Lydia Lindner, BSc

Automatic Brain Tumor Segmentation using Deep Neural Networks Trained with Synthetic MRI Data

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

September, 2018

In cooperation with

Computer Algortihms for Medicine Laboratory

Graz, Austria

University Hospital of Giessen
Department of Neurosurgery

Giessen, Germany

Abstract

In this master's thesis, an approach for fully automatic brain tumor segmentation is presented, utilizing a deep convolutional neural network trained with synthetically generated data. In particular, this work focuses on the segmentation of glioblastoma multiforme in MRI data. Medical image processing is an interdisciplinary field of research, which combines a large spectrum of knowledge from applied mathematics, computer science, engineering, statistics and medicine. In clinical practice, brain tumor segmentation is one of the most challenging tasks and the utility of manually obtained segmentations is limited, due to them being subjective, time-consuming and not reproducible. Following the success of deep learning methods in other real-world domains, they are now also attracting increasing interest in the field of medical image processing, since they provide an objective, robust and fast automated segmentation technique. A large dataset and the associated ground truth data are crucial for a high performance of deep learning based segmentation approaches, however, in domains like medicine, only limited data is available in the majority of cases. For this thesis, MR images of 14 patients suffering from glioblastoma multiforme were provided. Manual slice-by-slice segmentation has been performed by clinical experts to obtain the corresponding ground truth. Since such a small dataset is not sufficient to train a deep neural network, an approach for generating brain MR images containing synthetic tumors is presented. Furthermore, a modified U-Net architecture for automatic segmentation of the lesion is introduced. The presented network was trained solely with synthetic MRI data and was subsequently tested on real MR images of patients suffering from glioblastoma multiforme. With the proposed method, it was possible to obtain a Dice similarity coefficient of 0.824 and a Hausdorff distance of 9.002 on the test set.

Keywords: Deep Learning, Semantic Image Segmentation, Synthetic Brain Tumor, Glioblastoma Multiforme, Convolutional Neural Networks, Magnetic Resonance Imaging

Kurzfassung

In dieser Masterarbeit wird ein Ansatz zur vollautomatischen Hirntumorsegmentierung vorgestellt, bei dem ein tiefes, mit synthetisch erzeugten Daten trainiertes neuronales Netzwerk verwendet wird. Insbesondere konzentriert sich diese Arbeit auf die Segmentierung des Glioblastoma multiforme in MRT-Bildern. Die medizinische Bildverarbeitung ist ein interdisziplinäres Forschungsgebiet, das ein breites Spektrum an Kenntnissen aus der angewandten Mathematik, der Informatik, dem Ingenieurwesen, der Statistik und der Medizin vereint. In der klinischen Praxis ist die Segmentierung von Hirntumoren eine der schwierigsten Aufgaben und der Nutzen manueller Segmentierungen ist begrenzt, da diese subjektiv, zeitaufwändig und nicht reproduzierbar sind. Nach dem Erfolg von Deep-Learning-Methoden in anderen Bereichen gewinnen diese nun auch im Bereich der medizinischen Bildverarbeitung zunehmend an Bedeutung, da sie eine objektive, reproduzierbare und schnelle vollautomatische Segmentierungstechnik bieten. Ein umfangreicher Datensatz inklusive zugehöriger Segmentierungsmasken ist entscheidend für die Leistungsfähigkeit von Deep-Learning-basierten Segmentierungsansätzen. Im medizinischen Bereich sind jedoch in den meisten Fällen nur begrenzte Datenmengen verfügbar. Für diese Arbeit standen MR-Aufnahmen von 14 Patienten mit Glioblastoma multiforme zur Verfügung. Eine manuelle Slice-by-Slice Segmentierung wurde von klinischen Experten durchgeführt, um die entsprechenden Segmentierungsmasken zu erhalten. Da solch ein kleiner Datensatz allerdings nicht ausreicht, um ein tiefes neuronales Netzwerk zu trainieren, wird ein Ansatz zur Erzeugung von synthetischen MR-Bildern des Gehirns präsentiert, die künstlich generierte Tumore enthalten. Darüber hinaus wird eine modifizierte U-Net-Architektur zur automatischen Segmentierung der Läsion vorgestellt. Das Netzwerk wurde ausschließlich mit synthetischen MRT-Daten trainiert und anschließend mit realen MR-Bildern von Patienten mit Glioblastoma multiforme getestet. Mit dem präsentierten Verfahren wurden am Test-Datensatz ein Dice-Koeffizient von 0,824 und ein Hausdorff-Abstand von 9,002 erreicht.

Keywords: Deep Learning, Semantische Bildsegmentierung, Synthetischer Hirntumor, Glioblastoma Multiforme, Convolutional Neural Networks, Magnetresonanztomographie

Acknowledgements

I am gratefully indebted to all those who directly or indirectly contributed to the work presented in this thesis.

In particular, I would like to express my sincerest gratitude to my advisor Dr.rer.physiol. Dr.rer.nat. Jan Egger for the knowledge and constant support he has provided. His friendly guidance, encouragement and expert advice have been invaluable throughout all stages of this work. I have greatly benefited from the illuminating discussions with him, which have led me to the final completion of this thesis and gave me insights into research and scientific working in general.

Furthermore, I am very much obliged to my supervisor Univ.-Prof. Dipl.-Ing. Dr.techn. Dieter Schmalstieg for the opportunity to work in his group at the Institute of Computer Graphics and Vision. The excellent working facilities and resources he provided, greatly helped me to complete this thesis.

I would also like to offer my special thanks to Dr. med. Malgorzata Kolodziej from the University Hospital of Giessen for providing her medical expertise and evaluating the quality of the generated synthetic MRI data.

