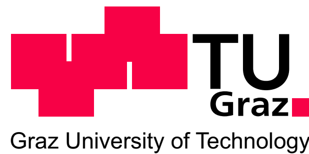


Christina Gsaxner, BSc

Automatic urinary bladder segmentation in CT images using deep learning

MASTER'S THESIS
to achieve the university degree of
Diplom-Ingenieurin
Master's degree programme: Biomedical Engineering

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg

Institute of Computer Graphics and Vision
8010 Graz, Inffeldgasse 16/II

Advisor

Dr.rer.physiol. Dr.rer.nat. Jan Egger

Graz, October 2017

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present masters thesis.

________________	____________________________
Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

________________	____________________________
Datum	Unterschrift

Abstract

We present an approach for fully automatic urinary bladder segmentation in CT images with artificial neural networks in this thesis. Automatic medical image analysis has become an invaluable tool in the different treatment stages of diseases. Especially medical image segmentation plays a vital role, since segmentation is often the initial step in an image analysis pipeline. Since deep neural networks have made a large impact on the field of image processing in the past years we use two different deep learning architectures to segment the urinary bladder. Both of these architectures are based on pre-trained classification networks that are adapted to perform semantic segmentation. Since deep neural networks require a large amount of training data, specifically images and corresponding ground truth labels, we furthermore propose a method to generate such a suitable training data set from Positron Emission Tomography/Computed Tomography image data. This is done by applying thresholding to the Positron Emission Tomography data for obtaining a ground truth and by utilizing data augmentation to enlarge the dataset. In this thesis, we discuss the influence of data augmentation on the segmentation results, and compare and evaluate the proposed architectures in terms of qualitative and quantitative segmentation performance. The results presented in this thesis allow concluding that deep neural networks can be considered a promising approach to segment the urinary bladder in CT images.

Keywords: Image Segmentation, Deep Learning, Convolutional Neural Networks, Urinary Bladder, Computed Tomography

Kurzfassung

In dieser Arbeit wird eine Methode zur vollautomatischen Harnblasensegmentierung in Computertomographiebildern mit künstlichen neuronalen Netzen vorgestellt. Die automatisierte Analyse von medizinischen Bilddaten hat sich als wertvolles Instrument in verschiedensten Behandlungsstadien von Krankheiten etabliert. Vor allem die Segmentierung von medizinischen Bildern spielt dabei eine wichtige Rolle, da sie oft den ersten Schritt in einer Bildanalysepipeline darstellt. Da tiefe, neuronale Netze in den letzten Jahren sehr erfolgreich im Bereich der Bildverarbeitung angewandt wurden, werden auch in dieser Arbeit "Deep Learning" Algorithmen zur Harnblasensegmentierung verwendet. Genauer werden zwei unterschiedliche Architekturen solcher Netze vorgestellt, die auf vortrainierten Klassifizierungsnetzwerken beruhen, welche für semantische Bildsegmentierung adaptiert werden. Da solche Netze eine große Anzahl an Daten benötigen, um von ihnen zu lernen, wird in dieser Arbeit ebenfalls ein Ansatz zur Generierung eines solchen Datensatzes aus Positronenemmissionstomopgraphie/Computertomographie-Bilddaten vorgestellt. Dabei wird durch ein Schwellenwertverfahren, welches auf die Positronenemmissionstomographie Bilder angewandt wird, eine Referenzsegmentierung gewonnen. Zusätzlich soll durch Augmentation der Bilder der Datensatz vergrößert werden. Der Einfluss von augmentierten Daten auf das Ergebnis der Segmentierung wird in der Arbeit diskutiert. Auerdem werden die vorgestellten Netzwerkarchitekturen bezüglich ihrer qualitativen und quantitativen Segmentierungsergebnissen ausgewertet und verglichen. Die Arbeit kommt zu dem Schluss, dass tiefe neuronale Netze einen vielversprechenden Ansatz zur Segmentierung der Harnblase in CT Bildern darstellen.

Schlüsselwörter: Bildsegmentierung, Deep Learning, Convolutional Neural Network, Harnblase, Computertomographie

Acknowledgements

I would like to start by thanking my supervisor from the Graz University of Technology, Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg, for giving me the opportunity to carry out this work in his department and for providing the resources and environment to complete this thesis. Furthermore, I owe my gratitude to my advisor, Dr.rer.physiol. Dr.rer.nat. Jan Egger, for his guidance, advice and enthusiasm throughout the course of my work. He not only shared his expertise in the field of medical imaging and image processing, but also gave me insight into research and scientific working in general. His support opened opportunities I'm sure I will benefit from throughout my career. I also want to express my thanks to Dr. Dr. Jürgen Wallner from the Medical University of Graz for providing me with medical insight in the topics covered in this thesis.

This acknowledgement would not be complete without thanking my family, especially my parents, for their unconditional understanding and support, for always believing in me and for giving me the opportunity to pursue a master's degree. I'm also grateful for my friends, who always make me laugh and provided diversion whenever university got too stressful. Thanks to them, I will always look back at my years at university with pleasure. Lastly, I am particular grateful to my boyfriend Andreas, who has been by my side throughout my entire student time with all it's ups and downs, and was always there for me.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Objective . . . 2

2 Medical Background **3**
2.1 The Urinary Bladder and Prostate . . . 3
2.1.1 Anatomy and Functionality . . . 3
2.1.2 Prostate Cancer . . . 5
2.1.3 Urinary Bladder Cancer . . . 6
2.1.4 Other relevant Pathologies . . . 7
2.2 Medical Imaging Techniques . . . 7
2.2.1 Computed Tomography . . . 7
2.2.2 Positron Emission Tomography (PET) . . . 9
2.2.3 Combined PET/CT . . . 10

3 Technical Background **12**
3.1 Artificial Neural Networks . . . 12
3.1.1 The Perceptron . . . 12
3.1.2 Multilayer Neural Networks . . . 13
3.1.3 Activation Functions . . . 14
3.1.4 Training a Neural Network . . . 16
3.1.5 Regularization . . . 18
3.1.6 Convolutional Neural Networks (CNNs) . . . 19
3.2 Medical Image Segmentation . . . 22
3.2.1 Common Segmentation Methods in Medical Imaging . 23
3.2.2 Deep Learning Artificial Neural Networks for Image Segmentation . . . 25
3.3 MeVisLab . . . 26
3.3.1 Visual Programming . . . 27
3.3.2 Creation of Macro Modules . . . 27
3.4 TensorFlow . . . 29
3.4.1 Tensors . . . 29
3.4.2 Computation Graphs and Sessions . . . 29
3.4.3 Training with Tensorflow . . . 30
3.4.4 TensorBoard . . . 32
3.4.5 TensorFlow-Slim . . . 34

4 Related Work **35**
4.1 Automatic Urinary Bladder Segmentation 35
4.2 Neural Networks for Image Classification and Segmentation . . 36
4.2.1 VGG Classification Network 36
4.2.2 ResNet Classification Network 37
4.2.3 Fully Convolutional Networks for Semantic Segmentation (FCNs) 39
4.2.4 Atrous Convolution for Semantic Segmentation 41

5 Methods **43**
5.1 Dataset and Preprocessing 43
5.2 Generation of Image Data 44
5.2.1 The DataPreperation Macro Module 46
5.3 Image Segmentation using Deep Neural Networks 52
5.3.1 Working with TFRecords files 53
5.3.2 Creating TFRecords files for Training and Testing . . . 54
5.3.3 Segmentation Networks using TF-Slim and pre-trained Classification Networks 54
5.3.4 Bilinear Upsampling 58
5.3.5 Training the Segmentation Networks 58
5.3.6 Testing the Segmentation Networks 61
5.3.7 Evaluation Metrics 62

6 Results and Evaluation **65**
6.1 Generation of Training and Testing Data 65
6.2 Training and Testing 67
6.3 Image Segmentation Results 68

7 Discussion **72**
7.1 Conclusion and Future Outlook 76

A Dataset Overview **86**

B Loss Development during Training **87**

C Seperate Segmentation Results **88**

List of Figures

1 Location of the urinary bladder in the male and female pelvis 3
2 Anatomy of the male urinary bladder and prostate in frontal and sagital view . . . 4
3 Measuring principle of CT . . . 8
4 Detection Principle of PET . . . 9
5 3D image data obtained from CT, PET and combined PET/CT of the torso and part of the head . . . 11
6 An artificial neuron . . . 13
7 A multilayer perceptron . . . 15
8 Non-linear, sigmoidal activation functions . . . 15
9 Graphical interpretation of an error function in two dimensions 17
10 Error backpropagation . . . 19
11 Convolution and pooling . . . 20
12 Rectified Linear Unit (ReLU) activation function . . . 21
13 Convolutional neural network with two convolution and pooling layers . . . 22
14 Graphical interpretation of thresholding and classification . . . 25
15 Basic module types and module connectors of MeVisLab . . . 27
16 Example network in MeVisLab . . . 28
17 Visualization of the development of sum-of-squares error during training a linear regression model in TensorBoard . . . 33
18 Graph visualization in TensorBoard . . . 33
19 VGG 16 architecture . . . 36
20 Comparison between ResNet building blocks . . . 38
21 Basic ResNet architecture with 34 layers . . . 38
22 Transition from image classification to image segmentation in CNNs . . . 39
23 FCN architectures . . . 40
24 Atrous convolution . . . 41
25 General Network for loading, processing and visualizing PET and CT data, implemented in MeVisLab . . . 46
26 The internal network of the DataPreperation macro module . 47
27 Panel of the DataPreperation macro module . . . 49
28 Network Graphs for FCN and upsampled ResNet visualized with TensorBoard . . . 57
29 Comparison between normal convolution and transposed convolution . . . 59
30 Graphical interpretation of Hausdorff distance . . . 64

31 Examples of overlays between CT data and generated ground truth labels 65
32 Example for an augmented dataset 66
33 Histograms of evaluation metrics 69
34 Qualitative segmentation result overlays for images scaled to 256×256 70
35 Qualitative segmentation result overlays for images with resolution 512×512 71
36 Development of the cross entropy loss during training 87
37 Segmentation results for images scaled to 256×256. 88
38 Segmentation results for images with original resolution of 512×512. 89

List of Tables

1 Parameters for data augmentation 50
2 Created TFRecords files for training and testing 55
3 Comparison of training and inference times 67
4 Segmentation evaluation results for images rescaled to 256×256 68
5 Segmentation evaluation results for images of resolution 512×512 68
6 Used datasets obtained from RIDER PET/CT 86

1 Introduction

1.1 Motivation

Since imaging modalities like computed tomography (CT) are widely used in diagnostics, clinical studies and, treatment planning and evaluation, automatic algorithms for image analysis have become an invaluable tool in medicine. Image segmentation algorithms are of special interest, since segmentation plays a vital role in various medical applications [1]. Typically, segmentation is the first step in a medical image analysis pipeline and therefore incorrect segmentation affects any subsequent steps heavily. However, automatic medical image segmentation is known to be one of the more complex problems in image analysis [2]. Therefore, to this day delinearation is often done manually or semi-manually, especially in regions with limited contrast and for organs or tissues with large variations in geometry. This is a tedious task, since it is time consuming and requires a lot of empirical knowledge. Furthermore, the process of manual segmentation is prone to errors and since it is highly operator dependent, not reproducible, which emphazises the need for accurate, automatic algorithms. One up-to-date method for automatic image segmentation is the usage of deep neural networks. In the past years, deep learning approaches have made a large impact in the field of image processing and analysis in general, outperforming the state of the art in many visual recognition tasks, e.g. in [3]. Artificial neural networks have also been applied successfully to medical image processing tasks such as segmentation. Therefore, in this thesis we propose an approach to automatic urinary bladder segmentation in CT images using deep learning.

Currently there are two main applications for segmentation of the urinary bladder. In clinical practice, it is used in radiation treatment planning. The delineation of organs at risk and target tissue is an important step in planning radiation therapy. The urinary bladder is considered such an organ at risk that should be protected against high doses of radiation in e.g. treatment of prostate cancer, which is the second most common cancer in men worldwide and the most common cancer in Europe for men [4], [5]. Furthermore, segmentation of the urinary bladder is a key step in computer-aided detection of urinary track abnormalities, such as bladder cancer, since the segmented bladder defines the search region for further detection steps. Bladder cancer currently ranks fourth in the most common cancers in men [6]. Additional applications include the measurement of parameters such as bladder wall thickness or bladder volume, which are critical indexes for many bladder-related conditions [7].

1.2 Objective

The goal of this thesis is to examine, implement and compare deep neural network models for semantic segmentation of medical image data, specifically CT scans of the urinary bladder. Deep Neural Networks usually require a large amount of labelled training data to specify all connections in the network. This is often problematic when working with medical image data. Compared to general image databases, which frequently offer millions of labelled entries, open medical image databases are generally small, often consisting of only one or a couple of patient datasets. Furthermore, for the segmentation task at hand a ground truth, e.g. images of the already segmented urinary bladder, are required as labels. Since segmentation is such a time consuming task, large medical image databases that contain already segmented images for a specific task are basically impossible to find. Therefore, a method for generating suitable training and testing data needs to be found.

In short, there are two main objectives pursued in this thesis:

1. Create a suitable dataset for training and testing artificial neural networks from an open medical image database;
2. Use the generated data to train and test different deep neural network architectures for semantic segmentation.

2 Medical Background

This chapter provides medical background information about the topics discussed in this thesis. In the first section, an outline of the anatomy and functionality of the urinary bladder and prostate is given. Furthermore, important pathologies of these organs, like bladder and prostate cancer, and how their detection and treatment could benefit from automatic urinary bladder segmentation are presented to provide further insight in the motivation behind this thesis. Section 2.2 explains relevant medical imaging techniques. Since medical data for this thesis was obtained from combined Positron Emission Tomography/Computed Tomography (PET/CT), both modalities are first described individually before the concept behind combined PET/CT is outlined. The purpose of this section is to clarify the characteristics as well as advantages of disadvantages of these techniques.

2.1 The Urinary Bladder and Prostate

2.1.1 Anatomy and Functionality

The urinary bladder is a hollow smooth muscle organ that is located at the pelvic floor. It acts as a receptacle for urine and it's capacity is approximately 350-500 ml. In males, it lies below the peritoneal cavity, between the pubis and the rectum, in females, the vagina and the uterus intervenes between. Figure 1 shows sagittal sections through the male and female pelvis to illustrate the position of the urinary bladder.

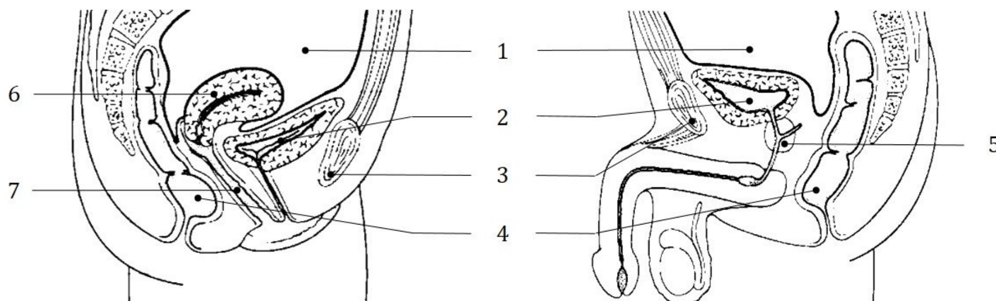


Figure 1: Location of the urinary bladder in the male and female pelvis. 1: peritoneal cavity, 2: urinary bladder, 3: pubic bone, 4: rectum, 5: prostate, 6: uterus, 7: vagina. Adapted from [8].

The anatomy of the urinary bladder is outlined in figure 2. The bladder is often described to have a pyramidal shape. The apex of the pyramid

points forward and forms a fibrous cord, the median umbilical ligament. The posterior surface of the bladder is called the base or fundus and contains the trigone, which is named after its triangular shape. Urine is transported into the urinary bladder through the ureters, which enter the trigone through two orifices. It exits the bladder through the urethra, which is directly connected to the neck of the urinary bladder. The muscle coat of the bladder is called detrusor muscle and is a composite of interlacing, disorganized muscle fibres. At the bladder neck, the detrusor is thickened to form the internal urethral sphincter. In the male, the prostate gland lies between the internal and external sphincter, in the female, the external sphincter lies just below the bladder neck. The inner walls are coated with a thick mucous membrane, that are thrown into folds when the bladder is empty and allow for the expansion of the bladder. This membrane is lined with transitional cells making up the urothelium or uroepithelium, which is a tissue type highly specific to the urinary tract [9] [8].

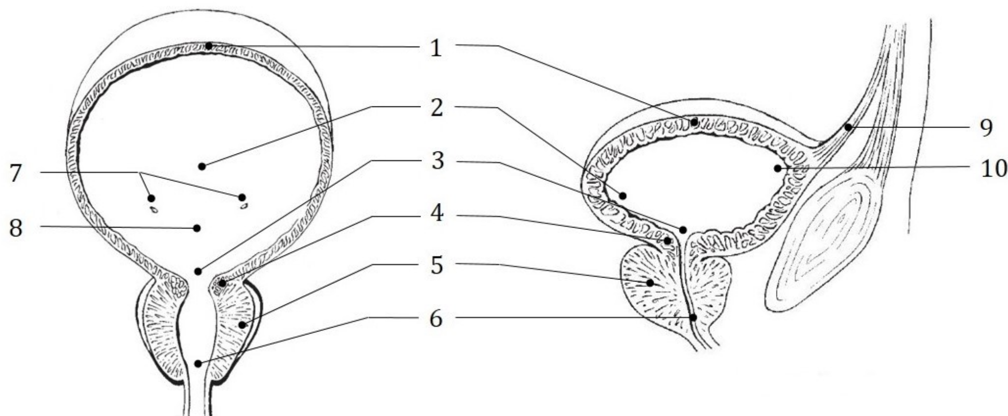


Figure 2: Anatomy of the male urinary bladder and prostate in frontal and sagital view. 1: detrusor muscle, 2: base or fundus of the bladder, 3: bladder neck, 4: internal urethral sphincter, 5: prostate, 6: urethra, 7: uretal orifices, 8: trigone of the bladder, 9: median umbilical ligament, 10: apex of the bladder. Adapted from [10].

The prostate is a compound tubuloalveolar exocrine gland of the male reproductive system. It surrounds the urethra and lies between the bladder neck and the urogenital diaphragm. In healthy individuals, it has approximately the size of a walnut. It comprises multiple lobes, that contain glands producing a secretion that is added to the seminal fluid [8].

2.1.2 Prostate Cancer

Prostate cancer is the most common cancer in northern and western European males, and the second most common cancer in men worldwide. In the European Union, 345.000 new cases were estimated in 2012, which accounted for 24% of all new cancers in this year. It is especially common in more developed countries, with about 68% of cases occurring in Europe, North America and Australia and New Zealand. Much of these variations in incidence rates might stem from differences in screenings, with regions with higher screening density having significantly higher incidence rates. However, the detected tumors are often clinically insignificant [11], [4].

The majority of prostate cancers (95%) are adenocarcinomas. Adenocarcinomas are cancerous tumours that occur in tissue that has glandular origin or characteristics. For diagnosis, there are three major tools: prostate-specific antigen screening of blood serum, digital rectal examination and transrectal ultrasonography. The tumor is staged and graded based on values obtained from these examinations, then treated accordingly [11].

External beam radiation therapy plays a critical part in the treatment of both localized and advanced diseases. It uses high energy radiation, like x-rays or gamma rays, to kill cancer cells, an therefore shrink tumours. It effectively damages the DNA of cells, causing them to stop dividing or to die. However, the radiation also damages normal cells. Modern 3D-conformal radiotherapy (3D-CRT) systems are able to apply large target doses, while excluding adjacent normal tissue from the high dose region. Through segmentation of organs in the target region, toxicities in the rectum and bladder can be minimized while treating prostate cancer. Intensity modulated radiotherapy (IMRT) poses a further technological advancement that allows improved coverage of the clinical target volume while minimizing the volume of bladder and rectal tissue exposed to high radiation doses. In 3D-CRT and IMRT, 3D image data obtained by computed tomography is used to segment tumours and normal tissue. However, the delineation is mostly done manually and comes with uncertainties. Therefore, margins extending into normal tissue are usually added to the planning target volume to decrease the rist of missing tumour cells. Obviously, the inclusion of healthy tissue limits the applicable radiation dose. Hence, accurate, reliable segmentation of tumours and organs at risk is crucial for effective radiation therapy [11], [12].

2.1.3 Urinary Bladder Cancer

Cancer in the urinary bladder is the ninth most common cancer in the world, with 430,000 new cases diagnosed in 2012. Men are four times more likely to get bladder cancer than women, with bladder cancer being the fourth most common type of cancer in men. As prostate cancer, it is more common in developed countries, with highest incidence in Northern America and Europe. Cancer occurance is often related to tobacco smoking [4]. Over 90% of bladder cancers are transitional cell carcinoma. It arises from the transitional cells making up the urothelium.

Medical imaging modalities, especially computed tomography, play a big role in detecting and staging bladder cancer. CT scans of the kidney, ureters and bladder is known as CT urogram. In these scans, radiologists can detect tumours in the urinary tract and gain detailed information, like their size, shape and position. However, the interpretation of CT urograms is tedious and time consuming, as each individual slice has to be evaluated for lesions. Furthermore, the process leads to a substantial variability between radiologists in the detection of cancer, and there is also the chance of missing small lesions due to the large workload. Computer aided detection (CAD) might aid radiologists in finding lesions in the bladder. The first step in a CAD system is to define a search region for further detection, specifically to segment the urinary bladder. By excluding non-bladder structures for the search process, the possibility of false positive detections is decreased. Therefore, accurate bladder segmentation is a critical component in the computer aided detection of bladder cancer [6].

In the treatment of urinary bladder cancer, radiation therapy, again, plays a critical part. However, since the urinary bladder is an organ which shows significant variations in size and position between patients and even within patients between individual therapy sessions, which limits the allowed radiation dose and results in large amounts of healthy tissue receiving the same radiation dose. Therefore, a technique called adaptive ratiotherapy is used to re-optimize the plan during treatment to account for deformations of the target. Usually, a cone beam CT is taken before every treatment to select the optimal plan for the day. For quick and accurate plan selection, automatic segmentation of the urinary bladder in these CT scans is desirable [13].

2.1.4 Other relevant Pathologies

Segmentation of the urinary bladder can also aid in the measurement of parameters such as bladder wall thickness or bladder volume. These parameters are critical indexes for many bladder-related issues [7]. For example, bladder wall thickness can be a useful parameter in the evaluation of benign prostatic hyperplasia, a non-cancerous enlargement of the prostate [14] and has been shown to be a useful predictor for bladder outlet obstruction and destrusor overactivity [15]. Furthermore, focal bladder wall thickening is a sign for bladder cancer. Measuring bladder volume can be useful to look for urinary retention.

2.2 Medical Imaging Techniques

2.2.1 Computed Tomography

Computed Tomography (CT) is an X-ray imaging modality. It uses the same principles of generation, interaction and detection as conventional X-ray, however, the generation of a sliced view of the body is enabled through computed reconstruction of X-ray attenuation inside the patient.

The basic measurement principle behind computed tomography relies on the rectilinear propagation and attenuation of X-rays through the patient. X-rays are generated by a X-ray tube and measured by an opposing detector array. The human body is inhomogeneous, consisting of regions with varying compositions and densities. Consequently, X-ray attenuation differs for each region. To image the inside of the human body, the spatial distribution of attenuation coefficients is reconstructed in CT imaging. For this reconstruction, various projections of the object are taken. Each projection consists of multiple beams, often in a fan-shape, and is characterized by it's projection angle. By backprojecting these projections taken at various angles all around the object, a sliced view can be obtained. The measurement principle of CT is outlined in figure 3.

