University of Innsbruck

Department of Mathematics

Master Thesis

Deep Learning in Data Annotation: Projection-based 2.5D U-net Structure for Biomedical Volumetric Segmentation

von

Christoph Helmut Angermann

Submission Date: July 3, 2019
Supervisor: Univ.-Prof. Dr. Markus Haltmeier
Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit/Dissertation eingereicht.

Datum: ______________________________

Unterschrift: __________________________
The world is one big data problem.

Andrew McAfee
Abstract:

This master thesis has been written in collaboration with the Group of Applied Mathematics Innsbruck and the Department of Neuroradiology Innsbruck. We give an introduction to the problem of image segmentation and discuss the idea of fully convolutional networks [16], which provided one of the first network structures to achieve state-of-the-art 2D segmentation masks. Based on them, we stepwise derive the U-net-architecture [18], which constitutes a fully automated method for very accurate semantic segmentation, especially for biomedical images. Additionally, given 3D MRA scans of 119 different patients, we apply all discussed models to them using 2D maximum intensity projections. Furthermore, we extend the U-net to construct volumetric segmentation masks using 3D convolutions [6, 10, 17].

Volumetric convolutions have huge storage requirements and therefore, end-to-end training is limited by GPU memory and data size. To overcome this issue, we introduce a network structure for volumetric data without 3D convolutional layers. The main idea is to include projections (e.g. maximum intensity projections or Radon-transforms [7]) from different random directions to transform the volumetric data to a sequence of images, where each image contains information of the full data. We then apply 2D convolutions to these projection images and lift them to volumetric data using a trainable reconstruction algorithm. The proposed network architecture can be applied to arbitrary big data volumes without cropping or sliding-window techniques, which is not possible for network structures using 3D convolutions. For the targeted volumetric binary segmentation of MRA scans, it even shows better performance than the 3D U-net and is much more consistent in terms of accuracy of the output segmentations.

Every network in this thesis is constructed with the programming language Python v3.6.8 (https://www.anaconda.com/download/#linux) using the machine learn-
ing library *Keras* [5] v2.2.4 with *TensorFlow*-backend [2] v1.13.1. We assume that the reader owns good knowledge about the construction and training process of neural networks. Especially understanding the building blocks of *convolutional neural networks* (i.e. convolutional layers, pooling layers, fully-connected layers et cetera) and the training process of such networks (i.e. loss functions, optimizers et cetera) is important. Information and background on these issues are given in [1, 3, 14].
Acknowledgements:

I would like to express my special appreciation and thanks to my supervisor Univ.-Prof. Dr. Markus Haltmeier, who aroused my interest into machine learning and always gave good and encouraging advice during my bachelor and master studies.

Also a special thanks to my beloved parents for their financial and mental support during my years of academic education. I am deeply grateful to you.
Contents

List of Figures I
List of Tables IV
Listings V

1 Introduction 1

2 Fully Convolutional Networks 7
  2.1 From VGG-nets to FCNs . . . . . . . . . . . . . . . . . . . . . . . . . 8
  2.2 Network Training . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  2.3 Skip-Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3 The U-net 23
  3.1 The 2D U-net . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23
  3.2 The Dice-loss Function . . . . . . . . . . . . . . . . . . . . . . . . . . 26

4 Volumetric Segmentation 29
  4.1 The 3D U-Net . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
  4.2 Slice-by-Slice 2D U-Net . . . . . . . . . . . . . . . . . . . . . . . . . . 31
  4.3 Projection-Based 2.5D U-Net . . . . . . . . . . . . . . . . . . . . . . 32
    4.3.1 Proposed 2.5D U-net Architecture . . . . . . . . . . . . . . . . 32
    4.3.2 Deterministic Projection Directions . . . . . . . . . . . . . . . 36
    4.3.3 Integration over Regions . . . . . . . . . . . . . . . . . . . . . 39
    4.3.4 Random 2.5D U-net . . . . . . . . . . . . . . . . . . . . . . . . 42
  4.4 Numerical Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45

5 Conclusion 48

Bibliography 50
List of Figures

1.1 **Image segmentation:** Left we have an image of the environment, in the middle we mark the different objects, and on the right we see the resulting segmentation mask with label 0 (black), label 1 (red), label 2 (green) and label 3 (blue). The pictures have been constructed with the freeware PixelAnnotationTool [4].

1.2 **3D MRA scan:** Here the three planes (a)-(c) of a volumetric MRA scan are displayed. The 2D visualizations of the 3D scan were generated with the freeware ITK-SNAP [22].

1.3 **3D segmentation:** In every plane (a)-(c) the blood vessels of interest are marked in red. In (d) we see the resulting segmentation mask. The segmentation was conducted with the freeware ITK-SNAP [22].

1.4 **Maximum intensity projection** of a 3D object. Taking the maximum values along the projection direction (in this case the first spatial axis) produces a 2D projection containing information of the 3D data.

1.5 **MIP images** of a 3D MRA scan with projection directions $\Theta = \{k \times 36^\circ | k = 0, \ldots, 5\}$. In the first row, we see the projections of the original scan, in the second row we see the corresponding projections of the ground truth.

2.1 **The FCN-11-bi net** gets an input scan (left) with 1 channel and outputs for each pixel the probability for being a vessel pixel. Applying a threshold of 0.5 generates the final segmentation mask (right). Total number of trainable parameters: $6.3 \times 10^6$.

2.2 **Performance of the FCN-11-bi net:** Above we see the corresponding ground truth, below the segmentation produced by the network.

2.3 **The FCN-11-bi-skip net** is the modified version of the FCN-11-bi net with skip architecture. The purple arrows show the combination of local information and fine prediction. For this model, again bilinear interpolation is used as upsampling technique.

2.4 **Performance of the FCN-11-bi-skip net:** Above we see the corresponding ground truth, below the segmentation produced by the network.

2.5 **Performance of the FCN-11-dc-skip net:** Above we see the corresponding ground truth, below the segmentation produced by the network.
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>The 2D U-net: Visualization of the ground structure of an U-net for semantic segmentation of 2D images</td>
<td>24</td>
</tr>
<tr>
<td>3.2</td>
<td>Performance of the 2D U-net: Above we see the corresponding ground truth, below the segmentation produced by the network</td>
<td>26</td>
</tr>
<tr>
<td>3.3</td>
<td>2D U-net with Dice-loss: Above we see the corresponding ground truth, below the segmentation produced by the network</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>2D U-net loss functions: Comparison between the training process (measured on test data) using 2D U-net with weighted binary cross-entropy (blue curves) and Dice-loss function (orange curves). Left: loss function, Middle: mean accuracy, Right: Dice-coefficient</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>The 3D U-net: Comparison between the original ground truth (first row) and the segmentation generated by the 3D U-net (second row). Here we see the projections on the $yz$-plane, the $xy$-plane and the $xz$-plane.</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>The slice-by-slice 2D U-net: Comparison between the original ground truth (first row) and the segmentation generated by the 3D U-net (second row). Here we see the projections on the $yz$-plane, the $xy$-plane and the $xz$-plane.</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Reconstruction operator $R^{2,{0,90}}$: The voxel value is defined as the sum over the corresponding 2D values, here illustrated for two MIP images.</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Filtrated backprojection: Left: Network’s output (before threshold) using backprojection without filtration. Right: Network’s output (before threshold) using filtrated backprojection.</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Number of projection directions: Performance of the 2.5D U-net for different number $p$ of projection directions.</td>
<td>36</td>
</tr>
<tr>
<td>4.6</td>
<td>The 2.5D U-net with 12 deterministic projection directions: Comparison between the original ground truth (first row) and the segmentation generated by the 2.5D U-net (second row). Here we see the projections on the $yz$-plane, the $xy$-plane and the $xz$-plane.</td>
<td>38</td>
</tr>
<tr>
<td>4.7</td>
<td>Summation over projection regions: We construct MIP images for every $\theta \in {-5^\circ, -4^\circ, \ldots, 174^\circ}$ and sum them over regions of $10^\circ$.</td>
<td>39</td>
</tr>
<tr>
<td>4.8</td>
<td>The 2.5D U-net with integration over projection areas: Comparison between the original ground truth (first row) and the segmentation generated by the 2.5D U-net (second row). Here we see the projections on the $yz$-plane, the $xy$-plane and the $xz$-plane.</td>
<td>41</td>
</tr>
</tbody>
</table>
4.9 **Structure of the random 2.5D U-net:** On the left side, we see the first path, in which the 2D convolutional part $U$ is trained with random MIP images $M_{\theta_1}, \ldots, M_{\theta_p}$. The output $\hat{y}_{aux}$ of the first path is a 3D reconstruction of these random projections. On the right side, we see the second path, in which $U$ is applied to all available projections images $M_{\theta_1}, \ldots, M_{\theta_m}$. The output $\hat{y}$ of the second path is also a 3D reconstruction and the main output of the model. Both paths are optimized via the joint Dice-loss function $\tilde{\ell}_c$.

4.10 **The random 2.5D U-net:** Comparison between the original ground truth (first row) and the segmentation generated by the random 2.5D U-net (second row). Here we see the projections on the $yz$-plane, the $xy$-plane and the $xz$-plane.