Additionally, I want to thank the University Hospital of Giessen and Marburg in Germany and the Human Connectome Project, WU-Minn Consortium (Principal Investigators: David Van Essen and Kamil Ugurbil) funded by the 16 NIH Institutes and Centers that support the NIH Blueprint for Neuroscience Research; and by the McDonnell Center for Systems Neuroscience at Washington University, for providing access to the data used in this work. I also received generous support from the interdisciplinary research institution CAFÉ lab - Computer Algorithms for Medicine Laboratory.

Finally, I must express my deepest heartfelt gratitude to my parents and to my partner, for providing me with unfailing support and continuous encouragement, love and enthusiasm throughout my years of study and during the accomplishment of this thesis. I undoubtedly could not have done this without you.

Thank you very much!

Contents

Abbreviations

2D	Two-dimensional
3D	Three-dimensional
ANN	Artificial neural network
API	Application programming interface
BN	Batch normalization
BSE	Brain surface extraction
CNN	Convolutional neural network
CNS	Central nervous system
CT	Computed tomography
DSC	Dice similarity coefficient
FCN	Fully convolutional network
FP	False positives
FN	False negatives
GBM	Glioblastoma multiforme
GPU	Graphics processing unit
HD	Hausdorff distance
MRI	Magnetic resonance imaging
NMR	Nuclear magnetic resonance
NSF	Nephrogenic systemic fibrosis
ReLU	Rectified linear unit
RF	Radio frequency
TN	True negatives
TP	True positives

1 Introduction

1.1 Motivation

Magnetic resonance imaging (MRI) is a non-invasive technique for producing detailed images of the human brain with good soft-tissue contrast and is widely used in clinical studies and diagnostics [1]. Key aspects of clinical neuro-oncology and diagnostic medicine in general are detection and localization of lesions, diagnosis, treatment planning and monitoring treatment response. It is vitally important to segment brain tumors before starting any therapy, in order to ensure that solely tumor cells are destroyed during treatment, while healthy brain tissue is left undamaged [2]. In daily clinical practice, evaluation and interpretation of MR images is typically conducted manually by medical experts, such as physicians or radiologists, who assess MR images based on qualitative criteria (for example, characteristic hyper-intense tissue occurance in post-contrast T1-weighted MR images), or by relying on simple quantitative measures such as the largest axial diameter of the tumor [3]. However, the survival rate of a patient can be significantly improved by early and precise detection and localization of the lesion and by accurate disease staging; hence, fully automatic image segmentation tools would be of considerable value for improved primary diagnosis of brain tumors, individual treatment plannig and evaluation of therapy results [4]. Segmentation of a lesion is usually the first step in an image processing chain and affects all subsequent steps significantly.

Manual segmentation done by medical experts is labor-intensive and prone to human error, misinterpretation and observer bias, which leads to intra- and inter-observer variability [5]. Therefore, it would be of enormous value to replace the current manual assessment with automatic segmentation techniques that ensure highly accurate, fast and reproducible segmentation results to improve disease diagnosis, treatment planning, and enable large-scale studies of the pathology [6]. However, automatic image segmentation, especially brain tumor segmentation, is one of the most complex tasks in medical image analysis, since tumor structures may deviate substantially across patients with respect to shape, size, localization and extension [7]. Hence, the effective application of strong priors is not possible [3].

1.2 Objective

The aim of this thesis was to perform fully automatic brain tumor segmentation using a deep neural network architecture, which was solely trained with synthetically generated MRI data.

Deep learning approaches have demonstrated immense success in computer vision tasks in the past few years, outperforming many previous state-of-the-art techniques, including fully automatic image segmentation. However, it is important to note that the medical domain is different from many other real world domains, since usually there are only limited data samples available. Compared to other databases that may contain millions of images, medical databases often only contain image data of one or a few subjects. Furthermore, no corresponding ground truth segmentation is available for most medical datasets, since manual delineation is such a time-consuming task.

Hence, one big challenge of using deep learning approaches for medical image segmentation lies in augmenting the available dataset and training deep architectures without overfitting the training examples. By generating synthetic training data (MR images containing synthetic brain tumors), this issue can be overcome.

The objective of this thesis is twofold:

1. Generation of synthetic MR images that comply with the basic features of glioblastoma multiforme, as they are visible in post-contrast T1-weighted brain MR images.

2. Implementation, modification and tuning of a deep neural network architecture for semantic image segmentation. Training the deep network using the generated synthetic MRI data and evaluating its performance on real MR images of patients suffering from glioblastoma multiforme.

2 Medical Background

2.1 Brief Overview of the Human Brain

The brain is the most complex organ of the human body and has the ability to perform higher-order cognitive functions. Together with the spinal cord it forms the central nervous system (CNS), which is responsible for most bodily functions, including movement, awareness, thoughts, sensations, memory, and speech [8]. The CNS can be divided into gray matter and white matter, whereby gray matter mostly consists of neurons (cell body, dendrites, axon) and white matter is primarily composed of myelinated axons (nerve projections) [9]. Furthermore, both types of brain tissue contain glial cells, which ensure support and protection of neurons [10]. The type of fat in myelin, which surrounds the axons in white matter, causes the characteristic whitish color, whereas gray matter exhibits the natural grayish color of neurons and glial cells, due to the absence of myelin [8]. Gray matter is mainly found in the outer cortex of the brain, however, it can also be distributed in deeper areas, such as the basal ganglia and midbrain, where groups of neurons form functional units. White matter is predominantly found in the inner area of the brain [9].