The spatial distribution of the attenuation coeficients is measured in Hounsfield units (HU)

$$I_{i,j} = 1000 \left(\frac{\mu(x_i, y_i)}{\mu_w} - 1 \right) HU \tag{1}$$

where $I_{i,j}$ is the resulting pixel value in the image matrix, $\mu(x_i, y_i)$ is the attenuation coefficient on the corresponding position, and μ_w is adjusted so as to give water a pixel value of zero. A normal CT contains Hounsfield values

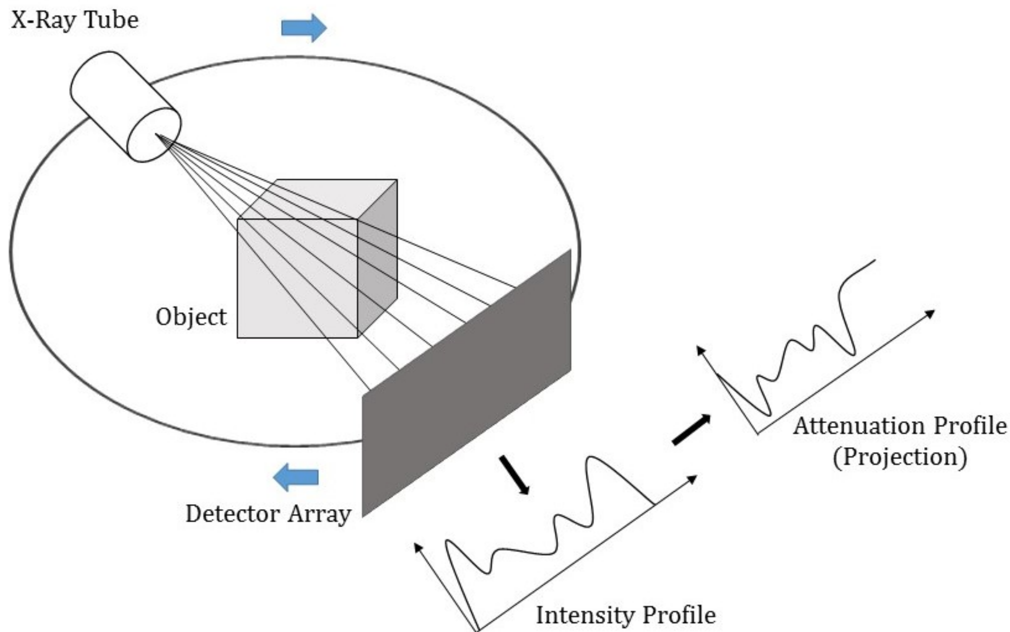


Figure 3: Measuring principle of CT. The object is radiographed with a fan-shaped x-ray beam. X-rays are attenuated inside the object and a detector array measures the remaining intensity. The intensity profile is then converted into an attenuation profile, also called projection. This is repeated for multiple angles as x-ray tube and detector array rotate around the object.

between -1024 HU (Air) and 3071 HU (Bone). Theoretically, the Hounsfield scale can be extended to even higher or lower values, but restricting the grey values to a range of 4096 allows 12 Bit representation. Since soft tissue contains mainly water, it usually ranges between -100 HU (tissue with fat) and 100 HU (blood clot). However, only 30-40 gray values can be discriminated by the human eye. To obtain high contrast, windowing has to be applied to display gray values that cover the structures and tissues of interest. Even with windowing, the contrast for soft tissue is still limited in CT scans [16].

A limitation of CT usage is it's high radiation dose, often delivering more than a hundred times the radiation dose of conventional X-ray scans. Alternative methods like magnetic resonance imaging (MRI) do not use any ionizing radiation, while offering comparable, if not better, image quality. Still, utilization of computed tomography has increased over the past several decades [17]. CT systems are widely available, much cheaper than MRI systems and scanning times are short. Furthermore, CT scanners are still the

best modality to image bone. That is why CT scanning is still as relevant as ever.

2.2.2 Positron Emission Tomography (PET)

The following chapter is based on my master's project report in [18].

Positron Emission Tomography (PET) imaging is based on measuring radiation emitted by a so called radiotracer injected into the patient. Naturally occuring biologically active molecules, like glucose, water or ammonia, are labelled with positron-emitting radioisotopes with short half-lives such as ^{11}C, ^{15}O and ^{18}F. The so formed compounds are called radiopharmaceuticals or radiotracers. The organism can not distinguish them from their non-radioactive pendants and therefore, radiotracers partake in the normal metabolism. This allows non-invasive imaging of functional and metabolical processes.

Radiotracers are chosen to accumulate in regions relevant for specific screening, like inflammatory sites or tumour cells. Although many radiotracers have been developed, the most commonly used is fluorine-18-labelled fluorodeoxyglucose (^{18}F-FDG) [19]. Metabolically active lesions show a higher glucose metabolism than surrounding regions. For example, the high rate of cell division in cancer or the immune response to infections requires glucose. FDG molecules act like glucose during their initial reactions within cells, but their altered structure prevents further metabolism, which causes 18F-marked glucose to accumulate in these areas [20]. A downside of using ^{18}F-FDG is, that normal uptake of FDG occurs in all sites of the body which may cause confusion in interpreting PET images. Such a physiological uptake of FDG usually appears in the brain, heart, active skeletal muscle and other areas with naturally high glucose levels and consumption [21]. Furthermore, FDG is not, like glucose, reabsorbed in the proximal tubules of the kidney, which leads to accumulation of the radioactive trace in the urine.

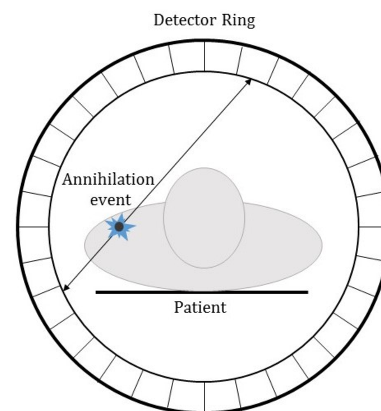


Figure 4: Detection Principle of PET. A detector ring array measures detection coincidences and the point of annihilation is determined along a straight line between these detector elements

This causes high FDG activity in the urinary bladder, even if the patient empties his bladder before the scan [22].

The basic principle behind PET systems is the detection of annihilation gamma rays following the decay of positrons. Positrons emitted by a radiotracer interact with electrons in the tissue, resulting in annihilation of the particles into a pair of 511-keV gamma photons that are emitted at approximately 180 relative to each other. These photon rays are detected by a detector array surrounding the patient. If a coincidence between two opposing detector elements is registered within a short time period (usually a couple of nanoseconds), the point of the annihilation event can be localized along a straight line of coincidence between the detector elements [23]. An illustration of this detection principle can be seen in figure 4. While PET images usually offer a good contrast, spatial resolution is rather low. Positrons travel some distance in the subject before annihilation, regions of high radiotracer uptake are blurred. Other factors to consider are the intrinsic spatial resolution of the detector and noncolinearity (deviations from the 180 emission angle). Therefore, spatial resolution in PET images is limited to 2-6 mm [24].

PET scanners usually measure the in vivo radioactivity concentration in [kBq/ml], which is directly linked to the radiotracer concentration in the tissue. However, there are many factors besides the tissue uptake of the tracer influencing this measure. The most significant sources of variation are the amount of injected radiotracer and the patient size. Therefore, the standardized uptake value (SUV) is commonly used as a measure for tracer uptake. The basic expression for SUV is

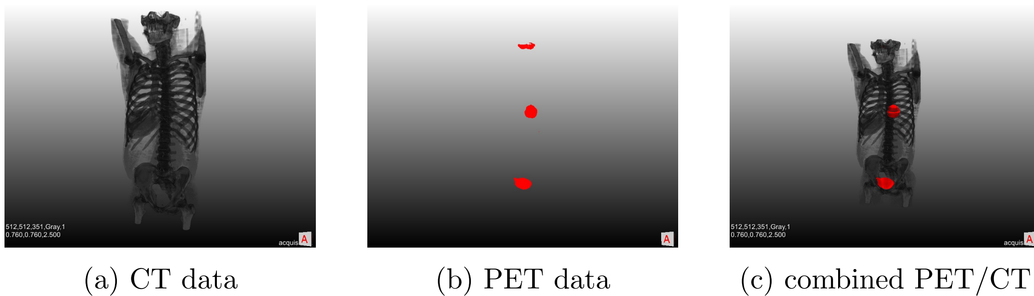
$$SUV = \frac{r}{(a'/w)} \tag{2}$$

where r is the radioactivity concentration a' is the amount of injected radiotracer and w is the weight of the patient [25].

2.2.3 Combined PET/CT

A combined Positron emission tomography/Computed Tomography (PET/CT) system unites the functionality of PET and CT into a single device with a shared operating system. While functional imaging with PET provides information about metabolic activities inside a patient, anatomical context is not obtained, which makes the task of anatomical registration difficult. By adding CT to PET, patients can be scanned with both modalities at the same time without moving the patient, which allows for the correct anatomical localization and quantification of tracer uptake. It also provides additional

diagnostic information, like accurate tumour size. Furthermore, CT data is used for attenuation correction of PET emission data. From the viewpoint of a radiologist, malignancies show up with higher sensitivity, since metabolical changes already show up where morphological change is still very little. In effect, the addition of PET to CT provides a metabolic contrast agent. Figure 5 highlights the advantage of a combined PET/CT scan for the three-dimensional case.



(a) CT data (b) PET data (c) combined PET/CT

Figure 5: 3D image data obtained from CT, PET and combined PET/CT of the torso and part of the head. While CT data in (a) shows important anatomical structures, the contrast for soft tissue, in example in the abdominal region, is poor. PET data in (b) only shows metabolical active regions, without providing anatomical context, making it impossible to accurately localize lesions. In the co-registered PET/CT scan in (c), it is possible to properly assign active regions anatomically. The urinary bladder, a lesion in the left lung and parts of the brain are highlighted via the PET data.

Most applications of PET/CT exams are related to oncology, since PET/CT systems enable a much better differentiation between healthy tissue and tumour tissue than single PET or CT scans. Especially in areas with high anatomical structure densitiy, like the head, neck and abdomen, this is very useful. Furthermore, PET/CT studies can help with the planning of biopsies, interventional procedures and radiation therapy [16], [23].

3 Technical Background

In this chapter, technical background information about subjects relevant to this thesis is given. The first section focuses on the principles of artificial neural networks. Basic concepts like neurons, layers and activation functions are explained. Then, this section goes into greater detail about how neural networks are trained by the minimization of an error function and error back-propagation. Lastly, basic principles of convolutional neural networks, like convolutional layers and pooling layers, as well as their importance in image processing, are presented. In the second section, some popular concepts in medical image segmentation are explained. Furthermore, this section provides and overview of image segmentation with deep learning artificial neural networks. It's purpose is to show how the deep learning approach proposed in this thesis could improve upon other state of the art methods. The last two sections give an introduction to two important software tools used for this thesis to give a better understanding about the presented methods. The medical imaging framework MeVisLab, as discussed in section 3.3, was used for the generation of training and testing data. Implementation of deep learning neural networks, their training and evaluation was performed using the machine learning software library TensorFlow, which is described in section 3.4.

3.1 Artificial Neural Networks

In machine learning, artificial intelligence (AI) systems acquire their own knowledge from raw data, rather than using information input by a human user. One algorithmic approach to this problem is the usage of artificial neural networks (ANNs), which are inspired by the human brain. The brain is a complex network made up of a large number of simple elements (neurons) that are connected via synapses to form complex networks that are able to process complex high-level information. Contrary to biological neural networks, where neurons can connect to any other neuron, ANNs consist of discrete layers with specific connections and directions of information propagation [26], [27].

3.1.1 The Perceptron

The basic building block of ANNs are artificial neurons, often referred to as nodes. An outline of its function can be seen in figure 6. It processes inputs with a set of three rules. First, the inputs x_i are multiplied with individual weights w_i, then, the weighted inputs are summed up (sometimes a bias b

is added). Lastly, the summed weighted inputs are passed to a transfer or activation function $f(.)$ that determines the output y of the node:

$$y(\boldsymbol{x}, \boldsymbol{\theta}) = f\left(\sum_{i=1}^{D} x_i w_i + b\right) \tag{3}$$

where D denotes the dimension of the input, $\boldsymbol{x}$ is a vectorized representation of inputs and θ is a vector concatenated from connection weights and biases.

The activation function is chosen depending on the problem the artificial neuron should solve. It is usually a step function, linear function or non-linear function [28]. Single-layer networks with only one output are commonly referred to as perceptrons. This simple model can be easily extended to multiple outputs (i.e. for multi-class classification) by adding multiple output units and their respective connection weights [27].

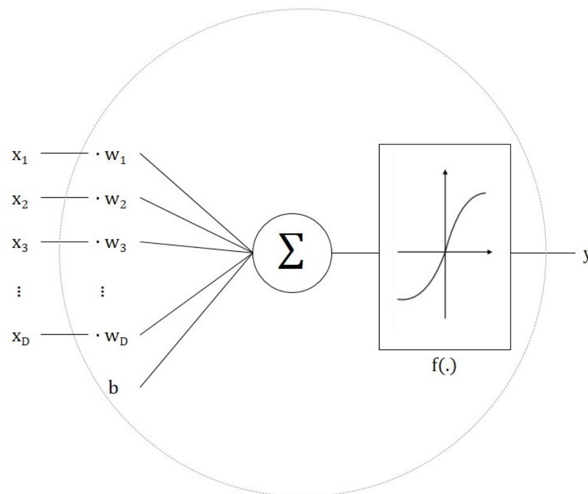


Figure 6: An artificial neuron. The inputs x_i are multiplied with the weights w_i and summed up. A bias b might also be added. The result is transformed by an activation function $f(.)$ that determines the output y. Adapted from [28].

3.1.2 Multilayer Neural Networks

Perceptrons are limited to linear separation tasks for which simpler techniques are more practical. The full potential of artificial neurons is exhausted

when so-called hidden layers are introduced between the input and output layer. While the design of the input and output layer are usually considered fixed (the number of input nodes is dependent on the dimensions of the input and the number of output nodes determines the number of classes), the amount of hidden layers and artificial neurons in each layer is dependent on application. Hidden layers have to be designed to be able to model all useful patterns in the input, while not over-fitting the data. Each hidden layer consists of several nodes that are connected to the nodes of the next layer in a specific way.

The most simple architecture of a multilayer network is the multilayer perceptron. It consists of one hidden layer and each layer is fully connected to the next one. A graphical representation can be seen in figure 7. Its composition function can be written as:

$$y(\boldsymbol{x}, \boldsymbol{\theta}) = f^{(2)}\left(\sum_{j=1}^{M} w_{kj}^{(2)} f^{(1)}\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b^{(1)}\right) + b^{(2)}\right) \tag{4}$$

where M is the number of nodes in the hidden layer and the subscript denotes the layer number. Note that the input layer is usually not counted, making the multilayer perceptron a 2-layer neural network. A network like the multi-layer perceptron in figure 7 is referred to as feed-forward neural network (FNN). FNNs propagate information from input to output in only one direction. The other main group of networks are recurrent neural networks (RNNs) that allow for information to flow in the opposite direction.

3.1.3 Activation Functions

Usually, the same type of non-linear, sigmoidal activation function is applied to the hidden layers. The most commonly used functions are either the logistic sigmoid function

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

or the hyperbolic tangent function

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{6}$$

A plot of these functions can be observed in figure 8. The major difference between these two activations is the range of output values. While the logistic sigmoid function results in values between $[0, 1]$, the hyperbolic tangent

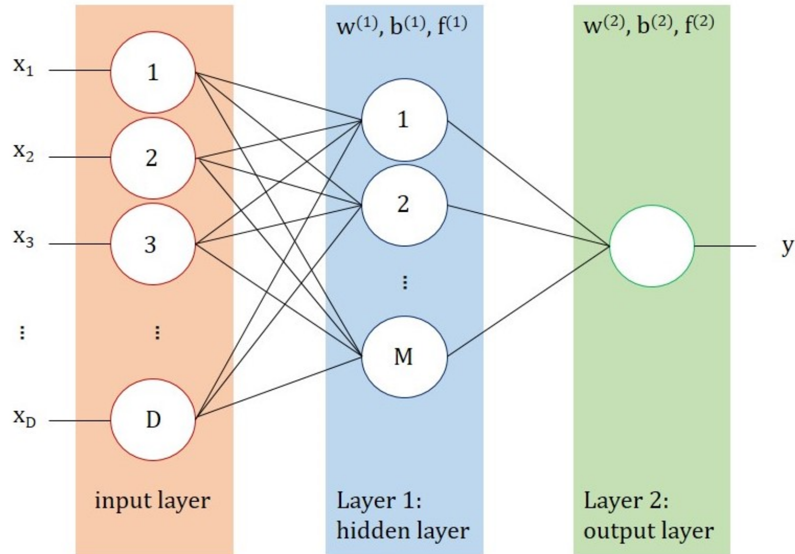
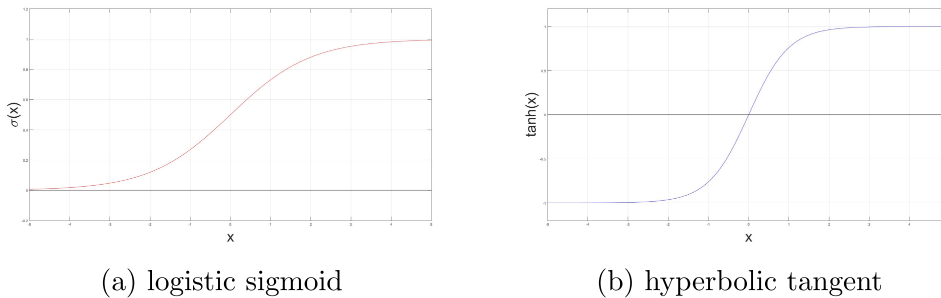


Figure 7: A multilayer perceptron. It has one hidden layer and all layers are fully connected, meaning that each node of a layer connects with all nodes of the next layer.

function yields values in the range of $[-1, 1]$. Because of its zero-centered output, tanh activation functions are usually preferred [27].



(a) logistic sigmoid (b) hyperbolic tangent

Figure 8: Non-linear, sigmoidal activation functions. The logistic sigmoid in (a) maps input values to a range of $[0, 1]$, the hyperbolic tangent function in (b) to values between $[-1, 1]$.

For output layers, the activation function depends on the problem being solved. While for binary classification, sigmoidal output funcitons as seen in equations 5 and 6 are appropriate, linear activation functions are chosen for

regression problems. For multi-class problems, a softmax activation function of the form

$$p(C_k|\boldsymbol{x}) = \frac{e^{a_k}}{\sum_j e^{a_j}} \tag{7}$$

with

$$a_k = \ln p(x|C_k)p(C_k) \tag{8}$$

is commonly used. Here, $p(C_k|\boldsymbol{x})$ is the posterior probability for class C_k, $p(x|C_k)$ are the class-conditional densities and $p(C_k)$ are the class priors. The softmax function is the generalization of the logistic sigmoid to $K > 2$ classes. Its name comes from the fact that it represents a smoothed version of the *max* function [29]. The outputs of a softmax funciton is the probability of an input $\boldsymbol{x}$ belonging in a class C_k, which are values between 0 and 1 that add up to 1. Therefore, the output of a softmax layer can be seen as a probability distribution.

3.1.4 Training a Neural Network

The fundamental problem in learning a neural network is the determination of network parameters. The connection weights an biases of each node together make up the parameter vector $\boldsymbol{\theta}$, which defines the overall behaviour of the network. Identification of the optimal parameters for a given problem can be formulated as the minimization of an error function $E(\boldsymbol{\theta})$.

The error function is a measure of discrepancy between the desired output and the actual output of the model. Its choice depends, similar to that of the activation function, on the type of problem being solved. For regression, a sum-of-squares error function

$$E(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}(y(x_i, \boldsymbol{\theta}) - t_i)^2 \tag{9}$$

where N is the number of observations $x_1, ..., x_N$ and $t_1, ..., t_N$ are the corresponding target values, is used. A cross-entropy function is used for binary classification. It has the form

$$E(\boldsymbol{\theta}) = -\sum_{i=1}^{N}(t_i \ln(y_i)(1 + t_i)\ln(1 - y_i)) \tag{10}$$

where y_i denotes $y(x_i, \boldsymbol{\theta})$. The cross-entropy error function can be generalized to multi-class problems with K classes using

$$E(\boldsymbol{\theta}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} t_{ki} \ln y_k(x_i, \boldsymbol{\theta}) \tag{11}$$

[29].

The error function can be viewed as a surface sitting over the parameter space defined by the vector $\boldsymbol{\theta}$, as seen in figure 9. It is a smooth and continuous, but also a non-linear, non-convex function. Therefore, the parameter set $\boldsymbol{\theta}$ that minimizes the function can not be computed analytically, because the function possesses local minima besides the global minimum [27].

Therefore, computation of the minimizing parameter vector is usually done iteratively, recalculating the parameter vector in each step:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} + \Delta\boldsymbol{\theta}^{(\tau)} \tag{12}$$

τ is the current iteration step and $\Delta\boldsymbol{\theta}^{(\tau)}$ is the update of the parameter vector in this step.

The most common way to calculate the update is by using gradient information:

$$\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \eta\nabla E(\boldsymbol{\theta}^{(\tau)}) \tag{13}$$

η is called learning rate. This procedure is called gradient descent algorithm. The gradient of the error function $\nabla E(\boldsymbol{\theta})$ always points into the direction of greatest rate of increase of the function. By moving in the opposite direction, the error is therefore reduced. Since $E(\boldsymbol{\theta})$ is a smooth, continuous function, a vector $\boldsymbol{\theta}$ for which the gradient vanishes exists [27], [29]. The learning rate η is used to control the length of each step taken in the current direction. This prevents over-correction of the current variables, which could lead to divergence. So to speak, the learning rate determines how fast a network changes established parameters for new ones. Often a decay function is used for the learning rate, so big steps are taken in the beginning of the algorithm, then learning rate is

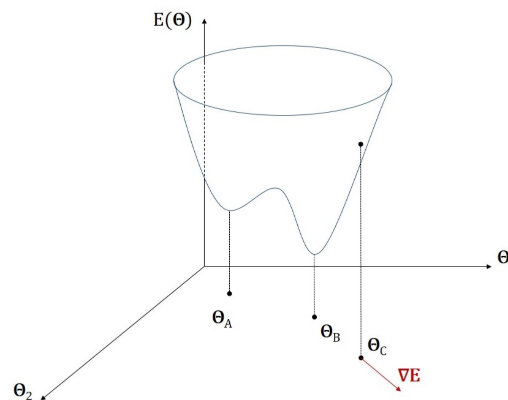


Figure 9: Graphical interpretation of an error function $E(\theta)$ in two dimensions. It can be viewed as a surface over the parameter space defined by the vector $\boldsymbol{\theta}$. Point $\boldsymbol{\theta}_A$ shows a local minimum, $\boldsymbol{\theta}_B$ is the global minimum. The local gradient of the error function is given by the vector $\nabla\boldsymbol{E}$ and can be calculated in every point, like in $\boldsymbol{\theta}_C$. Adapted from [29].

gradually decreased to make smaller steps.

There are two different approaches for timing the update of the parameter vector. In batch gradient descent, the parameters are updated based on the gradients ∇E evaluated over the whole training set. In stochastic gradient descent, the gradient is calculated for one sample in each iteration step and the parameters are updated accordingly. In deep networks, stochastic gradient descent is more commonly used [30].

The gradient of the error function for a network with L layers is given by

$$\nabla E(\boldsymbol{\Theta}) = \left[\frac{\partial E}{\partial \boldsymbol{\Theta}^{(1)}} \cdots \frac{\partial E}{\partial \boldsymbol{\Theta}^{(l)}} \cdots \frac{\partial E}{\partial \boldsymbol{\Theta}^{(L)}} \right], \tag{14}$$

where the superscript denotes the layer index. For the gradient descent algorithm, this gradient needs to be calculated in every update step. To compute this efficiently, error backpropagation is usually used. The idea behind error backpropagation is to propagate the resulting error from the output layer back to the input layer [31]. For this, a so-called error message $\delta^{(l)}$ is calculated for every layer $1, ..., l, ..., L$. The error message of each node can be seen as a measure for the contribution of said node to the output error. Since the states and desired outputs for hidden layers are not known, error terms can not be calculated but have to be estimated by propagating the error messages backwards through the network. Layer l receives an error message $\delta^{(l+1)}$ from layer $l+1$ and updates it using

$$\delta^{(l)} = f'(\boldsymbol{z}^{(l)}) \cdot \left[(\boldsymbol{\Theta}^{(l+1)})^{\intercal} \delta^{(l+1)} \right] \tag{15}$$

where $\boldsymbol{z}^{(l)}$ is the input vector of layer l and $f'(.)$ is the inverse of the activation function, and passes it on to layer $l-1$. Furthermore, the activation of layer l $\boldsymbol{a}^{(l)}$ is used to calculate the gradient of the error function with respect to the parameters of the current layer:

$$\frac{\partial E(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}^{(l)}} = \delta^{(l)} \boldsymbol{a}^{(l)}. \tag{16}$$

This is repeated for every layer [27]. The basic concepts of error backpropagation are illustrated in figure 10.

3.1.5 Regularization

As already mentioned, the number of hidden units in an ANN is a parameter which significantly influences the performance of the network, since it also

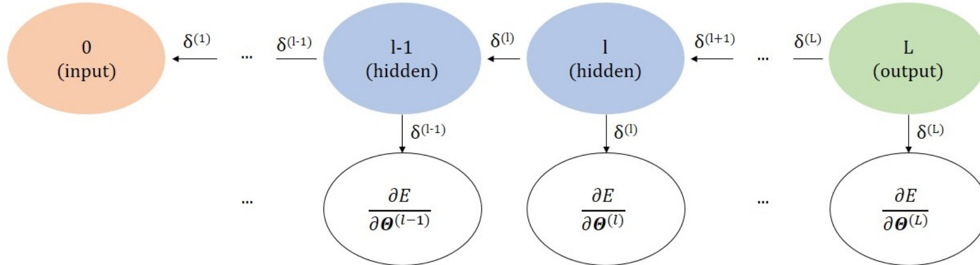


Figure 10: Error backpropagation. The error message δ is propagated from output to input and updated in every layer. In each layer, the partial derivatives with respect to the parameters $\mathbf{\Theta}^{(l)}$ are calculated using the error message. Together, these partial derivatives make up the gradient of the error function ∇E.

controls the number of trainable parameters. It has to be chosen to find a balance between over- and underfitting. If the number of hidden layers and parameters is to small, the model won't work accurately. If it is too big, it will lose its ability to generalize beyond the training data and become too specialized [29].

