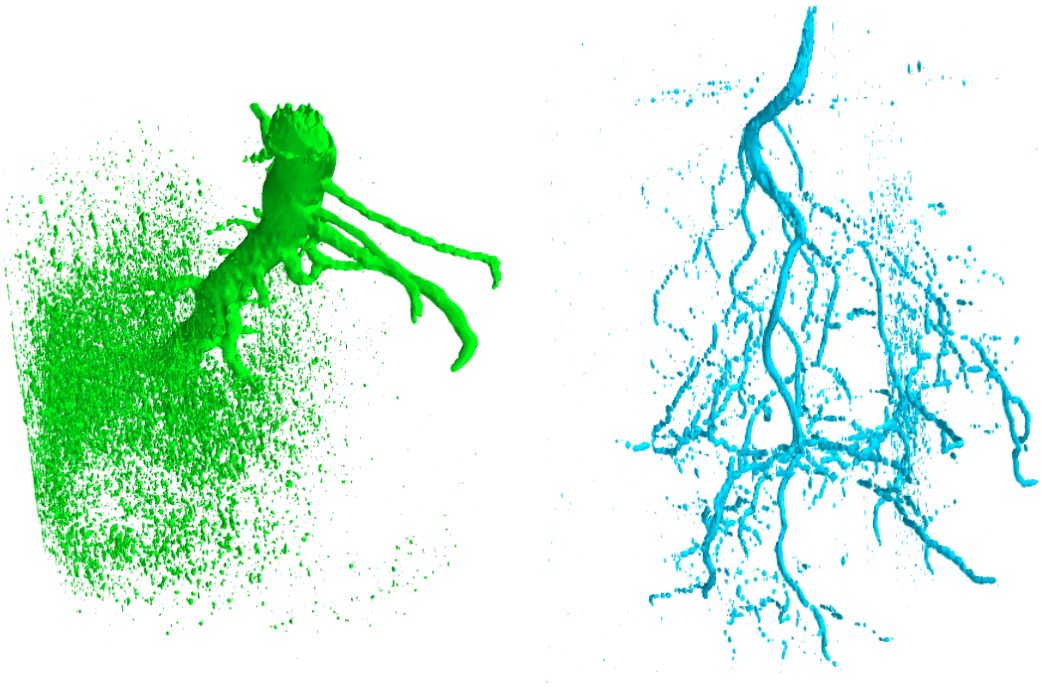
An algorithm to automatically extract the root architecture from MRI scans has already been developed (Schulz et al., 2013). The performance of this algorithm depends heavily on scan resolution and signal-to-noise ratio (SNR). Limited resolution may also lead to voxels being larger than thinner roots loosing these details. Scans with insufficient resolution and low SNR loose large portions of structure in reconstruction.

To allow for scans with low resolution or low SNR to be used, software post-processing may be applied.

This thesis seeks to explore the application of simpler convolutional neural networks to increase the resolution and SNR of 3D plant root MRIs. The problem is therefore split into:

1. **Increase SNR** by decreasing intensity of noise voxels while retaining or increasing intensity of root voxels. This can be understood as Image Segmentation problem.

2. **Increase scan resolution** while trying to recapture details not captured in the original scan due to low resolution. This is the problem of Super Resolution.



(a) Dataset: lupine_small. Voxel with intensity 127 or higher are depicted. This reduces noise to some extend.

(b) Dataset: lupine_22. Voxel with intensity 45 or higher are depicted. This reduces noise and also looses a lot of thinner roots which creates disconnected branches.

Figure 1.1: 3D visualization of thresholded plant root MRI

1.2 Structure and Contribution

Chapter 2 will describe the problems of image segmentation and super resolution together with related work on these two subjects. Theoretical background used in this thesis is described in chapter 3. This thesis contains the following contributions

1. A set of shallow 3D convolutional neural network architectures to be applied to the problem of plant root MRI scan segmentation.
2. Shallow convolutional neural networks used for scan segmentation and scan super resolution. Both are described in Chapter 5.
3. Described in Chapter 6 the evaluation of the proposed networks on the artificially created plant root datasets.
4. Application of trained networks on real plant root MRI scans at the end of chapter 6.

Chapter 7 will give an outlook on possible further work on this topic.

2 Problems

2.1 Image Segmentation

Increasing the signal-to-noise ratio of given plant root MRI scan necessitates the increase of signal intensity (root voxels) while decreasing the intensity of noise voxels. Therefore a way of identifying root or soil is needed. The voxel intensity should then be adjusted accordingly.

This can be understood as an object classification task. More specific dense voxel-wise classification is of interest. The problem of dense object classification is known as semantic segmentation which is part of image segmentation.

Image segmentation describes the problem of decomposing an image into disjunct sets or classes. The objective is to change the representation of the image while retaining its structural relation. Of interest for plant root MRI image segmentation are the classes root and soil.

Problem

Formally the problem of image segmentation for 3D plant root MRI can be defined as:

Let $I \in \mathbb{R}^{X,Y,Z}$ be a 3D plant root MRI. The function $int_I(x, y, z) \in [0.0, 1.0]$ denotes the voxel intensity at position x, y, z in I and $v_I(x, y, z)$ denotes the position itself with $0 \leq x < X \in \mathbb{N}$, $0 \leq y < Y \in \mathbb{N}$ and $0 \leq z < Z \in \mathbb{N}$. Create a representation I_{seg} mapping each position $v_I(x, y, z) \in I$ onto the modified intesity $int_{I_{seg}}(x, y, z)$, with

$$int_{I_{seg}}(x, y, z) = \begin{cases} 1, & v_I(x, y, z) \in R \\ 0, & v_I(x, y, z) \in S \end{cases}$$

With R denoting the set of voxels in I containing root signal and S denoting the voxels containing soil/noise. Therefore a transformation function f_{seg} has to be found with $f_{seg}(I) = I_{seg}$.

A classifier set for R and S can be employed to map each voxel onto these classes. As only two disjunct classes are of interest this is a binary classification

problem. This allows describing the classes R and S as R and $-R$. Therefore only one classifier is needed. Classifiers for semantic segmentation usually return a confidence value instead of a binary value. This return value for a root classifier c_R describes the conditional likelihood $P(v_I(x, y, z) \in R \mid c_R)$ of a voxel to carry root signal. To enhance the accuracy of the segmentation instead of a binary value for $I_{seg}(x, y, z)$ this confidence value is used as intensity value $int_{I_{seg}}(x, y, z)$.

Related Work

Deep learning based models are state-of-the art for image classification (Simonyan and Zisserman, (2014), He et al., (2016)) and semantic segmentation for 2D images (G. Lin, Shen, et al., 2016, Girshick et al., (2014)).

Long et al., (2015) uses a fully convolutional network for image segmentation. This approach allows for the input image to be of arbitrary size. Their approach uses convolution followed by pooling to reduce the image size while broadening the network. Then deconvolution with stride larger 1 is applied to upsample the image up to the original resolution.

By using pooling layers followed by upsampling, finer details in the image segmentation can be lost. G. Lin, Milan, et al., (2017) address this issue by using RefineNet blocks (He et al., 2016) to create different resolution feature maps. Lower resolution maps are then upsampled to the highest resolution and fused by summation. This multi-scale approach utilizes both low level features from the high resolution feature maps and high level features in the upsampled low resolution maps.

MRI segmentation is researched primarily in the context of medical imaging and has often been applied for brain region segmentation (Heckemann et al., (2006), Colliot et al., (2006), Despotović et al., (2015)). This includes methods for automatic MRI segmentation from simple thresholding algorithms for classification of different tissue classes based on thresholds recovered from scan histograms (Despotović et al., 2015) to utilizing handcrafted maps of the brain (atlases) for brain segmentation (Makropoulos et al., (2014), Despotović et al., (2015)).

Recently deep learning based methods have been developed for MRI image segmentation. Milletari et al., (2016) applied deep fully convolutional networks on prostate MRI. They constructed a fully convolutional end-to-end pipeline utilizing 3D convolution. To address the problem of class imbalance between fore- and background they proposed dice loss instead of re-weighting, outperforming the latter.

Kamnitsas et al., (2017) employ multi scale 3D convolutional networks followed by 3D Conditional Random Field layers Krähenbühl and Koltun, (2011) for final

classification while retaining the ability to input scans of arbitrary size. To address problems of computation time and memory requirements, scan bigger than a certain predefined size are computed in segments and each segment gets treated as full input, therefore increasing batchsize while keeping memory and computational requirements low.

2.2 Super Resolution

Increasing the resolution of given 3D plant root MRI while also increasing the amount of information entails the reconstruction of details lost in the low resolution scan. Doubling the resolution in each dimension of an MRI scan increases the number of voxels by 8. In the low resolution scan, these 8 values are fused together by some metric. Therefore a pipeline which is capable of inferring the missing details from the low resolution scans is needed. This problem is known as super resolution. More specifically single image or scan super resolution as only one scan is used as input to create one corresponding high resolution output.

As this problem occurs due to information fusion during the sampling during image acquisition, the super resolution problem can be described as the inverse problem of the image formation process. This is naturally ill posed as there are multiple different high resolution patches that can correspond to given low resolution patch. The problem of single scan super resolution can be described as:

Problem

Let I be a 3D plant root MRI scan as defined in Section 2.1. To solve the super resolution problem for upsample factor λ_{ups}, a representation I_{sr} is needed which maps each $v_I(x, y, z) \in I$ onto the voxel group $V_{I_{sr}}(x, y, z)$ with

$$V_{I_{sr}}(x, y, z) = \{v_{I_{sr}}(x_{sr}, y_{sr}, z_{sr}) | x_{sr} \in Coor_{\lambda_{ups}}(x), y_{sr} \in Coor_{\lambda_{ups}}(y), z_{sr} \in Coor_{\lambda_{z}(z)}\}$$

and

$$Coor_{\lambda_{ups}}(n) = \{n_{sr} \in \mathbb{N} | \lambda_{ups} \cdot n \leq n_{sr} < \lambda_{ups} \cdot (n + 1)\}$$

This mapping is done using a function f_{sr} with $f_{sr}(I) = I_{sr}$.

In particular f_{sr} has to map each voxel $v_I(x, y, z) \in I$ onto the group $\{int_{I_{sr}}(v_{I_{sr}}) | v_{I_{sr}} \in V_{I_{sr}}(x, y, z)\}$. The low resolution value $int_I(x, y, z)$ gets generated by some function g_{sr} with

$$g_{sr}(V_{I_{sr}}(x, y, z)) = int_I(x, y, z)$$

which usually is the mean intensity value of all elements of $V_{I_{sr}}(x, y, z)$. Simply

inversing this function is not possible because g_{sr} usually maps multiple different voxel groups onto the same value.

Related Work

Solutions for the super resolution problem can be divided into multi image and single image super resolution. The first approach uses multiple images of the same scene preferable offset by subpixel steps. These are then fused into one high resolution image.

This thesis looks at single image super resolution for plant root MRI. Therefore the details missing in low resolution have to be inferred from one single input. This can be done using an example based method which either uses self similarities within the image (Cui et al., 2014) or learns a mapping function from low to high resolution.

State of the art single image super resolution utilizes deep convolutional neural networks (Dong et al., (2016)). These usually learn an end-to-end mapping from linearly upsampled low resolution image (Kim et al., 2016a) to high resolution or directly from original low resolution to high resolution by upsampling the image after stages of feature extraction.

Kim et al., (2016b) use a deep recursive architecture to first extract features and reassembling them in higher resolution. Therefore a feed forward net extracts features from the input. By feeding these feature maps into a convolutional recursive stage, step by step larger image context is taken into account. In a last step, the feature maps created by recursion are sent into a reconstruction stage to create the high resolution output.

Instead of upsampling the low resolution image before inputting it into the network, Lai et al., (2017) employ a combination of feature extraction on low resolution images followed by a transposed convolution layer upsampling the image by a factor of 2. This is then combined with an output of another transposed convolution. For upsample factors larger then 2, multiple such networks are combined.

Ledig et al., (2017) use generative adversarial networks (Goodfellow, Pouget-Abadie, et al., 2014) for single image super resolution. Therefore the high resolution image is estimated by a generator network consisting of multiple residual blocks. A discriminator network is trained to discriminate between real data and generator output. This feedback is then used to train the generator.

Lately, deep convolutional neural networks have also been applied to single image super resolution for medical MRI (Pham, Ducournau, et al., 2017). Pham, Fablet, et al., (2017) examine multiple parameters and their impact on performance for brain MRI super resolution. Therefore deep convolutional neural networks with

residual connections are applied.

2.3 Class Imbalance

A problem often encountered in machine learning is the problem of class imbalance. Given a classification problem with 2 classes, the problem occurs if the datapoints corresponding to one class far outnumber the datapoint corresponding to the other.