4.11 **Visual comparison between the proposed networks for volumetric segmentation:** From left to right: Ground truth, random 2.5D U-net, 2.5D U-net with 12 deterministic projection directions, 2.5D U-net with integration over projection areas and 3D U-net. Images were constructed with the freeware ITK-SNAP [22].
List of Tables

3.1 **Summarization** of the evaluation results of all 2D networks proposed in this thesis. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

4.1 **Summarization** of the evaluation results of all 3D networks proposed in this thesis. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46

4.2 **Summarization** of time and storage observations of all 3D networks proposed in this thesis. . . . . . . . . . . . . . . . . . . . . . . 46
Listings

4.1 Implementation of the \texttt{reconstruction operator}. \hfill 34
4.2 Implementation of the \texttt{filtration layer}. \hfill 35
4.3 Implementation of the \texttt{sharpening layer}. \hfill 40
1 Introduction

Convolutional neural networks have become a powerful method for image recognition \[12, 20\]. For example, they show outstanding performance for classifying images due to recognizing different features in the 2D data. In the last few years they also exceeded the state-of-the-art in providing segmentation masks for 2D images. This is, for instance, a huge effort in autonomic driving. The car always records its environment and then has to know by which objects it is surrounded. The car needs the information: “Where is a street?” “Where is a tree?” “Where are passengers?” So the computer has to label each pixel of the current image which leads to a segmentation mask.

When we look at Figure 1.1 we get an idea of how such a segmentation mask looks like. We mark the different areas of the image and label them. So label 1 could be the colosseum, label 2 the vegetation, label 3 the sky and label 0 the rest.

![Figure 1.1: Image segmentation: Left we have an image of the environment, in the middle we mark the different objects, and on the right we see the resulting segmentation mask with label 0 (black), label 1 (red), label 2 (green) and label 3 (blue). The pictures have been constructed with the freeware PixelAnnotationTool \[4\].](image-url)
1 Introduction

In [16], the idea of using fully convolutional networks for segmentations came up. Based on this work, the U-net introduced in [18] provides a powerful 2D segmentation tool for biomedical applications. It has been demonstrated to learn highly accurate ground truth masks from only very few training samples.

Among others, the fully automated generation of volumetric segmentation masks becomes increasingly important for biomedical applications. This task still is challenging. One idea is to extend the U-net structure to volumetric data by using 3D convolutions, as it has been proposed in [6, 10, 17]. Significant drawbacks are the huge memory requirements and the resulting restrictions to deepness and filter number of the network. Deep learning segmentation methods therefore are often applied to 2D slice images. However, these slice images do not contain information of the full 3D data, which makes the segmentation task much more challenging. To address the drawbacks of existing approaches, we introduce a network structure which is able to generate accurate volumetric segmentation masks of very large 3D volumes. The main idea is to include projection layers from different directions which transform the data to 2D images containing information of the full 3D image. The projection method mainly depends on the application. Common used methods are maximum intensity projection, the Radon-transform [7] or a combination of both. We then apply the 2D U-net to these projection images and propose a learnable reconstruction algorithm to lift them again to volumetric data.

The main focus of this thesis lies on sparse binary segmentation of volumetric data. At the end, the reader can see, that this novel proposed 2.5D U-net architecture even generates more accurate segmentation masks than the 3D U-net for the following application:

We take as an example the magnetic resonance angiography (MRA), where blood vessels (arteries and veins) get imaged in a 3D scan. As targeted application, we
1 Introduction

focus on the generation of volumetric binary segmentation masks of these scans. In particular, we aim at segmentations of those blood vessels, which assist the doctor to detect abnormalities like stenosis or aneurysms. Furthermore, the medical sector is looking for a fully automated method to evaluate large cohorts in the future. Here, a good semantic segmentation of the blood vessels, in which the doctor is interested, is crucial for good detection.

The Department of Neuroradiology Innsbruck has provided volumetric MRA scans of 119 different patients. The data faces the arteries and veins between the brain and the chest (Figure 1.2).

![Figure 1.2: 3D MRA scan](image)

(a) Transversal plane. (b) Sagittal plane. (c) Coronal plane.

Figure 1.2: 3D MRA scan: Here the three planes (a)-(c) of a volumetric MRA scan are displayed. The 2D visualizations of the 3D scan were generated with the freeware ITK-SNAP [22].

Fortunately, also the volumetric segmentation masks (ground truths) of these 119 patients have been provided. These segmentation masks have been generated by hand which takes hours for one patient. The found segmentation mask for one of these scans is shown in Fig. 1.3.

Our goal is the fully automated generation of the 3D segmentation masks of the blood vessels. For that purpose we use deep learning and neural networks. At the first glance, this problem may seem to be quite easy because we only have two labels (0: background, 1: blood vessel). However, there are also arteries and veins which have label 0 which might confuse the network since we only want to segment those
1 Introduction

(a) Transversal plane.  
(b) Sagittal plane.  
(c) Coronal plane.  
(d) Volumetric segmentation.

Figure 1.3: 3D segmentation: In every plane (a)-(c) the blood vessels of interest are marked in red. In (d) we see the resulting segmentation mask. The segmentation was conducted with the freeware ITK-SNAP [22].

vessels the doctor is interested in. Other challenges are caused by the big size of the volumes (96 × 288 × 224 voxels) and by the very unbalanced distribution of the two labels (on average, 99.76% of all voxels indicate background).

To get a better understanding of these problems we will initially switch to the 2D-case. For very sparse binary segmentation of blood vessels, an appropriate way to reduce dimensions is the generation of maximum intensity projection (MIP) images as illustrated in Figure 1.4.
1 Introduction

Figure 1.4: **Maximum intensity projection** of a 3D object:

Taking the maximum values along the projection direction (in this case the first spatial axis) produces a 2D projection containing information of the 3D data.

This technique includes rotating the 3D data around the vertical axis ($z$-axis) using angles $\theta \in \Theta$ (projection directions) and taking the maximum values along the first spatial axis after each rotation. So if we choose for the amount of projection directions $\Theta = \{k \times 36^\circ \mid k = 0, \ldots, 9\}$, we get 10 different MIP images out of one 3D scan. Data corresponding to one patient are shown in Figure 1.5. So we augment the training data from 119 3D samples to $119 \cdot |\Theta|$ 2D samples and no further data augmentation process is necessary. Note that choosing $\theta$ too small is not beneficial for our problem due to the following reason:

In the 2D case, we will split our data into 70\% training data, 15\% test data and 15\% evaluation data. For $\theta \ll 1$ it happens that the model gets tested on an image which has very high similarity to another image which the model has seen in the training process. So we would get biased performance scores which we have to avoid.
1 Introduction

Figure 1.5: MIP images of a 3D MRA scan with projection directions \( \Theta = \{k \times 36^\circ \mid k = 0, \ldots, 5\} \). In the first row, we see the projections of the original scan, in the second row we see the corresponding projections of the ground truth.

After this introduction we start with the task of 2D segmentation by introducing the structure of fully convolutional networks in the next chapter.
2 Fully Convolutional Networks

As mentioned in the introduction, the aim of semantic segmentation is to find a classifier, that assigns each pixel of an image to its corresponding class. The naive approach would be to train a convolutional network that gets as input a certain region around each pixel. The output of this network will then be probabilities predicting the class of only this pixel. Segmentation of the whole image could then be made by sliding-window technique. The disadvantages of such an approach are clear:

- A lot of redundancy is caused by overlapping regions.
- Long computational time is needed to make a prediction for a large image, e.g. the 2D MIP images of the MRA dataset are of size 288 × 224. Thus, there would be $6.5 \times 10^4$ predictions to compute for one MIP image.
- Finding a good way to deal with border pixels in the application can be very time-consuming.

However, since for this approach classification happens for each pixel individually, determining the local position of each assigned pixel in the resulting classification map does not cause a problem.

In this chapter, we introduce a more effective approach, namely *fully convolutional networks*. The architecture of these network types and their application to semantic segmentation tasks gets discussed very precisely in [16]. This approach overcomes
the drawbacks of the naive one and, additionally, it shows that a precise localization is also possible without sliding-window techniques.

2.1 From VGG-nets to FCNs

Considering image classification using deep learning, the VGG-net architectures [20] have an essential impact on this issue. These are very deep convolutional networks for detecting objects in images. This is done by multiclass classification, i.e. we have \( n_c \) different objects and want to predict which of our \( n_c \) objects can be seen in this image. These VGG-nets were developed by the Visual Geometry Group of University of Oxford. The construction properties are simple:

i) The network gets an input image \( \mathbf{x} \in \mathbb{R}^{n_h \times n_w \times n_f} \), where \( n_h \) and \( n_w \) are the spatial dimensions and \( n_f \) describes the number of channels (e.g. for RGB-images denotes \( n_f = 3 \) the three different colour channels red, green and blue). The number \( n_f \) is also called the number of filters.

ii) The next part is called the convolutional part or also contracting part. It includes stacking over convolutional blocks consisting of two convolutional layers and one max-pooling layer considering the following properties:

- The very first convolutional layer raises the number of filters to 64.
- We only use \( 3 \times 3 \) filters to hold down complexity and use zero-padding to guarantee that all layer outputs have even spatial dimension.
- Each max-pooling layer has stride \((2, 2)\) to half the spatial dimensions. We must be aware that the spatial dimensions of the input image can get divided by 2 often enough without producing any rest.
- After each max-pooling layer we increase the number of filters by the factor 2.

iii) This is followed by the classification part, which consists of a stack of fully
connected layers for classification. The last layer of this part gives an output\( \mathbf{o} \in \mathbb{R}^{1 \times 1 \times n_c} \), where \( n_c \) denotes the number of classes of our classification problem. Applying softmax-activation-function to \( \mathbf{o} \) gives numbers \( p_0, \ldots, p_{n_c-1} \) in \([0, 1]\), where \( p_k \) denotes the probability that the image belongs to class \( k \) for \( k \in \{0, \ldots, n_c - 1\} \), i.e. the probability that the \( k \)-th object gets detected in the image.