The most common way to avoid overfitting is regularization. For this approach, a regularization term is added to the error function. This is often a quadratic term, resulting in a regularized error function of the form

$$\tilde{E}(\mathbf{\Theta}) = E(\mathbf{\Theta}) + \frac{\lambda}{2}\mathbf{\Theta}^\intercal\mathbf{\Theta}. \tag{17}$$

λ is called the regularization coefficient and can be used to model the complexity of the resulting network. By minimizing this regularized error function, one encourages the network parameters (weights and biases) to adopt small values, therefore the regularization term is often referred to as weight decay [32].

3.1.6 Convolutional Neural Networks (CNNs)

In the multi-layer networks discussed in the previous sections, inputs were in vector form and network layers were fully connected. However, this is not practical for image data. Each pixel in the input image counts as one input dimension. If each of these inputs were connected to a hidden layer with a few 100 hidden units, an image of size 200 $\times$200 would already result in

several 40,000 weights per neuron and over 4,000,000 weights for the whole layer. Training all these weights would require a large amount of training data and memory. Furthermore, a lot of information is gained from the topology of the input, in example local correlations among neighbouring pixels. This information is destroyed when vectorizing image data. Therefore, convolutional layers are introduced into the network architecture. These layers force the network to extract local features by restricting the receptive fields of hidden units to a certain neighbourhood of the input. Such features could be oriented edges, end-points or corners [33].

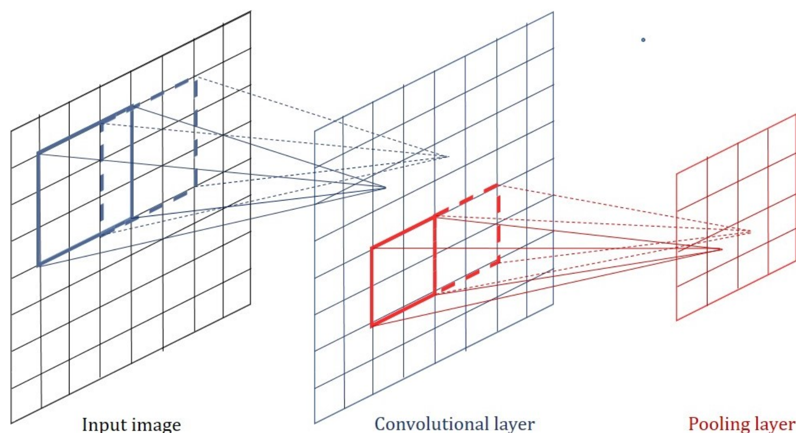


Figure 11: Convolution and pooling. An 8×8 image is convolved with a 3×3 filter kernel to make up a feature map. For a stride of 1 and with zero padding, the feature map has the same dimensions as the input. The feature map is then pooled using a receptive field of size 2× and a stride of 2, resulting in a 4×4 feature map. By using several different filter kernels, three-dimensional feature maps are obtained. Adapted from [29].

Each unit of a convolutional layer receives inputs from a small neighbourhood of units in the previous layer. This neighbourhood is also called receptive field. Sets of neurons whose receptive fields are located at different parts of the image are grouped together to have identical weight vectors ("weight sharing"). As an output, these sets produce so-called feature maps. A convolutional layer consists of several such unit sets, therefore, the output of these layers is a three dimensional volume where width and height are dependent on width and height of the input, and the depth is equivalent to the number of neuron sets in the layer. This process can also be understood as convolving the input with several different filter kernels, each kernel contributing one

feature map to the output. The filters are defined by their values, given by the weights of the network, and their size, defined by the size of the receptive fields. Besides the number of filters, there are two more parameters that influence the size of the feature map. Firstly, the stride is the number of pixels by which the kernel slides over the input image in each step. A larger stride results in reduced width and height of the feature map, a stride of 1 would leave widht and height unchanged. Secondly, zero-padding is sometimes applied to the borders of the image to allow for the application of the filter to bordering pixels. Since in convolutional layers many neurons share the same weights, the number of parameters to train is greatly reduced. Furthermore, if the input image is shifted, the feature maps will shift in the same way but remain otherwise unchanged. This makes the network invariant to small shifts [33], [29].

Since convolution is a linear operation, but the data a CNN should learn is mostly non-linear, non-linearity must be introduced via a suitable activation function. Convolutional layers can use non linear activation functions such as the hyperbolic tangent or logistic sigmoid, but another type of activation function has been found to perform better, namely the Rectified Linear Unit (ReLU) function

$$f(x) = ReLU(x) = max(0, x), \quad (18)$$

which can be seen in figure 12.

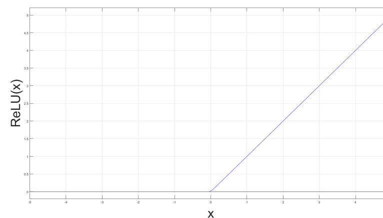


Figure 12: Rectified Linear Unit (ReLU) activation function. It is equal to thresholding the values at zero, setting all negative values to zero.

Subsequently, another new type of layer is introduced, the pooling layer. Its purpose is to downsample an input feature map by reducing the width and height of each map, but leaving the depth unchanged. Similar to the convolution layer, each unit of the pooling layer is connected to a receptive field of units from the previous layer. There are several variants for pooling, such as max pooling where the maximal value inside a receptive field is taken, or average or mean pooling, where the average/mean value is calculated. Pooling has the purpose of decreasing the feature dimensions and therefore, further reduce the number of parameters (weights, biases) of the network. Furthermore, it makes the network invariant to small scaling. Image 11 shows an illustration of convolving and pooling layers applied to an input image [27].

In a convolutional neural networks, several convolutional and pooling layers can be applied successively. The deeper the network becomes (the more hidden layers it has), the better the ability of the network to extract useful features. The outputs of such a network are high level, low dimensional features of the input that finally have to be classified. Fully connected layers with appropriate activation functions, as discussed in the previous sections, are usually used for this task. A typical neural network architecture can be seen in figure 13

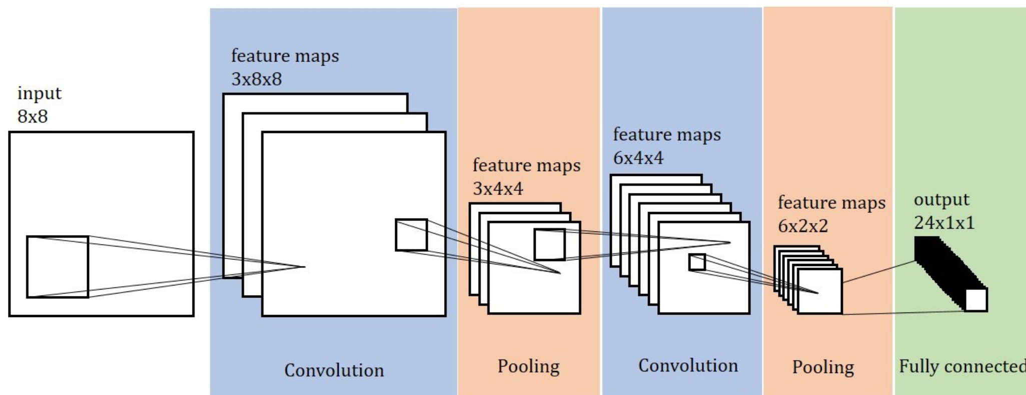


Figure 13: Convolutional neural network with two convolution and pooling layers. The input image of dimension 8×8 is convolved with three kernels to produce a 3×8×8 feature map. Width and height are reduced in the first pooling layer to the dimension 3×4×4. The feature map is then convolved and pooled for a second time in the same manner. The output layer is fully connected and produces a vector of 24×1×1. Adapted from [33].

3.2 Medical Image Segmentation

Segmentation is the process of subdividing an image in its constituent regions or objects. For non-trivial, natural images, automatic segmentation is one of the most challenging tasks in image processing. Since segmentation is often the first step in computerized image analysis algorithms, it's accuracy is determining the success of subsequent steps. Therefore, segmentation algorithms must be very precise. Generally, there are two basic approaches to automatic image segmentation:

1. Discontinuity-based approaches aim to find sharp, local changes of intensities in images, such as edges, and partition the image based on those discontinuities.

2. Similarity-based approaches partition an image into regions that are similar according to a pre-defined criterion. Intensity is a criterion that is most often used, but other characteristics like colour or texture might be used as well [34].

In medical image processing, segmentation plays an important role in many applications, such as the quantification of tissue volumes, the study of anatomical structures, the localization of pathologies, treatment planning (especially radiotherapy planning), computer-integrated surgery and computer aided diagnosis.

3.2.1 Common Segmentation Methods in Medical Imaging

Medical image data is very hard to segment automatically due to its complexity. Anatomical structures usually have a large variability in shape and location, and medical images are prone to artefacts and noise, furthermore their resolution is often limited. Therefore, in clinical practise, segmentation is mostly done manually or semi-manually. Manual segmentation is usually performed by one or more physicians who delineate regions of interest slice by slice using simple drawing tools. This process obviously has many shortcomings: First of all, the workload for individual physicians is huge since delineation is very time consuming, which in turn leads to errors. Furthermore, results are not reproducible. This, as well as the growing size and number of medical image data, has led to the increasing importance of automatic segmentation algorithms for the delineation of anatomical structures or other regions of interest.
The following section provides an overview over common medical image segmentation methods found in recent literature and is adapted from [1], which the reader might refer to for in-depth information.

- **Thresholding**
  Thresholding based methods create a binary partitioning of the image based on intensity values. In the thresholding procedure, the intensity histogram of an image is usually used to find an intensity value that best separates the desired classes from one another. This value is called the threshold. An example for this can be seen in figure 14(a). Subsequently, all pixels with intensity values below the threshold are grouped to one class (frequently labelled background) and the remaining pixels belong to the object. Thresholding is widely used due to its simplicity, however, it requires structures to have distinct contrasting intensities. Furthermore, it does not take spatial information into account, which makes it sensitive to noise and intensity inhomogeneities

which are often present in medical image data. Therefore, thresholding is seldom used alone. It is, however, often used as an initial step for further processing operations.

- **Region Growing**
  Region growing methods, in their simplest form, usually start from a seed point inside a region of interest, and then progressively include neighbouring pixels which satisfy a predefined similarity criterion. Usually, a fixed interval around a certain intensity is chosen as a criterion, which makes this approach similar to thresholding, but with the consideration of spatial information. Region growing is also most commonly used within a more complex segmentation pipeline. One disadvantage of region growing is that, in general, user input in form of a seed point is required for every region to extract.

- **Classifiers and Clustering**
  Classifier algorithms assign a certain label to an image or an image patch. For this, the classifier must be trained using image data with known labels. Usually, multidimensional features, which are abstract, reduced representations of images (i.e. edge directions), are calculated. Then the classifier is chosen in a way that best separates the feature space into the desired categories. This can be seen in figure 14(b). The classifier can then be used to decide in which category a new image or image patch belongs. A downside of classifiers is, that they use hand crafted features. This means that the user must decide which type of features best represent the information he wants to extract to obtain the best results.
  Clustering algorithms are very similar in their functionality to classifier methods, but they do not require training data. These algorithms group similar instances together, based on previously extracted features. Clustering methods, however, can't assign predefined labels to these groups.

- **Deformable Models**
  Deformable models use parametric curves or surfaces that are deformed by internal or external influences. For this, an initial model is placed near the boundary one wants to delineate. Then external influences drive the model towards the desired boundary, while internal constraints ensure that the model stays smooth.

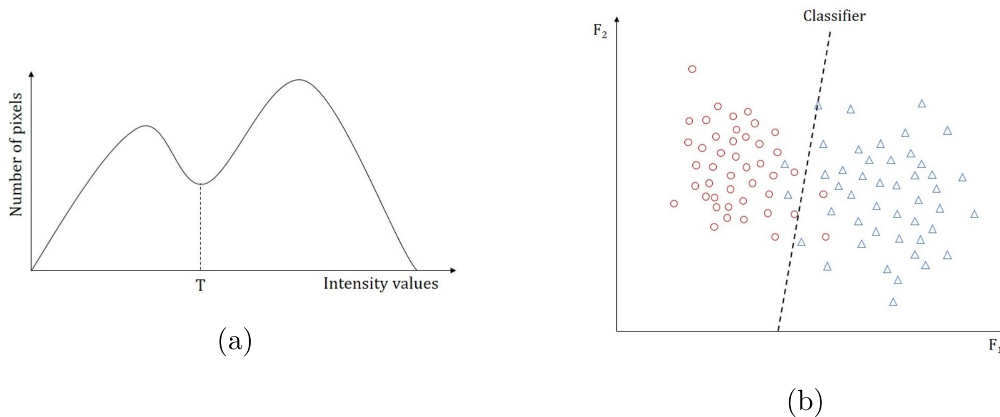


Figure 14: Graphical interpretation of thresholding and classification. (a) Finding a threshold based on the image histogram. The threshold T is chosen in a way that best separates the intensity values in an image. (b) Seperating a 2D-feature space with a linear classifier. First, features with known labels (circles, triangles) are seperated. This classifier can then be used to assign unlabelled objects to one of the two classes.

3.2.2 Deep Learning Artificial Neural Networks for Image Segmentation

Deep learning algorithms using ANNs have made a large impact on the field of automatic image analysis in the past years, outperforming state of the art methods in many visual recognition tasks. They possess various advantages over the established methods discussed before. Deep learning approaches don't require user interaction, like a seed point in region growing or a initial model like with deformable models. Furthermore, contrary to classifiers or clustering, no hand crafted features are needed. Deep ANNs learn multiple layers of representation. Input data can be represented in many ways, but certain representations are better suited to learn a task of interest (e.g. classification). Deep learning algorithms attempt to find the best representation of images by learning complex relationships amongst the input data. Usually, many layers of non-linear data processing are used in the course of this [35].

Convolutional neural networks are commonly used for classification tasks, where input data (e.g. an image) is assigned to a single class label. However, in many recognition tasks, especially in biomedical imaging, localization should be considered in the output, i.e. a label prediction should be made at every pixel. The CNNs as discussed in chapter 3.1 are not applicable on semantic segmentation since they produce low dimensional, non spacial

feature maps which don't allow pixel-wise labelling. An early idea to transition from classification to segmentation was to classify each pixel based on a patch around that pixel [36]. However, this approach is rather slow and has a trade-off between localization accuracy and context depending on the patch size [37]. A better solution to this problem is the application of fully convolutional neural networks (FCNs) as proposed by Long et al. in [38]. These networks take an arbitrary sized input and produce a segmented output of corresponding size. Contrary to traditional classification networks, FCNs use only convolution and pooling layers combined with non-linear activation functions, and no fully connected layers. FCNs will be discussed in greater detail in section 4.2. A further development in semantic segmentation using CNNs is the Encoder-Decoder Convolutional Neural Network (ED-CNN), like SegNet in [39] or U-Net in [37]. In an ED-CNN, the contracting path (encoder) is followed by a expansive path (decoder) that performs upsampling in many steps, using the stored pooling indices. This way, context information is propagated to higher resolution layers. Both paths are more or less symmetric, yielding a u- or v-shaped architecture. Higher level features from the encoder path can be combined with upsampled outputs from the decoder path to produce a more precise result.

3.3 MeVisLab

The following section follows my masters' project report in [18] and is adapted from the MeVisLab Getting Started Guide [40], the MeVisLab Reference Manual [41] as well as [42] and [43].

MeVisLab is a modular framework for the development of medical image processing algorithms and visualization of medical data. The framework offers a wide range of features, from basic image processing algorithms like filters and transformations, to more advanced medical imaging modules, in example for segmentation or registration. Development can be done on three levels. A visual level using graphical programming enables the creation of image processing networks via the inclusion of processing, visualization and interaction modules. On a scripting level, macro modules can be created using Python scripting to control interactions between network modules, the graphical user interface (GUI) and internal parameters. Lastly, new modules can be programmed and integrated using the C++ class library. Furthermore, the MeVisLab Definition Language (MDL) allows the design of GUIs. Visual programming and setting up macro modules will be explained in more detail in the following sections. Since module programming in C++ was not part of this project, no further information will be provided about it.

3.3.1 Visual Programming

Visual programming using modules integrated in the MeVisLab framework and connections between modules is the most basic form of implementing algorithms in MeVisLab. Three basic module types distinguished by their colors, are available. They are shown in figure 15. Blue modules represent algorithms from the MeVis Image Processing Library (ML). They contain page-based, demand-driven processing functions. Open Inventor modules (in green) provide visual, three dimensional scene graphs or scene objects. Finally, a brown color indicates macro modules, which represent a combination of other module types, connected by a specific hierarchy and scripted interactions. Most modules have connectors, the bottom ones indicating module inputs and the top ones module outputs.

Figure 15: Basic module types and module connectors of MeVisLab. A blue color indicates ML modules, green modules are Open Inventor modules and a brown color marks macro modules. Connectors can have the form of triangles (ML images), half circles (Open Inventor scenes) or rectangles (data structure pointers).

Modules can be connected in two basic ways. By connecting the module connectors, image data or Open Inventor information is transported between modules. Each connector shape defines a certain type: Triangles indicate the transportation of ML images, half circles of Inventor scenes and squares of pointers to data structures. These connector types can also be seen in figure 15. The second way to connect modules is via parameter connection. With this, any field of a module can be connected to a compatible field. This allows, for example, the synchronization of parameters between different modules. Figure 16 shows an example for a fully connected network in MeVisLab. ML, Open Inventor and macro modules are used. Data connections as well as parameter connections are shown.

3.3.2 Creation of Macro Modules

A macro module is a combination of several other modules that allows the implementation of hierarchies and scripted interactions between modules. To

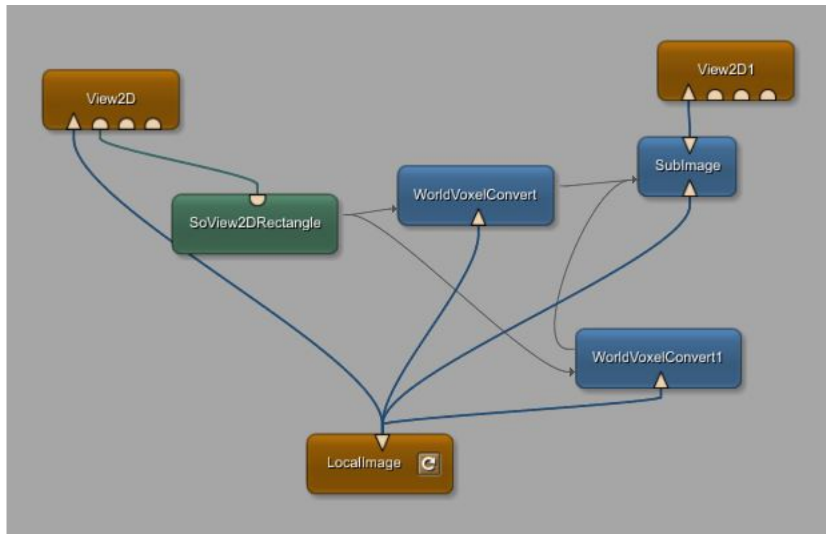


Figure 16: Example network in MeVisLab. All three types of modules are used. Blue and green lines indicate connetions between module connectors (data connections), grey arrows indicate parameter connections.

implement a macro module in MeVisLab, several files are necessary:

1. The module definition file (`*.def`). This file contains some general information about the module, like name, author and date of creation. It is also possible to define a genre and keywords of the module to make it easier to find for other users.

2. The MeVisLab Definition Language (MDL) script file (`*.script`). This file defines the interface of the module, like input-, output- and parameter fields. Furthermore, the graphical user interface (GUI) can be manipulated here.

3. The Python script files (`*.py`). Here, functions and interactions between modules can be implemented using Python scripting.

4. The internal network (`*.mlab`). This file contains the internal network structure of the macro module.

For creating macro modules, MeVisLab provides a tool named Project Wizard, which sets up all required files and their connections.

3.4 TensorFlow

TensorFlow is a platform independent open-source software library for numerical computation, mostly used for machine learning. It is both an interface for expressing machine learning algorithms, as well as an implementation for executing these algorithms. This section explains basic concepts of TensorFlow and is adapted from the TensorFlow whitepaper [44] and the TensorFlow developement guide [45].

3.4.1 Tensors

All data within a TensorFlow program is represented as a tensor, as the name suggests. Tensors can be thought of as multi-dimensional arrays or lists that are passed between operations. A tensor is characterized by its rank, which describes the dimensionality of the tensor, its shape and its data type, e.g. signed or unsigned integer types, IEEE float and double types or string type.

3.4.2 Computation Graphs and Sessions

In TensorFlow, programs are divided into two phases:

1. The construction phase
2. The execution phase

In the construction phase, a computation graph is assembled. The nodes of the graph represent operations, and information is passed between nodes along the edges of the graph in the form of tensors. Operations are abstract computations, in example "add" or "matrix multiply". Usually, computation graphs are started using operations that don't require inputs, e.g. a `tf.constant`:

```
1 const1 = tf.constant(3.0, dtype = tf.float)
2 const2 = tf.constant(4.0, dtype = tf.float)
3 print(const1, const2)
```

which would produce the output

```
Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(),
  dtype=float32)
```

It can be seen from this simple example, that `constant1` is not an actual value, but a node that outputs a tensor. It is important to note that a computation graph doesn't compute anything, but is rather a description of computations and operations. Computations are done in the execution phase by launching the graph in a session. Within the session, the graph is translated to executable operations which can be distributed across different

compute devices, such as the CPU or GPU. When running a session, the user can define a set of outputs that should be computed. TensorFlow then executes the appropriate nodes to compute the desired output:

```
sess = tf.Session()
print(sess.run([const1, const2]))
```

which now produces the expected values:

```
[3.0, 4.0]
```

Now, operations, like element-wise matematical operations can be applied to the outputs of these nodes:

```
const1 = tf.constant(3.0, dtype = tf.float)
const2 = tf.constant(4.0, dtype = tf.float)
sum = tf.add(const1, const2)
print(sess.run(sum))
```

which results in:

```
7.0
```

Another common way of starting a graph is to use `tf.placeholder`, to which external inputs can be assigned during running sessions. When running a session by specifying the desired output, the user has to feed values to every placeholder node that will be evaluated to compute this output:

```
a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
sum = tf.add(a, b)
print(sess.run(sum, {a: 2.0, b:6.0}))
print(sess.run(sum, {a: [1.0, 4.0], b:[3.0, 2.0]}))
```

Now, the result is:

```
8.0
[4.0, 6.0]
```

A graph can be executed multiple times within a session, in example when training deep neural networks with thousands of training data batches.

3.4.3 Training with Tensorflow

Since computation graphs are usually executed multiple times, `tf.Variable` operations are needed to be able to access tensors across multiple executions of a graph. For deep learning applications, the network parameters like weights and biases are usually stored in tensors held in variables. This way, when training a network with many data batches, those network parameters can be updated in every run of the training graph. Contrary to constants, which are initialized when calling `tf.constant` and their value never changes, variables are not initialized by calling `tf.Variable` but must be initialized within a session.