Training a classifier on such a dataset, using the amount of errors as metric, may lead to a significant fraction of the smaller class to be classified wrongly. This is due to any number of wrong classification for one class are compared to the number of wrong classification for the other. Given a difference of e.g. 1:1000 this would mean classifying 50% of the smaller class incorrectly, has as much of an impact on the learner as classifying 0.05% of the larger class incorrectly.

In the used plant root scans the number of soil voxels outnumbers the number of root voxels by a large amount.A learned classifier tries to minimize the loss over all datapoints, which leads to classifying 10% of soil correctly being more influential than classifying 10% of root correctly.

For full structure extraction classifying root correctly is most important, as voxels falsely classified as soil are also lost in the extracted structure. Soil voxels classified as root can more easily be interpreted as such if they are not part of connected structures. Therefore it may be preferable to accept a number of false positives for a smaller decrease in false negatives.

There are several approaches to address the problem of class imbalance for training artificial neural networks. Zhou and Liu, (2006) look into the effects of multiple different methods. The first being oversampling. This method samples a disproportionate amount of examples from the small class so that the effect of it on the overall training is equal to the bigger class. Opposite to this is undersampling. Therefore the number of bigger class examples is reduced. A third method discussed is threshold moving. Each class gets assigned a value dependent on the actual output class. During testing, the output values are multiplied by these values therefor shifting the thresholds between classes. The last method discussed is ensemble learning. Therefore multiple classifiers are trained with different kinds of resampling or threshold moving. The multiple classifications are then combined.

T.-Y. Lin et al., (2018) propose a focal loss for the problem of one-stage object detection. Focal loss is based on cross entropy and applies a dynamic loss scaling based on a confidence measure during training. This is done by reducing the loss the smaller it already is, therefor reducing the impact of datapoints that are already close to being correctly classified.

3 Theoretical Background

3.1 Convolutional Neural Networks

Convolutional neural networks are a specialized type of artificial neural networks primarily employed on data carrying a known grid like structure (Goodfellow, Bengio, et al., 2016) for example signal processing along time (1D structure) or image processing (2D structure). This type of network is inspired by the structure of the visual cortex. A convolutional neural network contains one or more convolutional layers.

3.1.1 Convolution

Convolution is a mathematical operation taking two functions as input and combining these into an output function. On two real valued functions convolution is defined as:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\lambda)g(t-\lambda)d\lambda \tag{3.1}$$

This operation is commutative. Convolution is defined for every function pair for which this integral is defined.

As computer usually work on segmented data, e.g. pixel in an image, the input parameter t can be restricted to integer values. Therefore 3.1 can be rewritten to:

$$(f * g)(t) = \sum_{\lambda=-\infty}^{\infty} f(\lambda)g(t-\lambda) \tag{3.2}$$

In the context of convolutional neural networks f is referred to as **input** I and g as **kernel** K.

Image or MRI scans have topological relationships along 2 and 3 axis respectively. Therefore convolution along multiple axis is of interest. Specifically convolution for inputs with 3 dimensions with input size X, Y, Z can be defined as:

$$(K * I)(x, y, z) = \sum_{m=0}^{X-1}\sum_{n=0}^{Y-1}\sum_{o=0}^{Z-1} I(x-m, y-n, z-o)K(m, n, o) \tag{3.3}$$

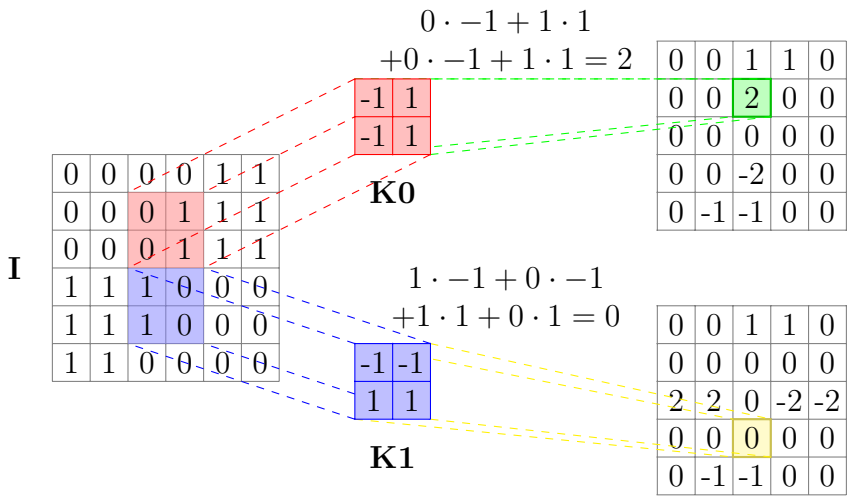


Figure 3.1: Convolution operation applying two 2D kernel $K0$, $K1$ onto a 2D input I

Usually the kernel is a very sparse matrix with the non-zero values accumulated in a small area around the input position x, y, z. This is due to the convolution kernel being much smaller than the input. By limiting the kernel size only a local neighborhood of the input is convolved. Figure 3.1 shows the application of convolution along two axis on a 2D input. Goodfellow, Bengio, et al., 2016.

3.1.2 Convolutional Layer

The convolutional layer l applies the convolution operation in the context of artificial neural networks. Therefore the kernel as well as a bias is trained together with the rest of the network. The output is also called **feature map**.

Given an input with $size_{inp} = x$ and an output with $size_{out} = y$: A fully connected layer uses matrix multiplication to transform an input into output. Therefore a $x \cdot y$ weight matrix is needed with $O(x \cdot y)$ operations. This means that every input unit interacts with every output unit.

Compared to fully connected layers convolutional layers operate on a different set of ideas. By using a small convolutional kernel with $size_K = k << x$ each input unit only interacts with a small amount of output units. This is called sparse interaction. Using this principle only k parameters are needed and only $k \cdot y$ operations.

Due to all outputs being computed using the same kernel weights the same feature is extracted at all positions in the input. Therefore the whole input is computed for a learned local structure. This also leads to one small kernel being trained over the whole input. The low number of parameters also reduces the

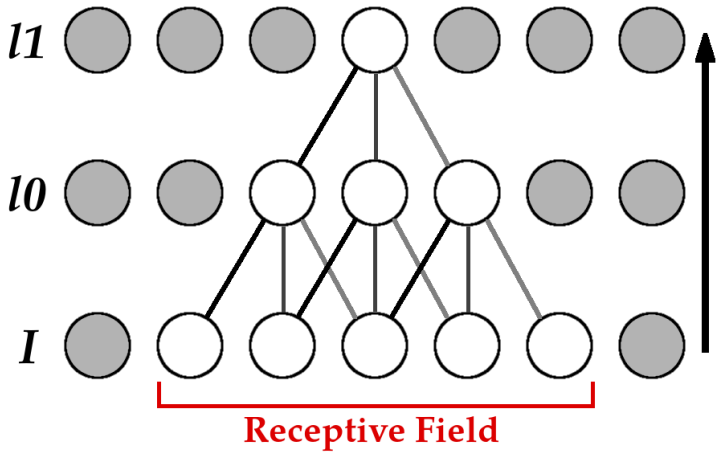


Figure 3.2: 1D Receptive field growing through 2 convolutional layers $l0$, $l1$ with kernel size 3, applied onto a 1D input I

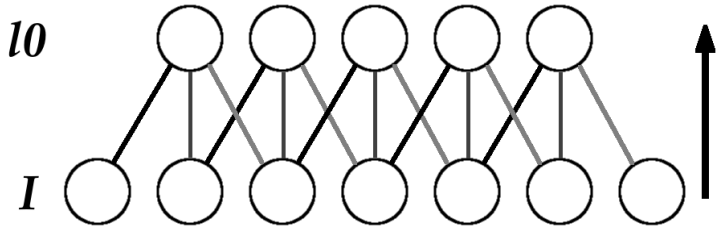


Figure 3.3: Applying the convolutional layer $l0$ with kernel size 3 onto input I with $size_{inp} = 7$ reduces the $size_{out}$ to 5

amount of memory needed for parameter storing by a large amount given that k is a lot smaller than $size_{inp}$.

The **receptive field** of a convolution is the area of input units that affect a given output unit. For a single convolutional layer this is the same as its kernel size. With each layer of convolution with a kernel size bigger than 1 the receptive field increases. Figure 3.2 shows the growing amount of input units affecting each output unit when using two convolutions with a kernel of size three. In the context of a convolutional neural network the receptive field is described by the receptive field of the output layer in context of the network input.

When applying convolutions with a kernel size larger then 1 not all positions within the input can be evaluated without the kernel leaving the bounds of the input. As can be seen in Figure 3.3 this leads to the output decreasing in size. This is known as valid padding. By adding zeros to the outside of the input the size can temporarily be increased. This allows to compute an output of the same size as the input. This can lead to unwanted effects along the borders in the output

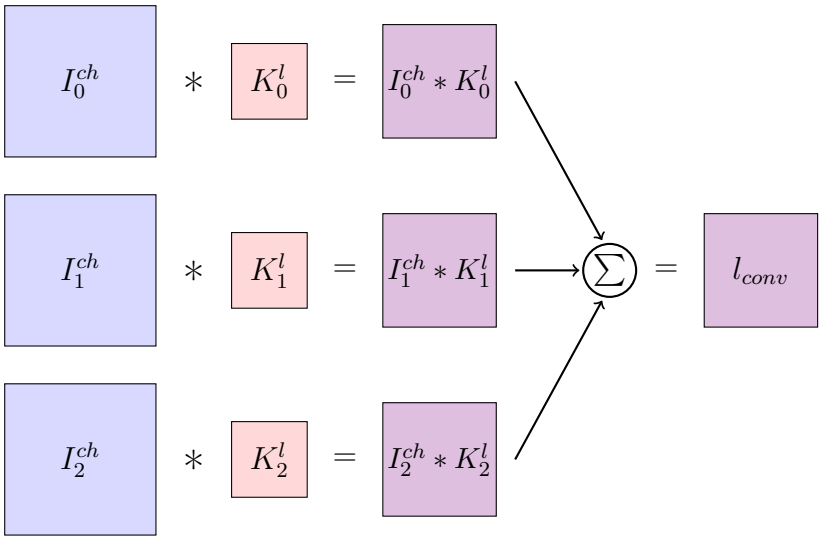


Figure 3.4: Applying a convolutional layer l with $ch_{in} = 3$ input channels and $ch_{out} = 1$ output channels onto an input I with channels I_0^{ch}, I_1^{ch} and I_2^{ch}. The parameters of the convolutional layer are the convolution kernels K_0^l, K_1^l and K_2^l.

as zero-padding introduces further structure to the input.

It is useful to extract more than one feature from given input. Therefore multiple **channels** per layer are employed. Each channel has its own weight kernel and bias which gets applied onto the input. This increases the dimensionality of the input. This multi channel output can be computed by further convolutional layers. Therefore for each output channel a distinct kernel is applied to each input channel. This necessitates $ch_{in} \cdot ch_{out} \cdot k$ weights. All activations are summed up.

The output vector $v_O \in \mathbb{R}^{ch_{out}}$ at position t with given input $v_I \in \mathbb{R}^{ch_{in} \cdot size_K}$ of a convolutional layer with non linearity $f()$, kernels $K_{0,0}, ..., K_{ch_{in},ch_{out}}$ and bias b can be defined as:

$$v_O = f(\sum_{i=0}^{ch_{in}-1} v_I[i] * K_i + b_i)$$

To allow a generic number of channels per layer most frameworks code inputs for convolutional layers as $[ch_{inp}, ch_{out}, input]$. See Figure 3.4 for an example of a convolution onto multiple input channels.

Convolution is a linear operation. This limits a network only using convolution with bias to only learning linear functions. To address this a non-linearity is applied element-wise on the output of each convolution. The non-linearities used in this thesis are:

- **ReLU:**

$$ReLU(x) = max(0, x)$$

- **Sigmoid:**

$$Sigmoid(x) = \frac{1}{1 - e^{-x}}$$

- **TanH:**

$$TanH(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

As the kernel size and therefore the amount of network parameters is independent of the input size, a fully convolutional neural network can handle inputs of different sizes without the need for retraining.

A basic convolutional layer can be described by the following parameters: ch_{in}, ch_{out}, $size_K$ and *non-linearity*.

3.2 Batch Normalization

Batch normalization (Ioffe and Szegedy, 2015) is a method to normalize the input of a neural layer to speed up training. This method seeks to reduce the internal covariate shift of network layers. Internal covariate shift describes the change in distribution of network layer activation occurring during training.

This normalization is done by subtracting the activation mean for given channel and by dividing this output by the activation standard deviation:

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x] + e}} \tag{3.4}$$

(Ioffe and Szegedy, 2015). $E[x]$ and $Var[x]$ are computed independently for every training iteration.