Although to their deepness (in practice between 11 and 19 convolutional layers), VGG-nets are able to keep complexity low. Their impressing performance on different classification challenges are mentioned in [20].

Recall that general deep networks like the VGG-nets are non-linear functions which map a spatial input like an image to non-spatial output. But if we remove the classification part and all layers consist of either convolution or pooling operations, then the network denotes a deep filter for the input image.

Such a deep filter for dense prediction problems (e.g. segmentation) is a function \( \mathcal{N} : \mathbb{R}^{n_h \times n_w \times n_f} \rightarrow [0, 1]^{n_h \times n_w \times n_f} \) and gets called fully convolutional network (FCN). To get an idea of the transformation of a VGG-net to a deep filter we consider the following construction:

i) We use the VGG-13-net (13 weight-layers) out of [20] Table 1, but for the benefit of lower computational costs we remove the last convolutional block with filter size 512 and reduce the dense sizes in the classification part from 4096 to 1024. From now on we call this reduced version the VGG-11-net.

ii) For deep filters, we only consider convolutional and pooling layers. Therefore, we transform each fully connected layer in VGG-11-net to a convolutional layer with the number of filters equal to the dense size. The size of these filters equals \( 1 \times 1 \).

iii) Our deep filter should construct a segmentation mask for the input, so we need
the output to have the same spatial size as the input. The solution is to stack an additional part onto the network, a so-called *upsampling part*. So if we add $k$ max-pooling layers in the contracting part, we need an upsampling by the factor $2^k$. There are two common ways to implement this:

- **Bilinear Interpolation:**
  Assume that we want to upsample an image $A \in \mathbb{R}^{4 \times 4}$ to an image with spatial dimensions $(4k, 4k)$, $k \in \mathbb{N}_{\geq 2}$. For simplicity we omit the channel dimension. Then we pad the smaller image the following way:

  $$A_{\text{pad}}[i, j] \triangleq \begin{cases} A[i, j], & \text{if } i \mod k = j \mod k = 1 \\ 0, & \text{else} \end{cases}$$

  for $i, j \in \{1, \ldots, 4k\}$.

  For all padded pixels we set

  $$A_{\text{pad}}[i, j] = \frac{w_1}{4} A[n_1(i, j)] + \frac{w_2}{4} A[n_2(i, j)] + \frac{w_3}{4} A[n_3(i, j)] + \frac{w_4}{4} A[n_4(i, j)],$$

  where $(i \mod k \neq 1) \lor (\mod k \neq 1)$, the functions $n_1, n_2, n_3, n_4$ denote the indices of the nearest 4 not padded cell neighbours, and $w_1, w_2, w_3, w_4$ are computed by the distance to these neighbours. Obviously, this is a linear upsampling with fixed parameters which need not to be learned.

- **Transposed Convolution:**
  A kind of non-linear upsampling can get achieved by transposed convolution [9]. It is basically upsampling convolution with learnable parameters. Again we omit the channel dimension. As an example: Given again an image

  $$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbb{R}^{2 \times 2},$$
for stride $s = (2, 2)$ we pad $A$ with zeros such that

$$A_{\text{pad}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a & 0 & b & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \in \mathbb{R}^{6 \times 6}.$$

Then we train a $3 \times 3$ filter $k$ and compute a standard convolution $B = A_{\text{pad}} \ast k \in \mathbb{R}^{4 \times 4}$ with stride 1 and no padding. So we augmented the spatial dimension with a trainable upsampling by factor 2. Due to the reason that in the above example the filter size is not equal to $s$, we add a zero row and a zero column to $A_{\text{pad}}$. In this case, we call it a transposed convolution with padding and the resulting output size of this operation is equal to the input size multiplied by the stride.

First we will consider the easier technique, namely linear upsampling, and later research if switching to trainable upsampling increases performance of deep filters on our targeted application.

iv) After resizing the image at the end of the contracting path back to the same spatial dimensions as the original input, the construction of a deep filter is completed. Because we need a segmentation mask, we have to apply an appropriate activation function pointwise on the last layer to get a prediction, which pixel belongs to which label. Since we want our deep filter to be able to annotate vessel pixels in MIP images of volumetric MRA scans, we only consider two labels (background and vessel) and therefore need binary output. This suggests the sigmoid-function $\sigma : \mathbb{R} \to (0, 1), \ x \mapsto (1 + e^{-x})^{-1}$ as output activation function. Such a deep filter outputs for each pixel the probability
of being a vessel pixel (foreground pixel). The resulting segmentation mask is then generated by the application of a threshold (usually 0.5).

We will call the transformed version of the VGG-13-net now the FCN-11-bi net, denoting the fully convolutional network with 11 convolutional layers and upsampling by bilinear interpolation. An overview of the structure is given in 2.1.

Figure 2.1: The FCN-11-bi net gets an input scan (left) with 1 channel and outputs for each pixel the probability for being a vessel pixel. Applying a threshold of 0.5 generates the final segmentation mask (right). Total number of trainable parameters: $6.3 \times 10^6$.

### 2.2 Network Training

All in all, the FCN-11-bi network has 11 convolutional layers with $6.3 \times 10^6$ different parameters that have to be adjusted during training.

- **Adam-Optimization:**

  We train with Adam-optimizer provided in the Keras module. Basically Adam algorithm is stochastic gradient descent with momentum, but while ordinary stochastic gradient optimization holds a fixed learning rate for all parameters during the whole training process, Adam algorithm maintains different learning rates for each parameter and adjusts them during training. Instead of adapting the parameters only in relation to the first moment (i.e. the mean) of the gra-
Fully Convolutional Networks

dients like in AdaGrad-optimization, Adam algorithm also makes use of the second moments (i.e. the uncentered variance) like RMSProp-optimization. So the combination of RMSProp and AdaGrad algorithm with an individual learning rate for each weight parameter provides an effective way of optimization for large problems (very much parameters), and also for noisy data and sparse gradients. A more detailed description of Adam-optimization and its benefits can be found in [15].

- **Weighted Binary Cross-Entropy:**

  Using the sigmoid-activation-function in the last layer for pixel classification, suggests the binary cross-entropy loss function [3, Section 5.2]. A problem we have to deal with is the very unbalanced data. Given the 2D MIP images of our volumetric MRA scans, only 5.5% of all pixels belong to the blood vessels our network has to segment. As a consequence, using ordinary binary cross-entropy loss would force the network to take special focus only on those pixels which denote background because this would even lead to a accuracy rate of 94.5%. So we have to make the network more sensible about the little amount of vessel pixels. This can be done by adding some class-weights to the loss function.

  The ordinary binary cross-entropy loss is given by

  \[
  \ell(y, \hat{y}) \triangleq - \frac{1}{\text{#pixels}} \sum_k \left( (y \odot \log(\hat{y}))_k + ((1 - y) \odot \log(1 - \hat{y}))_k \right),
  \]

where

- \( \odot \) denotes pixelwise multiplication,
- \( y \) denotes the ground truth (i.e. \( y \) is an image with values 0 for background and 1 for blood vessel) corresponding to the training sample \( x \),
- the sum is taken over all pixel locations (average sum over all pixels of \( y \)),

\[13\]
–  $\mathbf{1}$ is a tensor with same spatial dimension as $y$ and with all entries equal to 1,

– $\hat{y} = \mathcal{N}(x)$ are the predicted probabilities by the network $\mathcal{N}$ for the training sample $x$.

The only change we have to make for the weighted binary crossentropy loss is adding two parameters $w_0, w_1 \in \mathbb{R}_{\geq 0}$ and defining the new loss

$$\ell_w(y, \hat{y}) \triangleq -\frac{1}{\#\text{pixels}} \sum_k \left( w_1 (y \odot \log(\hat{y}))_k + w_2 ((1 - y) \odot \log(1 - \hat{y}))_k \right).$$

An efficient way to compute the class weights is

$$w_0 = \frac{|\mathcal{Y}|}{2(|\mathcal{Y}| - \sum \mathcal{Y})}, \quad w_1 = \frac{|\mathcal{Y}|}{2 \sum \mathcal{Y}},$$

where $\mathcal{Y} \subset (\{0, 1\}^{n_h \times n_w})^N$ denotes the amount containing the ground truths of all $N$ training samples. The sums are taken over the whole amount, i.e. over all ground truths. The error function which has to be minimized is then calculated by

$$E_N(\mathcal{N}) = \frac{1}{N} \sum_{i=1}^{N} \ell_w(y_i, \hat{y}_i),$$

where $N$ denotes the amount of training samples, $\hat{y}_i = \mathcal{N}(x_i)$ is the prediction for the $i$-th training sample $x_i$ and $y_i$ the corresponding ground truth.

• Callbacks:

The use of callbacks can have much positive influence on the training process. An example is learning rate scheduling. The Adam-optimizer takes as input an upper boundary for all individual learning rates. When the network shows no improvement after 3 epochs, we reduce the learning rate by the factor 0.5. This can help the network to optimize more precise after gradient plateaus get passed in the first few epochs.
Also making use of an early-stopping-technique is quite recommended. This helps to avoid the overfitting problem. The more a network learns the same stuff, the more it can perfectly memorize the learned things but it often loses the ability to generalize. To avoid this, stopping the training process when the error over the test samples starts increasing while the error over the training set still decreases seems to be a good solution. In this thesis, every network gets stopped after showing no improvement for 5 epochs and the best weights in terms of validation loss get restored.