The brain is suspended in cerebrospinal fluid and can be subdivided into three main parts: the cerebrum, the brain stem and the cerebellum [9] (see figure 1).

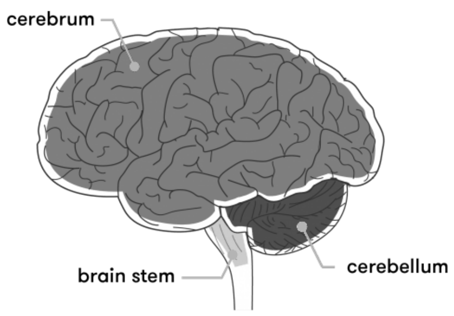


Figure 1: Cerebrum, cerebellum and brain stem. Adapted from [10].

The cerebrum, which is associated with higher-order brain function, can be divided into left and right hemisphere, while each hemisphere can again be subdivided into four different lobes, which are the frontal, parietal, temporal and occipital lobe [10] (see figure 2). The corresponding regions of the two cerebral hemispheres are connected via the corpus callosum, which contains a bundle of nerve fibres. The outer layer of the cerebrum is called the cerebral cortex and consists of the larger neocortex, which is only present in mammals, and the smaller allocortex [8]. The cortex is strongly folded in order to maximize the surface area of the brain and at the same time the number of neurons [10].

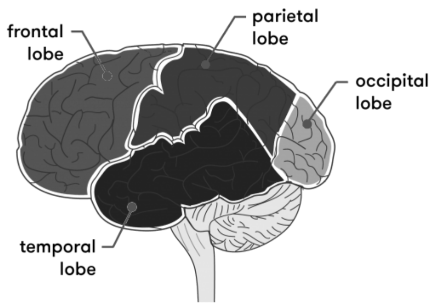


Figure 2: Sections of the cerebrum: frontal lobe, parietal lobe, temporal lobe, occipital lobe. Adapted from [10].

The cerebellum can also be divided into two hemispheres and is located at the back of the brain, behind the brainstem and between the cerebral hemispheres [8]. It is involved in a variety of tasks related to movement, coordination and balance [9]. The cerebellar cortex is also highly folded, but exhibits finer folds compared to the cerebral cortex. The brain stem is one of the simplest parts of the human brain, but still one of the most important, as it is responsible for basic vital functions, such as heartbeat, breathing and blood pressure [10]. It consists of the midbrain, pons, and medulla and is located directly above the spinal cord and below the cerebrum [8].

The brain is surrounded and protected against physical trauma by the skull and, in contrast to any other organ, it is additionally protected by the so-called blood-brain barrier, which allows to accurately regulate the exchange of molecules, ions and cells between blood and brain [11]. This mechanism protects the brain from toxins, pathogens and harmful substances, however, it is still vulnerable to damage, disease, and infection [12].

2.2 Gliomas

Gliomas are the most common and aggressive primary brain tumors in adults and presumably develop from glial cells [3]. Glial cells, also known as neuroglia, provide nerve cells with nutrients and energy and help maintain the blood-brain barrier [13].

Classification of gliomas is done according to the World Health Organization (WHO) as astrocytoma, oligodendroglioma, mixed oliogoastrocytoma, and ependymoma. Furthermore, the tumor grade is determined by the histopathological impression based on the WHO grading system that defines grades I to IV according to aggressiveness of the lesion, where grade I refers to least agressive and grade IV to most agressive tumors [14]. Grade and histopathological impression are usually associated with malignant potential, response to treatment and survival rate [15].

- **Grade I:** non-malignant and slow-growing tumors, associated with long-term survival
- **Grade II:** non-malignant or malignant tumors, rather slow-growing, can recur as higher grade tumors
- **Grade III:** malignant tumors, often recur as higher grade tumors
- **Grade IV:** most aggressive malignant tumors, fast reproduction

Around 70% of all gliomas are of grade III or grade IV and are therefore considered malignant gliomas [16].

Astrocytic tumors can be separated into two main groups that represent circumscribed astrocytomas (grade I) and diffuse astrocytomas (grade II-IV). Diffuse astrocytic tumors are prone to transform into grade IV astrocytomas, which are also known as glioblastoma multiforme [17].

2.3 Glioblastoma Multiforme

The most common type of glioma in adults is glioblastoma multiforme (GBM), which corresponds to WHO astrocytoma grade IV. It is named for its histopathological appearance and makes up approximately 55% of all gliomas [15].

GBMs can be categorized either as primary or secondary glioblastoma according to their clinical representation. Approximately 90% of all glioblastomas are of the primary type, which means that they develop very fast de novo in older patients without evidence of a less malignant previous lesion [18]. In the majority of cases, patients show first symptoms less than six months before diagnosis [15].

In contrast, secondary glioblastomas develop from low grade gliomas, such as diffuse astrocytoma or anaplastic astrocytomas and usually occur in younger patients [18]. In general, symptoms appear more than six months before diagnosis. Secondary GBMs are typically located in the frontal lobe, show less necrosis and usually carry a better prognosis. From a histological point of view, primary and secondary glioblastomas are basically indistinguishable; however, they are different in their genetic and epigenetic profiles [15].

Development of GBM could be associated with certain genetic abnormalities, either inherited such as the Li-Fraumeni syndrome, or resulting from mutation into an oncogene or inactivation of the so-called tumor-suppressor gene p53 [17].

Treatment approaches for glioblastoma multiforme usually include chemotherapy, radiation therapy and maximum resection of the lesion [16]. However, the median survival rate for patients with glioblastoma multiforme is still only approximately 15 months [19].