By doing this normalization the representational power of the network is reduced. To address this issue batch normalization adds two learnable parameters β, γ for each channel. These parameters are updated during normal training. To increase representational power of given network the parameters are applied as follows:

$$y = \gamma \cdot \hat{x} + \beta \tag{3.5}$$

This operation can revers the normalization given that: $\gamma = \sqrt{Var[x] + e}$ and $\beta = E[x]$ if useful for the trained task.

In the case of a fully connected layer each output value gets its own set of normalization parameters. Therefore to compute meaningful mean and deviation minibatches with size > 1 are needed. For convolutional layers mean and standard deviation are computed per full channel. This allows to employ batch normaliza-

tion on fully convolutional networks with a minibatch size of one.

Normalizing the output allows for the use of higher learning rates. Due to the normalization the scale of the output is unaffected by parameter scale. This also reduces the impact of exploding and vanishing gradients by for example stopping small parameter changes to accumulate and getting stuck in saturated areas of nonlinearities.

4 Plant Root Data

4.1 Real Root MRI scans

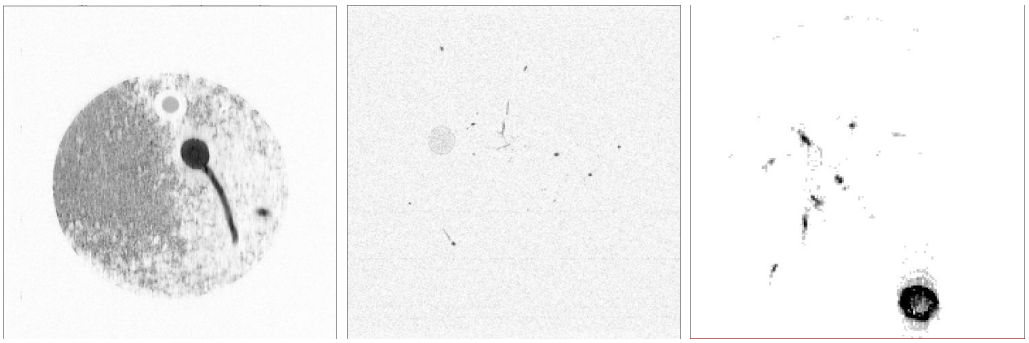
At the time of this thesis three types of real plant root MRI scans are available:

1. **lupine_small**: This scan contains one large, thick main branch from which multiple smaller ones protrude.
2. **lupine_22_august**: This scan contains a large amount of thin roots.
3. **gtk**: This scan contains multiple vertical small roots which are disjointed and not fully connected.

Figure 4.1 shows 2D slices of these real MRI scans showing the root structure as well as the noise present in these scans.

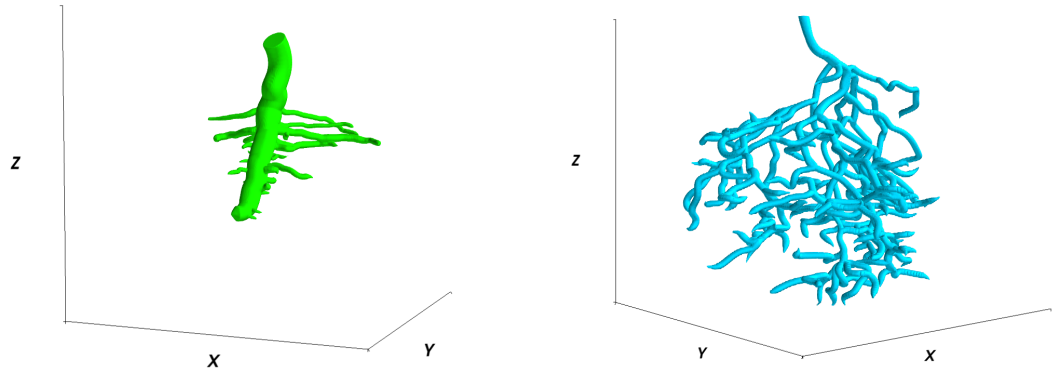
The root structure of these scans was manually extracted. Each root structure is encoded as tree with each branch being parametrized by root radius and end point. Figure 4.2 shows the 3D reconstruction of lupine_small and lupine_22.

These reconstructions are not perfect. Not all angles especially on small roots are completely correct. Also as only the radius is given, insufficient information is given for roots not having a circular profile. As voxel wise classification is of interest these small differences lead to the manual reconstruction being ill suited as ground truth for the real MRI scans.



(a) **(b)** **(c)**

Figure 4.1: Real plant root MRI scans: (a) lupine_small, (b) lupine_22, (c) gtk



(a) (b)

Figure 4.2: 3D reconstructed root from lupine_small and lupine_22. As the voxels in the original scan are not cubic the 3D image is upsampled by a factor of 2 along the z-axis. As this is not fully correct the reconstruction in this figure looks compressed along the z-axis.

4.2 Synthetic Plant Root MRI

Artificial neural networks need large amount of accurately labeled training data to train and later generalize well. As the three real datasets are insufficient and no usable voxel accurate ground truth for these is available, the data for training has to be synthesized.

The baseline for creating artificial data are the manual reconstructed roots. A tree structure is taken from the manual reconstruction file. Therefore splines are fitted onto the tree from the manual reconstruction. Then the radius is used to fit tubular structures over the splines. To increase variability in the dataset and increase the ability of trained networks to generalize well, the radius is iterated with four factors: 0.34, 0.71, 1.00 and 1.41. Also the structures are rotated by 60° and 120°.

To transform the reconstructed structures into usable input, 3D grids of different resolutions are imposed onto these structures. For each voxels in these grids the occupancy is computed. This is done by taking the percentage of reconstructed root within each voxel. The resolution used are dependent on the real MRI data, also see Table 4.1.

To further increase variability the reconstructed occupancy grid is flipped along the x-axis, along the y-axis and x and y coordinates are swapped. This creates a total of 96 ground truth files per root type per resolution.

This ground truth is then used to create data for training. Therefore the noise

Root Type	Natural resolution	Super resolution
lupine_small	256x256x128	512x512x256
lupine_22_august	256x256x120	512x512x240
gtk	183x183x613	366x366x1226

Table 4.1: Datasize of reconstructed plant root MRI

structure of the available real data is used as blueprint for noising these reconstructed ground truth.

Training data is created by applying a contrast reduction on the reconstructed occupancy grid. This is then offset by a small value. Doing so creates a background value bigger 0. Then a noise mask is added onto this data. In a last step the values are then cut off at 0 and 1.

The first scan to be analyzed is lupine_small. There seem to be 3 main types of structure in the lupine_small noise:

1. Large connected areas of solid noise. Dividing the scan into high and low intensity areas. See Figure 4.3c.

2. Small spots of connected intensity shifts. See Figure 4.3d.

3. Random salt-and-pepper noise primarily in areas of already high noise intesity. See Figure 4.3d.

To allow for the networks trained on synthetic data to work on real data, the reconstructed data has to be close to the data distribution the real scans are sampled from.

This noise distribution is approximated by combining three types of noise:

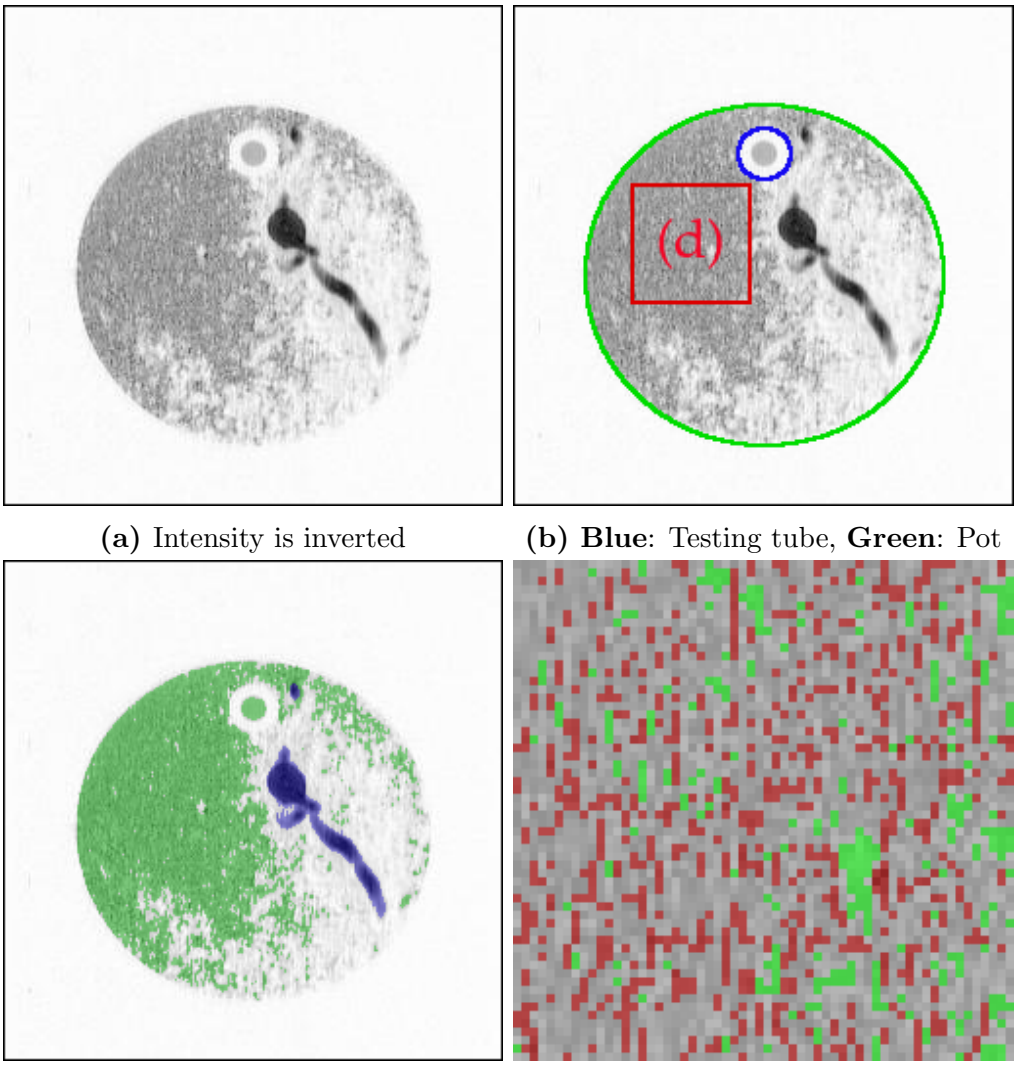
1. Perlin noise (Perlin, 2004) was selected to model the large connected areas of background noise.

2. A large number of small 3D gaussian blobs is then added

3. Uniformly distributed noise guided by the already existing noise intensity is added.

From these steps a noise mask of the same shape as the reconstructed occupancy grid is created.

The Perlin noise generates large connected patches of higher intensity noise. In practice 3 iterations with an initial number of 3D grid cells of 4x4x2 for lupine_small (size: 256x256x128 voxel) are applied. The intensity of the perlin noise is sampled



(a) Intensity is inverted

(b) **Blue**: Testing tube, **Green**: Pot

(c) **Blue**: Root (not pixel accurate), **Green**: Intensity larger 55

(d) **Red**: Areas with intensity larger 109, **Green**: Intensity smaller 70

Figure 4.3: 2D Slice from lupine_small (slice 75 along z-axis). Intensity refers to the intensity of the original scan scaled between 0 and 255. These intensity values are inverted for visualization.