• Metrics:

Due to the very unbalanced data distribution, monitoring the pixel accuracy for the whole prediction makes not much sense, especially since we are interested in a good prediction of the foreground pixels. Thus we make use of three metrics which are common in sparse semantic segmentation.

For $i, j \in \{0, 1\}$ let us denote by $p_{ij}$ the number of all pixels of class $i$ predicted to class $j$, and by $t_i$ the number of all pixels belonging to class $i$. With this notation, we evaluate the following evaluation metrics during training:

- **Mean Accuracy**:

  \[
  \text{MA} \triangleq \frac{1}{2}\left(\frac{p_{00}}{t_0} + \frac{p_{11}}{t_1}\right).
  \]

  This metric calculates the mean pixel accuracy for both classes, but in contrast to ordinary pixel accuracy it does not get influenced by the size of each class.

- **Mean Intersection over Union**:

  \[
  \text{IU} \triangleq \frac{1}{2}\left(\frac{p_{00}}{t_0 + p_{10}} + \frac{p_{11}}{t_1 + p_{01}}\right).
  \]

  This very commonly used metric is basically the same as mean accuracy, but it also takes care of the amount of false predicted pixels for each class.
– Dice-coefficient [3, 10, 17]:

\[ DC \triangleq \frac{2p_{11}}{2p_{11} + p_{01} + p_{10}}. \]

This metric named after Lee Raymond Dice is an indicator of similarity of two images. The Dice-coefficient especially focuses on the right predicted foreground pixels and compares them with the total number of foreground pixels in the ground truth. Therefore it gets preferred as performance measure for sparse binary 3D segmentation as marking important vessels in MRA scans or separating cancer affected regions in MRT scans. In the following, we will especially rely on the Dice-coefficient for performance comparison between models.

**Kernel Initialization:**

Note that a good initialization of the start weights is crucial for convergence of a network. The kernel initialization of each layer depends not only on the input scales but also on the layer size itself. After dealing with big convergence problems using the default initialization of Keras, defining our own initialization function solved the problem:

In FCN-11-bi net the start weights of each layer are normally distributed with mean 0 and deviation

\[ \frac{1}{f_\ell}, \quad \ell = 1, \ldots, 11, \]

where \( f_\ell \) denotes the number of filters of the \( \ell \)-th convolutional layer in FCN-11-bi net.

**Batch Normalization:**

In every network that gets constructed in this thesis, the first layer will be a batch normalization layer [13], to normalize every image before transferring it to the convolutional part. After each max-pooling layer we also make use
of batch normalization, i.e. we normalize the hidden units to speed up convergence. This ensures that we do not obtain very high or low values in the hidden layers and therefore batch normalization reduces the covariance shift of the hidden units.

A batch normalization layer in Keras conducts normalization by computing for each input tensor $x$ the mean $\mu_x$ and the variance $\sigma_x^2$ of $x$. These two scalars are computed for every input individually. The output of such a batch normalization layer is then of the form

$$\gamma \frac{x - \mu_x}{\sqrt{\sigma_x^2 + 10^{-8}}} + \beta,$$

where $\gamma$ is a trainable scale parameter and $\beta$ is a trainable offset parameter. Hence, a batch normalization layer consists of 4 parameters: $\gamma, \beta$ get adapted during training process and $\mu_x, \sigma_x^2$ are computed for each input individually.

Note that in the application to test data, $\gamma$ and $\beta$ remain the same, while $\mu_{\tilde{x}}, \sigma_{\tilde{x}}^2$ are also computed individually for each test sample $\tilde{x}$.

As explained in the introduction, volumetric MRA scans of 119 different patients are given. Transferring the problem into the 2D case using MIP images with projection directions $\Theta = \{k \times 36^\circ \mid k = 0, \ldots, 9\}$ augments our data set to 1190 samples. We crop each projection in order to obtain images of spatial sizes $288 \times 224$. This ensures, that the outputs of the hidden layers also have even spatial size after applying several max-pooling operations. Therefore, the spatial dimension at the end of the contracting part is $18 \times 14$. We randomly choose 70% of the samples as training data and split the remaining into 15% test data and 15% for evaluation.

We conduct whole image training on NVIDIA GeForce RTX 2080 GPU. The advantage of training on GPUs is that they are able to compute several gradients simultaneous and therefore speed up training process. The GPU also has an impact on the choice of the minibatch size. This size denotes the amount of training samples
used to make an update. Computing several gradients simultaneously and taking the average of them for updating reduces the variances between all stochastic gradient updates. Therefore, the optimization algorithm is able to make faster progress during training. Considering the dimensions of our inputs and the deepness of our model, a minibatch size of 8 seems to be an appropriate choice.

Deploying the FCN-11-bi net to the evaluation data results into a weighted loss of 0.488, MA of 90.2\%, IU of 51.5\% and DC of 36.3\%. Total training process lasts on average 6.02 minutes, the application time to a MIP image is less than 0.01 seconds. The results for the FCN-11-bi net on a few evaluation samples are monitored in Figure 2.2.

![Performance of the FCN-11-bi net](image)

Figure 2.2: **Performance of the FCN-11-bi net**: Above we see the corresponding ground truth, below the segmentation produced by the network.

Looking at the image plots in Figure 2.2 we are not satisfied with this result. Although the network recognizes the difference between background and blood vessel, it seems that we lose too much information when contracting the image to size $18 \times 14$ in the contracting part. The local positions of the pixels in the resulting segmentation map are very coarse. So the more diligent the FCN-11-bi learns the difference between labels, the more it limits the amount of localization details. To overcome this problem, we rely on a novel skip architecture proposed in [16], combining information of all convolutional layers and not only of the smallest one.
2.3 Skip-Architecture

So far, the FCN-11-bi net is able to gather very deep semantic information, but unfortunately in a coarse way. Hence a combination of each pixel’s information with its localization is necessary. Of course, using the naive way of predicting each pixel separately would not have to deal with this problem, but the disadvantages mentioned in the beginning of this chapter are too fatal. Long et al. proposed in [16] one of the first successful approaches to overcome this problem. The main idea is to link coarse layers having higher spatial dimensions in the contracting part with the finest layer at the end of the contracting part. So we overcome the disadvantage that the whole information for the final prediction only comes from a layer with 16× smaller spatial dimension. The solution is simple:

- After each max-pooling operation, we make a prediction applying a convolution with the amount of 1 × 1 filters equal to the desired channel size of the output.

- Instead of applying a 16× upsampling operation at once on the finest layer, we conduct stepwise 2× upsampling and combine each upsampled image with the corresponding prediction of the max-pooling layer with same spatial size (Figure 2.3).

- There are different ways of merging coarse layers with their upsampled counterparts, like concatenation or addition.

Long et al. [16] analyzed the benefits of this approach for different modifications of their fully convolutional filter. After changing the amount of links between coarse predictions and fine upsampled predictions, they concluded that linking each max-pooling layer via addition yields satisfying performance boost.

So if we apply this approach to the FCN-11-bi net, we obtain the new fully convolutional network FCN-11-bi-skip net, which is now able to read out local information in combination to its global structure (Figure 2.3).
Figure 2.3: The **FCN-11-bi-skip net** is the modified version of the FCN-11-bi net with skip architecture. The purple arrows show the combination of local information and fine prediction. For this model, again bilinear interpolation is used as upsampling technique.

Applying this new structure on the evaluation data yields improvement for weighted loss and all evaluation metrics. In detail:

Weighted loss of 0.41, MA of 94.9%, IU of 64.8% and DC of 55.1%. Again, total training process lasts on average 8.85 minutes, the application to a MIP image takes less than 0.01 seconds. The amount of adjusted parameters has not changed, compared to the FCN-11-bi net, and equals $6 \times 10^6$. Looking at Figure 2.4, we recognize that we also obtain visual improvement.

But we have to admit that the current result is far away from our expectation. In Figure 2.4 we discover that the FCN-11-bi-skip net produces a lot of artefacts. One also may see that the upsampling is very rough and could be much smoother. At the introduction of fully convolutional filters we mentioned that there exist different upsampling methods. So instead of further using bilinear interpolation, we include...
transposed convolution to double spatial dimensions in the upsampling part. Especially using this operation with kernel size $3 \times 3$ and stride $s = (2, 2)$ leads to significant improvement for all displayed performance measures and generates a smooth upsampling result (Figure 2.5). This changes lead to a modified version of the FCN-11-bi-skip net, namely the FCN-11-dc-skip net, where $dc$ denotes the new upsampling method deconvolution. Note that deconvolution is a colloquial term and not the right scientific term for this operation, because basically it is not the inverse of convolution.

At this moment we obtain the following already satisfying evaluation results:

Weighted loss of 0.085, MA of 96.8%, IU of 75% and DC of 70.2%. In average, total training lasts 8.73 minutes, the average application time is smaller than 0.01 seconds and the amount of weights slightly increased due to the transposed convolutional layers but remains approximately $6.3 \times 10^6$.