Therefore, fast and accurate segmentation and measurement of brain lesions, such as the glioblastoma multiforme, is critical for diagnosis, treatment planning and for monitoring therapy response.

2.3.1 MRI Features of Glioblastoma Multiforme

In T1 contrast-enhanced MR images, glioblastomas appear as large heterogeneous mass in the supratentorial white matter and usually cause significant tumor mass effect (see chapter 2.3.2). They can sometimes also occur near the dura mater or in the corpus callosum, posterior fossa or spinal cord [17]. Typically, glioblastomas show a thick, irregular-enhancing margin with a central area of necrosis, surrounded by large edema [20]. In post-contrast T1-weighted MR images, enhancement of the tumor is most commonly present and represents the uptake of gadolinium-based contrast agent in the lesion, which leads to a bright appearance of the tumor wall. Intravascular, flow-related enhancement as well as interstitial, permeability-related enhancement is involved [21]. In contrast, the necrotic core appears as dark area in post-contrast T1-weighted MRI data [20]. Some GBMs can also show small areas of enhancement inside the margin, typically in irregular patterns that represent areas of surviving tumor cells within necrotic regions [21].

A schematic representation of a glioblastoma multiforme in an axial gadolinium-enhanced T1-weighted MR image can be seen in figure 3.

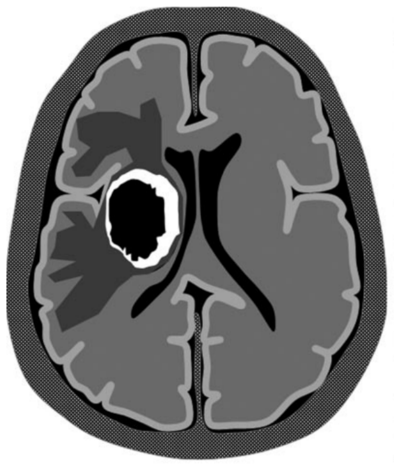


Figure 3: Axial gadolinium-enhanced T1-weighted MR image showing the irregular, heterogeneous tumor mass with enhancing margin and necrotic core. Adapted from [21].

2.3.2 Tumor Mass Effect

The vascular system of glioblastomas is unstructerd and highly permeable, due to an overexpression of vascular growth factors, which in addition to the fast and large expansion of the tumor, lead to strong inflammation, fluid leakage and edema [22]. Due to the surrounding skull, it is not possible for the brain to make room for a growing mass, like a glioblastoma. Therefore, the tumor compresses and displaces the adjacent brain tissue, often leading to fatal damage. This process is called the tumor mass effect [22]. The force that is applied to the brain by a growing tumor is an outward radial force that originates from the central tumor area and weakens by distance [23].

Figure 4 shows this effect caused by a glioblastoma multiforme. The deformation and compression of the brain mass is clearly visible by looking at the body of the right lateral ventricle. In a healthy brain the left and right lateral ventricle are nearly symmetric.

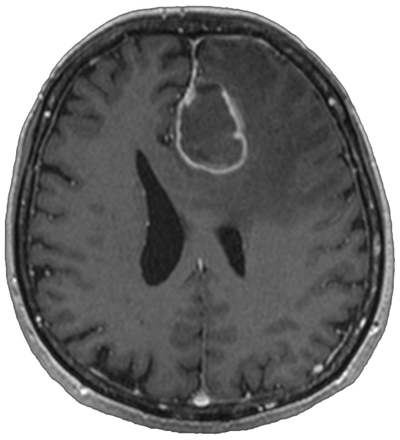


Figure 4: Tumor mass effect in the brain, caused by glioblastoma. Data source: J. Egger [24].

3 Technical Background

3.1 Magnetic Resonance Imaging

Management and evaluation of brain tumors is usually performed by various medical experts, such as medical oncologists, neurosurgeons, neurologists and radiologists, who all heavily depend on imaging of the central nervous system to diagnose and characterize lesions [25]. Magnetic resonance imaging is the standard method for brain tumor imaging, since it provides a non-invasive technique for producing detailed images of the human brain with good soft-tissue contrast [1].

This section gives a conceptual overview of the principles of magnetic resonance imaging.

The nuclear magnetic resonance phenomenon, which is the basis for magnetic resonance imaging, was independently discovered by Felix Bloch and Edward Mills Purcell in 1946, for which they received the nobel prize in physics. Magnetic resonance imaging (MRI) is a non-invasive medical diagnosis technique used to generate pictures from inner parts of the body. In contrast to Computed Tomography (CT), it does not expose the patient to harmful radiation [26].

Magnetic resonance imaging was initially called nuclear magnetic resonance imaging and works based on the principle that protons in the human body are able to absorb and emit radio frequency energy when subjected to an external magnetic field [27]. MRI is a two-step process, where in the first step the spin-orientation of a proton is manipulated by a variety of applied strong magnetic fields and, in the second step, alterations of the proton's spin-orientation can be measured by a detector coil, due to interaction with the protons magnetic field [26]. The signal of a single proton is diminutive, however, the sum of all magnetic fields from all involved protons produces a considerable signal, which can be detected [28].

3.1.1 General Principle

The human body and living tissue in general consists of approximately 60%-80% water. Therefore, typically hydrogen atoms are targeted in MRI to produce detectable radio-frequency signals [28]. All atomic nuclei with an odd number of protons posses an intrinsic angular moment (often called spin) and a magnetic dipole moment, which is proportional to the angular momentum [26]. However, it is important to note that protons are not actually spinning, instead, spin is a fundamental property of nature. When the magnetization has a direction different from the main magnetic field, this leads leads to precession of the proton spin around the field direction [27]; see figure 5.