TensorFlow provides a large variety of optimizers that slowly change each variable in order to minimize the loss function within the `tf.train` module, in example a simple gradient descent optimizer as described in section 3.1. A full example for training a simple linear regression model is shown in Code 1. At first, model parameters (weight `W` and bias `b`) are defined as variables with the values $[0.3, -0.3]$. Then, the input of the model `x` and the desired output `y` are initialized as placeholders and the model function is defined. By defining the input and output as a placeholder, the model can be evaluated for several values at the same time. Next, the error function (the discrepancy between the desired output and the actual model output) is calculated using sum-of-squares error. An optimizer to minimize the loss returned from the error function is defined. Then, a training data set is defined by providing inputs and desired outputs of the model. A session is started and training is done for 1000 iterations, changing model parameters each iteration and consequently minimizing the error function via gradient descent.

```
import tensorflow as tf

# define model parameters as variables:
W = tf.Variable([.3], tf.float32)
b = tf.Variable([-.3], tf.float32)

# define model input and desired output as placeholders:
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# define the model:
linear_model = W * x + b


# calculate an error function (sum of squares)
loss = tf.reduce_sum(tf.square(linear_model - y))

# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)

# provide training data
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]

# start the session and initialize the variables:
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
# train for 1000 iterations:
for i in range(1000):
    sess.run(train, {x:x_train, y:y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x:x_train, y:y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```

Code 1: Training a simple linear regression model in TensorFlow

Running this program displays the final model parameters as well as the final loss:

```
W: [-0.9999969] b: [ 0.99999082] loss: 5.69997e-11
```

3.4.4 TensorBoard

TensorBoard is suite of visualization tools for inspecting and understanding TensorFlow graphs and sessions. It is designed to enable easier understanding, debugging and optimization of TensorFlow programs.

TensorBoard requires a log file, called a summary, from which it reads data to visualize. While creating the TensorFlow graph, the user can annotate nodes with summary operations which will export information about the node they are attached to. Like any other node, a summary operation needs to be run within a session to actually generate data. The collected summary data can then be written to a log directory, from which TensorBoard acquires all the information it needs for visualization. A popular application for TensorBoard is to monitor the development of the loss during training. For the linear regression model above, a summary node can be added with the following line:

```
tf.summary.scalar('sum_of_squared_differences', loss)
```

Then, all summary nodes are merged and a summary file writer is defined:

```
merged = tf.summary.merge_all()
train_writer = tf.summary.FileWriter("path/to/log/folder", sess.graph)
```

and within the session, this file writer is run:

```
for i in range(1000):
    sess.run(train, {x:x_train, y:y_train})
    summary = sess.run(merged, {x:x_train, y:y_train})
    train_writer.add_summary(summary, i)
```

Now, a log file will be created in the specified folder. When opening this file with TensorBoard, the progress of loss during training can be observed and manipulated using different visualization tools, as seen in figure 17.

TensorBoard also enables the visual inspection of the constructed graphs by displaying all operation nodes and how tensors are passed between them. The example for the linear regression model graph in figure 18 (a) shows, that even for simple models those graphs contain numerous nodes, making visualization confusing. Therefore, TensorFlow allows to name nodes, and to scope nodes together, defining a hierarchy in the graph. Figure 18 (b) shows a graph of the same linear regression model, but this time, nodes are named and grouped together using name scopes.

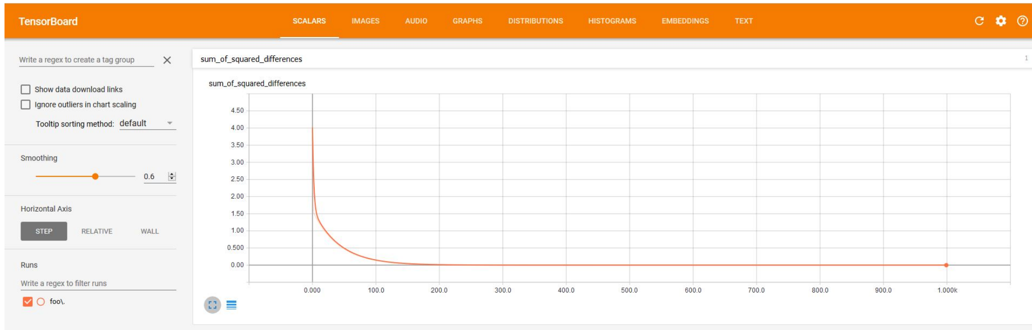
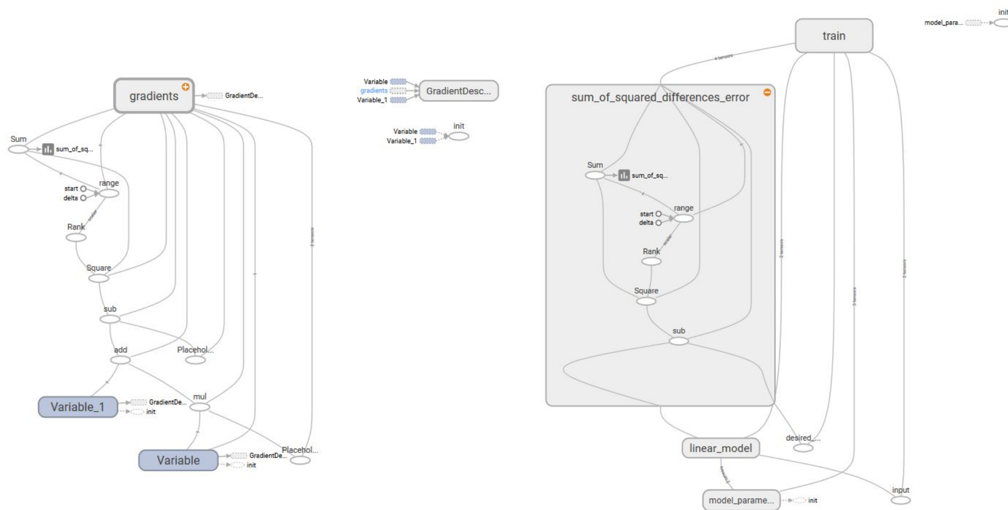


Figure 17: Visualization of the development of sum-of-squares error during training a linear regression model in TensorBoard. Development is shown for 1000 training iterations. It can be seen how the error quickly decreases with every iteration.



(a) Unedited Graph of a linear regression model

(b) Graph of a linear regression model, structured by using name scopes and named nodes

Figure 18: Graph visualization in TensorBoard. Image (a) shows an unedited graph, which can get confusing quickly. (b) shows the same graph, structured using name scopes and with some named nodes, which makes understanding the graph easier. Nodes in one name scope are displayed collapsed, but scopes can be expanded to reveal their inner structure, like the sum of squared differences error scope in this example.

3.4.5 TensorFlow-Slim

TensorFlow-Slim (TF-Slim) is a high-level library for defining, training and evaluating complex models in TensorFlow. It is designed to make programming with TensorFlow simpler and to improve the readability of code by

- the usage of argument scoping, allowing the user to define default arguments for specific operations, reducing repetitive code,
- providing high level functions for defining layers, variables, regularizers, losses etc,
- providing network definitions for popular computer vision models, e.g. VGG-Net or ResNet, including pre-trained parameters in the form of model checkpoints.

Since TensorFlow is very low-level, defining complex models can result in complicated scripts with a lot of boilerplate code. TF-Slim offers an easy way around this and can also be mixed with native TensorFlow and other TensorFlow frameworks.

4 Related Work

Important publications related to the work in this thesis are discussed in this chapter. At first, previous attempts at automatic urinary bladder segmentation in CT scans are considered. It is shown that the presented approach has not yet been applied to this task. In section 4.2, the architectures of deep neural networks for image classification and segmentation on which the segmentation concept in this thesis is based on are reviewed. There are two main designs for semantic segmentation networks explored in this thesis: The first one uses fully convolutional neural networks (FCNs), the second is based on the utilization of so-called atrous convolution. Both of these designs are build upon deep neural networks for image classification, VGG Net and ResNet. Insight is given in all of these four network architectures.

4.1 Automatic Urinary Bladder Segmentation

Even though urinary bladder segmentation is still mostly done manually, many different techniques have been attempted in the past years for automatic segmentation of the bladder in CT images. Earlier concepts used deformable models to delineate the urinary bladder wall in 3D, in example in [46] or [47]. Shi et al. [5] proposed a three step algorithm for segmentation of the urinary bladder. The first step is to apply a mean shift algorithm to obtain a rough contour of the bladder, then, again, region growing is used, followed by a rolling ball algorithm to refine the obtained contour. However, a major problem with region growing approaches is the leaking problem, especially at interfaces of the bladder and other soft tissue, like the prostate. A segmentation attempt using convolutional neural networks was made by Cha et al. [48]. They extracted patches of 32×32 pixels from CT urography scans and classified them using the Convolutional Neural Network by Krizhevsky et al. [3]. However, they found that DL-CNN segmentations alone don't follow detailed structures of the bladder sufficiently. Therefore, in [6], a CNN was trained to output the likelihood that an input region of interest is inside the bladder and assembled a bladder likelihood map from this information. This map was then used as a initial contour for a cascading level-set based bladder segmentation. All of these publications rely on manually segmented CT images for training, evaluation and validation of their algorithms. Obviously, this limits the size of validation data sets.

The idea of using combined PET/CT scans for generating a reference standard/ground truth from PET images to train an automatic segmentation algorithm, as introduced in this thesis, is, to the best of our knowledge, a

new approach. Furthermore, advanced deep learning techniques for semantic segmentation, relying on fully convolutional neural networks and upsampling of coarse feature maps instead of patch-wise classification, have not yet been applied to the problem of urinary bladder segmentation in CT images, as far as we know.

4.2 Neural Networks for Image Classification and Segmentation

4.2.1 VGG Classification Network

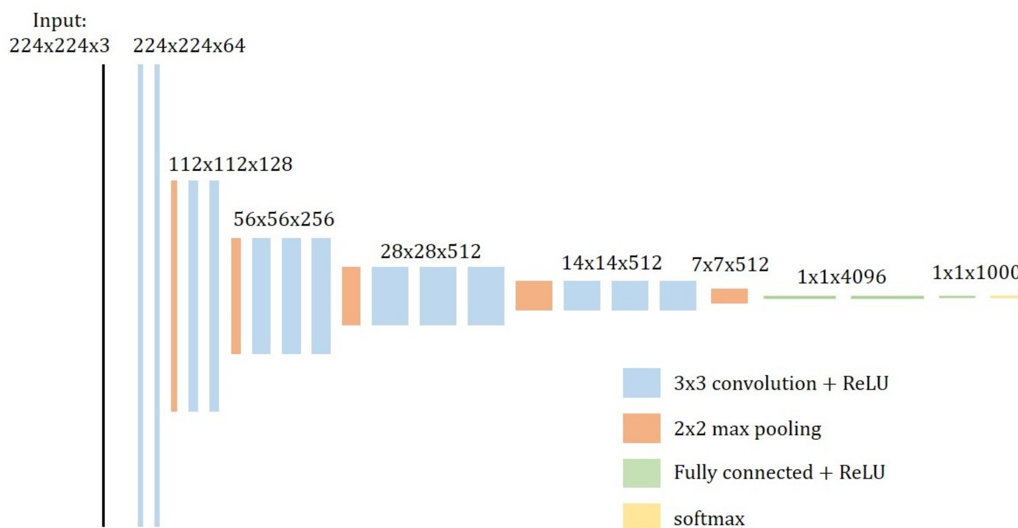


Figure 19: VGG 16 architecture. The network consists of five convolution blocks, each containing several convolutions, and followed by a max pooling layer. After the stack of convolutional layers, three fully connected layers are applied. The last layer uses a softmax activation function to map the output of the network to probabilities.

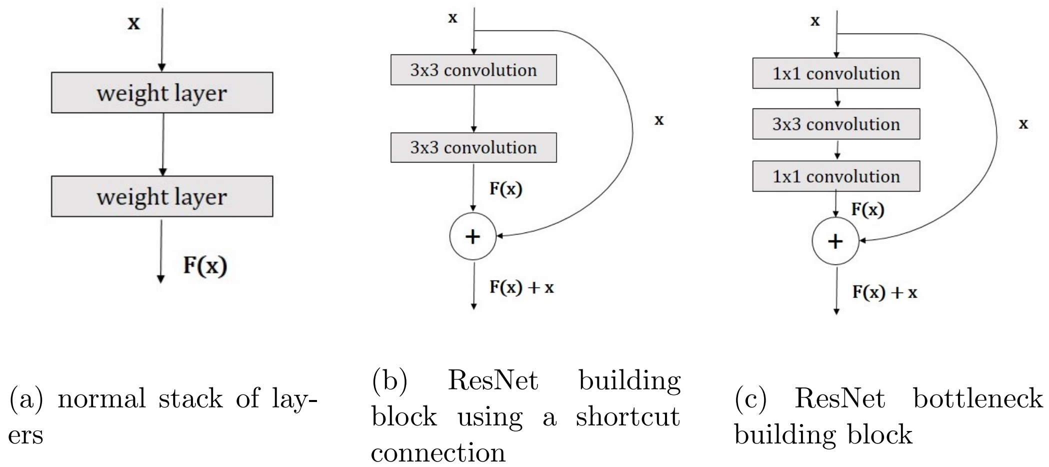
The very deep convolutional networks for large-scale image recognition [49], developed by the Visual Geometry Group of the University of Oxford (hence, the short term VGG-Net) were a runner-up in ImageNet Large Scale Visual Recognition Competition (ILSVRC) 2014 and have since then been used for various tasks such as object detection and image segmentation. The CNN is trained on fixed-size 224×224 RGB images. Convolution layers use filters with a kernel size of 3×3 and a stride and pad of one pixel. Therefore, the spatial resolution of a convolution layer input is preserved after the convolution. Pooling is carried out by max-pooling layers over a 2×2 pixel window,

with a stride of two and without padding, dividing the spatial resolution of an input by two. While different configurations are introduced in the paper cited above, the most popular architecture is the VGG 16 architecture, featuring 13 convolutional layers, five max-pool layers and three fully connected layers, the first two having 4096 channels, the last one 1000 channels which corresponds to the number of classes in the ILSVRC classification problem the VGG 16 Net was trained on. All hidden layers are equipped with a ReLU activation function. The last layer utilizes a softmax activation function to map the output of the last layer to class probabilities. While deeper architectures were tested, this configuration offered the best trade-off between performance and accuracy. The full network architecture can be seen in figure 19. In total, the network has approximately 138 million parameters.

4.2.2 ResNet Classification Network

ResNet architectures were introduced by He et al. in [50]. In this paper it is shown that extremely deep network architectures of up to 152 layers can be trained through the use of residual modules. Their deepest networks won the first place in the ILSVRC 2015.

A major problem with very deep architectures is the degradation problem: the training accuracy of a network saturates with increasing depth, and eventually degrades very quickly. He et al. introduce so-called shortcut connections that skip one or more layers, perform identity mapping and add their output to the output of a stack of layers to overcome this degradation. Figure 20 shows the transition from a normal stack of network layers in (a) to a residual module, the building block of ResNet architectures, in (b). This makes the resultant mapping easier to optimize without adding additional parameters or computational complexity to the model. The basic ResNet architecture uses 34 weight layers and is inspired by VGG Net, using mainly 3×3 convolution filters. Several convolutions are applied successively, then downsampling is performed. Instead of using max pooling layers for downsampling, convolutional layers with a stride two are applied. At the end of the network, global average pooling is performed, followed by a fully connected layer and a layer with softmax activation function. Additionally, shortcut connections are inserted after each pair of 3×3 filters. The architecture can be seen in figure 21.



(a) normal stack of layers

(b) ResNet building block using a shortcut connection

(c) ResNet bottleneck building block

Figure 20: Comparison between ResNet building blocks. While an output $F(x)$ is computed from and input x in a traditional network as seen in (a), a ResNet building block in (b) performs identity mapping of the input and adds it to the output, producing and output $F(x) + x$. Building blocks using bottleneck design as seen in (c) speed up the training process by first reducing the input dimensions using 1×1 convolution, then convolving with a 3×3 kernel, and then upsampling the dimensions using 1×1 convolution again. Adapted from [50].

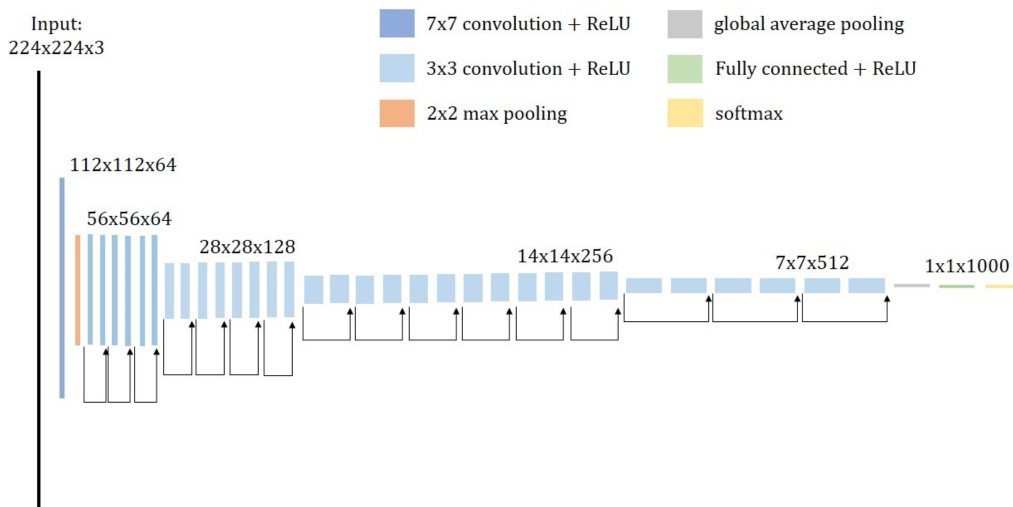


Figure 21: Basic ResNet architecture with 34 layers. After an initial 7×7 convolution and 2×2 max pooling, the input is convolved with a series of 3×3 kernels. Downsampling is performed using these filters and a stride of two. After each pair of convolutions, a shortcut connection is added. At the end of the network there is a global average pooling layer, followed by a fully connected layer and a softmax activation function.

To implement even deeper networks, building blocks are modified to a bottleneck design for faster training. This design uses three layers instead of two, where the dimension of the input is first reduced by a 1×1 convolution, then convolved with a 3×3 kernel, and then dimensions are restored with another 1×1 convolution. This building block can be seen in figure 20 (c). The best performing ResNet uses 152 layers, grouped together in such three-layer bottleneck blocks.

4.2.3 Fully Convolutional Networks for Semantic Segmentation (FCNs)

The FCNs introduced by Long et al. in [38] made a large impact on semantic segmentation with neural networks by providing end-to-end, pixel-to-pixel segmentation networks. The basic idea behind this approach is to transform established classification networks to fully convolutional networks, suitable for semantic segmentation. They achieved best results with the VGG 16 network. For this, fully connected layers are transformed into convolution layers, which allows the network to output a spacial heatmap. This is illusrated in figure 22. However, these output maps are considerably downsampled from the input image because of pooling layers, making the output very coarse. It can be seen from figure 19, that in case of VGG 16 Net, an input image is downsampled by a factor of 32 after the last max pooling layer.

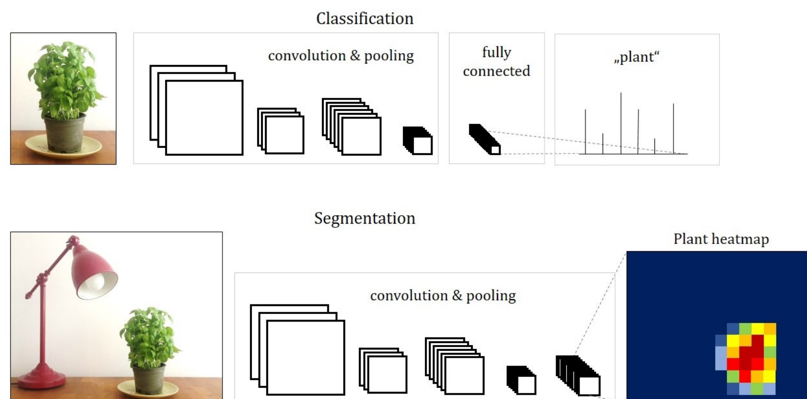


Figure 22: Transition from image classification to image segmentation in CNNs. In the classification network, the last layer is fully connected, in the segmentation network, this layer is transformed to an additional convolutional layer. This enables the network to output a heatmap for the object of interest. Adapted from [38].

To map these coarse outputs to dense predictions, learnable upsampling layers, which resample the image to its original size, are proposed. For upsampling, not only the features from the last downsampling layer are used. Instead, so-called skip connections are introduced to use convolutional features from different layers in the network, which have different scales. Since shallower layers produce bigger feature maps where more spatial information is preserved, this helps capturing finer details from the original image. Upsampling layers are learned for each of these skip connections individually. The approach from Long et al. resulted in three network architectures. The basic FCN 32s architecture simply upsamples the feature map obtained from the adapted VGG 16 network by a factor of 32. For the FCN 16s architecture, the two times upsampled VGG 16 feature map is combined with the features from the fourth pooling layer of VGG 16 and then upsampled by factor 16. The best performing architecture is their FCN 8s architecture, where the combination of upsampled VGG 16 features and fourth pooling layer results is combined with features obtained from the third pooling layer of VGG 16 and then upsampled by a factor of eight. An illustration of these architectures can be seen in figure 23.

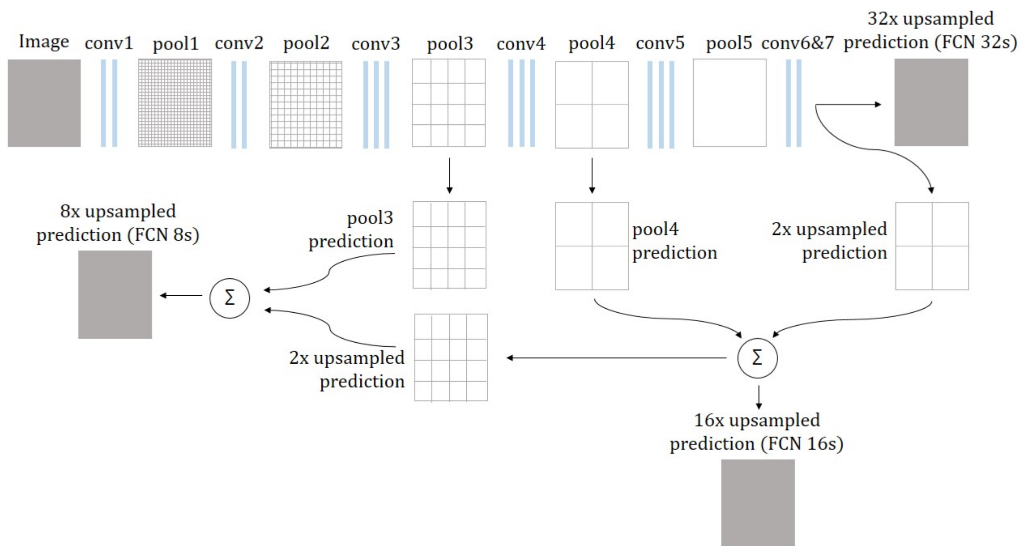


Figure 23: FCN architectures. The input image is downsampled by max pooling layers, getting coarser from layer to layer. In the FCN 32s, the output of the last pooling layer is upsampled by factor 32 in a single step. In FCN 16s, features from the last layer and pool4 layer are combined and then upsampled by factor 16. For FCN 8s, predictions from pool3 layer are inlcuded. Adapted from [38].

4.2.4 Atrous Convolution for Semantic Segmentation

DeepLab systems as proposed in [51] follow a different approach to deal with the problem of considerably downsampled feature maps resulting after a traditional classification networks. They utilize so-called atrous convolution, also called dilated convolution, which is convolution with upsampled filters. By replacing convolutional layers in classification networks such as VGG 16 or ResNet with atrous convolution layers, the resolution of feature maps of any layer within a CNN can be controlled. Furthermore, the receptive field of filters can be enlarged without increasing the number of parameters.

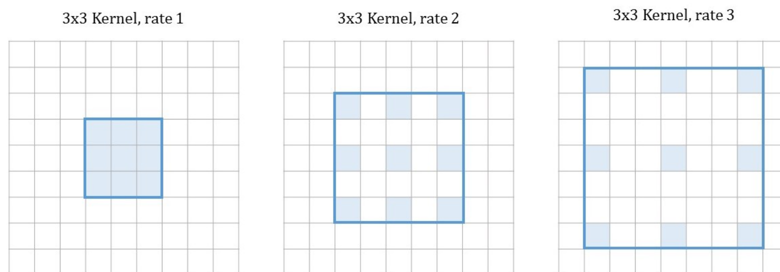


Figure 24: Atrous convolution. In all examples the kernel size is 3×3, but the rate differs. The rate defines by which factor the filter is dilated. Empty values are filled with zeroes. The larger the rate, the larger the receptive field of the filter becomes. For a rate of one, atrous convolution corresponds to standard convolution. Adapted from [52].

In atrous or dilated convolution layers, instead of downsampling the input image resolution by a factor of e.g. two and then performing convolution with a certain kernel, the full resolution image is convolved with a filter 'with holes'. This filter can be seen of an upsampled version of the kernel applied to the downsampled image, where zeroes are inserted in between filter values. Convolution with such kernels is called atrous convolution and can be defined for a 1-D input signal $x[i]$ as

$$y[i] = \sum_{k=1}^{K} x[i + r \cdot k]w[k] \tag{19}$$

where $w[k]$ is a filter of lenght K and r denotes the rate which is equivalent to the stride with which the input signal is sampled. For a rate of one, atrous convolution is equivalent to normal convolution. Atrous convolution in 2-D is illustrated in figure 24.

This convolution allows the construction of many layered networks without decreasing resolution, and since only non-zero values have to be accounted for in convolutions, the number of filter parameters does not increase. However, computing feature maps at the original image resolution is not very efficient, therefore, hybrid approaches are mostly used. Atrous convolution layers are applied in a way to downsample the original image by a factor of eight in total (compared to a factor of 32 in VGG Nets or ResNets), followed by bilinear interpolation to recover feature maps of the original image resolution. Compared to approaches using fractionally strided convolution for upsampling, no new parameters are learned within the network, which leads to faster training.

5 Methods

This chapter discusses the methods with which underlying ideas were set into practice. The first section describes the used Reference Image Database to Evaluate Therapy Response as well as the image data obtained from it. Section 5.2 explains how suitable training and testing data was generated from this image data using a macro module implemented in MeVisLab. The macro module generates ground truth labels from PET image data and applies data augmentation. The implementation of deep neural networks for image segmentation using the TensorFlow library under Python is outlined in section 5.3. At first, the TensorFlow-own file format TFRecords, and how it is used to process input data efficiently, is introduced. Next, the implemented segmentation network architectures are described, which were created using the TF-Slim library and pre-trained classification networks. Lastly, the process of training, testing and evaluating these networks is demonstrated.

5.1 Dataset and Preprocessing

The implemented methods were trained and tested on the Reference Image Database to Evaluate Therapy Response (RIDER) [53]. RIDER is a collection of Computed Tomography, Magnetic Resonance Imaging (MRI) and PET/CT data.

The RIDER PET/CT dataset provides serial patient studies (without metadata) as well as data from multi-vendor and multi-parameter calibration phantoms. It consists of de-identified Digital Imaging and Communications in Medicine (DICOM) serial PET/CT lung cancer patient data and provides serial scans of 28 lung cancer patients (a total of 65 scans), as well as data from studies with a long half-life calibration phantom. As radiotracer, fluorine-18-labelled fluorodeoxyglucose was used in all PET scans. The public database can be downloaded from the National Cancer Imaging Archive (NCIA) at [54].

To obtain an overview of the database, the DICOM datasets were loaded into the medical image visualization software 3DSlicer [55]. For each patient, several CT and PET scans with different scanning and visualization parameters are available. For this project, CT scans with a resolution of 512×512 and a slice thickness of 2.5 mm were selected. The chosen PET scans have a resolution of 128×128 pixel and a slice thickness of 3.27 mm, and use standardized uptake values to measure radiotracer uptake. Patient data with very high noise levels in either the CT or PET data or with unusually low

contrast in the region of interest, as well as the phantom data, were rejected. The DICOM images were converted to Nearly Raw Raster Data (NRRD) file format. This format has the advantage of representing three dimensional data in a single file, in contrary to the DICOM format which saves each slice individually.

After removing patient data with low contrast and high noise from the RIDER PET/CT database, a total of 29 patient datasets were obtained. The CT datasets offer between 148 and 358 transversal slices, yielding a total of 8754 CT scans. Since these scans cover the whole torso the urinary bladder is only visible on a fractional amount of the images, with an average of 25 transversal slices per dataset covering the bladder. It would not be sensible to train our deep network with such a large amount of negative training examples, therefore, a total amount of 845 CT image slices around the urinary bladder were extracted from the whole dataset. A full overview of the used datasets, including the number of total slices and slices showing the urinary bladder can be seen in table 6 in appendix A.

5.2 Generation of Image Data

Parts of this section follow my master's project report [18]. A paper about the exploitation of ^{18}F-FDG enhanced urinary bladder in PET data for deep learning ground truth generation in CT scans was accepted at SPIE Medical Imaging 2018 [56].

The 845 images obtained from the RIDER PET/CT database are probably not enough to train a deep neural network with. It also has to be considered that not all data can be used for training, since testing data must also be taken from the same dataset. Furthermore, a reference standard in the form of segmented CT images is not available. The first problem, the small size of the databases, is overcome by using data augmentation. The basic idea behind data augmentation is to apply plausible changes, which preserve label information, to the existing data to generate new, additional training data. These images are similar to, but not the same as the existing data [57]. To enlarge our dataset we applied rotation and scaling to CT images as well as the masks generated from the corresponding PET data. Furthermore, zero-mean Gaussian noise was added to CT images.

The generation of segmentations of the urinary bladder as a ground truth for training a deep neural network is performed by using combined positron emission tomography-computed tomography scans. The most commonly

used radiotracer, ^{18}F-FDG, accumulates in the urinary bladder, therefore, the bladder always shows up in these PET scans. Contrary to CT, PET images exhibit high contrast and are therefore comparably easy to segment automatically. We automatically segment PET images using a simple thresholding approach to generate binary masks of the urinary bladder that match CT data acquired from the same patients at the same time.

At first, the available 29 patient datasets containing a total of 845 image slices showing the urinary bladder were split into training and testing data. A standard for this split commonly found in literature is 80% training data, 20% testing data. Loosely following this guideline, 630 images were used for training and 215 images were reserved for testing, corresponding to 21 and 8 patient datasets, respectively. Next, the 21 patient datasets for training were processed with the proposed MeVisLab network to obtain individual, augmented CT slices as well as corresponding ground truth labels. For the 8 datasets reserved for testing, only a ground truth label was created and no augmentation was applied. To additionally be able to analyse the effect of data augmentation, a training dataset containing only the 630 unaugmented, original images and labels as well as a dataset with only transformed image data (without noise) was put together.

The necessary steps for data generation were implemented with the medical imaging framework MeVisLab. The corresponding Macro module and Python source code is freely available under [58]. The general network constructed in MeVisLab can be seen in figure 25.

The purpose of this network is to load, process and visualize the given input data using modules already integrated in the MeVisLab framework, as well as a self-implemented macro module.

Corresponding PET and CT image data is loaded into the module in integer type for CT images and double type for PET data using the `itkImageFileReader`. This module permits accessing file formats supportet in the Insight Segmentation and Registration Toolkit (ITK), like the NRRD files generated in the first step. These three-dimensional images are then fed into the `DataPreperation` macro module. This self-implemented module is the centrepiece of the data generation step in this thesis. It calculates binary masks from the PET data, performs data augmentation as specified by the user and saves the created training data. It will be explained in greater detail in section 5.2.1. Furthermore, the PET data is segmented using the `Threshold` module and the threshold calculated within the `DataPreperation` module.

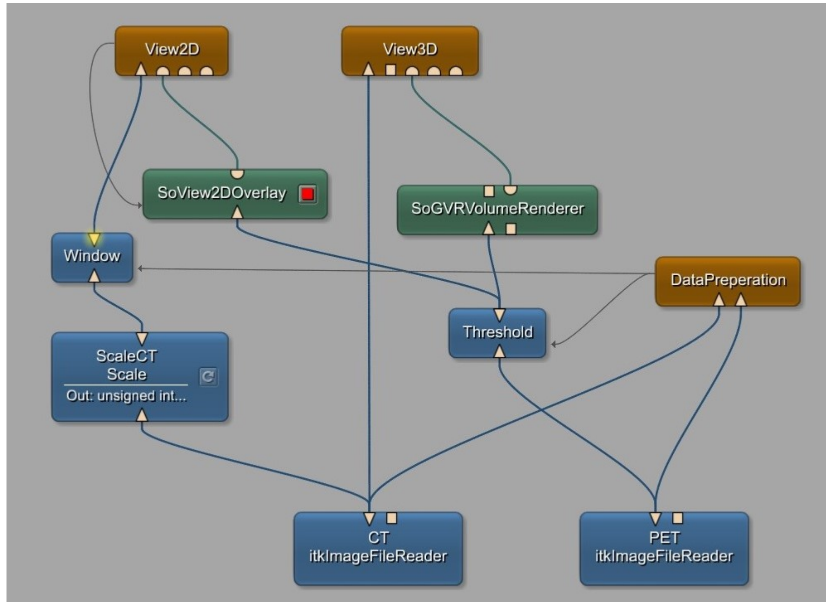


Figure 25: General Network for loading, processing and visualizing PET and CT data, implemented in MeVisLab.