Although the amount of artefacts decreased significantly, we must not neglect that the FCN-11-dc-skip net still produces some artefacts in its predictions. Our next approach is based on [18]. Ronneberger et al. extended the idea of fully convolutional filters presented here, which led to a new network architecture providing state-of-
Figure 2.5: **Performance of the FCN-11-dc-skip net**: Above we see the corresponding ground truth, below the segmentation produced by the network.

the-art methods for semantic segmentation till today: the *U-net*. 
3 The U-net

3.1 The 2D U-net

Basically, the U-net proposed in [18] is closely related to the deep filters we discussed so far. Ronneberger et al. have taken the idea of fully convolutional networks with skip connections and convolutional upsampling (transposed convolution) out of [16] and improved performance significantly by changing basically two things in the structure:

- So far, the upsampling part only consists of merging layers and transposed convolutions. To obtain similarity to the contracting part, we also make use of convolutional blocks consisting of two convolutional layers with kernel size $3 \times 3$ after each upsampling layer. Instead of using a convolutional layer with filter number 1 at the end of the contracting part, we decrease channel size stepwise in the same way we augmented it in the contracting part (Figure 3.1).

- Instead of using summation layers to link the contracting part with the upsampling part, we use concatenation layers, where two images with same spatial dimension get concatenated over their channel dimension. Therefore it is not longer necessary to reduce the channel size to 1 before merging, and the output of each convolutional block can get used directly.

Obviously, this new structure has much more weights to be adjusted through the fully convolutional upsampling part. The symmetry in this structure leads to the
fitting name U-net. To avoid running out of memory, Ronneberger et al. only trained the network in [18] with minibatch size 1. Fortunately, due to the memory capacity of our GPU we enjoy freedom of choice for the depth of the 2D network and minibatch size during training. Ronneberger et al. also suggest to augment spatial dimension of the input images by mirroring and to use convolutional layers without padding to make maximum use of the information in the border pixels. Since in our application we do not have any useful information in the border pixels, we do not follow this approach.

It is important to note, that the U-net structure provides state-of-the-art methods for segmentation till today, especially in biomedicine. The reason why first of all the biomedical sector benefits from it, is that a U-net is able to learn very accurate segmentations with only very few training images. Due to the reason that
the amount of data samples in biomedical approaches is often very small, the U-net gained popularity rapidly in different applications, like cancer detection, cell annotation, vessel segmentation, et cetera.

Now it is time to apply this new structure to the 2D MIP images. The U-net in our implementation has channel size 32 in the first convolutional block and channel size 512 at the end of the contracting part (4 downsampling steps in the contracting part). This leads to a total number of $8.6 \times 10^6$ trainable parameters. Remember that the channel size gets doubled after each max-pooling operation and halved after each concatenation layer.

Due to the huge amount of training samples (we received in total 1190 MIP images out of the 3D MRA scans) and the property of this architecture to learn fast, we have to be careful about overfitting. Therefore we insert dropout layers [21] between the convolutional layers. This type of layer chooses a random amount of entries of the input tensor corresponding to the dropout rate and sets these entries to 0. Using a dropout rate of 0.5 in the deepest convolutional block and a dropout rate of 0.2 in the two second deepest blocks dims the overfitting problem satisfyingly. We train the network with minibatch size 6 and use again the self-generated weighted binary crossentropy loss function. Using again the (70, 15, 15) split into training, test and evaluation data leads to the following results:

Weighted loss of 0.052, MA of 98.2\%, IU of 83.1\% and DC of 81.1\%. Average total training is 7.1 minutes and application time slightly increases to 0.013 seconds.

Having a careful look on Figure 3.2, we observe that the blood vessels of the network’s segmentations are always a little bit thicker than those of the ground truth. This result is not surprising due to the choice of our loss function:

We have to remember that we modified the binary crossentropy loss function such that it puts high weight to detecting foreground pixels denoting vessels. This leads to a network that wants to make sure to detect all pixels denoting foreground. And
3 The U-net

Figure 3.2: **Performance of the 2D U-net**: Above we see the corresponding ground truth, below the segmentation produced by the network.

to ensure that really all these pixels have been detected, the network adds at the border of each vessel some pixels also denoting vessels. So the idea is to find a loss function, which not only takes special care of the foreground pixels, but also ensures that the number of those pixels in the prediction is pretty the same as in the ground truth. This leads to an interesting approach using the *Dice-loss* mentioned in [17].

3.2 The Dice-loss Function

In Section 2.2 we introduced a metric indicating the similarity between two images, that is often used for binary segmentations problems in medicine, namely the Dice-coefficient. This metric measures similarity by comparing all correctly predicted vessel pixels with the total number of vessel pixels in the prediction. Obviously we are not able to train the network with this metric since it takes Boolean inputs. Therefore we would have to apply a binarization operation to the model’s output resulting to a non differentiable loss function. Milletari et al. modified in [17] the Dice-coefficient a little bit such that it becomes a differentiable loss function, the
3 The U-net

**Dice-loss:**

\[
\ell(y, \hat{y}) = 1 - \frac{2\sum_k (y \odot \hat{y})_k}{\sum_k \hat{y}_k + \sum_k y_k},
\]

(3.1)

where \( \odot \) denotes pixelwise multiplication, the sums are taken over all pixel locations, \( \hat{y} = U(x) \) are the probabilities predicted by the U-net, and \( y \) is the corresponding ground truth.

Training our U-Net with this new loss function yields following results:

Dice-loss of 0.088, MA of 95.7\%, IU of 91.6\% and DC of 91.3\%.

![Figure 3.3: 2D U-net with Dice-loss](image)

Figure 3.3: **2D U-net with Dice-loss:** Above we see the corresponding ground truth, below the segmentation produced by the network.

Figure 3.3 shows, that the visual results are more satisfying now. While average application time remains the same, with an average total training time of 13.5 minutes we obtain a significant increase for the amount of training epochs. Obviously, training with the new loss function slightly decreases the mean accuracy on validation data, but it has a positive impact on the Dice-coefficient (see Figure 3.4). Since we aim for high similarity for sparse binary data, all networks will be trained with the just introduced Dice-loss function from now on.
While the main task of this thesis is the volumetric segmentation of 3D data, we will skip any further fine-optimization in the 2D case and move on with the extension of our networks to three spatial dimensions in the next chapter. The evaluation results of all 2D networks introduced so far on our targeted application are summarized in Table 3.1. In the table also the amount of trainable parameters, the average total training and the average application time on one MIP sample are displayed.

Table 3.1: **Summarization** of the evaluation results of all **2D networks** proposed in this thesis.

<table>
<thead>
<tr>
<th>Network</th>
<th>Weights</th>
<th>Training</th>
<th>Application</th>
<th>loss</th>
<th>MA %</th>
<th>IU %</th>
<th>DC %</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCN-11-bi</td>
<td>$6.3 \times 10^6$</td>
<td>6.02 min.</td>
<td>&lt; 0.01 sec.</td>
<td>0.488</td>
<td>90.2</td>
<td>51.5</td>
<td>36.3</td>
</tr>
<tr>
<td>FCN-11-bi-skip</td>
<td>$6.3 \times 10^6$</td>
<td>8.85 min.</td>
<td>&lt; 0.01 sec.</td>
<td>0.410</td>
<td>94.9</td>
<td>64.8</td>
<td>55.1</td>
</tr>
<tr>
<td>FCN-11-dc</td>
<td>$6.3 \times 10^6$</td>
<td>8.73 min.</td>
<td>&lt; 0.01 sec.</td>
<td>0.085</td>
<td>96.8</td>
<td>75.0</td>
<td>70.2</td>
</tr>
<tr>
<td>U-Net</td>
<td>$8.6 \times 10^6$</td>
<td>7.10 min.</td>
<td>0.013 sec.</td>
<td>0.052</td>
<td>98.2</td>
<td>83.1</td>
<td>81.1</td>
</tr>
<tr>
<td>U-Net Dice-loss.</td>
<td>$8.6 \times 10^6$</td>
<td>13.5 min.</td>
<td>0.013 sec.</td>
<td>0.088</td>
<td>95.7</td>
<td>91.6</td>
<td>91.3</td>
</tr>
</tbody>
</table>
4 Volumetric Segmentation

4.1 The 3D U-Net

Now we aim at generating binary segmentation masks of sparse volumetric data using a 3D version of the prior introduced U-net. The resulting 3D U-net follows the same structure as in Section 3, the only difference is the usage of 3D convolutions and 3D pooling layers. For the 3D U-net we have to take special care about overfitting \cite{11} and about memory space. Especially the second point causes restrictions to the number of filters and the amount of downsampling steps in the contracting part. Therefore, for the 3D U-net we choose channel size 8 at the beginning and channel size 32 at the end of the contracting part (2 downsampling steps). This results into a network consisting of $9.8 \times 10^4$ trainable parameters. Also, the use of high dropout rates \cite{21} (0.5 in the deepest convolutional block) is necessary to ensure an efficient training process. Due to the huge size of our training samples ($96 \times 288 \times 224$ voxels), we train the network on a single batch, i.e. with minibatch size 1. During training, the 3D U-net allocates a memory space of at least 4.87 gigabyte on the GPU. The total training process takes on average 33.9 minutes. Since the number of 3D training samples is only 119, we conduct 7 training-runs using cross-validation:

i) We split the whole data into 7 subsets.

ii) For every training-run, we leave out one subset and train the model on the remaining 6 sets.
iii) We evaluate the model on the left-out subset and repeat the process, but now we leave out another subset.

Using the 3D U-net, we obtain on average the following results:

Dice-loss of $0.162 \pm 0.042$, MA of $91\% \pm 2.4\%$, IU of $87.1\% \pm 2.36\%$ and DC of $83.9\% \pm 4.23\%$.