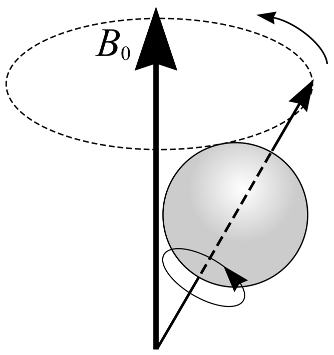


Figure 5: The interaction of the proton's spin with the magnetic field produces a torque, causing the nucleus to precess around the field direction B_0.

Protons can be seen as small magnets that align along the external magnetic field, either parallel or anti-parallel, and the resulting magnetic moment is called net magnetization [27]. On average, more protons align in parallel to the external magnetic field lines, since the anti-parallel direction requires slightly more energy than the parallel direction [28]. This equilibrium value leads to an NMR effect, however, a small non-vanishing spin excess is not enough to guarantee that the signal can be detected. Therefore, the magnetization vector is rotated away from the direction of the main magnetic field, which is called excitation [27].

The contrast of an MR image is determined by the relaxation time of a certain tissue and the proton density [29]. Relaxation is the process of restoring the equilibrium situation after excitation [28]. Water can either be free or bound to the surface of macro-molecules, and the relaxation properties of a tissue are determined by the corresponding interaction [27].

MR scanners produce a strong, constant, homogeneous magnetic field with common electric field strengths of 1.5 to 3 Tesla, which is used to align spins, producing magnetization [27]. In figure 6, a typical MR scanner can be seen.

Figure 6: MR scanner.

It is important to note that MRI comprises a release of energy from the spin system. According to Albert Einstein's quantum mechanical description of the emission of energy from atomic systems, the emission can either be induced or spontaneous [30]. In MRI, energy emission must always be induced. Therefore, RF (radio frequency) pulses are applied perpendicularly to the main magnetic field, oscillating with a specific frequency that corresponds to the resonance frequency (Larmor frequency) of nuclei [27]. The application of such RF pulses results in a rotation of the net magnetization vector. By varying duration and amplitude of the RF pulse, different flip angles can be achieved. Additionally, a so-called gradient field is created by gradient coils in x-, y- and z-direction, which minimally distort the main magnetic field in a predictable way [29]. This leads to a slightly different resonance frequency of protons as a function of their position. It is desired to achieve linear gradients (linear change in field) that can be used for spatial encoding of the investigated body by using frequency and phase properties [26]. The characteristic knocking noise of an MR scanner is produced by the fast switching of these coils. As soon as the excitation via the RF-pulse stops, relaxation sets in, which leads to re-establishment of the equilibrium situation [27]. Excited hydrogen nuclei emit a radio-frequency signal that can be measured using RF receiver coils, which are placed near the investigated tissue [29]. The obtained signal can subsequently be transformed into an image by using inverse Fourier transformation. The rate at which excited hydrogen nuclei return to the equilibrium situation defines the contrast between different tissue types [27]. There is no standard image intensity scale in MRI, which may cause difficulties in image display and analysis. The resulting intensity depends on many factors like scanner settings and coil imperfections [29].

3.1.2 Relaxation

Different tissues return to equilibrium state at different rates, defined by the two independent relaxation times T1 and T2.

- **T1 relaxation:** Longitudinal relaxation or spin-lattice relaxation is the process by which the net magnetization vector returns to its initial maximum value in direction of the main magnetic field [26]. Time constant T1 reflects the time required for the longitudinal component of the net magnetization vector to regain about 63% of its maximum value, see figure 7.

- **T2 relaxation:** Transverse relaxation or spin-spin relaxation is the process by which the transverse components of the magnetization vector dephase [26]. Time constant T2 reflects the time required for the transverse magnetization to decay to around 37% of its initial maximum value, see figure 8.

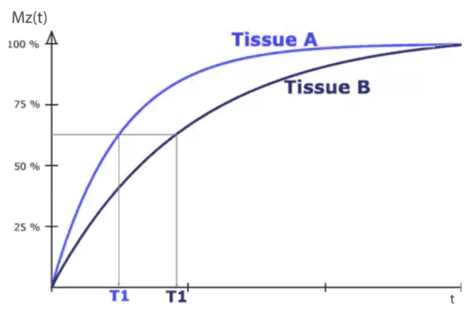


Figure 7: T1 relaxation process for two different tissues.

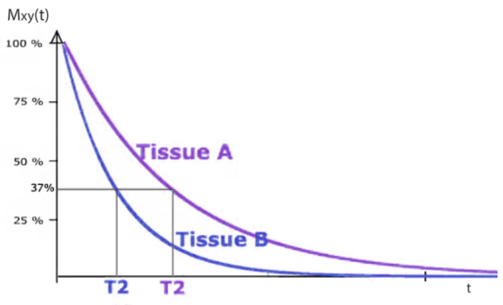


Figure 8: T2 relaxation process for two different tissues.

The relaxation rate depends on the local environment of hydrogen nuclei, hence, different relaxation rates give information about the type of tissue of a sample. It is possible to acquire T1-weighted or T2-weighted MR images, as described in the following sections.

3.1.3 T1-weighted images

To obtain a T1-weighted MR image, short repetition time T_R (time from one excitation pulse to the next one) and short echo time T_E (time between the RF excitation pulse and the peak of the signal induced in the coil) must be used to enhance the T1 differences between tissues [28]. The magnetization of each tissue is partially recovered before measuring, depending on the corresponding T1 time constant. The measurement is carried out by changing the repetition time [26]. T1-weighted images are also called "anatomy scans" since they have a significant tissue contrast [28]. Fluids appear very dark, water-based tissues are gray and fat-based tissues look very bright. In summary, it can therefore be said that tissues with long T1 times are darker than tissues with short T1 times, see figure 9 left.