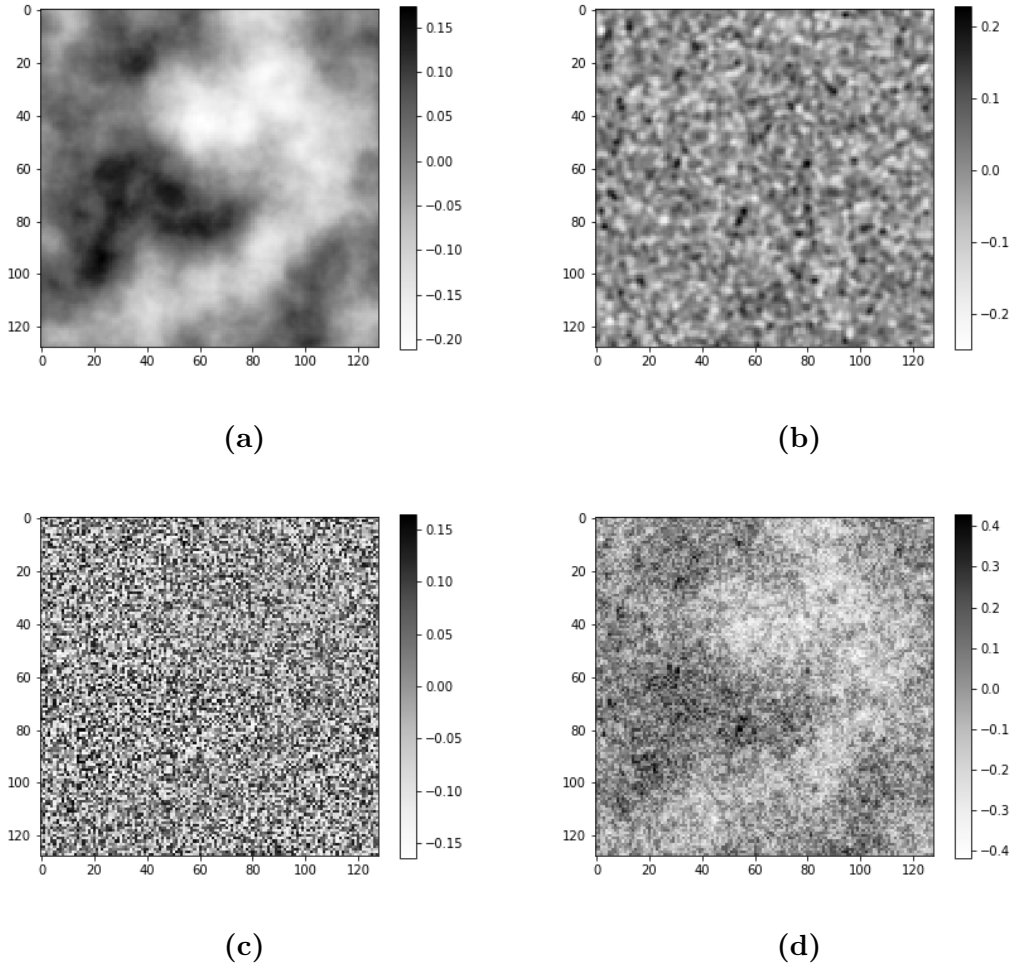


Figure 4.4: 2D Slice taken from a randomly generated noise mask. The figure only shows a quarter of the original slice. a) shows Perlin noise generated using 3 iterations with starting frequency 4,4,2. b) shows the added small gaussian blobs. c) shows the guided uniform noise. d) shows the complete noise mask.

uniformly once per scan from the interval $[0.25, 0.4]$. The sampled value serves as upper bound and its inverse as lower bound for the perlin noise therefore keeping it zero-mean. Each iteration after the first doubles the number of grid cells along all axis and halves the intensity. See Figure 4.4a for a 2D split of this perlin noise

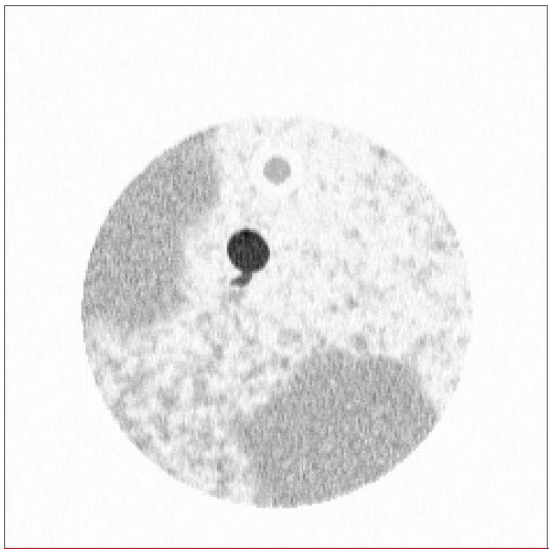
3D Gaussian blobs simulate small connected intensity shifts. To keep these shifts zero-mean the mean value of each blob is uniformly sampled from the interval $[-0.15, 0.15]$. A standard deviation is sampled from $[1, 3]$ voxels. Overall $2 \cdot 10^6$ to $2.5 \cdot 10^6$ are added to each synthetic scan. Figure 4.4b shows a 2D slice of a volume filled with these parameters.

As areas with high intensity background noise in the real lupine_small scan also hold more seemingly independent noise voxel-wise noise is not applied uniformly to the whole volume. Instead the already generated noise consisting of perlin and gaussian blobs is used as guide. Therefore the maximum int_{max} and minimum int_{min} noise intensity values in the current noise mask are determined. For each voxel a value $n \in [-0.3, 0.3]$ is uniformly drawn. This value is then scaled by $\sqrt{(int(x, y, z) - int_{min})/(int_{max} - int_{min})}$. In doing so the noise added to areas with low background intensity have low intensity while areas with high background intensity create high frequency noise.

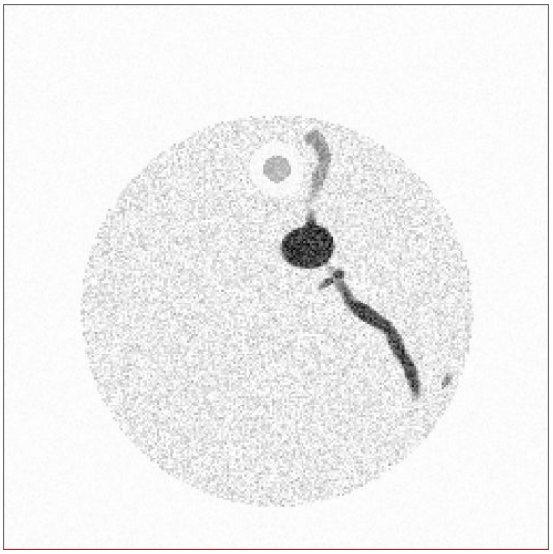
4.3 Employed Datasets

Three more types of noise masks are generated to further increase variability in the training set. The noise types used for training and testing are d and g Also structures simulating the plant pot and testing tube (Figure 4.3b) are added. This is beyond the scope of this thesis. For each root type, ground truth and noise type a total of five training sets with different signal-to-noise ratio is produced. Creating a total of 5760 synthetic plant root MRI scans. These scans are saved with 1 byte per voxel.

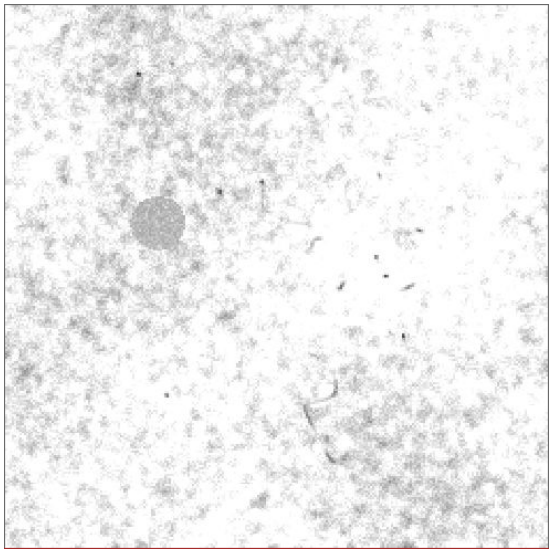
Of the four noise types two are used for training. This is due the forth noise type combining the approaches of two others. Figure 4.5 shows the different kinds of synthetic noise generated and added to the lupine_small and lupine_22 reconstructions. This leaves a dataset of 2880 synthetic scans for training and evaluation purposes. 480 scans are randomly selected as validation set. These datapoints are not used for training. The remaining 2400 scans are available as training data.



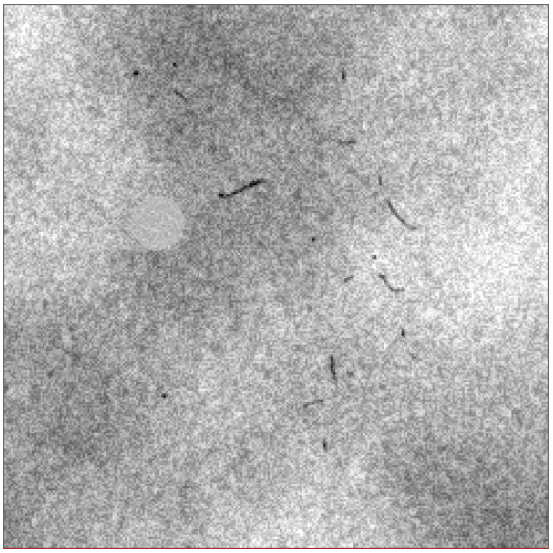
(a) Noise type d, which is based on the noise found in lupine_small, applied to the lupine_small reconstruction



(b) Noise type g, based on the lupine_22 noise, applied to the lupine_small reconstruction



(c) Noise type h, based on the smaller noise blobs in lupine, applied to the lupine_22 reconstruction



(d) Noise type l, which has been discussed in this thesis, applied to the lupine_22 reconstruction

Figure 4.5: The 4 different noise types d, g, h, i generated as training data. As it can be seen no pot is generated for the lupine_22 reconstruction. This is due to the reconstruction not allowing a cylindric pot to be added without loosing root information.

5 Networks and Training

5.1 Network Architectures

In this thesis fully convolutional neural networks are applied to the problem of image or scan segmentation and super resolution of plant root MRI. Each convolutional layer consists of a 3D-convolution stage followed by an activation function.

The 3D-convolution is described by kernel size, number of input channels and number of output channels, compare also Section 3.1.2. The number of input channels is dependent on the number of output channels of the previous layer and therefore this parameter is set. Optionally batch-normalization can be applied to the input of each channel and skip-connections carrying information from layer earlier in the network can be added.

Therefore each convolutional layer is defined by:

- kernel size $size_K \in \{3, 5, 7\}$
- number of output channels $ch_{out} \in \{4, 8, 16\}$
- activation function $\in \{ReLU, Sigmoid, TanH\}$
- apply batch normalization

The number of input channels ch_{in} is defined by the previous layer and the number of input channel for the first layer is one. A 3D-convolutional layer therefore has $num_{weights} = {size_K}^3 \cdot ch_{in} \cdot ch_{out}$ and $num_{biases} = ch_{out}$. If batch normalization should be applied $2 \cdot ch_{in}$ parameters are added.

Unless otherwise stated the activation function applied to all layer besides the output is ReLU. The last layer always has the sigmoid activation function. All layers but the input layer apply batch normalization on there input.

5.1.1 Image Segmentation Networks

As described in Section 2.1 an image segmentation network seeks to solve the problem of 3D image segmentation for the 3D scan I. Therefore a mapping $f_{seg}(I)$ has to be found which maps each $v_I(x, y, z)$ onto the likelihood of given voxel

carrying root signal. The assumption is made that local voxel wise context is sufficient to solve this problem on a per voxel basis.

As discussed in Section 3.1 this assumption is fulfilled by convolutional layers, by mapping the weighted sum of local context onto a value. In the output layer this value is then scaled between 0 and 1 using the sigmoid function and can therefore be interpreted as a likelihood.

To keep the networks simple only 2 and 3-layer fully convolutional neural networks are evaluated for image segmentation. See also Figure 5.1 (a) and (b).

5.1.2 Super Resolution Networks

The problem of super resolution described in Section 2.2 is solved by creating a higher resolution representation I_{sr} of an input scan I. As the ground truth generated is an upsampled, clean reconstruction without noise, the super resolution networks seeks to combine an image segmentation stage with a super resolution stage.

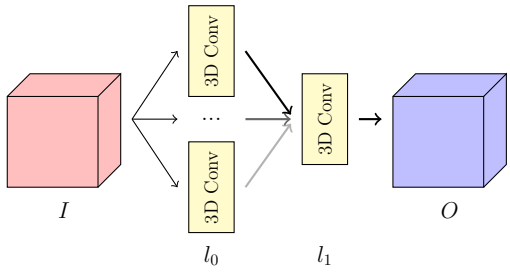
An upsampling factor of 2 along each axis increases the amount of voxel and memory requirements by a factor of 8. Due to the already comparatively large input data, GPU memory is a major bottleneck for super resolution networks. Therefore the image segmentation stage keeps the network in the original resolution. The segmentation stage consists of 3 convolutional layers.

The results generated by the segmentation stage is then upsampled with a factor of 2 along each axis using nearest neighbor interpolation. To recapture missing details representable in the new resolution, the upsampled scans are sent through 1 or 2 further convolutional layers. The output layer applies the sigmoid function allowing the output to be interpreted as probability. See Figure 5.1 (c) and (d).