For three-dimensional visualization of the data, the `SoGVRVolumeRenderer` is used for volume rendering of the segmented PET data. The `View3D` macro module overlays the three dimensional CT data with the rendered PET volumes. To visualize the data in 2D, the `SoView2DOverlay` module is used to blend the thresholded PET data over the CT data in the two-dimensional viewer `View2D`. For visualization, CT data is converted to 16 bit unsigned integer file type using the `Scale` module to match the images generated within the `DataPreperation` module. Because CT images offer rather low contrast for soft tissue, it is common practice to apply windowing to them. This means that only a certain range of pixel values around a defined center are displayed over the full range from white to black, and pixels outside of this window are displayed in all black or white. This greatly improves the contrast for soft tissue in the CT datasets. MeVisLab provides the `Window` module for the application of windowing. The center and width for this window are calculated for each dataset individually within the `DataPreperation` module.

5.2.1 The DataPreperation Macro Module

Internal Network

The internal network of the DataPreperation macro module can be seen in

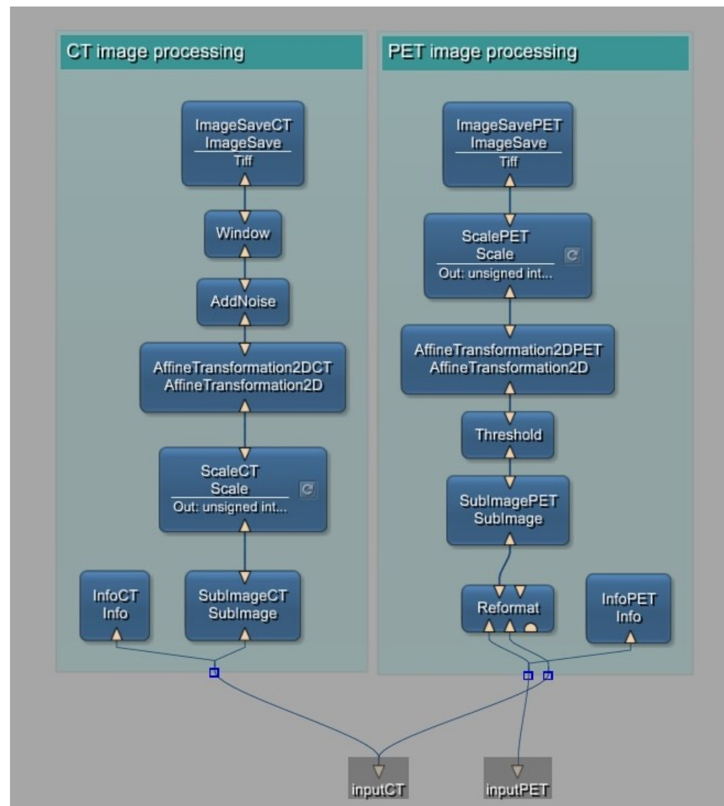


Figure 26: The internal network of the DataPreperation macro module.

figure 26. Generally, the network consists of two groups, one for CT and one for PET image processing. In this section, the MeVisLab Image Processing Library (ML) Modules used and their purpose are explained.

- PET image processing:
  First, general information about the input image is extracted via the `InfoPET` module. This module provides information like image size, image type, maximal and minimal pixel value, etc. The `Reformat` module reformats the PET dataset to the local coordinate system of the CT dataset using trilinear interpolation. This step is necessary because the PET and CT datasets do not have the same dimensions. The reformated image is then fed into the `SubImagePET` module. This module allows the extraction of subregions from input images. By iterating over the z-coordinate of a dataset and leaving the other parameters unchanged, individual transversal slices can be selected and manipulated. Next, the extracted PET slices are segmented by the ap-

plication of the `Threshold` module. A fixed threshold is calculated for each dataset within the Python script. Every pixel above the threshold is considered foreground, all other pixels are labelled as background. The output of this module is a binary image of the urinary bladder. Data augmentation of the binary masks is performed using the `AffineTransformation2DPET` module. This module enables the application of several affine transformations in 2D. Translation, rotation, shearing and scaling can be entered, however, for this macro module, only rotation and scaling are enabled. Before saving the image, it is converted to 8 bit unsigned integer and scaled to a range between 0 and 255 using the `ScalePET` module. At last, the produced slices are saved in TIFF format using the `ImageSavePET` module.

- CT image processing:
  Again, image information is obtained using the `InfoCT` module. Transversal slices are extracted from the three-dimensional image data using the `SubImageCT` module. The CT images are converted and scaled to 16 bit unsigned integer right away, because the `AffineTransformation2D` module requires a datatype of 16 bit unsigned integer or lesser to work. In the `AffineTransformation2DCT`, the same transformations as with the PET masks are applied to the CT slices. Additionally, the `AddNoise` module enables the addition of noise from various distributions to the input image. In this project, uniformly distributed, zero-mean Gaussian and Salt and Pepper noise can be chosen. Eventually, windowing is applied to the CT images to improve contrast for the urinary bladder. Then again, the slices are saved as TIFF using the `ImageSaveCT` module.

MDL Script and Panel
The MDL Script `DataPreperation.script` consists of three main sections:

- The interface section defines the inputs (CT and PET image data) and outputs (no outputs are used for this module) of data connections. Furthermore, the parameters fields of the implemented macro module are declared here. The fields declared in the interface section can either be independent script fields or they can be defined as aliases for internal fields of the internal network.

- The commands section defines the scripting file containing the functions to be executed upon the activation of certain fields. Also, the commands for calling these functions are defined.

- The window section can be used to create a panel for the macro module, as seen in figure 27.

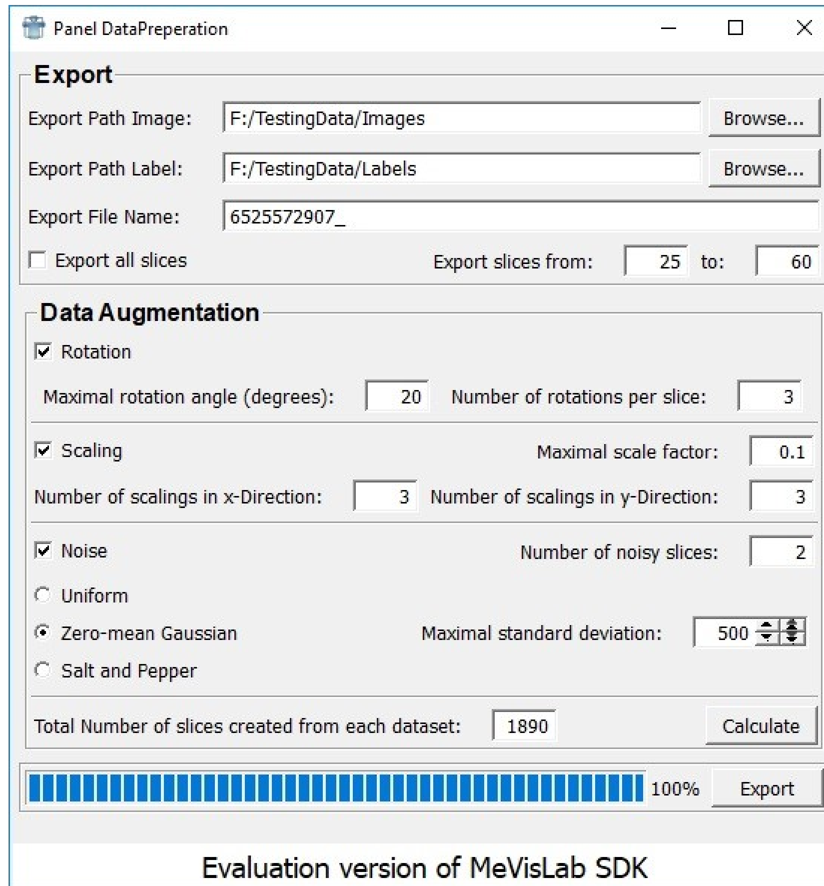


Figure 27: Panel of the DataPreperation macro module.

The DataPreperation module's panel is shown in figure 27. It is used to set the desired parameters and to monitor the progress of the file export. In the export section of the panel, the user can enter or browse to the desired export paths to which the created images and labels will be saved. Furthermore, the file name can be specified. It is also possible to choose between exporting all slices of the dataset or to only select certain slices.

The data augmentation section of the panel is for choosing the desired data augmentation steps and parameters. For rotation, the user can specify the maximal angle (in degrees) to which the images will be rotated. Furthermore, the number of rotations performed on each slice can be entered. The algorithm will then produce images that are rotated to equally distributed angles

between the negative and positive maximal rotation angle. When choosing scaling, the user can enter the desired maximal scale factor as well as the number of scaling steps performed in x- and y-direction. The algorithm will calculate scale factors in a range defined by the maximal scale factor around one and will apply these scalings to the input images. If noise should be applied to the input images, the number of slices generated can be specified. Other parameters depend on the noise type. For uniform noise, the maximal noise amplitude can be entered. For zero-mean Gaussian noise, one must define the maximal standard deviation and for salt and pepper noise, the maximal noise density is used as an input. An overview over all input parameters, their default values and their minimal and maximal values is shown in table 1. Minimal, maximal and default values were chosen such to limit the user to parameters that produce meaningful outputs.

By pressing the Calculate button, the number of images and labels created with the given parameters and for the specified slices are pre-calculated. This allows an easy overview over the amount of data obtained. Once all the parameters are entered and the user is satisfied with the number of created data, the algorithm can be started by pressing the Export button. Progress of the data export can be monitored via the progress bar at the bottom of the panel.

Table 1: Parameters for data augmentation. This table shows all parameters specifiable by the user, as well as their default, minimal and maximal values.

Augmentation Type		Parameter	Default	Minimum	Maximum
Rotation		Maximal Rotation Angle	20	0	180
		Number of Rotations per Slice	3	1	9
Scaling		Maximal Scale Factor	0.1	0.05	0.15
		Number of Scalings in x-Direction	3	0	5
		Number of Scalings in y-Direction	3	0	5
Noise		Number of Noisy Slices	2	1	6
	Uniform	Maximal Amplitude	1500	500	2500
	zero-mean Gaussian	Maximal Standard Deviation	500	100	1000
	Salt and Pepper	Maximal Density	0.2	0.05	0.5

Python Script
Within the Python script, interactions between module fields, inputs, panel fields and outputs can be defined. This section gives an overview of the most important functions implemented in the `DataPreperation.py` file.

When pressing the Export button on the panel, the function that is called is the `setThreshold()` function. This function calculates the threshold T for segmenting PET data and passes it to the `Treshold` module. The threshold is defined as a fixed percentage, in this case 20%, of the maximal SUV within the dataset:

$$T = SUV_{max} \cdot 0.2 \tag{20}$$

The next function to be executed is the `setWindow()` function, which calculates a suitable window center c for contrast enhanced visualization of soft tissue in the CT data. For this, the following formula was derived:

$$c = \frac{65535}{|I_{min}| + I_{max}} \cdot 2800 \tag{21}$$

where I_{min} and I_{max} are the minimal and maximal intensity value of the input image, respectively. As already mentioned, a CT contains Hounsfield values between -1024 HU and 3071 HU and the window center for soft tissue usually lies around 0 HU. Our input images, however, have variable intensity ranges around -3000 to +3000 and are already scaled to 16 bit unsigned integer, ranging between 0 and 65535. Therefore, these conversions have to be made. The factor 2800 was determined empirically to obtain optimal contrast for the urinary bladder.

Next, the functions `exportCT(field)` and `exportPET(field)` are called. These functions check whether the export path specified by the user is valid and then iterates over the slices to export, calling the main function `augmentAndSafe` for each slice. Furthermore, the progress bar is updated in this function. The `augmentAndSafe(...)` performs the augmentation steps specified by the user and safes the generated data to the export directory. The parameter `name` defines, whether PET or CT images are processed. Rotation angles, scaling factors and noise levels are determined using the respective functions. Then, the algorithm loops over all these values and passes them to the corresponding fields in the `AffineTransformation2D` and `Addnoise` modules. The file name is assembled by concatenating the input parameters and the transformation parameters, then the images or labels are saved.

The angles to which the input image should be rotated are calculated in the `getRotationAngles()` function. If rotation is disabled, the rotation angles will be set to zero. The function `getScales` returns all scaling factors to which an image will be transformed. It returns scaling factors of one, if scaling is disabled. In the function `getNoiseLevels`, noise type and noise levels are determined. For uniformly distributed noise, a range of noise amplitudes added to the input image is returned. In case of gaussian noise, a range of standard deviations is calculated. For salt and pepper noise, a variation of different densities is returned. If the adding of noise is disabled, noise type is set to `none` and all parameters are set to zero.

The function `switchNoiseType()` enables the input of different noise parameters in the module panel, depending on which noise type was selected. If the user activates the Calculate button in the panel, the function `calcSlice Number()` is called to display the number of total slices generated from both PET and CT data.

5.3 Image Segmentation using Deep Neural Networks

Algorithms for image segmentation using deep neural networks were implemented using TensorFlow 1.3 under Python 3.5. The code is divided up into several files:

- `tf_records.py`: Contains functions for creating and reading from the TensorFlow recommended file format TFRecords. Those functions are used to transform image data into a file format that is easy and fast to process in TensorFlow.
- `make_tfrecords_dataset.py`: This script can be used to put together a TFRecords dataset from a directory of image files.
- `networks.py`: Includes the model definitions for the implemented image segmentation networks. These functions can be run in training or testing mode.
- `upsampling.py`: Contains tools for creating bilinear upsampling filters used for upsampling the predictions made by the networks using transposed convolution.
- `FCN_training.py` and `ResNet_training.py`: These scripts are used for training the deep neural network models defined in `networks.py`.

- `FCN_testing.py` and `ResNet_testing.py`: These scripts can be used for testing the previously trained deep neural networks.
- `metrics.py`: Provides metrics for evaluating the segmentation results by calculating similarity measures between network prediction and ground truth.

5.3.1 Working with TFRecords files

There are two main ways of providing a TensorFlow program for training a deep neural network with data:

1. Feeding: The computation graph is constructed using placeholders, then data is loaded using python code and fed into the graph during the session.
2. Reading from files: The user provides a list of file names from which a file queue is created. The data is then read and decoded from this file queue, during which batching and shuffling can be performed.

Since loading image data is a time-consuming operation, data is provided in this project via reading from files. The standard, recommended TensorFlow format for this is a TFRecords file. A TFRecords file represents a sequence of binary strings, which enables the user to store images and ground truth annotations in a single binary file which is efficient to read. This way, the TFRecords file has to be created only once, and can then be used during experimenting with and fine tuning the network without having to open image data individually each time.

Tools for writing and reading TFRecords files are found in the file `tf_records.py`. Writing image and label data into a TFRecords file is performed with the function `write_to_tfrecords(image_path, label_path, tfrecords_filename, height, width)`. The first step is to gather the filenames of each image specified in `image_path` and `label_path`. Those images are then opened, resized to a resolution specifiable by the `height` and `width` parameters, and converted to string. An `Example` called protocol buffer is put together from the data one wishes to store. Properties of the data set, like images, labels and image size, are stored within the protocol buffer as protocols called `Features`. Features have one of three base types: bytes, float or int64. The example buffer is then serialized and written to a TFRecords file specified by `tfrecords_filename`.

To read image data from the previously created TFRecords file, the function `read_and_decode(tfrecords_filename)` was created. The name of a suitable TFRecords file is passed to this function in `tfrecords_filename` from which records are read and decoded by taking the serialized examples and mapping the features back to tensor values. At first, image and label data is parsed in string format, then this raw data is decoded to unsigned integer 8. Since image dimensions are lost when converting image data to string, images and labels have to be reshaped to their original dimensions. For this, image height and width are parsed from the TFRecords file as integer values.

5.3.2 Creating TFRecords files for Training and Testing

The file `make_tfrecords_dataset.py` contains a script for writing image data to TFRecords using the function described above. Paths to image and corresponding label data must be provided. Furthermore, the user can specify the target resolution of images and labels. This script was used to produce several TFRecords files used for further training and testing. From each raw image dataset TFRecords files containing images and labels downsampled by a factor of two to a resolution of 256×256 were created. Furthermore, two TFRecords files were generated from the unaugmented and transformed training data. Due to the unsatisfactory performance of training data with additional noise in the case of downsampled images and the long training times, we refrained from training our networks with fully augmented images at their full resolution.

Testing data was also converted to TFRecords files with one file containing downsampled images of resolution 256×256 and one file with original image resolution. Table 2 provides an overview over the created files. It shows the filenames of the TFRecords files, the augmentations applied, the resolution as well as the total number of labelled images within the dataset.

5.3.3 Segmentation Networks using TF-Slim and pre-trained Classification Networks

Neural networks for complex tasks like image segmentation need to be large and deep, resulting in many thousands of parameters. This means that training such networks requires huge datasets and a lot of computational power, and the training process might still require days or even weeks to complete. Therefore, pre-trained models were used for the segmentation task at hand. The TF-Slim API contains a set of standard model definitions implemented

Table 2: Created TFRecords files for training and testing.

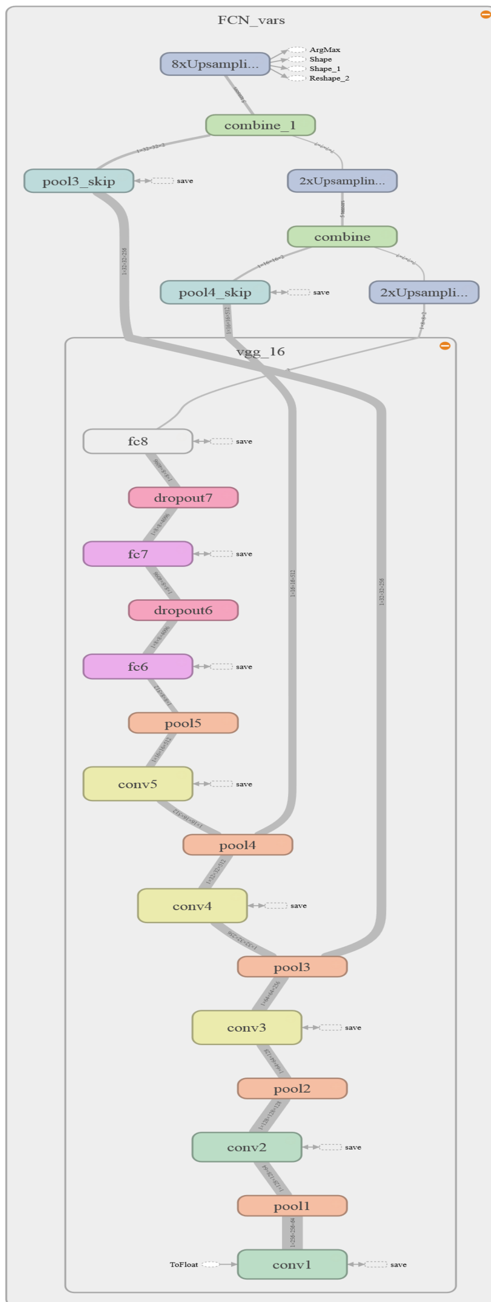
File Name	Augmentations	Image Resolutions	Number of Images
Training_NoAug_256.tfrecords	none	256x256	630
Training_NoAug_512.tfrecords	none	512x512	630
Training_TransAug_256.tfrecords	rotation, scaling	256x256	17010
Training_TransAug_512.tfrecords	rotation, scaling	512x512	17010
Training_FullAug_256.tfrecords	rotation, scaling, noise	256x256	34020
Testing_256.tfrecords	none	256x256	215
Testing_512.tfrecords	none	512x512	215

with TF-Slim as well as checkpoints for pre-trained parameters. An overview of the implemented models, corresponding code and links to the model checkpoints can be found at [59]. The models were trained on the ILSVRC-2012 dataset for image classification. The TF-slim model library contains implementations of VGG 16 and ResNet V2, which are adapted for segmentation tasks, as explained in section 4.2.

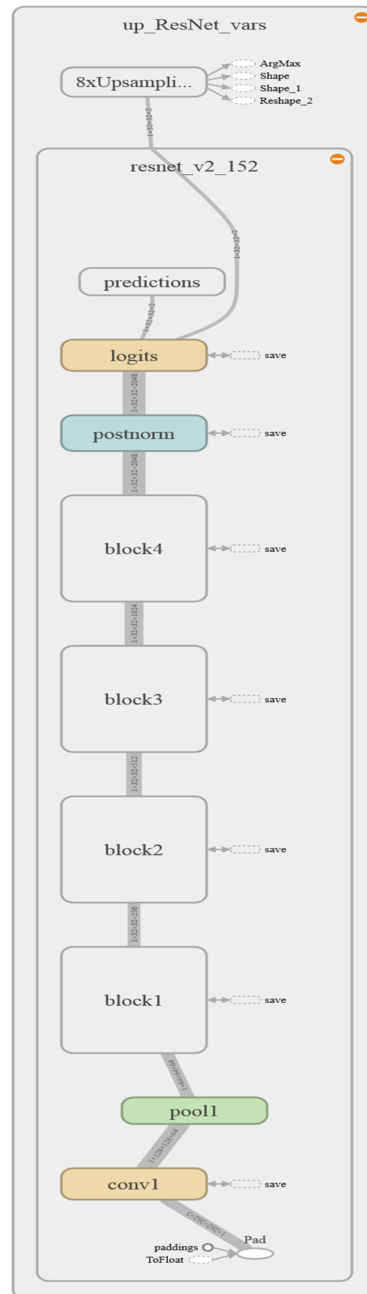
The model definitions of the implemented segmentation networks are found in the file `networks.py`. The implementation of these models is based on the implementation found in [60]. The first network definition, `FCN(...)` uses a pre-trained version VGG 16, which is already implemented in a fully convolutional way in the TF-slim model library, which means that the last three fully connected layers are also implemented as convolutional layers. By calling the `vgg.vgg_16(...)` function using `'SAME'` padding, a map of unscaled log probabilities of the classes (logits), downsampled by a factor of 32, is returned. Furthermore, a dictionary called `end_points` is obtained, which contains the logits of each network layer. These coarse feature maps are then upsampled using bilinear interpolation and skip connections, resulting in a FCN 8s architecture as described by Long et al. in [38]. In the first step, logits of the last layer are upsampled by factor two. Features produced by the `pool4` layer of the VGG 16 classification network are extracted from the endpoints and combined with the upsampled final logits. The combination is once again upsampled by a factor of two, and then combined with the features obtained from the `pool3` layer. As a last step, this combination is upsampled by a factor of eight, to produce an output of the same size as the original input to the classification network. This output represents the final unscaled log probabilities of the FCN 8s model and is returned by the function. Note that while upsampling filters could be defined as a learnable variable, they are kept fixed in this model, since Long et al. stated in their paper that

learnable upsampling kernels didn't significantly improve the performance of the model, while making computation more expensive. The network architecture was illustrated using TensorBoard, which can be seen in figure 28 (a). For easier further processing, the `FCN(...)` function also returns a dictionary which maps the variables from the VGG 16 name scope to the new name scope of our segmentation model by removing the `'vgg_16` part of the variable names. Since the variables for the skip connections, `pool4_skip` and `pool3_skip`, should be initialized and not restored from the checkpoint file, these variables are ignored in the mapping. This dictionary is later used to initialize the model's parameters by using the pre-trained parameters of the VGG 16 network.

The second model definition, `upsampled_ResNet(...)` utilizes a pre-trained version of ResNet V2 with 152 layers and atrous convolution to perform image segmentation. To run ResNet in fully-convolutional mode, the model definition `resnet_v2.resnet_v2_152(...)` must be called with the parameter `global_pool` set to `False`, otherwise global average pooling will be performed before computing the unscaled log probabilities of classes. Furthermore, `spatial_squeeze` must also be set to `False`, to ultimately obtain output logits which are of the same shape as the inputs. With the parameter `output_stride` the user can specify the factor by which the logits produced by ResNet V2 will be downsampled. For this task, a factor of eight was chosen to obtain a good trade-off between density of the computed feature maps and computational and memory expenditure. Once the downsampled logits are obtained, they again need to be upsampled by a factor of eight to obtain a result of the same size as the input image. Again, a visualization of the network architecture was obtained using TensorBoard, as seen in figure 28 (b). Once more, a variable mapping dictionary is created to map variable names to the right name scope by removing the `'resnet_v2'` from variable names.



(a) FCN architecture



(b) upsampled ResNet architecture

Figure 28: Network Graphs for FCN and upsampled ResNet visualized with TensorBoard. Figure (a) shows the implemented FCN architecture, which is based on a pre-trained VGG 16 network and upsampling with skip connections. Figure (b) shows our upsampled ResNet architecture, which is based on a pre-trained ResNet V2 152 network. The layers of this network are condensed within 4 building blocks.

5.3.4 Bilinear Upsampling

It was stated by Long et al. in [38], that upsampling can be performed by using transposed convolution. It is also often referred to as fractionally strided convolution or deconvolution. Convolution can be seen as sliding a convolution filter over an image and computing the dot product between filter and input in every step, which gives one element of the output image. Depending on the stride of the convolution filter, the resolution of the output image might be of lower resolution than the input. Fractionally strided convolution performs the opposite operation, going from a small resolution input to a bigger resolution output. One element in the input image defines the weights for the convolution filter, which is then copied to the output. Where filter regions overlap, the filter values are added. This operation is actually equivalent to the backward pass of a traditional convolution performed during error backpropagation. An illustration of this can be seen in figure 29.

TensorFlow has a built in function for this operation, `conv2d_transpose(value, filter, output_shape, strides)`. This function requires a `filter` in the form of a 4-D tensor with shape `[height, width, out_channels, in_channels]`. To calculate a filter kernel of specified shape, the file `upsampling.py` contains a function adapted from [60]. At first, the function `bilinear_upsampling_weights` determines the size of the filter kernel which is calculated from the stride s as

$$k = 2 \cdot s - s\%2 \tag{22}$$

where % is the modulo operator. An appropriate weight matrix is initialized. Then, the scaling factor is determined and the center point c is calculated as

$$c = \begin{cases} s - 1 & \text{if } k\%2 = 1 \\ s - 0.5 & \text{else} \end{cases} \tag{23}$$

Next, grid g containing values from 0 to kernel size k is created. The elements of the weight matrix w are calculated with the formula

$$w_{i,j} = \left(1 - \frac{|g_i - c|}{s}\right) \cdot \left(1 - \frac{|g_j - c|}{s}\right) \tag{24}$$

5.3.5 Training the Segmentation Networks

Training of the deep neural networks is performed using the files `FCN_training.py` and `ResNet_training.py`. Both files follow the same pattern, however,

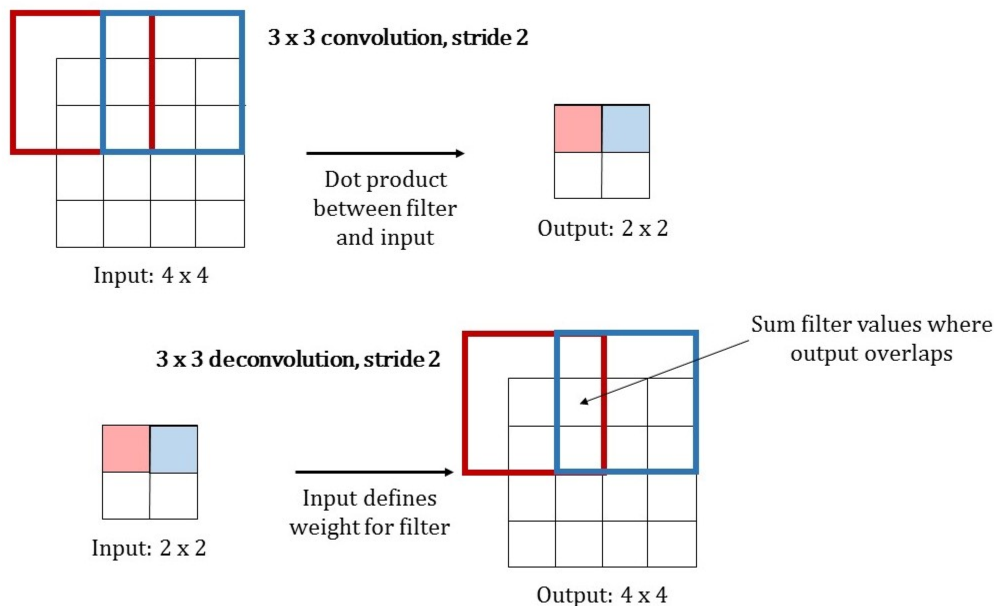


Figure 29: Comparison between normal convolution and transposed convolution. Both operations use a 3×3 kernel and a stride of two. Traditional convolution determines the output value as the dot product between filter and input, by moving the filter kernel for two pixels in every step, the input is downsampled by factor two. For transposed convolution, the input value determines the filter values that will be written to the output. Where filters overlap, the values are summed up. The stride defines the movement of the filter kernel in the output image, and therefore influences the factor of upsampling.