3.1.4 T2-weighted images

To acquire T2-weighted MR images, long repetition time T_R and long echo time T_E must be used, which means that they take longer than T1-weighted images, in fact the scan time directly depends on T_R [28]. The magnetization of each tissue is partially decayed before measuring, depending on the corresponding T2 time constant. The measurement is carried out by changing the echo time [26]. T2-weighted images have highest intensities for fluids, whereas water-based and fat-based tissues appear gray. They are also called "pathology scans", since aggregation of pathological fluid is significantly brighter than dark normal tissue [28]. In general, it can be said that tissues with long T2 time constants appear brighter than tissues with short ones, see figure 9 right.

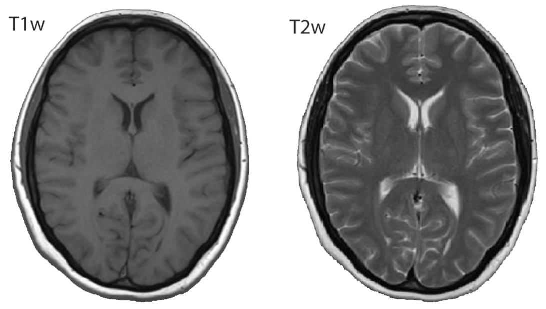


Figure 9: Left: Example of T1-weighted MR image. Right: Example of T2-weighted MR image.

3.1.5 Contrast Agents

MRI is very sensitive to pathological conditions, which means that abnormalities can be easily spotted, however, it is not really specific, in the context of differentiating between various pathologies [28]. Therefore, external contrast agents can be used, which act to reduce the T1 time constant of the affected tissue [26]. This helps to improve specificity.

Typically tissue enhancement takes place when vascular walls, which in healthy condition are a barrier to large molecules, are leaky due to pathological conditions. With intra-vascular agents the enhancement is usually visible between blood and surrounding tissue. Contrast agents can also improve image quality due to an increase of the signal-to-noise-ratio [28].

The most frequently used contrast media for brain tumor enhancement are based on gadolinium chelates. Since free gadolinium is a highly toxic heavy metal, the ion must be tightly bound or chelated to a molecular ligand to ensure high kinetic stability [31]. Gadolinium possesses a strong paramagnetic susceptibility, which leads to T1 shortening in tissues where it accumulates [26]. Consequently, these tissues will have enhanced intensity in post-contrast T1-weighted MR images.

The contrast medium is usually injected intravenously and initially stays in the vascular system, is then distributed into the extracellular space and finally excreted via the kidney; furthermore, there is the possibility of oral preparation for gastro-intestinal imaging purposes [28].

In general, gadolinium-based contrast agents are considered safe, however, for patients with severe renal impairment, a strong association between nephrogenic systemic fibrosis (NSF) and gadolinium-containing contrast agents has been found [32].

Healthy brain tissue is not enhanced by gadolinium agents, since the contrast medium cannot pass the intact blood-brain barrier. As opposed to this, brain tumors, which are typically highly vascular, will appear brighter on T1-weighted MR images, as the contrast agent will leak into abnormal tissue via a pathologically permeable blood-brain barrier [28].

3.2 Artificial Neural Networks

Biological neural networks are deemed to be the most well-organized information processing systems, which are able to process high-level information by interconnecting numerous simple computing elements called neurons [33]. One single neuron can only execute very simple tasks, such as responding to an input signal; however, when an abundance of neurons is connected to a network it is possible to perform highly complex tasks [34]. In the human brain approximately 10^{11} neurons can be found, which communicate via a dense connection network of synapses and axons [35].

Artificial neural networks (ANNs) are a machine learning technique, inspired by the functionality of the human brain [36]. Although the concept of ANNs is loosely based on biological neural networks, the key idea is not to build an exact replication of its physiological counterpart, but instead to use the knowledge regarding its functionality to solve highly complex problems [38].

ANNs are able to learn through experience instead of having fixed instructions to perform specific computations. They gather their knowledge by finding patterns and relationships in data [37]. Comparable to their biological counterpart, the basic building block of ANNs is the artificial neuron, which is a simple mathematical model with three elementary sets of rules: multiplication, summation and activation [39].

A visual comparison between biological and artificial neurons is shown in figure 10. It can be seen that some conceptual similarities between the two structures exist. Biological neurons have highly branched dendrites to receive signals, a cell body for information processing and an axon to send signals protruding from the cell body. Artificial neurons have input nodes, a processing stage and an output node.

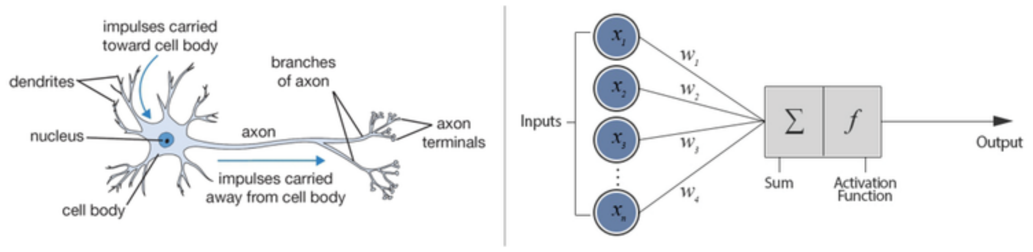


Figure 10: Biological and artifical neurons. Adapted from [40].

3.2.1 Feedforward Neural Networks

In feedforward artificial neural networks, information only flows in one direction, from input to output, and there are no feedback connections in which outputs of the network are fed back into itself [41].