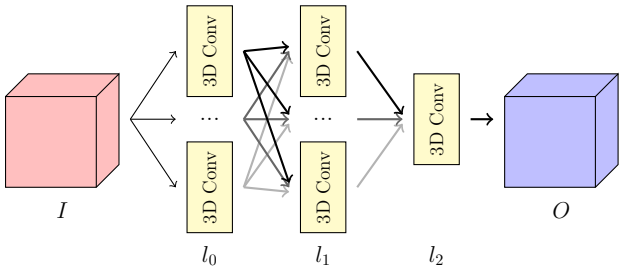
5.2 Training Environment

ADAM optimization (Kingma and Ba, 2014) is a training method for neural networks based on stochastic gradient descent. Ruder, (2016) suggests that ADAM might be the best optimizer to use. The parameters used are $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

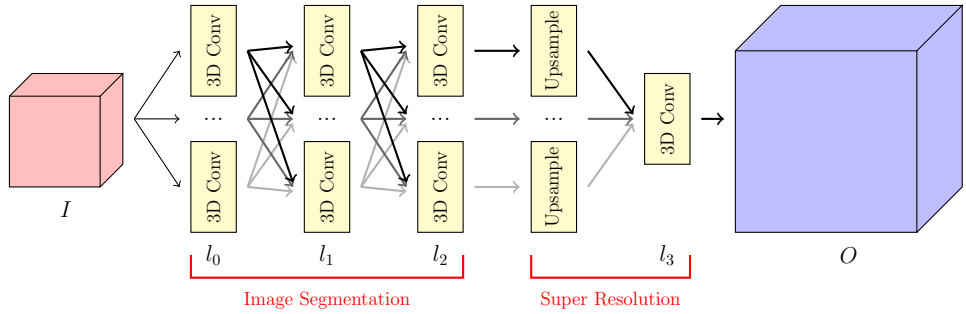
Of the 2400 scans for training a set amount is randomly drawn. Per epoch this training-subset is shuffled and then fed into the network as input, ground truth pair. After each input the loss is scaled by $\frac{100}{size_{subset}}$. As the minimum $size_{subset}$ used is 240 this scale is always smaller than 1. Therefore the scale of the loss and corresponding training adjustments is independent of the drawn subset. Therefore increasing the size of the training-subset can be used to increase the accuracy while



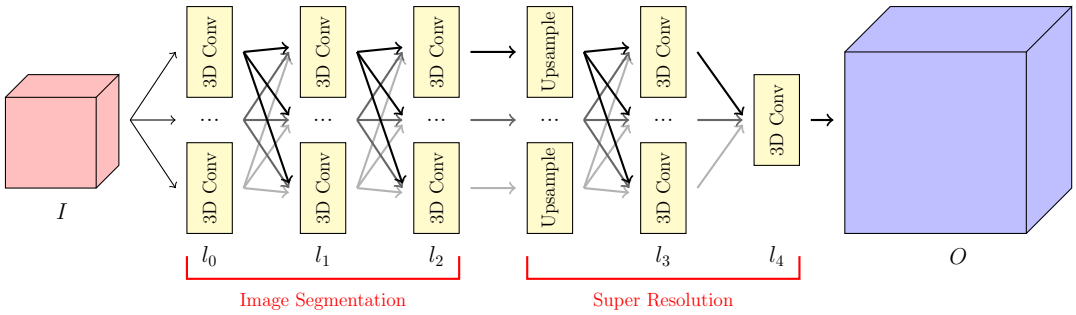
(a) The smallest image segmentation network tested is a 2 convolutional layer network with one multichannel convolutional layer l_0 and one single channel output layer l_1.



(b) The 3-layer image segmentation network has two multichannel convolutional layers l_0 and l_1 and one single channel output layer l_2.



(c) The smallest super resolution network tested is a 4 convolutional layer network. It consists of a 3 layer image segmentation stage with the 3 multi channel convolutional layers l_0, l_1 and l_2 followed by a multichannel upsampling layer and a single channel convolutional output layer l_3.



(d) The 5 layer super resolution network consists of the same image segmentation stage as the 4 layer variant. The super resolution stage consists of the upsampling layer followed by a multichannel convolutional layer l_3 and a single channel convolutional layer l_4 generating the output O.

Figure 5.1

increasing training time. This subset size is dependent on the time a networks needs for training. Subset sizes used are 310, 620, 1240 and 240.

5.3 Loss

As the problem is a binary classification problem the loss function used is binary cross-entropy loss:

$$-(y \cdot \log(p)) + (1 - y) \log(1 - p) \tag{5.1}$$

with y being the ground truth label for either root (1) or soil (0) and p being the prediction.

As discussed in Section 2.3 the amount of soil voxels heavily outnumbers the number of root voxels. More exactly: In lupine_22 0.648% of all voxels are root and in lupine_small 0.445% of voxels are root. Therefore a network can achieve a low per voxel error rate by classifying every voxel as soil. Also the network prioritizes a small mean soil error to a small mean root error.

To address this issue two kinds of loss reweighting during training are applied:

1. Multiplying a set weight $w_{root} > 1$ to the root loss troughout the whole training.

2. Changing weighting over training from small weighting for soil voxels to no weighting being applied.

For set weigthing Equation (5.1) is modified during training with the set weighting w_{root} to:

$$w_{root} \cdot -(y \cdot \log(p)) + (1 - y) \log(1 - p) \tag{5.2}$$

w_{root} is a hyper parameter and does not change during training.

The changing weighting approach uses two gates g_{str} and g_{end}. These are defined over training epochs. While the current training epoch $e < g_{str}$ holds, the loss is computed as:

$$-(y \cdot \log(p)) + w_{soil}(1 - y) \log(1 - p) \tag{5.3}$$

$$w_{soil} = \frac{num_{root}}{num_{soil}} \tag{5.4}$$

with num_{root} and num_{soil} being the number of roots and soil voxels in the ground truth.

While $g_{str} < eg_{end}$ the w_{soil} is scaled by $\frac{e - g_{str}}{g_{end} - g_{str}}$. When $e = g_{str}$ the soil weighting therefore is 1. For further epochs no more weighting is applied.

5.4 Data Splitting

Due to the large size of the input data the memory requirements, especially for super resolution networks, exceeds the available modern hardware. Also the used framework is unstable for training on 3D data with size > 250 along one or more axes.

To address these issues the input data has to be split into a number of subvolumes with a size that can be handled by hard- and software. As mentioned in Section 3.1.1 areas around the edges of the volume cannot be evaluated. This leads to problems if the input data is splitted naively as shown in Figure 5.2a. The reconstructed output is missing the information between the subvolumes.

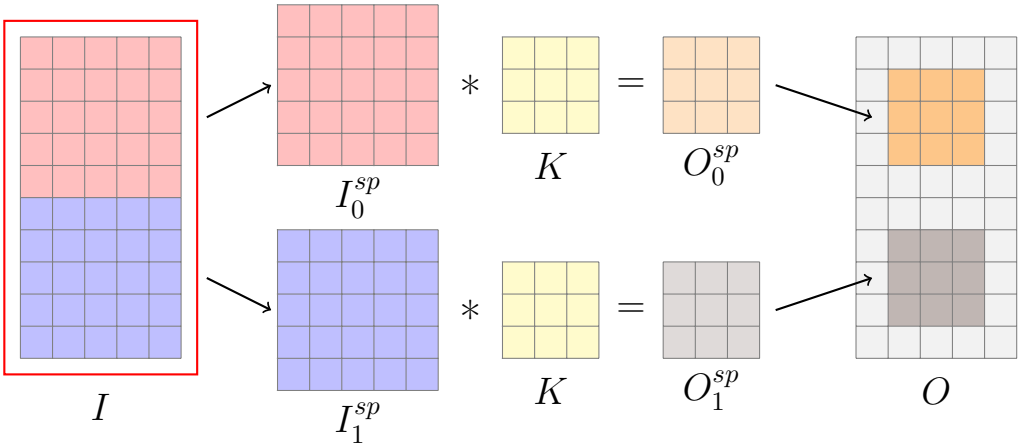
While the information lost on the scan edges cannot simply be retrieved the information between the subvolumes are part of the scan. Therefor to evaluate the input scan completely the input should be split depending on the output. As can be seen in Figure 5.2b the input is split into overlapping subvolumes. This allows to reconstruct a complete output scan from the created subvolume outputs.

The amount of computations needed per layer run are dependent on the output size. As the output of the split data has the same size as the output generated by unsplit data the number of computations does not change. However loading data into the graphics card takes time depending on the number of volumes. Therefore splitting the data increases the time needed per network run dependent on the number of subvolumes. Also the split operations as such increases computation time.

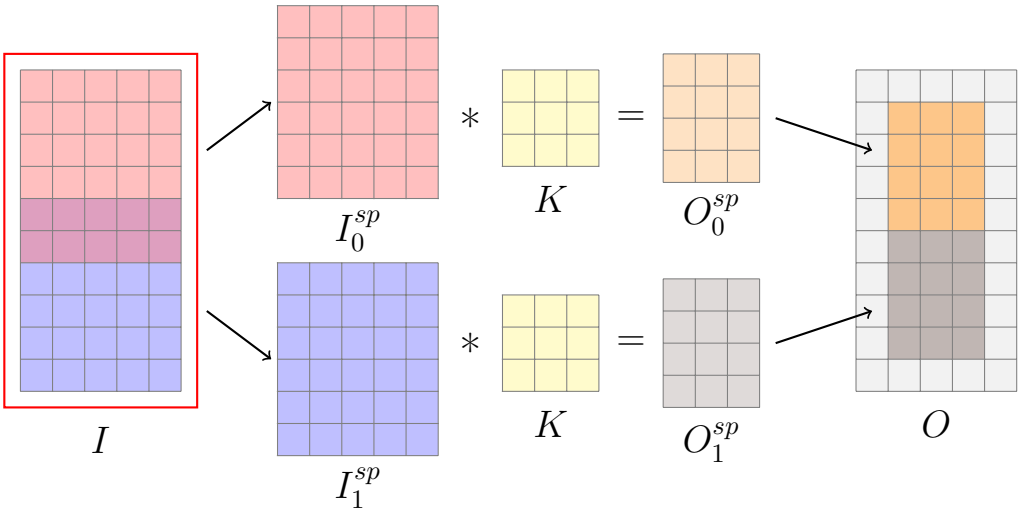
To minimize this overhead the amount of splits should be kept as small as possible. This is done by maximizing the volume per split. As the software frameworks seems to be the limiting factor for the size of single dimensions the ideal subvolume size is 250x250x250.

5.5 Implementation

The framework is implemented in the popular neural network framework PyTorch version 0.4 using Python 3. The GPUs employed are Nvidia Titan X with 12 GB of memory.



(a) An input I with size 5x10x1 is split into two 5x5x1 subvolumes I_0^{sp} and I_1^{sp}. These subvolumes are convolved with kernel K with $size_K = 3$. This results in two 3x3 outputs O_0^{sp} and O_1^{sp} which when set back together leave a 2x3 gap between the subvolumes. The grayed areas in O are the missing voxels.



(b) The split edges in the output are restored by computing the area along the edge twice. Once in I_0^{sp} and once in I_1^{sp}. The outer border cannot be computed without padding I.

Figure 5.2

6 Evaluation

6.1 Evaluation Framework

There are four metrics of interest to compare networks with different parameters:

1. Training Loss
2. Validation Loss
3. F1-Score
4. Training Time

6.1.1 Training Loss

The training loss is the mean cross-entropy loss between ground truth and network output. As discussed in Section 2.3 the number of soil voxels has a great influence on this value while the root signal is of more interest generally. Therefor not only the mean overall loss is computed but also the mean loss for root voxels and the mean value for soil voxels. Both of these values are computed before any weighting is applied.

As the training is done without any input padding the amount of voxels in the output depends on the amount of convolutions and their respective kernel size. The overall loss varies depending on any weighting applied. All this makes the train loss unsuitable to use as metric for comparing results, but it is useful to look into the behavior during training itself.

6.1.2 Validation Loss

As discussed in Section 4.3 the validation set contains 480 scans. Just like with the training loss the validation loss is the mean cross entropy loss between network output and corresponding ground truth. Also mean root loss and mean soil loss are computed.

To make the values comparable the validation is done with input padding. Therefore the amount of voxels that are compared per scan are independent of any loss of scan size during convolution.

Due to the amount of scans in the validation set, the validation is not done at every training epoch.

6.1.3 F1-Score

It is of interest how much of the root is recovered as well as how accurate these predictions are. The F1-Score is a metric describing this. The F1-Score is defined as:

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}}$$

Let E be a set of examples. Let $E_+ \subset E$ be the subset of examples of the class of interest and let $C_+ \subset E$ be the set of all examples that have been classified as the class of interest. Recall is defined as the ratio of correct as member of a given class identified examples, of the overall amount of correct examples, therefore giving the fraction of all correct examples identified. Recall can therefore be defined as:

$$recall = \frac{|E_+ \cap C_+|}{|E_+|}$$

The precision is defined as the ratio of correct as member of given class identified examples, of the amount of all examples that have been classified as given class. Therefore precision can be defined as:

$$precision = \frac{|E_+ \cap C_+|}{|C_+|}$$

Let O be a 3D scan. A voxel $v_O(x, y, z)$ is classified as root if its value is larger than 0.5. Therefore C_+ is defined as $C_+ = \{v_O(x, y, z) | x, y, z \in O, int_O(x, y, z) \geq 0.5\}$. Using this threshold the scan is binariest and then computed for recall and precision.