since networks are trained from different checkpoint files, there are some differences between training a fully convolutional network and an upsampled ResNet network. While the FCN architecture requires a VGG 16 checkpoint to work, a ResNet V2 152 checkpoint is needed for the upsampled ResNet architecture. Segmentation is performed into two classes, background and foreground (urinary bladder), so the number of classes is chosen accordingly. Batch size is chosen as 1 as it was in the original architectures. A larger batch size would result in tensor sizes exceeding GPU memory. A batch size of 1 means that each image will be processed individually and no batch normalization is performed. Then, images and labels are loaded from a specified TFRecords training data file.

Next, random batching of the obtained image data is performed. The func-

tion `tf.train.shuffle_batch(tensors, batch_size, capacity, min_after_dequeue, num_threads)` takes a list of tensors defined in `tensors`, creates a file queue of maximum size `capacity` from these tensors and then returns batches of size `batch_size` randomly sampled from this queue. The parameter `min_after_dequeue` controls the minimum number of elements left in the queue after a dequeue and can be used to ensure that elements still get mixed.

As a next step, label tensors of shape `[batch_size, height, width, number_of_classes]` are assembled. This is done since labels up to this point are of shape `[batch_size, height, width]` and are therefore not comparable to the predictions made by the network models. However, this comparison is necessary to calculate a loss function which then can be minimized by the optimizer to find the optimal model parameters.

The generated batch of images is then fed to one of the network models, either `FCN(...)` or `upsampled_ResNet(...)` which return the network's logits with the current parameters, as well as the variable mapping dictionary. A cross entropy error function is calculated between the logits of the model and the labels. As an optimizer, instead of the simple gradient descent algorithm, the more sophisticated optimizer following the Adam algorithm proposed in [61] with a learning rate of 10^{-4}, as suggested in the paper, is used. While this optimizer requires more computations for each parameter in each training step, it usually converges more quickly without the need to fine tune the learning rate. The learning rate is chosen as a fixed value, since Adam optimizer performs learning rate decay internally.

Before checkpoint variables are loaded into the network using the variable mapping created by the model, the mapping must be adjusted to the current task. Since the pre-trained VGG 16 and ResNet V2 152 models are trained for image classification with 1000 classes, the last layer of these networks which is responsible for the number of classes must be omitted. For VGG 16, this last layer is called `fc8`, for ResNet V2 152 it is called `logits`, so all parameters containing this name scope are ignored. With the updated dictionary, the variables from the pre-trained Checkpoints can now be loaded and mapped to the right name scope using the TF-Slim function `slim.assign_from_checkpoint_fn(...)`.

Next, a variable initializer and a summary writer are created. Then, the session can be launched and the actual training is started. Training is performed for a given set of iterations. The loss function is evaluated and the summary

is updated in each step. Every 100th iteration, the current loss is printed to allow easy monitoring of the training process. Furthermore, the current model parameters are saved to the checkpoints directory every 1000th step, as well as after training has finished.

5.3.6 Testing the Segmentation Networks

For testing the neural network models, the files `FCN_testing.py` and `ResNet_testing.py` are provided. Similar to the scripts for training, both files have the same structure and only differ in the used model definition and checkpoint files. After defining the paths to checkpoint files and the testing data, once again, image data is read and decoded from the specified TFRecords file. This time, random batching is not performed. However, since the neural network model definitions require a tensor of shape **`[batch_size, height, width, depth]`** to work, an additional dimension has to be added to the image and label tensors. Afterwards, inference is performed to obtain unscaled log probabilities of classes. To obtain a binary prediction, meaning a tensor containing zeroes and ones, from these logits, **`tf.argmax`** is applied. For calculating normalized probabilities of classes, the softmax function **`tf.softmax`** is used. Next, the similarity metrics true positive rate, true negative rate and dice coefficient are calculated. Those metrics will be discussed in more detail in the next section. Before starting the session, variables and checkpoint saver are initialized.

Within the session, parameters trained in the previous step are loaded into the model. Vectors for calculating the evaluation metrics are initialized and the number of iterations performed, corresponding to the size of the testing dataset, is specified. The current image, label, prediction, scaled probabilities as well as the Dice coefficient are evaluated at every testing iteration. The Hausdorff distance similarity metric is calculated within the session using numpy arrays instead of Tensors, which proved to be more efficient since the numpy library already contains many operations needed for calculation. Note that Hausdorff distance is only calculated for samples where both prediction and label are not empty, e.g. containing at least one pixel belonging to the foreground. Metric values are stored in their corresponding vectors. Images, corresponding labels and predictions are saved in jpeg file format to a specifiable directory. Furthermore, the option to display a sample of input images, their corresponding label and the prediction made by the segmentation network is implemented here. After the testing loop is finished, the mean of true positive rate (TPR), true negative rate (TNR), Dice coefficient (DSC) and Hausdorff distance (HD) are calculated. Furthermore,

histograms of TPR, TNR, DSC and HD are created. Histograms of these evaluation metrics are useful for evaluating per-image scores of the dataset. Per-image scores are important because measures averaged over the whole testing dataset are not suitable to distinguish between algorithms that perform mediocre on all images and algorithms that perform very well on some images, but very bad on others [62].

5.3.7 Evaluation Metrics

To evaluate the results achieved with the proposed neural networks, several metrics that are commonly applied for measuring similarity between the ground truth and the segmentation result when working with medical image data are calculated. True Positive Rate, True Negative Rate and Dice coefficient are metrics commonly found in literature. However, these metrics might be a poor measure for images with a lot of background and small object segments. Therefore, the Hausdorff distance is additionally measured. It is also a useful estimate when the boundary delineation of the segmentation is of special interest, as it is the case in this thesis. However, it should be noted that Hausdorff distance is very sensitive to outliers [63]. Functions for calculating evaluation metrics are found in the file `metrics.py`

True Positive Rate and True Negative Rate

The true positive rate (TPR), commonly referred to as sensitivity, measures the amount of positive pixels (foreground pixels) in the ground truth that are also identified as positives by the segmentation algorithm. The true negative rate (TNR), also called specificity, on the other hand, measures the portion of negative pixels (background pixels) in the ground truth segmentation that are correctly identified as such by the algorithm. The measures are defined as

$$TPR = \frac{TP}{TP + FN} \tag{25}$$

and

$$TNR = \frac{TN}{TN + FP} \tag{26}$$

where

- TP are the true positives, meaning pixels which are correctly classified to the foreground
- FN are false negatives, pixels that are incorrectly classified to the background

- TN are true negatives, pixels which are correctly assigned to background
- FP are false positives, meaning pixels that are incorrectly identified as foreground [64].

Calculation of TPR and TNR can be performed with the functions `tpr(label, prediction)` and `tnr(label, prediction)`, respectively.

Dice Coefficient
The Dice coefficient [65], also called Sorensen-Dice coefficient (DSC), is the most used metric for validating medical image segmentation. It is an overlap based metric. For a ground truth segmentation S_g and a predicted segmentation S_p, the DICE can be calculated as

$$DSC = \frac{2|S_g \cap S_p|}{|S_g| + |S_p|} \tag{27}$$

where $|S_g \cap S_p|$ is the intersection between ground truth segmentation and predicted segmentation. This intersection corresponds to the true positives TP. $|S_g|$ and $|S_p|$ denote the total amount of pixels classified to foreground in the ground truth and the prediction, respectively. The DSC takes values between 0 and 1, where 1 equals a perfect match. DSC calculation is implemented in the function `DSC_coeff(label, prediction)`.

Hausdorff Distance
The Hausdorff distance (HD) is a spatial distance based similarity measure, which means that the spatial position of pixels are taken into consideration. The Hausdorff distance between two point sets A and B is defined as

$$HD(A, B) = max(h(A, B), h(B, A)) \tag{28}$$

where $h(A, B)$ is the directed Hausdorff distance. It describes the maximal distance of point set A to the closest point in point set B. It's mathematical definition is

$$h(A, B) = \underset{a \in A}{max}\, \underset{b \in B}{min} \|a - b\| \tag{29}$$

where a and b are points of point set A and B respectively and $\|...\|$ is a norm, in example ℓ_2 norm to calculate Euclidian distance between the two points. A graphical representation of the Hausdorff distance and directed Hausdorff distance can be seen in figure 30. The Euclidian distance can be calculated by

$$\|a - b\|_2 = \sqrt{\sum_i (a_i - b_i)^2} \tag{30}$$

[66].

Calculation of the HD is included in the function `HD_distance(label, prediction)`. In this function, the indices of pixels classified as foreground are extracted from labels and predictions to produce the point sets A and B. These are then passed to the `directed_hausdorff(A, B)` function to calculate the directed Hausdorff distance. At first the algorithm iterates over all points found in point set A. Then, the Euclidian distance between the current point in point set A and all points in point set B is calculated. The minimal value of these Euclidian distances is then stored in a vector and the same is repeated for the next point in point set A, until all points have been evaluated. The directed Hausdorff distance is the maximal value of all distances stored in this vector. This distance is returned from the function. Directed Hausdorff distance is calculated from point set A to B as well as from B to A, and the greater of these two values is the final Hausdorff distance.

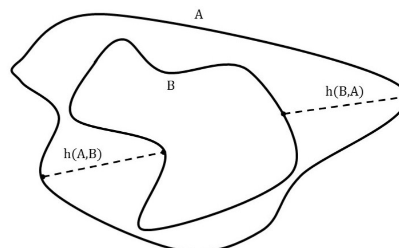


Figure 30: Graphical interpretation of Hausdorff distance. $h(A, B)$ is the distance between the most distant point of point set A from the closest point of point set B. For $h(B, A)$, it is opposite. HD is the maximum between $h(A, B)$ and $h(B, A)$. Adapted from [67].

6 Results and Evaluation

In this chapter, we present the results of our experiments. It starts with some examples taken from the training datasets to document the agreement between image data and generated labels, as well as the effects of data augmentation. In the second section 6.2, training and testing times using the two different architectures and different training datasets are evaluated. Segmentation results obtained from the various models described in this thesis are presented in section 6.3. Quantitative results are given in the form of various segmentation evaluation scores as well as their histograms to allow per-image evaluation. Furthermore, examples of qualitative segmentation results are shown.

6.1 Generation of Training and Testing Data

To illustrate the agreement between the ground truth labels obtained from thresholding the PET data and corresponding CT images, overlays between CT images and generated labels were produced. Some examples of these overlays can be seen in figure 31.

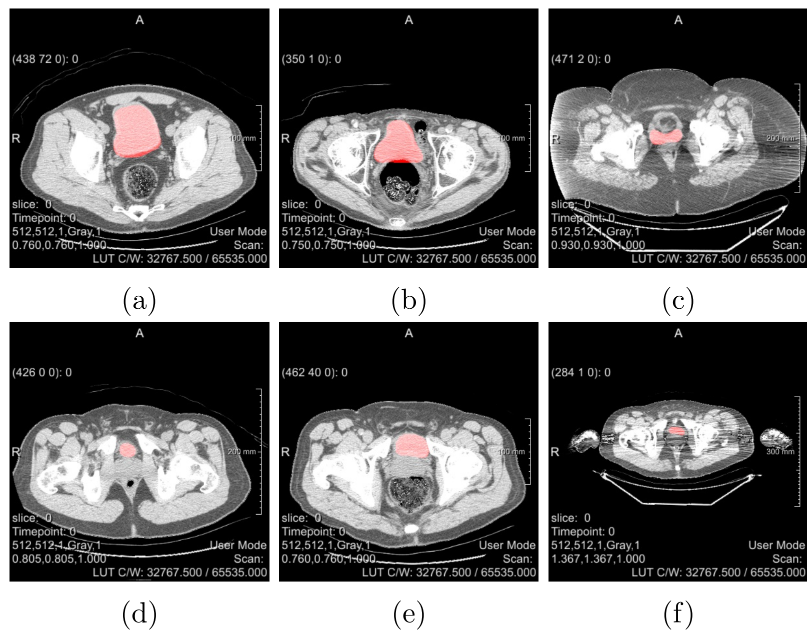
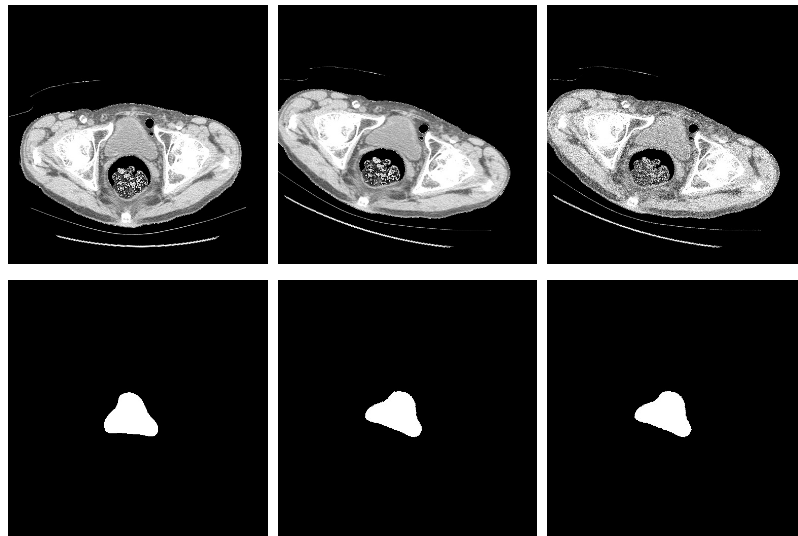


Figure 31: Examples of overlays between CT data and generated ground truth labels. The underlying CT images are shown in greyscale, while the ground truth labels obtained from PET segmentation are added in red.

By applying data augmentation with the default parameters specified in table 1 to the original 630 training images, a total amount of 34,020 augmented images and labels were obtained. This equals a magnification factor of the original dataset by 54. A magnification factor of 27 was achieved by the transformation, specifically the combination of rotations and scaling of the input images. The amount of transformed data was then doubled by the addition of zero-mean Gaussian noise on each image slice. The remaining noise types included in the `DataPreperation` MeVisLab macro module have not yet been explored. By fully exploiting the maximal parameters defined in table 1, a magnification factor of up to 1350 could be achieved, which would result in a total of 850,500 augmented image slices. However, since such large amounts of data are hard to handle with the available resources, such a large dataset was not created. A sample taken from the augmented training data is shown in figure 32.



(a) Original (b) Rotated and scaled (c) Added noise

Figure 32: Example for an augmented dataset. Image (a) shows the original CT image and ground truth label without augmentations. Image (b) shows the dataset after applying a rotation of 20° and scaling to a scale factor of 0.9 in x-direction and 1.1 in y-direction to both the CT image and the binary mask. In image (c), zero-mean Gaussian noise with a standard deviation of 500 was added to the transformed image. The binary mask remains unchanged by the addition of noise.

6.2 Training and Testing

Training was performed for 34,020 iterations (corresponding to the size of the largest training data set) on a server equipped with a NVIDIA Tesla K20Xm with 5 GB memory size. Testing was executed using a NVIDIA GeForce GTX 960 with 2 GB of memory.

On average, FCN models took 21 hours to train with images with a resolution of 256×256, while upsampled ResNet architectures took an average of 39 hours of training time with the same datasets. For images at their original resolution, training an FCN architecture required 71 hours on average, while for a ResNet architecture, 115 hours were needed. Table 3 gives a comparison over the time required to train our network models with the stated training data. Furthermore, the average time needed to perform inference, which is the average time needed to calculate a prediction for a single input image from the training dataset, is stated.

Table 3: Comparison of training and inference times. Training time is given in hours needed to complete the training process, inference time is averaged over all testing samples and given in milliseconds.

Network Model	Image Resolution	Training Data	Training Time (hours)	av. inference time (ms)
FCN	256×256	no augmentation	24	9.7
		transformed images	14	9.9
		fully augmented images	25	9.8
	512×512	no augmentation	56	18.4
		transformed images	86	17.7
upsampled ResNet	256×256	no augmentation	62	108.2
		transformed images	27	108.6
		fully augmented images	28	108.0
	512×512	no augmentation	99	219.4
		transformed images	130	222.6

A visualization of the development of cross-entropy loss during training obtained from TensorBoard can be seen in appendix B, figure 36.

6.3 Image Segmentation Results

Table 4 shows the results of segmentation evaluation for models trained with different training datasets at a resolution of 256×256. Table 5 presents the same metrics for segmentation results obtained from models with images at their original resolution. True positive rate, true negative rate, Dice coefficient, each in percent, and Hausdorff distance, in pixels, averaged over all 215 training datasets are listed.

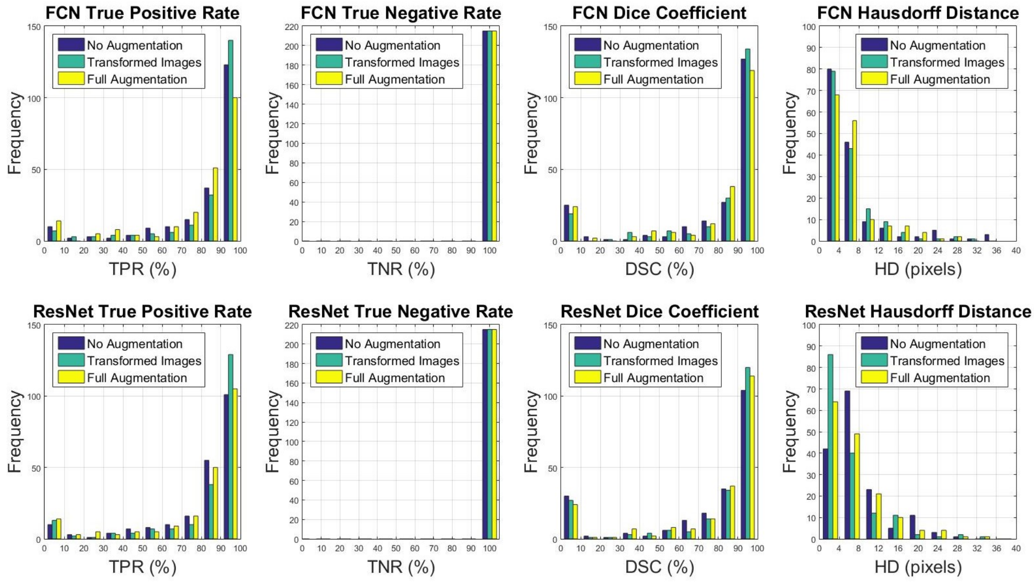
Table 4: Segmentation evaluation results for images rescaled to 256×256. This Table compares evaluation metrics for FCN and upsampled ResNet architectures trained using unaugmented training data, transformed training data (rotation, scaling) and fully augmented training data (transformations and zero-mean Gaussian noise).

Network Model	Training Data	mean TPR (%)	mean TNR (%)	mean DSC (%)	mean HD (pixel)
FCN	no augmentation	82.7	99.9	77.6	6.9
	transformed images	**85.0**	**99.9**	**80.4**	**6.1**
	fully augmented images	79.2	99.9	77.6	6.7
upsampled ResNet	no augmentation	80.7	99.9	73.5	7.9
	transformed images	82.5	99.9	76.9	6.3
	fully augmented images	79.7	99.9	76.7	7.7

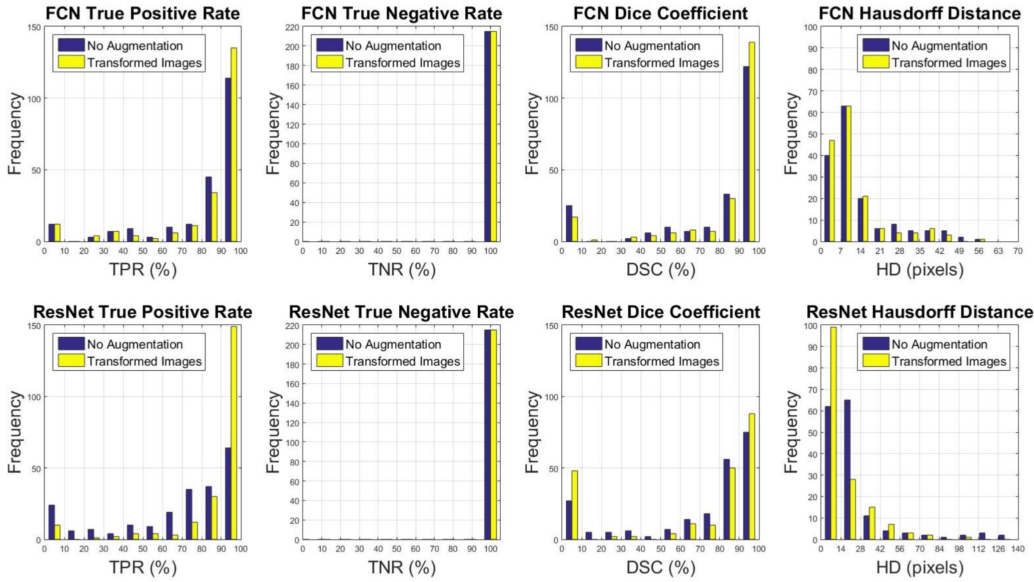
Table 5: Segmentation evaluation results for images of resolution 512×512. This Table compares evaluation metrics for FCN and upsampled ResNet architectures trained using unaugmented training data and transformed training data (rotation, scaling).

Network Model	Training Data	mean TPR (%)	meanTNR (%)	mean DSC (%)	mean HD (pixel)
FCN	no augmentation	80.9	99.9	77.6	13.3
	transformed images	**83.1**	**99.9**	**81.9**	**11.9**
upsampled ResNet	no augmentation	68.7	99.9	71.1	23.9
	transformed images	86.5	99.8	67.1	16.9

To visualize per-image evaluation scores of our models, the Histograms in figure 33 of evaluation metrics (true positive rate, true negative rate, Dice coefficient and Hausdorff distance) were plotted.



(a)



(b)

Figure 33: Histograms of evaluation metrics. Figure (a) shows the results for images of resolution 256×256, figure (b) for 512×512 images. The histograms plot the frequency of certain evaluation metric value ranges for TPR, TNR, DSC and HD and for both FCN and upsampled ResNet models and different data augmentation approaches.

Figures 34 and 35 show several representative examples of the obtained segmentation results for images downsampled to a resolution of 256×256 and images at original resolution of 512×512, respectively. For better illustration, original image data was overlaid with the contour of the ground truth in green as well as the prediction made by our deep networks in red. Original image, label and prediction data is shown in appendix C, figures 37 and 38.

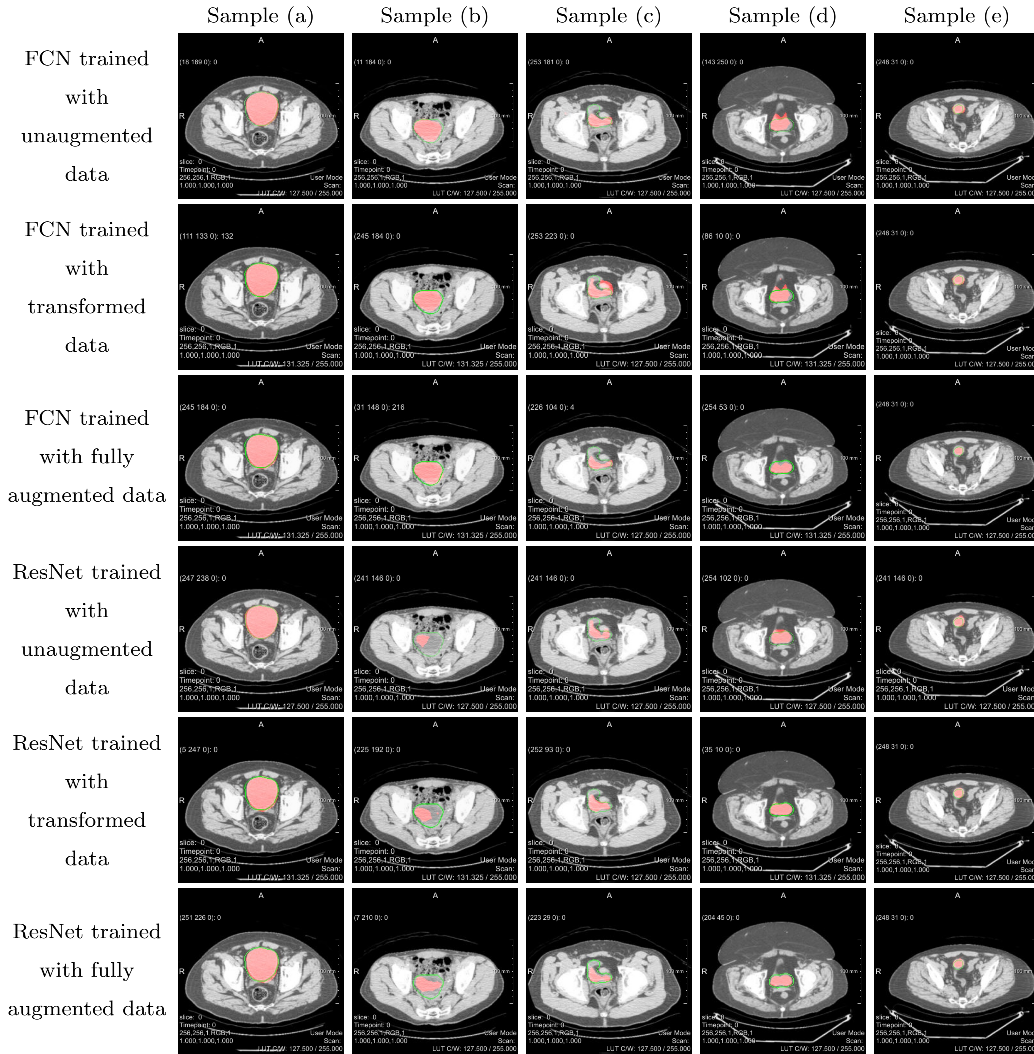


Figure 34: Qualitative segmentation result overlays for images scaled to 256×256. Ground truth labels are shown by the contours in green, the predictions made by the deep learning models are overlaid in red.

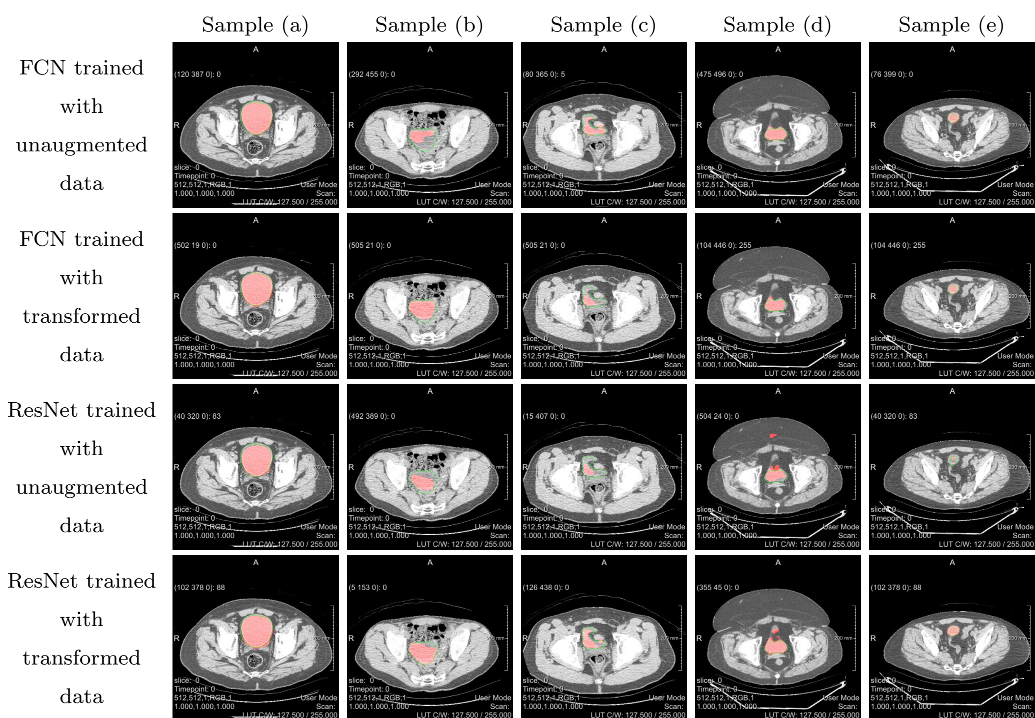


Figure 35: Qualitative segmentation result overlays for images with resolution 512×512. Ground truth labels are shown by the contours in green, the predictions made by the deep learning models are overlaid in red.

7 Discussion

Agreement between Image Data and Ground Truth

Concerning the evaluation of agreement between CT data and the ground truth labels generated from PET data, it can be observed that agreement is generally good, but not perfect. Examples are shown in figure 31. However, accuracy differs from dataset to dataset and even within individual slices. It can be observed that accuracy is worse in images were the urinary bladder only covers a small area surface of the image, like in image 31 (d) and (f). This is due to the nature of PET imaging, which has low spatial resolution and therefore, object boundaries might appear blurred. This is especially problematic when objects are small.

Figure 31 also shows some of the unique challenges one is confronted with when automatically segmenting the urinary bladder in CT images. It can be noted that size and position of the urinary bladder is varying between patients. In some image slices, for example in figure 31 (c), the shape of the urinary bladder highly differs from it's conventional, round form. Furthermore, low contrast between the bladder and surrounding soft tissue, as seen in figure 31 (b), poses a big difficulty. This especially occurs at the ambiguous bladder-prostate interface, as shown in figure 31 (e). It also becomes evident that not all CT data offers the same quality. In example, images 31 (c) and (f) show noticeable streak artefacts. Those artefacts are commonly found in CT scans and appear between dense objects like bone or metal due to beam hardening. Furthermore, since feature maps are significantly downsampled within our network architectures, images with a small area surface of the urinary bladder, as seen in figure 31 (f) might pose a problem, since small details could be lost as a result of downsampling.

Influence of Data Augmentation on Segmentation Performance

Segmentation evaluation results in tables 4 and 5 show that the application of data augmentation in the form of scaling and rotation to the original dataset does improve segmentation performance significantly. For mean TPR and DSC an increase of 2.3% and 2.8%, respectively, was achieved with the FCN architecture and training and testing images with resolution of 256×256. For data with resolution 512×512 a similar increase of 2.1% mean TPR and 4.3% mean DSC was achieved. With the upsampled ResNet architecture the corresponding enhancement was 1.8% and 3.5% with downsampled images or 17.8% TPR and a decrease of 4% DSC with original image resolution. Average Hausdorff distance was decreased by 0.8 pixels and 1.4 pixels in FCN

models, as well as 1.6 pixels and 7 pixels using upsampled ResNet for the 256×256 and 512×512 datasets, respectively. The mean true negative rate exhibited very high values regardless of the used network model and shows no significant variations between different training data sets. Although transformation of training data did increase segmentation performance, it can be noted that segmentation results obtained from the network trained with the original dataset consisting of only 630 images and labels are also quite satisfactory for models trained with downsampled image resolution. This shows that when using pre-trained networks, one can achieve passable results with only a small amount of training data.

From the evaluation scores achieved with models trained with fully augmented data in table 4, it can be seen that the addition of noise to the transformed training dataset does not improve the performance of of our proposed networks. In fact, all metrics show a worse performance of networks trained with artificially noisy training data compared to networks trained with only transformed training data. Using the FCN architecture, mean TPR even showed higher results when the network was only trained with 630 un-augmented training sets, with a TPR of 82.7% compared to 79.2% achieved with the fully augmented training set. The same is true for the upsampled ResNet architecture, although to a lesser extend, with an achieved TPR of 80.7% using un-augmented data compared to 79.7% using fully augmented data. One explanation for this could be that the applied Gaussian noise is not meaningful in the presented context. Therefore, the network learns spurious patterns that are not present in the training data. Another reason for the decrease in performance might be that the added noise is not strong enough. This results in the model seeing very similar images repeatedly, which might lead to overfitting. The model starts to fit too specific to the training set and loses it's ability to generalize to the new examples found in the testing set. Since we didn't obtain satisfactory results with the noisy training data, network architectures were not trained with this data at it's original resolution of 512×512.

Segmentation Results

The comparison of our FCN and upsampled ResNet architectures trained with differently augmented datasets in table 4 shows that in case of images rescaled to 256×256, best results can be achieved with the FCN architecture trained with images that are augmented with scaling and rotation. This network resulted in the highest true positive rate (85.0%) and Dice coefficient (80.4%) as well as the lowest Hausdorff distance (6.1 pixels). The true nega-