A feedforward neural network consists of input nodes, hidden nodes (not mandatory) and output nodes. The number of network layers, types of activation functions and the number of connections between individual neurons is not limited [39].

3.2.1.1 Perceptron

The perceptron is the earliest type of artificial neural network, developed by F. Rosenblatt in 1957 [42], and represents the simplest form of feedforward network. It is based on the McCulloch-Pitts neuron and the Hebbian learning rule for weight adjustment [43],[44]. As a linear classifier, it is only capable of solving linearly separable problems.

As can be seen in figure 11, the perceptron consists of input nodes x_i, adjustable connection weights w_i, bias w_0 and an output node y. The bias is used to shift the decision boundary away from its origin. Since the perceptron has one single layer of output neurons, it is also known as single-layer perceptron [33].

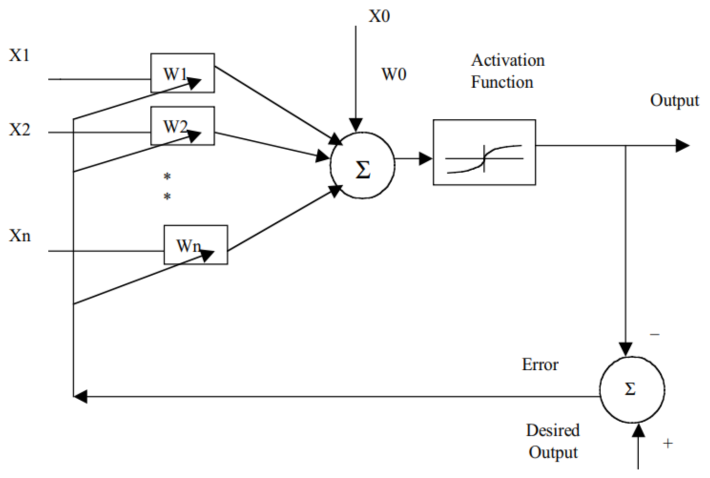


Figure 11: Perceptron with sigmoid activation function. Adapted from [45].

Given a vectorized observation of inputs $\boldsymbol{x}$ with dimension D, the resulting output y is obtained from an activation function f, by taking the weighted sum of inputs x_i [33], see formula 1.

$$y(\mathbf{x};\theta) = f\left(\sum_{i=1}^{D} x_i w_i + w_0\right) = f(\boldsymbol{w}^T\boldsymbol{x} + w_0) \tag{1}$$

where θ denotes the parameter set concatenated from adjustable connection weight vector $\boldsymbol{w}$ and bias w_0.

Typically, the connection weights of the perceptron are initialized to small non-zero values. The perceptron can be trained by calculating the error between the predicted output and the desired output and by adjusting the connection weights accordingly. Therefore this algorithm represents a form of gradient descent. The weights are only changed if an error occurs. This training process is repeated until the classification of all inputs is completed. If the input data is linearly separable, the algorithm converges and the decision hyperplane is placed between the two classes [45].

3.2.1.2 Multi-Layer Perceptron

As already mentioned in the previous section, the main limitation of a single-layer perceptron is its linear decision boundary, despite the use of a non-linear activation function. This limitation can be overcome by simply adding an additional layer of neurons, known as hidden layer, between the input and the output [33]. The described concept is implemented in the multi-layer perceptron, which is the most widely used neural network and consists of at least one input layer, one hidden layer and one output layer, representing a generalization of the single-layer perceptron [34],[45]. Therefore, multi-layer perceptrons can form arbitrarily complex non-linear decision boundaries and are able to separate various input patterns [45].

It has been proven by K. Hornik, M. Stinchombe and H. White [46] that for any continuous function f on a compact set K, there exists a feedforward neural network with only one hidden layer of finite size, which uniformly approximates f to within an arbitrary $\epsilon > 0$ on K. In other words, a feedforward neural network with a single hidden layer is in theory capable of approximating any continuous function to arbitrary accuracy. This statement is known as the universal approximation theorem.

An example of a multi-layer perceptron with one hidden layer and full interconnection can be seen in figure 12.

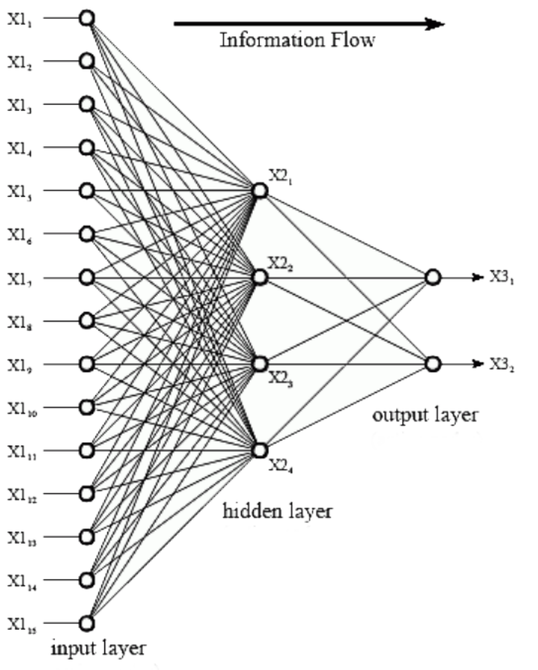


Figure 12: Multi-layer perceptron.

- **Input Layer:** A layer of input neurons, which obtain information from the outside world and pass it to the network for further processing [45].
- **Hidden Layer:** A layer of hidden neurons, which obtain information from the input layer (or other hidden layers) and process it. All connections of the hidden layer are to other layers within the network [45].
- **Output Layer:** A layer of output neurons, which obtain processed information and pass information to the outside world [45].