6.2 Image Segmentation

6.2.1 Training

Figure 6.1 shows the training loss value during training for multiple networks. It can be observed that the larger networks have sudden local increases in training

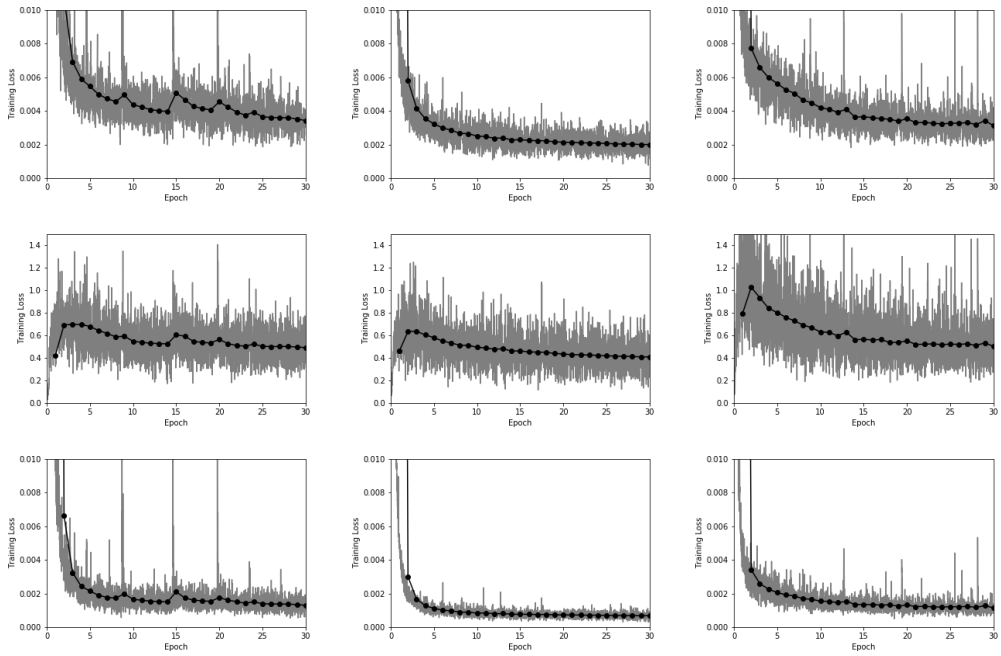


Figure 6.1: Training loss of three 3-layer networks with different kernel sizes $size_K$ or ch_{out}. The left column shows training loss, root loss and soil loss during training for a network with $size_K$ =5 5 3 and ch_{out} =4 4 1. The middle column shows the same for a network with $size_K$ =3 3 3 and ch_{out} =4 4 1. The right column shows the three losses for a network with $size_K$ =3 3 3 and ch_{out} =4 8 1. The black plot is the mean loss per epoch, while the gray plot is the mean loss computed every 1% of an epoch. The learning rate used for all is 0.0003

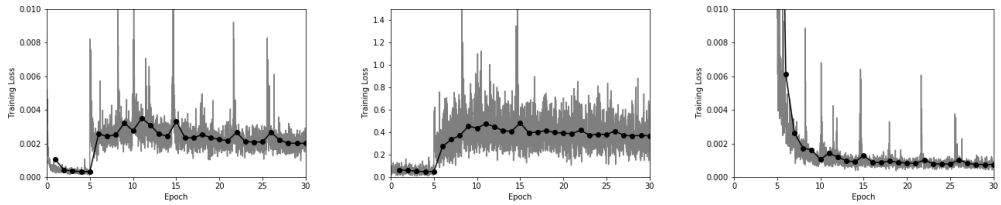


Figure 6.2: 3-layer image segmentation network with $size_K$ =5 5 5 and ch_{out} = 8 8 1 trained with dynamic loss weighting. From epoch 0-5 the full weighting is applied. From epoch 5-10 w_{soil} increases linearly to reach 1 at epoch 10+

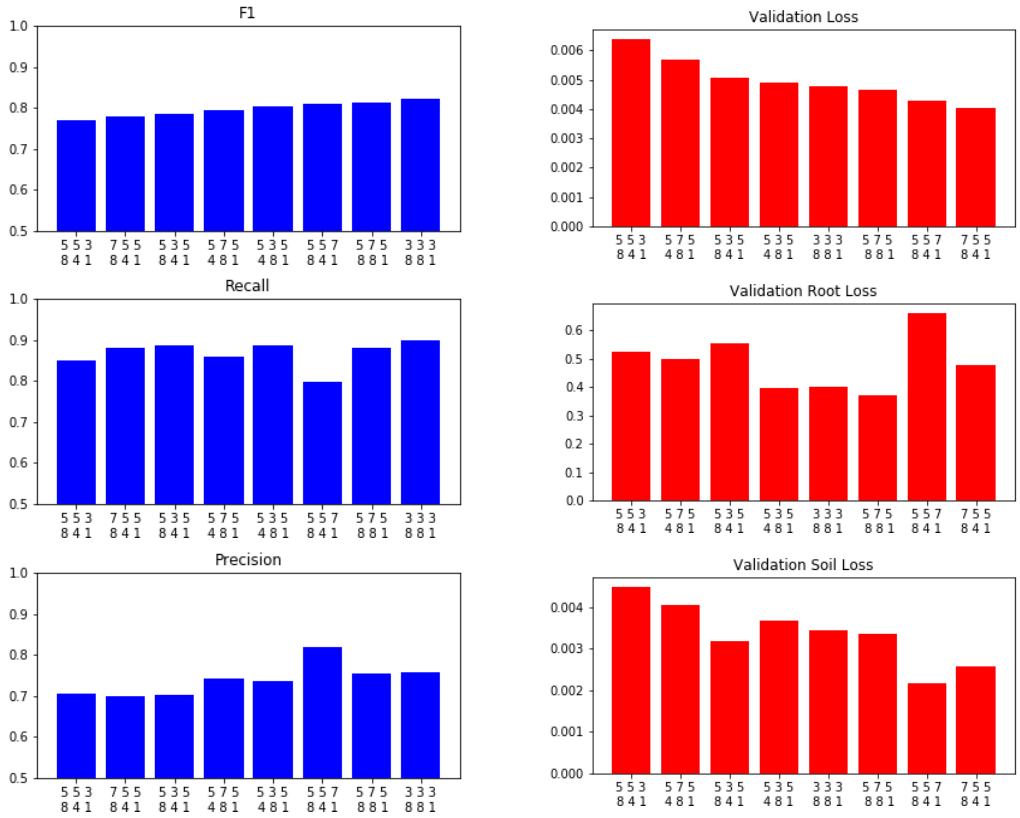
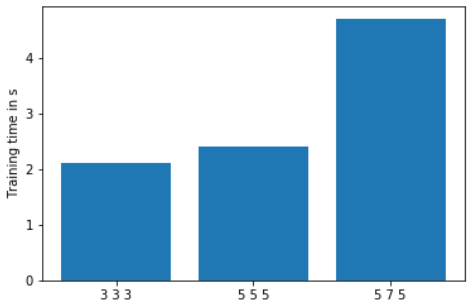


Figure 6.3: Best F1-score with corresponding recall and precision as well as the best validation loss for 3-layer networks with different kernel sizes and number of channels. Shown is the best F1-score achieved per network and the corresponding recall and precision values. The upper row of each label shows the kernel sizes and the lower row the number of channel per layer.

loss. This is probably the cause of the learning rate being too high. This may also impact the network performance considering validation loss and F1-score, as addressed in Section 6.2.3 with regards to the performance of the smallest network.

As the soil is the largest part of the loss, it is the first of the two loss parts that gets reduced, while the root loss increases. After the soil loss reaches a low value only small changes happen. Most of the changes in error now happen for the root loss. Therefore most of the training time is needed to reduce the root loss and increase recall.

The networks seem very volatile with respect to learning rate. Therefor further research in this area may be advised.



(a) The bars show the mean runtime per scan during training for layers with 8x8x1 channels. The numbers indicate the kernel size per layer.

Networks			Training time
l_0	l_1	l_2	
8	8	1	4.7s
4	8	1	3.2s
16	4	1	7.9s
8	4	1	4.7s
8	2	1	4.5s
4	2	1	2.6s

(b) This table shows the runtime for training of 3-layer networks with kernel sizes 5,7,5. The number indicate the channels per layer

Figure 6.4: Training runtime for multiple 3 layer networks depending on kernel size or channels. Training runtime is measured using the mean runtime per scan when running the training framework.

6.2.2 Kernel Size

Changing the kernel size $size_K$ for the 3D convolutions seems to have little effect on the validation error or the F1-score on its own. As can be seen in Figure 6.3. This might be due to the smallest kernel K with $size_K = 3$, already having $3^3 = 27$ parameters, while larger kernels have a lot more parameters ($size_K = 5$ has 125 parameters, $size_K = 7$ has 343) increasing the search space.

A concern with using large kernels is runtime. As can be seen in Figure 6.4a the runtime of 3-layer networks using kernels with $size_K = 3$ and $size_K = 5$ is very close, while the runtime with a single kernel with $size_K = 7$ increases massively during training. As the inference times are comparatively close the difference has to be in the backpropagation through the large kernel. Figure 6.4b shows that this effect is heavily dependent on the number of input channels into the layer and to a lesser extend the number of output channels. As discussed in Section 6.2.3 networks with larger amounts of channels usually perform better than narrower networks. As this effect is larger than the advantages of large kernels, $size_K > 5$ is not advised.

A concern about only using small convolutions is that the receptive field of the network becomes to small and cannot access nearby information vital for the task. To test this, 2-layer networks with growing receptive field have been tested to see if there is a minimum receptive field below which performance reduces sharply. As can be seen in Figure 6.5 the smallest network with a receptive field

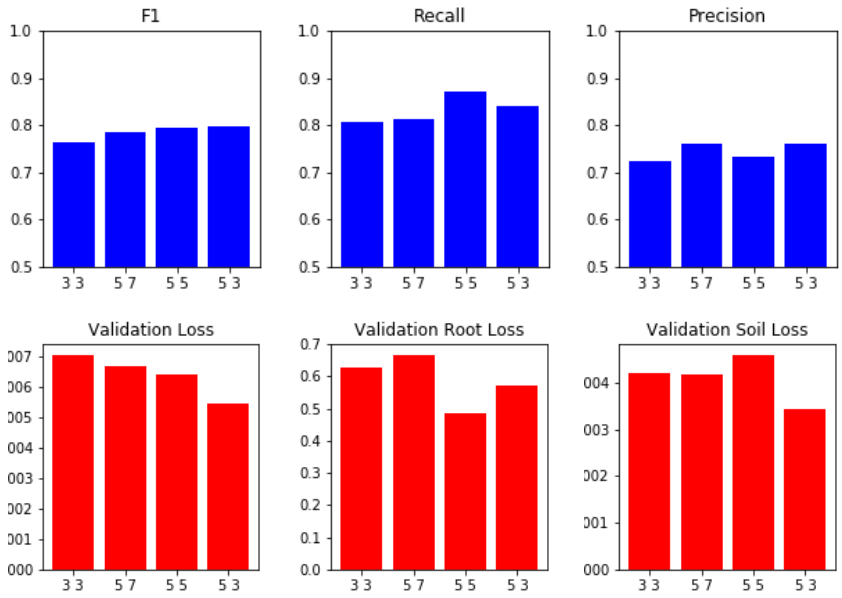


Figure 6.5: F1-score and validation loss for 4 2-layer image segmentation networks. The numbers correspond to the kernel sizes. There seems to be no order based on receptive field size besides the 3 3 network being worse than its counterparts.

of 5x5x5 achieves worse results than networks with larger receptive field. While not conclusive it may be advised to construct networks with a receptive field of at least 7x7x7.

Larger kernels need lower learning rates to converge properly as they have more parameters. This effect is noticable between networks with $size_K = 3$ towards $size_K = 5$. Figure 6.1 shows two 3-layer networks trained with the same learning rate. The network utilizing kernels with $size_K = 5$ has sudden drastic rises during training while the network with $size_K = 3$ converges better and without sudden increases in loss value. See also Section 6.2.1.

Given these reasons smaller kernels of $size_K = 3$ or $size_K = 5$ are preferable for smaller networks while $size_K = 3$ is the best choice for deeper networks. $size_K > 5$ is not advised, as the possible increase in performance is offset by long runtime per scan and the necessity for lower a learning rate both increasing the training time.

Networks	Mean memory cached	Runtime per training scan
4 4 1	2850MiB	2.1s
8 8 1	4240MiB	2.4s
16 16 1	4900MiB	4.2s

Table 6.1: 3-layer networks with different amounts of channels and there respective cached memory. The memory amount is read using PyTorch's cuda module while training. The values are not completely accurate as reading the amount of memory occupied by a network during training seems to not be exact all the time therefor the mean memory cached is used.