![Figure 4.1: The 3D U-net: Comparison between the original ground truth (first row) and the segmentation generated by the 3D U-net (second row). Here we see the projections on the $yz$-plane, the $xy$-plane and the $xz$-plane.](image)

Although the 3D U-net demonstrates high precision in our very sparse application (see Figures 4.1 and 4.11), it produces nasty artefacts, which constitutes an essential drawback of this architecture. Additionally, we are very limited in the choice of convolutional layers and the corresponding number of filters due to the huge size of the input data. So it is hardly possible to conduct end-to-end volumetric segmentation for even larger biomedical scans without using cropping or sliding-window techniques.
4 Volumetric Segmentation

4.2 Slice-by-Slice 2D U-Net

As mentioned in the introduction, the naive approach for accelerating volumetric segmentation and reducing memory requirements is to process each of the slice images independently through a 2D network (compare [6]). However, this causes the loss of connections between the slice images.

We take 17 randomly chosen MRA scans and extract their slices to a dataset, which now consists of $96 \cdot 17 = 1632$ images of size $288 \times 224$. The next step is to train the same 2D U-net architecture as described in Section 3 with this dataset. This time we also make use of the Dice-loss function in Equation (3.1). Obtaining a trained model for MRA slices, we choose further 17 volumetric MRA scans and apply the 2D network to each slice independently. Stacking these segmented slices to 3D volumes again yields the following disappointing results:

Dice-loss of $0.216 \pm 0.13$, MA of $87.5\% \pm 8.43\%$, IU of $83\% \pm 7.02\%$ and DC of $78.7\% \pm 12.7\%$. Total training time is 8.58 minutes and application to all slices of one 3D scan lasts in total 0.48 seconds.

As we observe in Figure 4.2, this approach even generates more artefacts. Also the high standard deviation of the metrics between different evaluation batches shows, that this architecture does not generate segmentation masks with consistent accuracy, which is a huge drawback. In terms of the evaluation metrics, the slice-by-slice approach is not able to compete with the 3D U-net. Therefore, we are looking for another efficient 3D segmentation approach.
4 Volumetric Segmentation

Figure 4.2: **The slice-by-slice 2D U-net:** Comparison between the original ground truth (first row) and the segmentation generated by the 3D U-net (second row). Here we see the projections on the $yz$-plane, the $xy$-plane and the $xz$-plane.

4.3 Projection-Based 2.5D U-Net

4.3.1 Proposed 2.5D U-net Architecture

As we have seen in Section 3 for our targeted application the 2D U-net does very well on the MIP images. So instead of processing 96 images of size $288 \times 224$ through a network at once, we aim for a method using some of the MIP images in connection with a learnable reconstruction algorithm.

Recall that a network for binary volumetric segmentation is a function $\mathcal{N} : \mathbb{R}^{a \times b \times c} \rightarrow [0, 1]^{a \times b \times c}$ that maps the 3D scan to the probabilities that a voxel corresponds to
the desired class. For a 3D input \( x \), the proposed 2.5D U-net takes the form

\[
\mathcal{N}(x) = \mathcal{T} \circ \mathcal{R}_{p, \Theta} \circ \mathcal{F}_p \circ \left[ \begin{array}{c}
\mathcal{U} \circ \mathcal{M}_{\theta_1}(x) \\
\vdots \\
\mathcal{U} \circ \mathcal{M}_{\theta_p}(x)
\end{array} \right],
\]

(4.1)

where

- \( \mathcal{M}_{\theta_i} : \mathbb{R}^{a \times b \times c} \rightarrow \mathbb{R}^{b \times c} \) are MIP images for different directions \( \theta_1, \ldots, \theta_p \),
- \( \mathcal{U} : \mathbb{R}^{b \times c} \rightarrow [0, 1]^{b \times c} \) is exactly the same 2D U-net as in Section 3 producing probabilities,
- \( \mathcal{F}_p : ([0, 1]^{b \times c})^p \rightarrow (\mathbb{R}^{b \times c})^p \) is a learnable filtration,
- \( \mathcal{R}_{p, \Theta} : (\mathbb{R}^{b \times c})^p \rightarrow \mathbb{R}^{a \times b \times c} \) is a reconstruction operator using \( p \) linear backprojections for directions \( \theta_1, \ldots, \theta_p \in \Theta \) as illustrated in Figure 4.3 (implementation of \( \mathcal{R}_{p, \Theta} \) can be found in Listing 4.1),
- \( \mathcal{T} : \mathbb{R}^{a \times b \times c} \rightarrow [0, 1]^{a \times b \times c} \) is a fine-tuning operator (average pooling followed by a learnable normalization followed by the sigmoid activation).

![Diagram](image)

**Figure 4.3:** **Reconstruction operator** \( \mathcal{R}_{2,\{0,90\}} \): The voxel value is defined as the sum over the corresponding 2D values, here illustrated for two MIP images.
Listing 4.1: Implementation of the reconstruction operator.

```python
import tensorflow as tf
import keras.backend as K
import numpy as np

def Reconstruction(x, directions, final_shape):
    result = tf.zeros(final_shape)
    x = tf.reshape(x, tf.shape(x)[1:5])
    i = tf.constant(0)
    c1 = lambda res, i: tf.less(i, len(directions))
    b1 = lambda result, i:
        temp = tf.zeros(final_shape)
        temp = tf.add(temp, x[i, :, :, :])
        y = tf.contrib.image.rotate(tf.transpose(temp, perm=[1, 2, 0, 3]), angles=(directions[i] / 180. * np.pi, interpolation='NEAREST'))
        y = tf.transpose(y, perm=[2, 0, 1, 3])
        return (result + y, i + 1)
    result, i = tf.while_loop(c1, b1, [result, i])
    return (K.reshape(result, [1, *final_shape]))
```

The backprojection operator $\mathcal{R}_{p,\theta}$ causes a kind of shadow (Figure 4.4, left), so we have to think about a filtrated backprojection. Therefore, we apply a convolutional layer $\mathcal{F}_p$ before backprojection. Using $1 \times 2$ filters, which get adapted during training for each projection direction $\theta \in \Theta$ individually, leads to a more satisfying result (Figure 4.4, right). The implementation of the filtration layer can be found in Listing 4.2.

![Figure 4.4: Filtrated backprojection:](image)

*Left:* Network’s output (before threshold) using backprojection without filtration.