tive rate at 99.9% is the same for all tested models. The very high specificity indicates that our models are very accurate when it comes to correctly labelling background. However, this measure is highly dependent on segment size. Images with a lot of background, as it is the case in our examples, naturally show a higher TNR. Regardless of the used training data, the FCN architecture outperforms the upsampled ResNet architecture in all evaluation metrics. Moreover, the comparison of training and inference times in table 3 shows that our FCN architecture is significantly faster. On average, it took 21 hours to train our FCN models with images of resolution 256×256, while it took 39 hours to train the upsampled ResNet models with the same datasets. When calculating a prediction from a single input image, our FCN models show an average inference time of 9.8 ms, which is over 10 times faster than the average inference time of the upsampled ResNet architectures with an average inference time of 108.3 ms.

Inspecting the evaluation results using images at their original resolution of 512×512 in table 5, again, the FCN architecture generally performs better in terms of our evaluation metrics. Especially Dice coefficient is notably higher at 81.9% for FCN than for ResNet at 67.1% for our best performing models. Also, the Hausdorff distance is shorter by 5 pixels, indicating that our upsampled ResNet architecture produces more outliers. Only in terms of true positive rate, the ResNet architecture achieved better results with a TPR of 86.5 % compared to 83.1% for the FCN architecture. Again, training and inference times are also considerably lower for FCNs than for upsampled ResNets.

Looking at per-image evaluation of our models illustrated by histograms in figure 33, it is evident that frequency distribution of our evaluation metrics is similar for FCN and upsampled ResNet. It is still recognizable that FCN performance is in general slightly better than upsampled ResNet performance. Looking at the True Positive Rate, it can be seen that most testing images by far were segmented with a TPR between 90% and 100%. There is also a significant amount of testing sets with a TPR between 80% and 90%, which can also be considered a good result. However, a not negligible number of images resulted in a very low TPR between 0% and 10%. The same is even more clear when looking at the Dice coefficient. While most testing images resulted in a DSC of 80% or higher, up to more than 25 images from our testing dataset were segmented with a DSC between 0% and 10%. After inspecting our qualitative segmentation results, it became apparent that most of these incorrectly segmented images are false positives. Our models frequently produced a prediction for testing datasets in which the CT im-

age did not show a segment of the urinary bladder and correspondingly, the ground truth label was empty. Additionally, a couple of false negatives were produced, primarily in image slices were the urinary bladder had a small area surface plus the image possessed rather low soft tissue contrast. Looking at Hausdorff distance distribution, it can be seen that most testing datasets resulted in a HD of zero to four pixels. After that, frequency exponentially decreases. However, Hausdorff distance reaches very high values for up to 40 pixels in some images which indicates that our models sometimes produce outliers in the segmentation.

The qualitative segmentation results for images scaled to 256×256 shown in figure 34 illustrate that for input images with good soft tissue contrast, a large, homogeneous area surface and a regular shape of the urinary bladder, as seen in sample (a), all network models perform well. In sample (b), contrast between the urinary bladder and surrounding tissue is not ideal, moreover, the bladder itself includes varying grey values, meaning that the area surface is not homogeneous. It is evident that while our FCN architecture has no trouble in detecting the urinary bladder in these images, the upsampled ResNet architecture performs poorly. Apparently the upsampled ResNet models are more sensitive against contrast and grey values. However, it can be seen from sample (c), that the ResNet models are better in adapting to distinct shapes. Sample (d) is interesting because our generated ground truth annotation does not follow the very unusual shape of the bladder very well in this example. While the ResNet models seem to fit better to the ground truth label, the segmentation predicted by the FCN models, especially the network trained with transformed data, seems to correspond better to the actual outline of the bladder. Sample (e) shows, that despite our initial concerns, the proposed models are able to identifying the urinary bladder when only a small portion of it is visible in a slice, as long as contrast is good. In fact, the predictions made by our models in some cases even follow the outline of the urinary bladder better than our underlying ground truth segmentation.

The same observations can be made when looking at the qualitative segmentation results for images of higher resolution in figure 35. For input images of high quality, both architectures perform well. FCN does better when segmenting images with low contrast and inhomogeneous grey values as seen in sample (b), while ResNet adapts better to unusual shapes as in sample (c). Again, sample (d) allows for some very interesting observations. Here, our upsampled ResNet architecture does a very good job in detecting the urinary bladder, even recognising the small, detached portion of the bladder

at the top. The FCN architecture also produces a more accurate segmentation than our underlying ground truth, but in this case, upsampled ResNet trained with augmented data performs very well. It is also notable that qualitative results for upsampled ResNet trained with unaugmented data of images with resolution 512×512 are worst amongst all achieved predictions. Segmentation results appear very uneven and edged, also they show a lot of outliers which is supported by the high Hausdorff distance of averagely 23.9 pixels for this model. Obviously, a network for higher resolution images also has more parameters that need to be tuned, and in this case, the 630 unaugmented training images apparently did not provide sufficient information to specify all these parameters correctly.

It can be noted that while our networks trained with images of higher resolution don't necessarily show a better segmentation performance in terms of our evaluation metrics, as seen in tables 4 and 5, qualitative results are to some extend much better for images with resolution 512×512, especially for network models trained with augmented data. The reason for this is our non-perfect ground truth. In many cases, predictions made by our models don't fit the ground truth we compare it to accurately, which results in low evaluation scores. Nevertheless, looking at the image data one can see that the predictions correspond well with the actual outline of the urinary bladder. This subjectively improved segmentation performance comes at the cost of significantly longer training times. With an average training time of 71 hours, training a FCN architecture with images of full resolution requires more than three times the training time than with images of downsampled resolution. The same applies to training the upsampled ResNet architecture, where 130 hours were required to train the network with 512×512 images.

7.1 Conclusion and Future Outlook

We introduced an approach to generate suitable training and testing datasets for deep learning algorithms by exploiting ^{18}F-FDG accumulation in the urinary bladder to produce ground truth labels, and by the application of data augmentation to enlarge a small dataset. Although in general, our method for generating ground truth labels from PET data produces good results, the agreement of image data with ground truth labels obtained by this approach is most certainly not as high as with manually created ground truth segmentations obtained from experienced physicians. Obviously, these non-perfect labels influence the results produced by our segmentation networks. It is shown in this thesis that when comparing segmentation results with these non-perfect ground truth labels, evaluation metrics like Dice coefficient and

true positive rate do not suffice to assess model performances, instead, qualitative evaluation by comparison with the underlying image data is preferable. Again, manually created ground truth labels would be beneficial to calculate more representative evaluation scores. However, since already segmented medical image data is very rare and databases are generally very small, our method is promising for creating a good starting position for training and testing. In future work, it would be interesting to re-train our networks with one or two accurately, manually segmented image datasets to analyse whether the segmentation performance increases, and to compare segmentation results with manually created labels.

We demonstrated that training data augmentation in the form of transformations, like rotation and scaling, can significantly improve the performance of segmentation networks, however, the addition of zero-mean Gaussian noise to the training data did not result in an enhanced performance in our case. Subsequent work could go into further exploring the effects of data augmentation on the segmentation results, by generating even bigger augmented datasets and by applying different noise types to the original image data.

Furthermore, we implemented and compared two different well-known deep learning models for semantic image segmentation and tested them on our created datasets. Our qualitative results show that the proposed segmentation methods can accurately segment the urinary bladder in CT images and are in many cases more accurate than the ground truth labels obtained from PET image data. It is shown that the used FCN architecture generally performs better in terms of evaluation metrics than the proposed ResNet architecture. We achieved the best segmentation performance with our FCN network which was trained with transformed image data. A proposal for future work is the implementation of post-processing algorithms. In many publications, including in [51] by Che et al., fully-connected conditional random fields are used to accurately recover object boundaries that are smoothed within the deep neural network. In our case, this might especially improve performance in cases were the urinary bladder has irregular, distinct shapes.

Semantic image segmentation using deep learning algorithms is an ongoing topic of research. Therefore, new network architectures for semantic segmentation are frequently introduced. Most of these architectures are based on the approaches used in this thesis and rely on a deep contracting path, followed by a symmetric expansive path that performs upsampling in many steps, for example U-Net [37], SegNet [39] or, more recently, the networks introduced in [68]. One final proposal for future work is to apply such incom-

ing architectures to our dataset. However, unless pre-trained networks can be usefully integrated in these architectures, larger image datasets and more computational power are probably required to train such networks efficiently.

References

[1] Dzung L Pham, Chenyang Xu, and Jerry L Prince. Current methods in medical image segmentation 1. *Annual review of biomedical engineering*, 2(1):315–337, 2000. 1, 23

[2] Isaac Bankman. *Handbook of medical image processing and analysis.* academic press, 2008. 1

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 35

[4] World cancer research fund international. `http://www.wcrf.org/`. Accessed: 2017-10-09. 1, 5, 6

[5] Feng Shi, Jie Yang, and Yue-min Zhu. Automatic segmentation of bladder in ct images. *Journal of Zhejiang University-Science A*, 10(2):239–246, 2009. 1, 35

[6] Kenny H Cha, Lubomir Hadjiiski, Ravi K Samala, Heang-Ping Chan, Elaine M Caoili, and Richard H Cohan. Urinary bladder segmentation in ct urography using deep-learning convolutional neural network and level sets. *Medical physics*, 43(4):1882–1896, 2016. 1, 6, 35

[7] Zhen Ma, Renato Natal Jorge, and João Manuel RS Tavares. A comparison between segmentation algorithms for urinary bladder on t2-weighted mr images. *Computational Vision and Medical Image Processing: VipIMAGE 2011*, page 371, 2011. 1, 7

[8] Omar Faiz, Simon Blackburn, and David Moffat. *Anatomy at a Glance*, volume 66. John Wiley & Sons, 2011. 3, 4

[9] Richard Cohan. Imaging and urodynamics of the lower urinary tract, 2012. 4

[10] Heinz Feneis and Wolfgang Dauber. *Pocket Atlas of Human Anatomy.* Thieme, 2000. 4

[11] Sergio Bracarda, Ottavio de Cobelli, Carlo Greco, Tommaso Prayer-Galetti, Riccardo Valdagni, Gemma Gatta, Filippo de Braud, and Georg Bartsch. Cancer of the prostate. *Critical reviews in oncology/hematology*, 56(3):379–396, 2005. 5

[12] Michael J Zelefsky, Zvi Fuks, Laura Happersett, Henry J Lee, C Clifton Ling, Chandra M Burman, Margie Hunt, Theresa Wolfe, ES Venkatraman, Andrew Jackson, et al. Clinical experience with intensity modulated radiation therapy (imrt) in prostate cancer. *Radiotherapy and Oncology*, 55(3):241–249, 2000. 5

[13] Xiangfei Chai, Marcel van Herk, Anja Betgen, Maarten Hulshof, and Arjan Bel. Automatic bladder segmentation on cbct for multiple plan art of bladder cancer using a patient-specific bladder model. *Physics in medicine and biology*, 57(12):3945, 2012. 6

[14] Oliver W Hakenberg, Clemens Linne, Andreas Manseck, and Manfred P Wirth. Bladder wall thickness in normal adults and men with mild lower urinary tract symptoms and benign prostatic enlargement. *Neurourology and urodynamics*, 19(5):585–593, 2000. 7

[15] Carlo Manieri, Simon St C Carter, Gianfranco Romano, Alberto Trucchi, Marco Valenti, and Andrea Tubaro. The diagnosis of bladder outlet obstruction in men by ultrasound measurement of bladder wall thickness. *The Journal of urology*, 159(3):761–765, 1998. 7

[16] Arnulf Oppelt. *Imaging systems for medical diagnostics*. Publicis MCD, 2005. 8, 11

[17] Rebecca Smith-Bindman, Jafi Lipson, Ralph Marcus, Kwang-Pyo Kim, Mahadevappa Mahesh, Robert Gould, Amy Berrington De González, and Diana L Miglioretti. Radiation dose associated with common computed tomography examinations and the associated lifetime attributable risk of cancer. *Archives of internal medicine*, 169(22):2078–2086, 2009. 8

[18] Christina Gsaxner. Exploit 18f-fdg enhanced urinary bladder in pet data for deep learning ground truth generation in ct scans, 2017. Project Report. 9, 26, 44

[19] Evelina Miele, Gian Paolo Spinelli, Federica Tomao, Angelo Zullo, Filippo De Marinis, Giulia Pasciuti, Luigi Rossi, Federica Zoratto, and Silverio Tomao. Positron emission tomography (pet) radiotracers in oncology–utility of 18f-fluoro-deoxy-glucose (fdg)-pet in the management of patients with non-small-cell lung cancer (nsclc). *Journal of Experimental & Clinical Cancer Research*, 27(1):52, 2008. 9

[20] Prantika Som, Harold L Atkins, et al. A fluorinated glucose analog, 2-fluoro-2-deoxy-d-glucose (f-18): nontoxic tracer for rapid tumor detection. *J Nucl Med*, 21(7):670–675, 1980. 9

[21] Sazilah Ahmad Sarji. Physiological uptake in fdg pet simulating disease. 2006. 9

[22] Justin K Moran, Hyo B Lee, and M Donald Blaufox. Optimization of urinary fdg excretion during pet imaging. *The Journal of Nuclear Medicine*, 40(8):1352, 1999. 10

[23] Miles N Wernick and John N Aarsvold. *Emission tomography: the fundamentals of PET and SPECT*. Academic Press, 2004. 10, 11

[24] William W Moses. Fundamental limits of spatial resolution in pet. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 648:S236–S240, 2011. 10

[25] Paul E Kinahan and James W Fletcher. Positron emission tomography-computed tomography standardized uptake values in clinical practice and assessing response to therapy. In *Seminars in Ultrasound, CT and MRI*, volume 31, pages 496–505. Elsevier, 2010. 10

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 12

[27] Shaohua Kevin Zhou, Hayit Greenspan, and Dinggang Shen. *Deep Learning for Medical Image Analysis*. Academic Press, 2017. 12, 13, 15, 17, 18, 21

[28] Andrej Krenker, Andrej Kos, and Janez Bešter. *Introduction to the artificial neural networks*. INTECH Open Access Publisher, 2011. 13

[29] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006. 16, 17, 19, 20, 21

[30] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012. 18

[31] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988. 18

[32] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012. 19

[33] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995. 20, 21, 22

[34] C Rafael Gonzalez and Richard Woods. *Digital image processing*. 2002. 23

[35] Li Deng and Dong Yu. Deep learning: Methods and applications. Technical report, May 2014. 25

[36] Dan Ciresan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012. 26

[37] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015. 26, 77

[38] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 26, 39, 40, 55, 58

[39] Vijay Badrinarayanan, Ankur Handa, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015. 26, 77

[40] Mevislab getting started guide. `http://mevislabdownloads.mevis.de/docs/current/MeVisLab/Resources/Documentation/Publish/SDK/GettingStarted.pdf`. Accessed: 2017-10-09. 26

[41] Mevislab reference manual. `http://mevislabdownloads.mevis.de/docs/current/MeVisLab/Resources/Documentation/Publish/SDK/MeVisLabManual.pdf`. Accessed: 2017-10-09. 26

[42] Jan Egger, Junichi Tokuda, Laurent Chauvin, Bernd Freisleben, Christopher Nimsky, Tina Kapur, and William Wells. Integration of the openigtlink network protocol for image-guided therapy with the medical

platform mevislab. *The international Journal of medical Robotics and Computer assisted Surgery*, 8(3):282–290, 2012. 26

[43] Jan Egger, Markus Gall, Jürgen Wallner, Pedro Boechat, Alexander Hann, Xing Li, Xiaojun Chen, and Dieter Schmalstieg. Htc vive mevislab integration via openvr for medical applications. *PloS one*, 12(3):e0173972, 2017. 26

[44] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 29

[45] Tensorflow developement guide. `https://www.tensorflow.org/get_started/`. Accessed: 2017-10-09. 29

[46] Benjamin Haas et al. Automatic segmentation of thoracic and pelvic ct images for radiotherapy planning using implicit anatomic knowledge and organ-specific segmentation strategies. *Physics in medicine and biology*, 53(6):1751, 2008. 35

[47] Maria Jimena Costa, Hervé Delingette, and Nicholas Ayache. Automatic segmentation of the bladder using deformable models. In *Biomedical Imaging: From Nano to Macro, 2007. ISBI 2007. 4th IEEE International Symposium on*, pages 904–907. IEEE, 2007. 35

[48] Kenny H Cha, Lubomir M Hadjiiski, Ravi K Samala, Heang-Ping Chan, Richard H Cohan, and Elaine M Caoili. Comparison of bladder segmentation using deep-learning convolutional neural network with and without level sets. In *SPIE Medical Imaging*, pages 978512–978512. International Society for Optics and Photonics, 2016. 35

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 36

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 37, 38

[51] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016. 41, 77

[52] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 41

[53] Samuel G Armato, Charles R Meyer, Michael F McNitt-Gray, Geoffrey McLennan, Anthony P Reeves, Barbara Y Croft, and Laurence P Clarke. The reference image database to evaluate response to therapy in lung cancer (rider) project: A resource for the development of change-analysis software. *Clinical Pharmacology & Therapeutics*, 84(4):448–456, 2008. 43

[54] National cancer imaging archive. `http://www.cancerimagingarchive.net/`. Accessed: 2017-10-09. 43

[55] Jan Egger, Tina Kapur, Andriy Fedorov, Steve Pieper, James V Miller, Harini Veeraraghavan, Bernd Freisleben, Alexandra J Golby, Christopher Nimsky, and Ron Kikinis. Gbm volumetry using the 3d slicer medical image computing platform. *Scientific reports*, 3, 2013. 43

[56] Christina Gsaxner, Birgit Pfarrkirchner, Lydia Lindner, Jürgen Wallner, Dieter Schmalstieg, and Jan Egger. Exploit 18f-fdg enhanced urinary bladder in pet data for deep learning ground truth generation in ct scans. 2018. 44

[57] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? In *Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on*, pages 1–6. IEEE, 2016. 44

[58] cgsaxner at github. `https://github.com/cgsaxner`. Accessed: 2017-10-16. 45

[59] Tensorflow-slim image classification library. `https://github.com/tensorflow/models/tree/master/research/slim`. Accessed: 2017-10-09. 55

[60] Daniil Pakhomov, Vittal Premachandran, Max Allan, Mahdi Azizian, and Nassir Navab. Deep residual learning for instrument segmentation in robotic surgery. *arXiv preprint arXiv:1703.08580*, 2017. 55, 58

[61] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 60

[62] Gabriela Csurka, Diane Larlus, Florent Perronnin, and France Meylan. What is a good evaluation measure for semantic segmentation?. In *BMVC*, volume 27, page 2013. Citeseer, 2013. 62

[63] Abdel Aziz Taha and Allan Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. *BMC medical imaging*, 15(1):29, 2015. 62

[64] Kai Ming Ting. *Sensitivity and Specificity*, pages 901–902. Springer US, Boston, MA, 2010. 63

[65] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945. 63

[66] Ralph Tyrrell Rockafellar and Roger J-B Wets. *Variational analysis*, volume 317. Springer Science & Business Media, 2009. 64

[67] Filippo Molinari, Guang Zeng, and Jasjit S Suri. A state of the art review on intima–media thickness (imt) measurement and wall segmentation techniques for carotid ultrasound. *Computer methods and programs in biomedicine*, 100(3):201–221, 2010. 64

[68] Gabriel L Oliveira, Claas Bollen, Wolfram Burgard, and Thomas Brox. Efficient and robust deep networks for semantic segmentation. *The International Journal of Robotics Research*, page 0278364917710542, 2017. 77

A Dataset Overview

Table 6: Used datasets obtained from RIDER PET/CT. The name of the datasets, the used PET and CT data as well as the number of total slices and the number of slices showing the urinary bladder are stated.

RIDER Dataset	PET	CT	Total Slices	Bladder Slices
1284094278	PET FDG SUV	CT 2.5MM STD	358	40
1542248368	PET FDG SUV	CT 2.5MM STD	351	35
1887858289	PET FDG SUV	CT 2.5MM STD	354	35
1940675042	PET FDG SUV	CT 2.5MM STD	354	40
2069446030	PET FDG SUV	CT 2.5MM STD	291	25
2112049538	PET FDG SUV	CT 2.5MM STD	295	30
217238498-1	PET FDG SUV	CT 2.5MM STD	293	20
217238498-2	PET FDG SUV	CT 2.5MM STD	291	25
2189009649	PET FDG SUV	CT 2.5MM STD	293	20
2310941115	PET FDG SUV	CT 2.5MM STD	291	30
2414443006-1	PET FDG SUV	CT 2.5MM STD	293	25
2414443006-2	PET FDG SUV	CT 2.5MM STD	298	25
2479814957	PET FDG SUV	CT 2.5MM STD	298	35
2491061956	PET FDG SUV	Recon 2: CTAC 2.5 THICK	291	25
2609389147-1	PET FDG SUV	CT 2.5MM STD	292	35
2609389147-2	PET FDG SUV	CT 2.5MM STD	291	25
2609389147-3	PET FDG SUV	CT 2.5MM STD	297	25
2610856938	PET FDG SUV	CT 2.5MM STD	294	25
2624615528	PET FDG SUV	CT 2.5MM STD	358	65
2736200846	PET FDG SUV	CT 2.5MM STD	338	25
2766484014	PET FDG SUV	CT 2.5MM STD	295	20
2779755504	PET FDG SUV	CT 2.5MM STD	304	20
2852173628	PET FDG SUV	CT 2.5MM STD	295	25
2871069045	PET FDG SUV	CT 2.5MM STD	291	25
3117097391	PET FDG SUV	CT 2.5MM STD	303	25
3270637687	PET FDG SUV	CT 2.5MM STD	296	20
3640513565	PET FDG SUV	CT 2.5MM STD	306	30
5572739776	PET FDG SUV	CT 2.5MM STD	295	35
6525572907	PET FDG SUV	Recon 2: CTAC 2.5 THICK	148	35

B Loss Development during Training

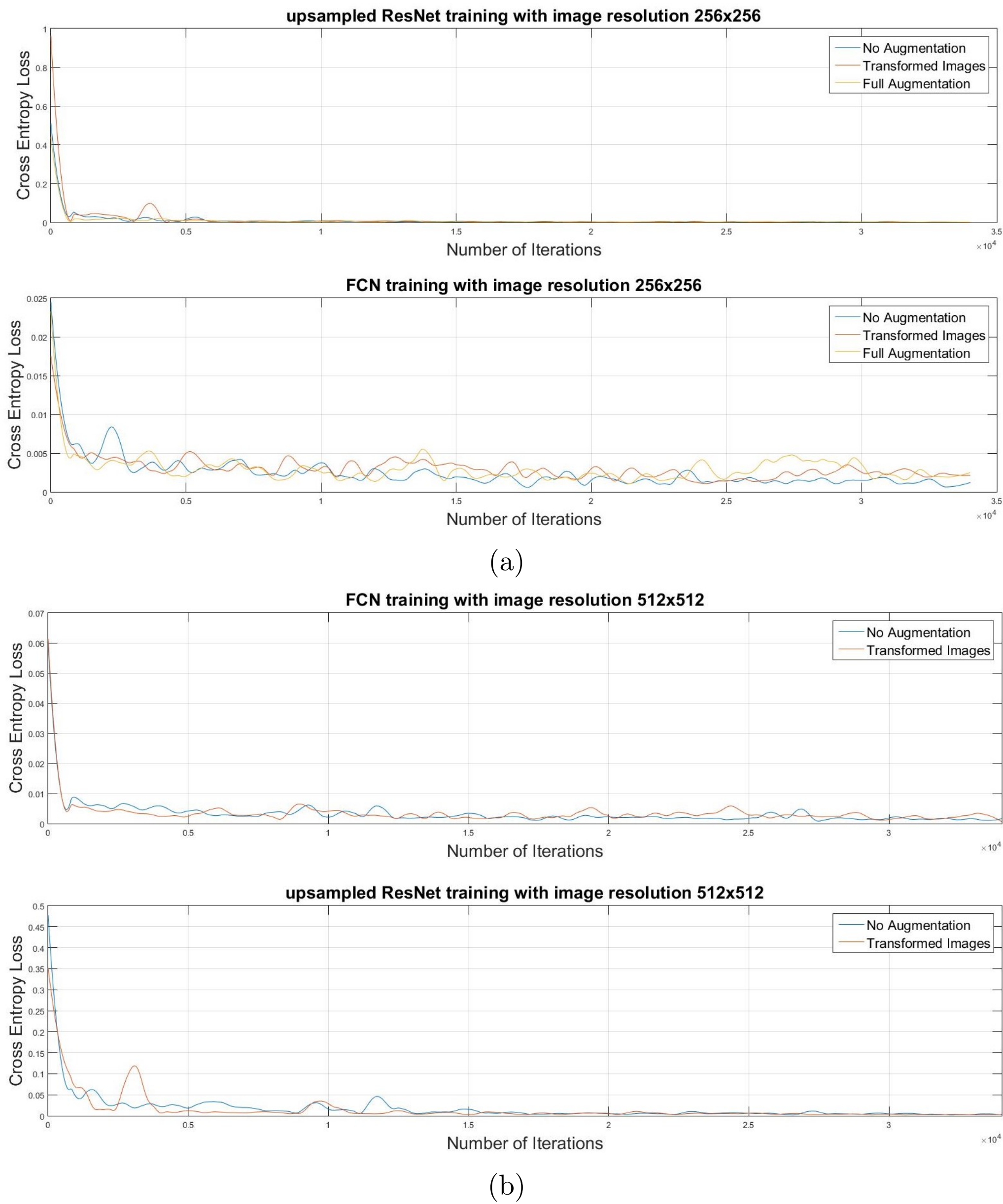


Figure 36: Development of the cross entropy loss during training. Figure (a) displays the development of loss for training with images of resolution 256×256, while figure (b) shows the same for 512×512 image resolution. Loss development is shown over the whole training process (34,020 iterations).

C Seperate Segmentation Results

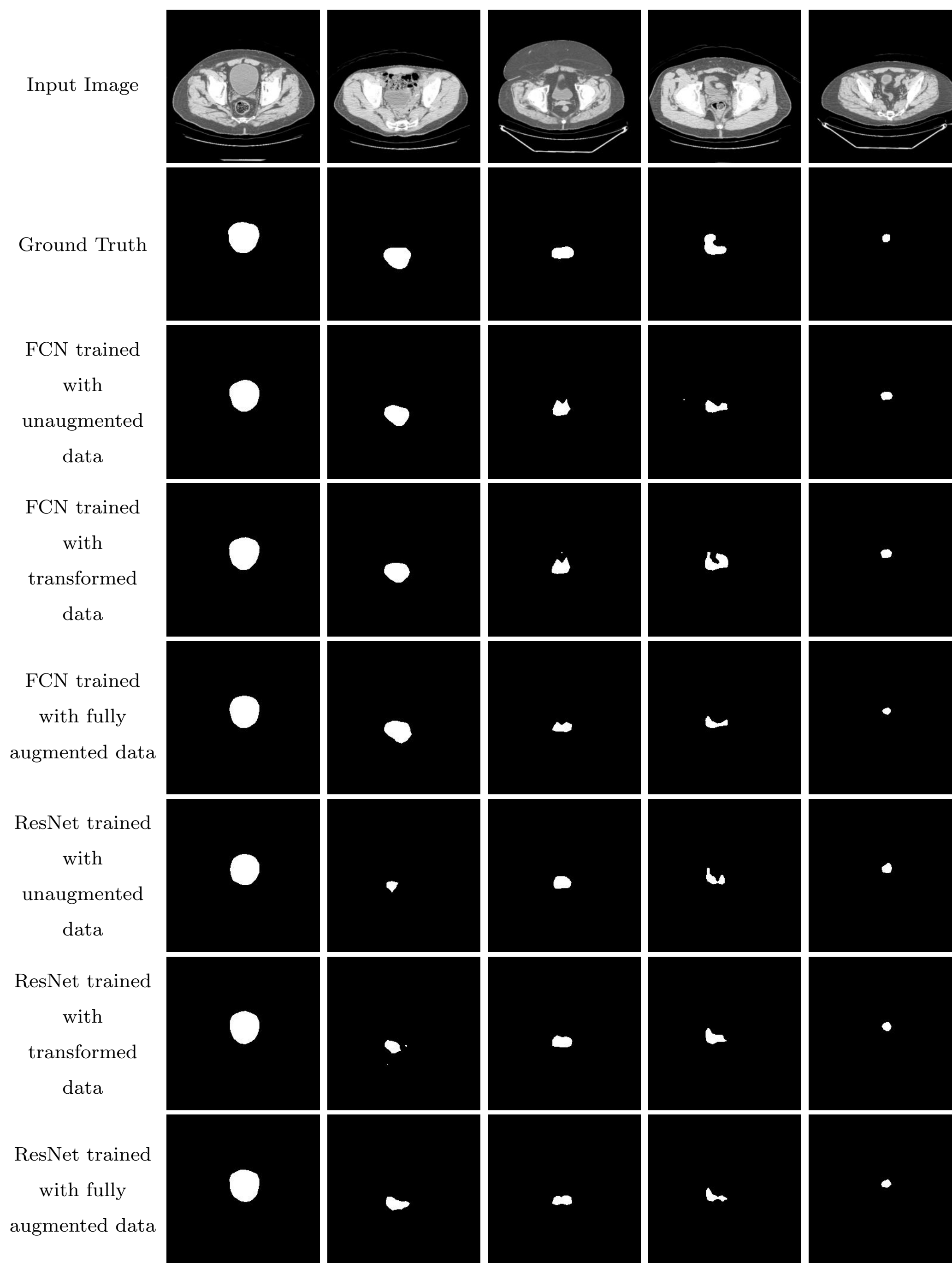


Figure 37: Segmentation results for images scaled to 256×256.

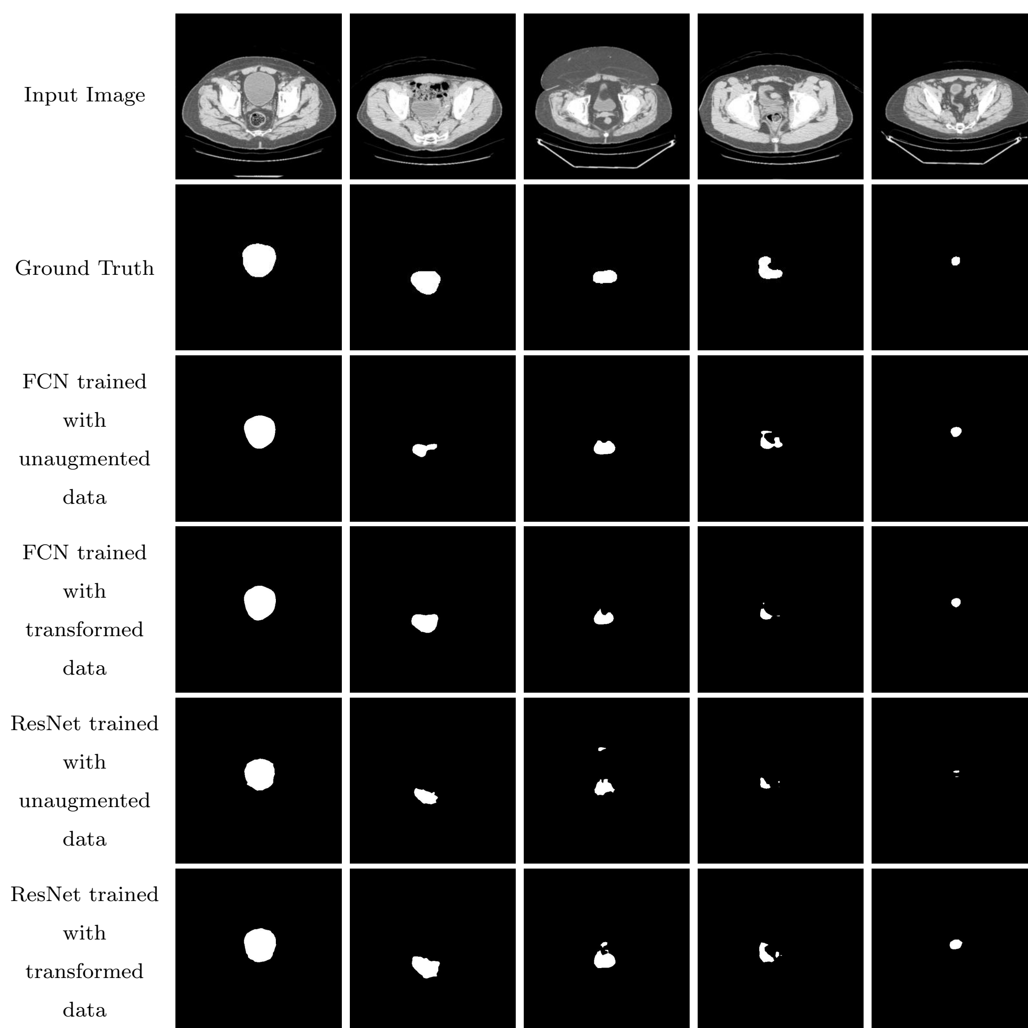


Figure 38: Segmentation results for images with original resolution of 512×512.