6.2.3 Number of Channels

As mentioned in Section 6.2.2 the number of channels and therefore extracted features seems to have a large impact on F1-score performance and validation error. As can be seen in Figure 6.3 networks with a larger number of channels in later layers outperform networks with more channel in the early layers. This also holds for the data in Figure 6.6. The 4x4x1 network seems to overperform by having a low soil loss and high precision coupled with low recall while also being less impacted by higher learning rates. While ch_{out} seems more impactful on later layers, missing capacity in early layers have an adverse effect as well. It therefore is advised to increase ch_{out} by starting with later channels and then moving towards the input layers.

A problem with increasing the numbers of channels is that the amount of memory needed increases, see Table 6.1. This is due to the network saving intermediate feature maps. While this does not pose problems for image segmentation networks, it can be problematic for super resolution networks, see also Section 6.3.

The main advantages of increasing the amount of channels instead of increasing kernel size is that the runtime gets less impacted, as can be seen by comparing Table 6.1 to Figure 6.4b. Increasing ch_{out} has an impact on training behaviour as can be seen in Figure 6.1, again the network with more channels is prone to sudden local peaks in training loss.

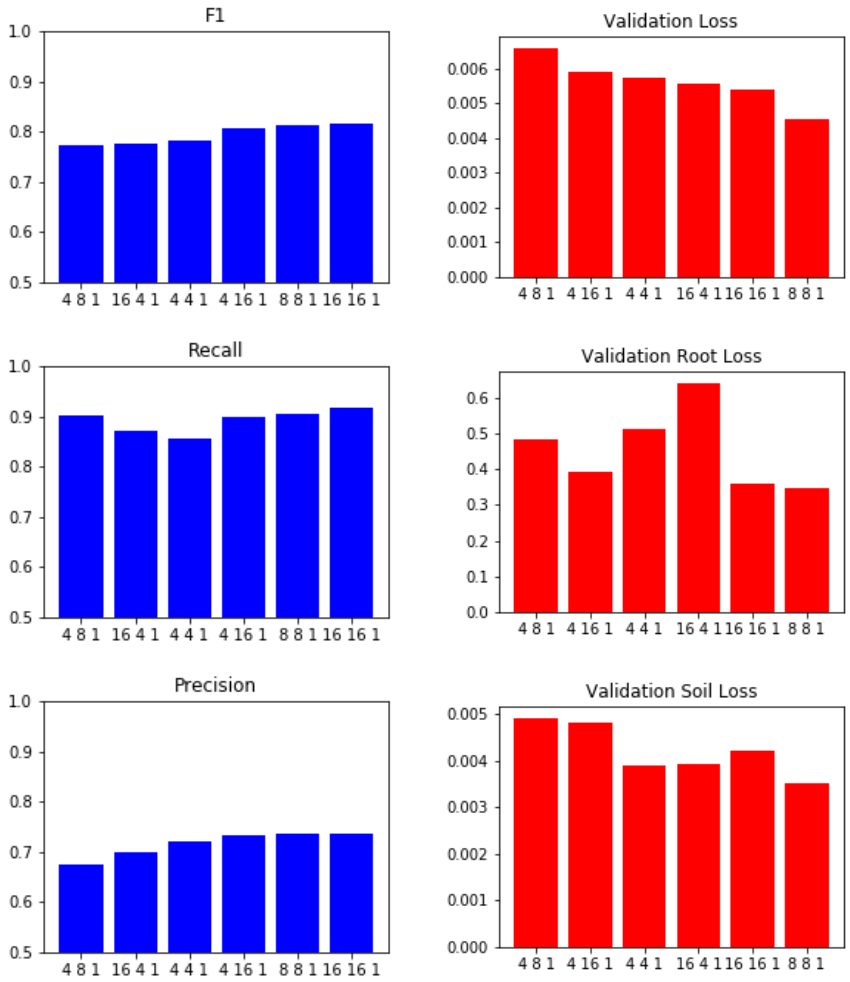


Figure 6.6: Best F1-score with corresponding recall and precision as well as the best validation loss for 3-layer networks with different numbers of channels. The kernel sizes in each network is $size_K = 5$ besides the 16x16x1 network were the kernels have $size_K = 3$. Shown is the best F1-score achieved per network and the corresponding recall and precision values. The numbers show the number of output channels per layer.

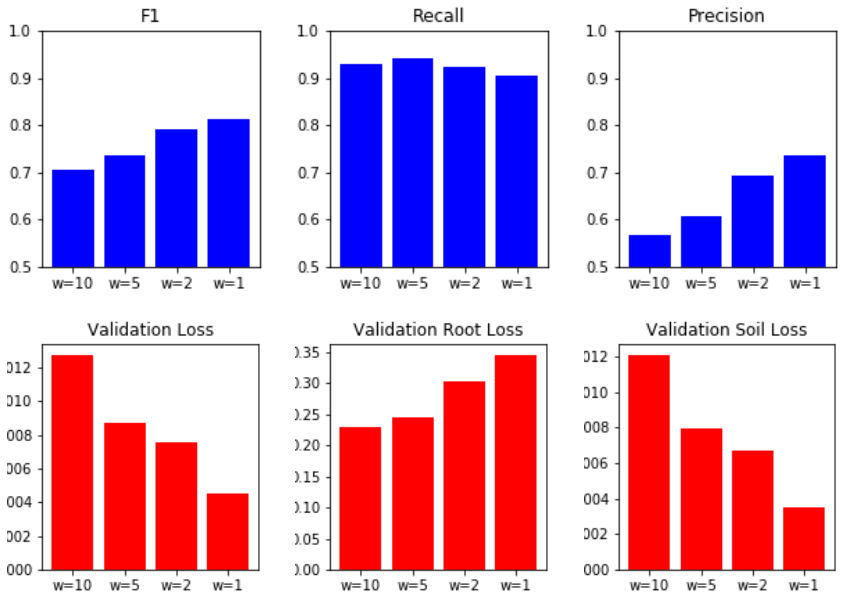


Figure 6.7: F1-score and validation loss for 4 3-layer image segmentation networks with $size_K$ =5 5 5 and ch_{out} =8 8 1. The numbers correspond to the set weight applied to root loss.

6.2.4 Loss Reweighting

The goal of loss reweighting is to increase the amount of recovered root. This should increase recall while reducing precision.

As can be seen in Figure 6.7 increasing the root loss does lead to an increase in recall but a larger decrease in precision, therefore reducing F1-score. Using a weighting of larger $w_{root} = 5$ seems to not further increase recall and seems to have the adverse effect. As the root loss for the network with $w_{root} = 10$ is the lowest, the lower recall might be due to the network trying to minimize on voxels which already have been classified as root as even small errors on root can have high impact using such weighting. This may be addressed by reshaping the loss function. Further testing on this is needed.

For now using a weighing $w_{root} < 5$ is advised. Loss reweighting if applied in this manner is an efficient way of increasing the recall.

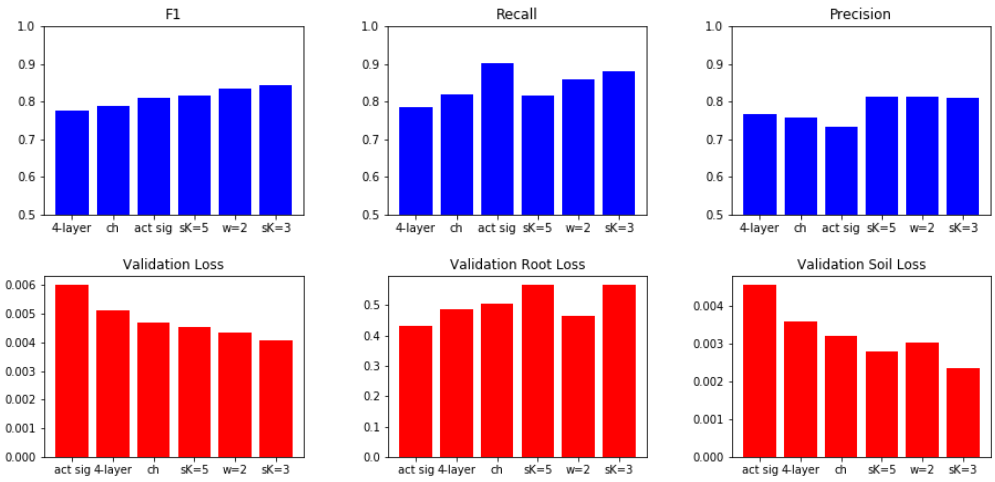


Figure 6.8: F1-score and validation loss for different super resolution networks. 4-layer means the network only has 4-layers. ch means the network has a decreased channel size ch_{out} = 8 8 4 4 1. All remaining networks have ch_{out} =8 8 4 8 1. sK means that the networks have this kernel size for the whole network. w=2 is a network trained with a root loss reweighing of 2 and $size_K$ =5 5 5 5 5. ac sig means that the last layer of the image segmentation stage has the sigmoid activation function.

6.3 Super Resolution

6.3.1 Performance

Due to the large amount of time super resolution networks need to train coupled with hardware and software problems (Section 6.3.2) the amount of networks tested is limited.

Figure 6.8 shows the performance of some super resolution models. Loss reweighting (w=2) for super resolution seems to increase the F1-score due to a much higher recall while the precision is only marginally affected. Changing the activation function while generating a good recall value has an adverse effect on validation loss. As expected the networks with the lowest capacity have the lowest F1-score but not the lowest validation loss.

The best performing network is again the one with the smallest kernel size and larger channel number. This may be because the network has an easier time converging when trained with higher learning rate. To get the other models, primarily the one with larger kernel size, to the same or better performance more training with more carefully selected learning rate is needed.

Number of splits	Runtime per scan
1	2.2s
2	2.7s
4	3s
8	3.2s

Table 6.2: Effect of data splitting on trainng runtime per scan. The network used is a 3-layer network with $size_K$ =5 5 5 and ch_{out} =8 8 1

6.3.2 Hardware and Software Limitations

One major problem regarding the super resolution problem and to a smaller extend image segmentation, are problems originating from the used hard- and software. As briefly alluded to in Section 5.4 the input scans are very large. Upsampling these large input scans using super resolution, the resulting feature maps and outputs take a large amount of memory. This heavily limits the amount of channels per layer and the number of layers as such unless the data is split into a lot of subvolumes increasing training time, see Table 6.2.

The upsampling layer also takes a large amount of memory. This leads to networks running out of memory if the number of channels upsampled is to large. Therefore ch_{out} of the last layer of the image segmentation stage is limited to 4.

There are some problems arising from the used framework, mainly due to PyTorch not being optimized for large 3D data:

1. Memory management is hard, as the only information provided by PyTorch is the amount of cached data or the amount of allocated data after execution and memory clean-up.

2. Some object hold large amounts of data. E.g. the computed loss value holds a large amount of data about gradients and feature maps.

3. The learning algorithm cannot handle 3D data with a single dimension larger around 250.

These problems have been dealt with using data splitting and manual memory management, ensuring that all objects on the GPU that are no longer needed are deleted and the memory freed.