*Right:* Network’s output (before threshold) using filtrated backprojection.
Listing 4.2: Implementation of the filtration layer.

```python
import tensorflow as tf
import keras.backend as K
from keras.engine.topology import Layer

class Filtration(Layer):
    def __init__(self, kernel, **kwargs):
        self.kernel = kernel
        super(Filtration, self).__init__(**kwargs)
        # p = number of projection directions
        def build(self, input_shape):
            self.w = self.add_weight(name='weights', shape=[p, 1, self.kernel, 1, 1],
                                      initializer='uniform',
                                      trainable=True)
            self.b = self.add_weight(name='bias', shape=[p],
                                      initializer=keras.initializers.Constant(0),
                                      trainable=True)
            super(Filtration, self).build(input_shape)
            def call(self, x):
                weights = self.w
                i = tf.constant(1)
                c1 = lambda i, x: tf.less(i, p)
                x_temp = tf.reshape(x, tf.shape(x)[1:5])
                result = K.conv2d(tf.expand_dims(x_temp[0, :, :, :, :], 0), weights[0, :, :, :, :],
                                  padding='same', data_format='channels_last')
                def b1(i, res):
                    temp = K.conv2d(tf.expand_dims(x_temp[i, :, :, :, :], 0), weights[i, :, :, :, :],
                                    padding='same', data_format='channels_last')
                    result = tf.concat([[result, temp]], axis=0)
                    return (i + 1, result)
                i, result = tf.while_loop(c1, b1, [i, result],
                                           shape_invariants=[i.get_shape(),
                                                             tf.TensorShape([[None, None, None, None, None]]))
                bias = tf.ones(tf.shape(x_temp))
                bias = tf.transpose(bias, perm=[3, 1, 2, 0]) * self.b
                result = tf.transpose(result, perm=[3, 1, 2, 0])
                return (K.reshape(result, tf.shape(x)))

For the fine-tuning operator \( T \) we use average pooling with pool-size \((2, 2, 2)\). This is followed by a learnable normalization, that additionally shifts the pooled data by an adjusted parameter since the decision boundaries have been changed by the operator \( R_{p, \Theta} \). This ensures, that at the end the application of the sigmoid function delivers accurate probabilities.

Looking at Equation (4.1), there exist different approaches for the computation of \( M_{\theta_i} \). Our first attempt will be using MIP images for a certain amount \( \Theta \) of...
deterministic projection angles, i.e. we compute the projections by rotating the 3D data around the vertical axis for deterministic angles $\theta \in \Theta$.

### 4.3.2 Deterministic Projection Directions

For the amount of the projection directions we choose equidistant angles $\Theta = \{ k \times \frac{180}{p} \mid k = 0, \ldots, p - 1 \}$ to ensure that we do not obtain too similar information of the 3D data for different projection directions. This causes the task of finding the best value for $p$ in Equation (4.1). Therefore we train the proposed network $\mathcal{N}$ for different values of $p$ and compare performance in terms of the evaluation metrics (Figure 4.5).

![Figure 4.5: Number of projection directions: Performance of the 2.5D U-net for different number $p$ of projection directions.](image)

Looking at Figure 4.5, we observe that $\Theta = \{ k \times \frac{180}{12} \mid k = 0, \ldots, 11 \}$ seems to be a good choice for the amount of projection directions, since that for higher values only computational costs increase but not the metrics. For our targeted application, again we only process one 3D sample through the network per iteration (minibatch size 1). The start weights of the convolutional part $\mathcal{U}$ in Equation (4.1) are initialized in the same way as in Section 2.2. The parameters of $\mathcal{F}_p$ and $\mathcal{T}$ are initialized empirically.
We notice that our proposed network $N$ sometimes has problems to find the right “direction” to a good local minimum during the first epochs and therefore does not converge. So we change the loss function to help the network during the first few adjustments. We minimize the new joint loss function

$$\tilde{\ell}_c(y_{aux}, y, \hat{y}_{aux}, \hat{y}) \triangleq c \cdot \ell(y_{aux}, \hat{y}_{aux}) + (1 - c) \cdot \ell(y, \hat{y}).$$  \hspace{1cm} (4.2)

Here, $\ell$ is the Dice-loss function in Equation (3.1), $\hat{y} = N(x)$ is the prediction of our proposed network, $y$ is the corresponding 3D ground truth, $\hat{y}_{aux} = (U \circ \mathcal{M}_{\theta_i}(x))_{i=1,...,p}$ are the predictions of the 2D U-net for the MIP images and $y_{aux}$ are the corresponding 2D ground truths.

So we combine the temporary 2D output with the final volumetric output. Since the adjustment of the 2D convolutional layers takes special care of the reconstruction part in Equation (4.1), we have to reduce the balance parameter $c$ after each epoch. So we are able to give the network the right direction at the beginning without losing accuracy at the end of the training process. For parameter $c$ we choose the regularization

$$c(n_{epoch} + 1) = c(n_{epoch}) \cdot 2^{-n_{epoch}},$$

where $n_{epoch}$ denotes the number of the current epoch and $c$ gets initialized with $c(1) = 0.2$.

Again we conduct 7 training-runs using cross-validation and obtain on average the following results:

Dice-loss of $0.161 \pm 0.014$, MA of $91.5\% \pm 1.02\%$, IU of $87\% \pm 1.02\%$ and DC of $84.8\% \pm 1.47\%$. In average, total training lasts 38.5 minutes and the application of the network to a 3D sample scan lasts 0.11 seconds. During training, the 2.5D U-net with 12 deterministic projection directions allocates a memory space of 3.96
gigabytes on the GPU.

![Figure 4.6: The 2.5D U-net with 12 deterministic projection directions: Comparison between the original ground truth (first row) and the segmentation generated by the 2.5D U-net (second row). Here we see the projections on the $yz$-plane, the $xy$-plane and the $xz$-plane.](image)

In terms of the evaluation metrics we already outperform the 3D U-net. Also accuracy over several predictions seems to be more consistent due to the smaller standard deviation for the metrics between evaluation batches. Looking at Figure 4.6 approves, that the proposed 2.5D U-net does not produce any artefacts. Note that these segmentations get produced by a network, which only uses 12 deterministic chosen MIP images of the volumetric input data and still is more reliable than the 3D U-net approach.

However, due to the fact we only use a few deterministic projection directions, we feel that we lose too much information of the volumetric data. The next step will be considering much more projection directions with the aim to further increase accuracy on the targeted application.
4 Volumetric Segmentation

4.3.3 Integration over Regions

On the one hand, we are concerned about information loss choosing MIP images with a projection distance of \( \frac{180^\circ}{12} = 15^\circ \). On the other hand, our proposed structure needs a discretization of the projection directions. Since the difference between two MIP images with projection angles \( \theta \) and \( \theta + 1 \) is marginal, we set \( \Theta = \{-5^\circ, -4^\circ, \ldots, 174^\circ\} \) and construct MIP images for all \( \theta \in \Theta \).

Due to memory issues we are not able to process all \(|\Theta| = 180\) MIP images through the proposed network \( \mathcal{N} \). Because we do not want to give up any information, we sum the MIP images over \( 10^\circ \) (compare Figure 4.7) each and therefore obtain \( p = 18 \) in Equation (4.1).

Summed up, we integrate the MIP images over the projection direction for 10 degrees each and therefore derive a new value for \( p \), which is manageable by our GPU.

Figure 4.7: Summation over projection regions: We construct MIP images for every \( \theta \in \{-5^\circ, -4^\circ, \ldots, 174^\circ\} \) and sum them over regions of \( 10^\circ \).
One may suspect that the network loses some accuracy because the input MIP images of \( \mathcal{U} \) in Equation (4.1) are a little bit blurred due to the summation over different projection directions. Indeed, introducing a convolution with sharpening matrices \( S_i \) after \( \mathcal{M}_{\theta_i} \) for \( i = 1, \ldots, 18 \) increases accuracy significantly.

Partially, for \( i = 1, \ldots, 18 \) and 3D input \( x \), we convolve each integrated MIP image \( \mathcal{M}_{\theta_i}(x) \) with a \( 3 \times 3 \) matrix \( S_i \) to sharpen the image. Here, the values of \( S_i \) are adapted each during training to ensure we obtain for each integrated projection a performance-boosting individual sharpening (compare Listing 4.3).

Listing 4.3: Implementation of the sharpening layer.

```python
import tensorflow as tf
import keras.backend as K
from keras.engine.topology import Layer