6.4 Outputs

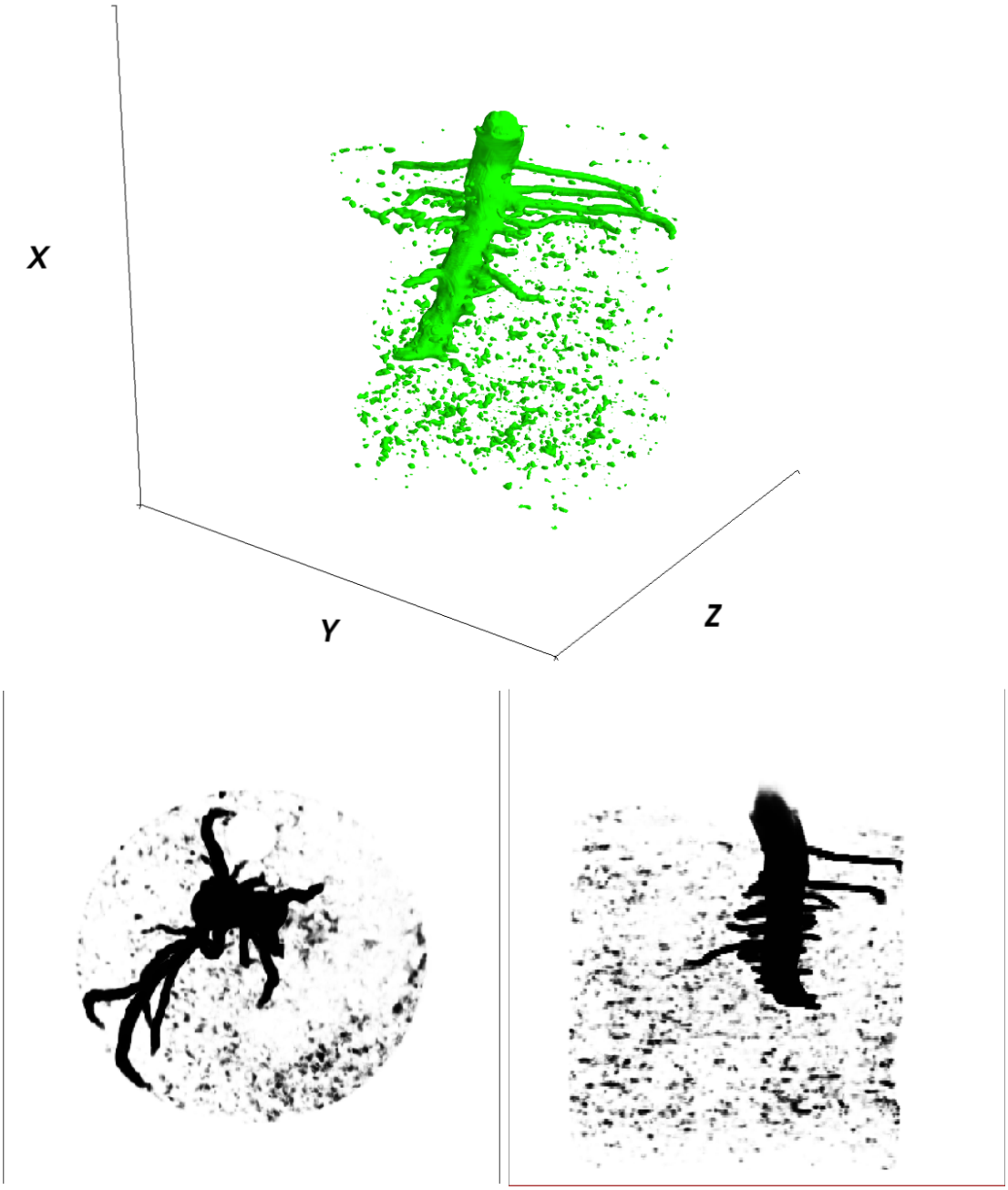


Figure 6.9: Output of the best performing super resolution network sK=3. Input is the real lupine_small scan. The upper picture shows a 3D model of the output thresholded at 0.5. Lower left is a 2D slice generated by taking the maximum values along the X axis for 51 slices. The lower right is the maximum accumulated along the Y Axis for 79 slices.

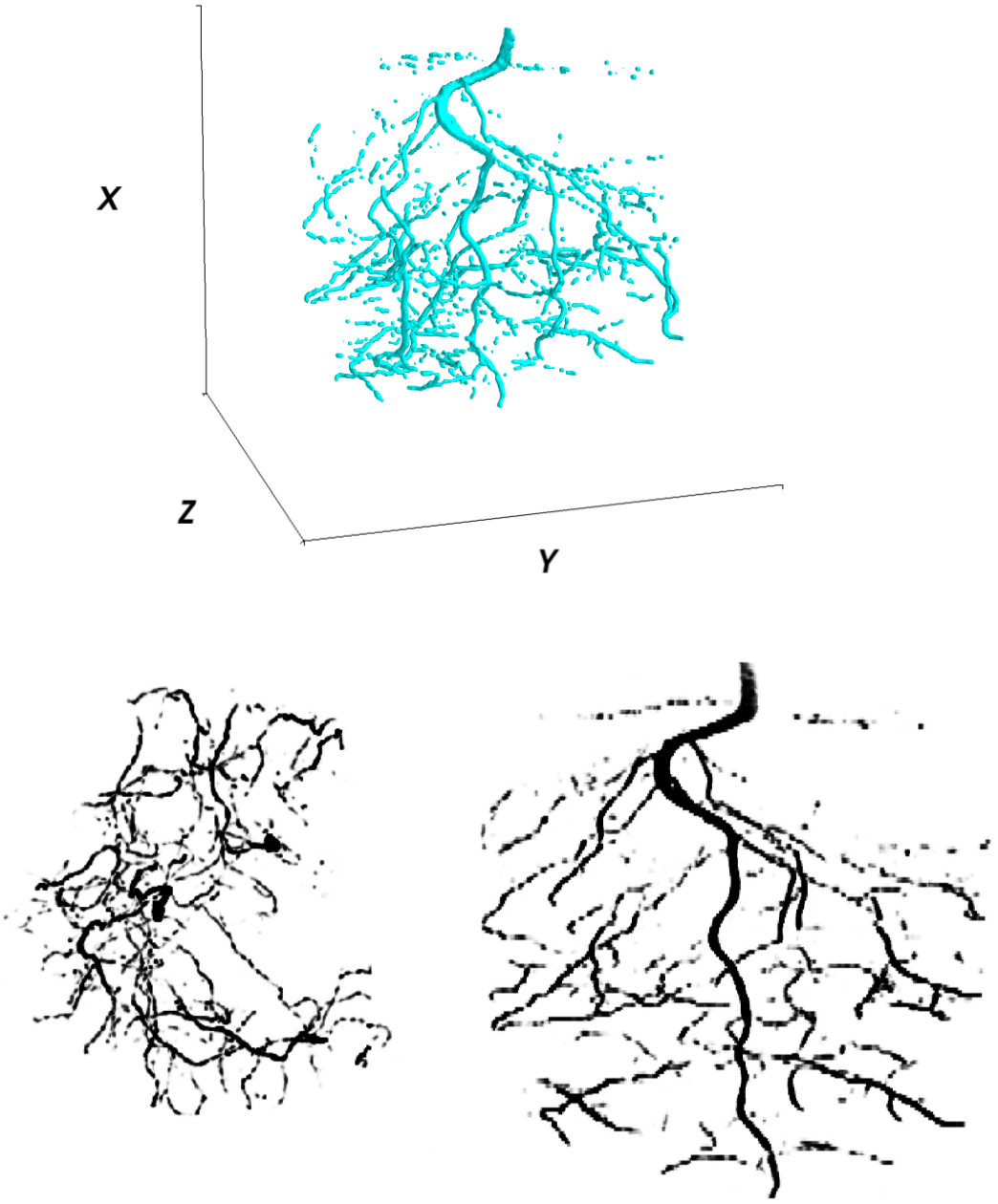


Figure 6.10: Output of the best performing super resolution network sK=3. Input is the real lupine_22 scan. The upper picture shows a 3D model of the output thresholded at 0.5. Lower left is a 2D slice generated by taking the maximum values along the X axis for 119 slices. The lower right is the maximum accumulated along the Y Axis for 119 slices.

7 Conclusion

In this thesis multiple different network architectures to solve the problem of image segmentation and the joint problem of image segmentation and super resoltion for 3D plant root MRI have been presented. This is in the form of shallow convolutional networks using 3D convolutions.

These networks are analyzed and evaluated by looking at the effect of different parameters on the F1-score and validation loss. Specifically the effect of kernel size, number of channels and loss reweighting have been looked into and guidelines for these have been extracted.

Also a way of synthesizing noise based on lupine_small has been shown. While this method is no longer in use, the sections shows the way of how the underlying data was analyzed to create synthetic data.

Further work can be done with respect to optimizing the learning rate more, as some larger networks may have underperformed due to the learning rate being to high. Also some more information on the super resolution problem is of interest. While most hard- and software problems have been resolved a more accurate learning rate might yield better results here. Also different ways for upsampling can be looked into. Further methods of output post processing can be explored e.g. connecting paths. Overall the usage of more complicated and deeper networks is of interest to fully solve the problem of 3D image segmentation and super resolution for plant root MRI.

Bibliography

Colliot, O., O. Camara, and I. Bloch (2006). "Integration of fuzzy spatial relations in deformable models—Application to brain MRI segmentation". In: *Pattern recognition* 39.8, pp. 1401–1414 (cit. on p. 6).

Cui, Z., H. Chang, S. Shan, B. Zhong, and X. Chen (2014). "Deep network cascade for image super-resolution". In: *European Conference on Computer Vision*. Springer, pp. 49–64 (cit. on p. 8).

Despotović, I., B. Goossens, and W. Philips (2015). "MRI segmentation of the human brain: challenges, methods, and applications". In: *Computational and mathematical methods in medicine* 2015 (cit. on p. 6).

Dong, C., C. C. Loy, K. He, and X. Tang (2016). "Image super-resolution using deep convolutional networks". In: *IEEE transactions on pattern analysis and machine intelligence* 38.2, pp. 295–307 (cit. on p. 8).

Girshick, R., J. Donahue, T. Darrell, and J. Malik (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587 (cit. on p. 6).

Goodfellow, I., Y. Bengio, A. Courville, and Y. Bengio (2016). *Deep learning*. Vol. 1. MIT press Cambridge (cit. on pp. 11, 12).

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative adversarial nets". In: *Advances in neural information processing systems*, pp. 2672–2680 (cit. on p. 8).

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778 (cit. on p. 6).

Heckemann, R. A., J. V. Hajnal, P. Aljabar, D. Rueckert, and A. Hammers (2006). "Automatic anatomical brain MRI segmentation combining label propagation and decision fusion". In: *NeuroImage* 33.1, pp. 115–126 (cit. on p. 6).

Ioffe, S. and C. Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (cit. on p. 15).

Kamnitsas, K., C. Ledig, V. F. Newcombe, J. P. Simpson, A. D. Kane, D. K. Menon, D. Rueckert, and B. Glocker (2017). "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation". In: *Medical image analysis* 36, pp. 61–78 (cit. on p. 6).

Kim, J., J. Kwon Lee, and K. Mu Lee (2016a). "Accurate image super-resolution using very deep convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1646–1654 (cit. on p. 8).

— (2016b). "Deeply-recursive convolutional network for image super-resolution". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1637–1645 (cit. on p. 8).

Kingma, D. P. and J. Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (cit. on p. 26).

Krähenbühl, P. and V. Koltun (2011). "Efficient inference in fully connected crfs with gaussian edge potentials". In: *Advances in neural information processing systems*, pp. 109–117 (cit. on p. 6).

Lai, W.-S., J.-B. Huang, N. Ahuja, and M.-H. Yang (2017). "Deep laplacian pyramid networks for fast and accurate superresolution". In: *IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 2. 3, p. 5 (cit. on p. 8).

Ledig, C., L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. (2017). "Photo-realistic single image super-resolution using a generative adversarial network". In: *arXiv preprint* (cit. on p. 8).

Lin, G., A. Milan, C. Shen, and I. Reid (2017). "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation". In: *IEEE conference on computer vision and pattern recognition (CVPR)*. Vol. 1. 2, p. 3 (cit. on p. 6).

Lin, G., C. Shen, A. Van Den Hengel, and I. Reid (2016). "Efficient piecewise training of deep structured models for semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3194–3203 (cit. on p. 6).

Lin, T.-Y., P. Goyal, R. Girshick, K. He, and P. Dollár (2018). "Focal loss for dense object detection". In: *IEEE transactions on pattern analysis and machine intelligence* (cit. on p. 9).

Long, J., E. Shelhamer, and T. Darrell (2015). "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440 (cit. on p. 6).

Makropoulos, A., I. S. Gousias, C. Ledig, P. Aljabar, A. Serag, J. V. Hajnal, A. D. Edwards, S. J. Counsell, and D. Rueckert (2014). "Automatic whole brain MRI segmentation of the developing neonatal brain". In: *IEEE transactions on medical imaging* 33.9, pp. 1818–1831 (cit. on p. 6).

Milletari, F., N. Navab, and S.-A. Ahmadi (2016). "V-net: Fully convolutional neural networks for volumetric medical image segmentation". In: *3D Vision (3DV), 2016 Fourth International Conference on*. IEEE, pp. 565–571 (cit. on p. 6).

Perlin, K. (2004). "Implementing improved perlin noise". In: *GPU Gems*, pp. 73–85 (cit. on p. 19).

Pham, C.-H., A. Ducournau, R. Fablet, and F. Rousseau (2017). "Brain MRI super-resolution using deep 3D convolutional networks". In: *Biomedical Imag-*

ing (ISBI 2017), 2017 IEEE 14th International Symposium on. IEEE, pp. 197–200 (cit. on p. 8).

Pham, C.-H., R. Fablet, and F. Rousseau (2017). "Multi-scale brain MRI super-resolution using deep 3D convolutional networks". In: (cit. on p. 8).

Ruder, S. (2016). "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747. arXiv: `1609.04747`. URL: `http://arxiv.org/abs/1609.04747` (cit. on p. 26).

Schulz, H., J. A. Postma, D. van Dusschoten, H. Scharr, and S. Behnke (2013). "Plant root system analysis from MRI images". In: *Computer Vision, Imaging and Computer Graphics. Theory and Application*. Springer, pp. 411–425 (cit. on p. 1).

Simonyan, K. and A. Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (cit. on p. 6).

Zhou, Z.-H. and X.-Y. Liu (2006). "Training cost-sensitive neural networks with methods addressing the class imbalance problem". In: *IEEE Transactions on Knowledge and Data Engineering* 18.1, pp. 63–77 (cit. on p. 9).