class Sharpen(Layer):
    def __init__(self, kernel,**kwargs):
        self.kernel=kernel
        super(Sharpen,self).__init__(**kwargs)
    def build(self,input_shape):
        self.w=self.add_weight(name='weights',shape=[cc,self.kernel,self.kernel,1,1],initializer='uniform',trainable=True)
        self.b=self.add_weight(name='bias',shape=[p],
            initializer=keras.initializers.Constant(0),
            trainable=True)
        super(Sharpen,self).build(input_shape)
    def call(self,x):
        weights=self.w
        i=tf.constant(1)
        cl=lambda i,x: tf.less(i,p)
        x_temp=tf.reshape(x,tf.shape(x)[1:5])
        result=K.conv2d(tf.expand_dims(x_temp[0,:,:,:],0),weights[0,:,:,:],padding='same',data_format='channels_last')
        def bl(i,result):
            y=K.conv2d(tf.expand_dims(x_temp[i,:,:,:],0),weights[i,:,:,:],
                padding='same',data_format='channels_last')
            result=tf.concat([result,y], axis=0)
            return (i+1,result)
        i, result = tf.while_loop(cl,b1,[i, result],
            shape_invariants=[i.get_shape(),tf.TensorShape([None,None,None,None]))
        bias=tf.ones(tf.shape(xx))
        bias=tf.transpose(bias,perm=[3,1,2,0])*self.b
        result=result+tf.transpose(bias,perm=[3,1,2,0])
        return (K.reshape(result,tf.shape(x)))
```

4 Volumetric Segmentation

Since we do not propagate a MIP image through the network for every \( \theta \in \Theta \) but only 18 integrated samples, the reconstruction operator \( R_{18,\tilde{\Theta}} \) only has to conduct 18 linear backprojections for the projection directions \( \tilde{\Theta} = \{ k \times 10^\circ \mid k = 0, \ldots, 17 \} \).

We again conduct 7 training-runs using cross-validation and obtain on average the following results:

Dice-loss of \( 0.179 \pm 0.019 \), MA of \( 91.7\% \pm 0.98\% \), IU of \( 85.5\% \pm 1.28\% \) and DC of \( 82.9\% \pm 1.88\% \). On average, total training lasts 27.1 minutes and the average application time to a 3D sample is 0.17 seconds. During training, a memory space of 5.8 gigabytes gets allocated.

![Figure 4.8: The 2.5D U-net with integration over projection areas](image)

Figur 4.8: The 2.5D U-net with integration over projection areas: Comparison between the original ground truth (first row) and the segmentation generated by the 2.5D U-net (second row). Here we see the projections on the \( yz \)-plane, the \( xy \)-plane and the \( xz \)-plane.

Obviously we do not improve accuracy in terms of the evaluation metrics. Hence we come up with a third approach, using a random selection of MIP images during the training process.
4.3.4 Random 2.5D U-net

Now we set $\Theta = \{k \times \frac{180}{m} \mid k = 0, \ldots, m - 1\}$ and construct MIP images for all $\theta \in \Theta$, where $m$ denotes the number of projections we want to use for the volumetric reconstruction. Due to memory issues we are not able to update the proposed model $\mathcal{N}$ using all $|\Theta| = m$ projection images if $m$ is large. But since our goal is to use information from as many directions as possible, we propose the following random approach using a weighted joint loss function:

The proposed model consists of two paths and therefore produces two outputs:

i) **Path 1:**
We consider for $k = 0, \ldots, p - 1$, $p \ll m$, the set $\Theta_k \triangleq \{k, k+1, \ldots, k+\frac{180}{p} - 1\}$ and choose for each epoch one angle $\hat{\theta}_k \in \Theta_k$ uniformly at random. For that $p$ projection directions, we generate the corresponding MIP images out of the 3D input $x$ and process them through the network $\mathcal{N}$, that has the same structure as in Equation (4.1). The first output $\hat{y}_{aux}$ is then the learned reconstruction of the $p$ randomly chosen MIP images. Note, that especially the 2D convolutional part $\mathcal{U}$ in Equation (4.1) gets also trained with these random projections.

ii) **Path 2:**
Now we generate for all $m$ projection directions in $\Theta$ the corresponding MIP images out of the same 3D input $x$. We take the 2D convolutional network $\mathcal{U}$ out of the first part and apply it to the $m$ projection images. Note, that $\mathcal{U}$ is only needed for evaluation and we do not update parameters of $\mathcal{U}$ in this path (Figure 4.9). Given these $m$ predictions $\mathcal{U}(M_{\theta_1}(x)), \ldots, \mathcal{U}(M_{\theta_m}(x))$, we are able to train a filtration layer $\mathcal{F}_m$ and a new fine-tuning operator $\mathcal{T}$. These are the only layers which get trained in this path. The final output of the second path is then computed by

$$\hat{y} = \mathcal{T} \circ \mathcal{R}_{m,\Theta} \circ \mathcal{F}_m \circ \left[\mathcal{U}(M_{\theta_1}(x)), \ldots, \mathcal{U}(M_{\theta_m}(x))\right].$$
Note that this output contains information of all $m$ projection directions.

Therefore, the final segmentation will be generated by applying a threshold of 0.5 to $\hat{y}$.

In conclusion, although we only use $p$ random directions for adjustment of the 2D convolutional part in $\mathcal{N}$ (first path), we are able with the help of $\mathcal{U}$ to construct simultaneously volumetric segmentation masks using available information from all $m$ projection directions $\theta \in \Theta$ (second path) with $m \gg p$. A huge advantage of this technique is precision:

The random structure only annotates a voxel as vessel, if there are enough of these $m$ projections, which also annotate the pixel of the corresponding 2D images as vessel. So the random 2.5D U-net has a more reliable decision boundary than the architectures presented before.

According to Section 4.3.2, $p = 12$ in the first path seems to be an appropriate choice. For the number $m$ of projection directions in the second path, we choose $m = 60$, i.e. $\Theta = \{k \times 3 \mid k = 0, \ldots, 59\}$, to hold down computational complexity.

Since the proposed random architecture generates two outputs, we make use of the following joint Dice-loss function:

$$\tilde{\ell}_c(y, \hat{y}; \hat{y}_{aux}) = c \cdot \ell(y, \hat{y}_{aux}) + (1 - c) \cdot \ell(y, \hat{y}),$$

where $y$ is the ground truth related to input tensor $x$, $\hat{y}_{aux}$ is the model’s first output (path 1), $\hat{y}$ is the model’s second output (path 2) and therefore the final segmentation prediction, and $\ell$ is the Dice-loss function in Equation (3.1). For balance parameter $c$, we choose the following regularization:

$$c(n_{epoch} + 1) = c(n_{epoch}) \cdot 0.99^{n_{epoch}},$$

where $n_{epoch}$ denotes the number of the current epoch and $c$ gets initialized with
4 Volumetric Segmentation

c(1) = 0.99. Therefore, the network has main focus on the optimization of the 2.5D U-net in path 1 during the first few epochs, and gradually takes more care of the reconstruction process in path 2 (Figure 4.9).

We again conduct 7 training-runs using cross-validation and obtain on average the following results:

Dice-loss of 0.148 ± 0.019, MA of 92.7% ± 1.21%, IU of 87.6% ± 1.34% and DC of 85.6% ± 1.91%. Average total training time of the random 2.5D U-net is 58.7 minutes on GPU and it allocates a memory space of 6.23 gigabyte. Application
time increases to 0.35 seconds.

Figure 4.10: The random 2.5D U-net: Comparison between the original ground truth (first row) and the segmentation generated by the random 2.5D U-net (second row). Here we see the projections on the $yz$-plane, the $xy$-plane and the $xz$-plane.

Considering evaluation results, the random approach outperforms all prior discussed models. Furthermore, this proposed architecture hardly produces artefacts (Figure 4.10).

4.4 Numerical Results

The evaluation results of all proposed networks in this chapter on our targeted application are summarized in Table 4.1. Also the amount of weights, the average total training time, the average application duration on a test sample and the need of memory during training are displayed in Table 4.2. Visual comparison between the hand segmentation of a 3D MRA scan and the generated segmentations by random 2.5D U-net, by 2.5D U-net with 12 deterministic projection directions, by 2.5D U-net with integration over projection areas and the 3D U-net is given in Figure 4.11.
4 Volumetric Segmentation

Table 4.1: **Summarization** of the evaluation results of all **3D networks** proposed in this thesis.

<table>
<thead>
<tr>
<th>Network</th>
<th>loss</th>
<th>MA %</th>
<th>IU %</th>
<th>DC %</th>
</tr>
</thead>
<tbody>
<tr>
<td>slice-by-slice 2D U-net</td>
<td>0.216 ± 0.130</td>
<td>87.5 ± 8.43</td>
<td>83.0 ± 7.02</td>
<td>78.7 ± 12.7</td>
</tr>
<tr>
<td>3D U-net</td>
<td>0.162 ± 0.042</td>
<td>91.0 ± 2.40</td>
<td>87.1 ± 2.36</td>
<td>83.9 ± 4.23</td>
</tr>
<tr>
<td>2.5D U-net (p = 12)</td>
<td>0.161 ± 0.014</td>
<td>91.5 ± 1.02</td>
<td>87.0 ± 1.02</td>
<td>84.8 ± 1.47</td>
</tr>
<tr>
<td>2.5D U-net (integration)</td>
<td>0.179 ± 0.019</td>
<td>91.7 ± 0.98</td>
<td>85.5 ± 1.28</td>
<td>82.9 ± 1.88</td>
</tr>
<tr>
<td>random 2.5D U-net</td>
<td>0.148 ± 0.019</td>
<td>92.7 ± 1.21</td>
<td>87.6 ± 1.34</td>
<td>85.6 ± 1.91</td>
</tr>
</tbody>
</table>

Table 4.2: **Summarization** of time and storage observations of all **3D networks** proposed in this thesis.

<table>
<thead>
<tr>
<th>Network</th>
<th>Weights</th>
<th>Tot. training</th>
<th>Application</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>slice-by-slice 2D U-net</td>
<td>8.6 × 10⁶</td>
<td>8.58 min.</td>
<td>0.48 sec.</td>
<td>1.70 gb</td>
</tr>
<tr>
<td>3D U-net</td>
<td>9.8 × 10⁴</td>
<td>33.9 min.</td>
<td>0.25 sec.</td>
<td>4.87 gb</td>
</tr>
<tr>
<td>2.5D U-net (p = 12)</td>
<td>8.6 × 10⁶</td>
<td>38.5 min.</td>
<td>0.11 sec.</td>
<td>3.96 gb</td>
</tr>
<tr>
<td>2.5D U-net (integration)</td>
<td>8.6 × 10⁶</td>
<td>27.1 min.</td>
<td>0.17 sec.</td>
<td>5.80 gb</td>
</tr>
<tr>
<td>random 2.5D U-net</td>
<td>8.6 × 10⁶</td>
<td>58.7 min.</td>
<td>0.35 sec.</td>
<td>6.23 gb</td>
</tr>
</tbody>
</table>
4 Volumetric Segmentation

Figure 4.11: Visual comparison between the proposed networks for volumetric segmentation:

From left to right: Ground truth, random 2.5D U-net, 2.5D U-net with 12 deterministic projection directions, 2.5D U-net with integration over projection areas and 3D U-net. Images were constructed with the freeware ITK-SNAP [22].
5 Conclusion

In this thesis we investigated the development of fully convolutional networks \[16\] up to the 2D U-net architecture \[18\], which provides state-of-the-art generation of 2D segmentation masks for various applications, especially for the biomedical sector. We also applied all presented network structures to 2D MIP images generated out of volumetric MRA scans. An import result is, that for very sparse segmentation tasks like the annotation of blood vessels in 2D MIP images, the choice of the loss function is very crucial. Therefore we introduced the Dice-loss function \[17\] based on the similarity measure Dice-coefficient.

When the focus is on the generation of volumetric ground truths for big 3D data, there are some essential challenges to deal with. Although the construction of volumetric segmentation masks with the help of a 3D U-net \[6, 10, 17\] delivers very satisfying results, generation of artefacts and the restrictions to deepness and number of filters due to memory issues are hardly sustainable. Therefore we proposed the 2.5D U-net structure. Although there are only projection images used for a few deterministic directions, this architecture is able to conduct volumetric segmentation of very big biomedical 3D scans even more reliable than the 3D U-net. Furthermore, the proposed network hardly produces artefacts and can be trained without any concern about input size.

To increase accuracy, we also investigated the integration of projection images over deterministic projection areas. As a result, the 2.5D U-net could not improve in
5 Conclusion

terms of evaluation metrics although it used more available information of the 3D data.

Finally, choosing a random approach increased accuracy on our sparse binary segmentation task. For our targeted application, the random 2.5D U-net enables the generation of 3D segmentations in a storage efficient way more accurate than approaches using 3D convolutions.

Note that the proposed network structure can be used for different projection techniques. For blood vessel segmentation, maximum intensity projection seemed to be the most fitting approach due the high sparsity of the data and the orientation of the vessels. For volumetric segmentation of other biomedical scans displaying organs or cells, other projection types can be used, e.g. a distance-weighted maximum intensity projection, the Radon-transform \cite{7} or combinations.

Further tasks will be to investigate whether applying data augmentation techniques to the 3D samples increases accuracy of the 3D U-net. Considering our application, data deformation could cause problems due to the fact, that the orientation of the vessels has a huge impact to the network’s prediction. In future work, we will investigate the use of the random 2.5D U-net architecture for other projection types and applications, e.g. for volumetric image reconstruction.
Bibliography


URL: http://tensorflow.org/


Bibliography


Bibliography