For a neural network with one hidden layer, the composition function can be written as follows; the bias term is omitted for simplicity:

$$y_k(\mathbf{x};\theta) = f^{(2)}\left(\sum_{j=1}^{M} W_{kj}^{(2)} f^{(1)}\left(\sum_{i=1}^{D} W_{ji}^{(1)} x_i\right)\right) \tag{2}$$

where $\theta = \{W^{(1)} \in \mathbb{R}^{MxD}, W^{(2)} \in \mathbb{R}^{KxM}\}$, D denotes the input dimension, M is the number of nodes in the hidden layer, K represents the output dimension and the subscript represents a layer index.

If the nodes were linear elements, then a single-layer perceptron with suitable weights could equally well be used instead of a multi-layer perceptron [45]. Hence it can be concluded that the great potential of multi-layer neural networks further lies in the utilization of non-linear activation functions.

3.2.2 Deep Learning

Deep learning in general refers to deep feedforward neural networks. The phrase "deep" indicates that the network contains several hidden layers to learn representations of data with multiple levels of abstraction. However, there is no precise definition of how many hidden layers a neural network must contain to be considered "deep" [47].

In the context of machine learning, the term "deep learning" was first used by R. Dechter in 1986 and was introduced to the field of artificial neural networks by I. Aizenberg et al. in 2000 [48],[49],[50].

It has been stated and proven that an artificial neural network with a single hidden layer of finite size, is able to approximate any continuous function to an arbitrary accuracy. Moreover, it has been proven that a neural network with two hidden layers of finite size is capable of approximating any function to an arbitrary accuracy [46],[51]. This raises the question why deeper neural networks with more hidden layers are commonly used.

The main advantage of using a deep neural network lies in its node-efficiency, which means that a shallow network would need much more total nodes (very large hidden layers) compared to a deeper one in order to approximate a complex function to the same accuracy [36]. Consequently, also a smaller dataset is required to train a deep network, due to the lower number of trainable parameters [36],[52]. Therefore, it is much more efficient to solve complex problems with deeper architectures.

However, it has not been possible to use the full potential of deep neural networks for a long time, due to the vanishing gradient problem - identified 1991 by S. Hochreiter [53] - which caused very slow training when more than one or two hidden layers were used [36]. The vanishing gradient problem occurs due to the nature of the backpropagation algorithm, where errors are propagated backwards through the network to perform weight updates, see section 3.2.5. With standard activation functions, the cumulative backpropagated error decays exponentially with the number of layers, due to repeated multiplication [47]. A simple solution to this issue is to use the rectified linear activation function (ReLU, see chapter 3.2.4), which does not reduce the error when it is propagated back through the network, as proposed 2011 by Glorot et al. [54].

Deep learning is a form of representation learning (feature learning), which means that the network is fed with raw data and automatically discovers the required features [55]. Deep neural networks learn such representations of the input by learning levels of abstraction with increasing complexity. Since information is passed through the network, lower level abstractions of shallow layers give rise to higher level abstractions of deeper layers [56], see figure 13.

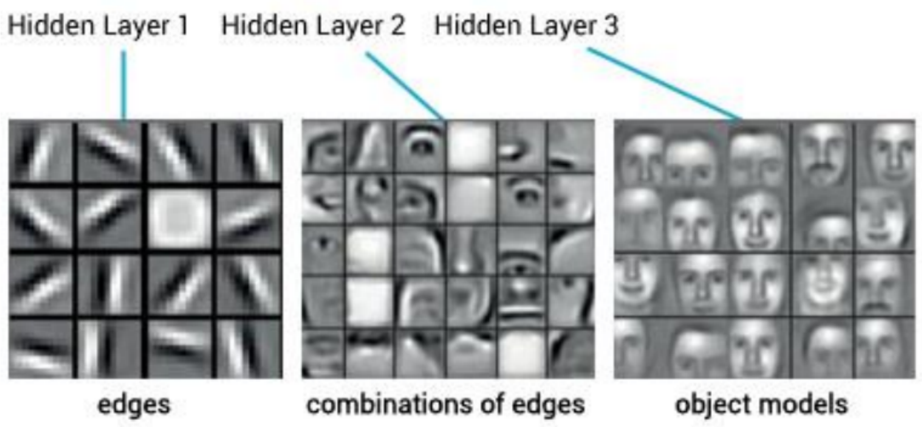


Figure 13: Levels of abstraction in a deep neural network.

Convolutional neural networks, which are discussed in the following section, are a prime example for deep neural networks.

3.2.3 Convolutional Neural Networks

A main property of image data is the spatial correlation between pixels, which means that neighbouring pixels are more strongly correlated than distant ones; however, in fully connected multi-layer neural networks, as discussed in the previous sections, the complex 2D spatial structure of images is not exploited due to vectorization of the input [34],[57]. Furthermore, due to the full connectivity between neurons in a traditional multi-layer neural network, the number of connection weights increases rapidly when larger images are processed [41]. Therefore, the use of fully connected neural networks is impractical for real-world image processing tasks.

One way to overcome these problems is to use a convolutional neural network (CNN), which is a special kind of deep feedforward neural network, widely used in the field of computer vision [58]. The three main mechanisms incorporated into CNNs are: local receptive fields, weight sharing, and subsampling (pooling) [57]. Besides, convolution allows working with inputs of variable size [41]. A typical CNN is composed of three main types of layers: convolutional layers, pooling layers, and fully-connected layers. A common architecture for a convolutional neural network can be seen in figure 14.

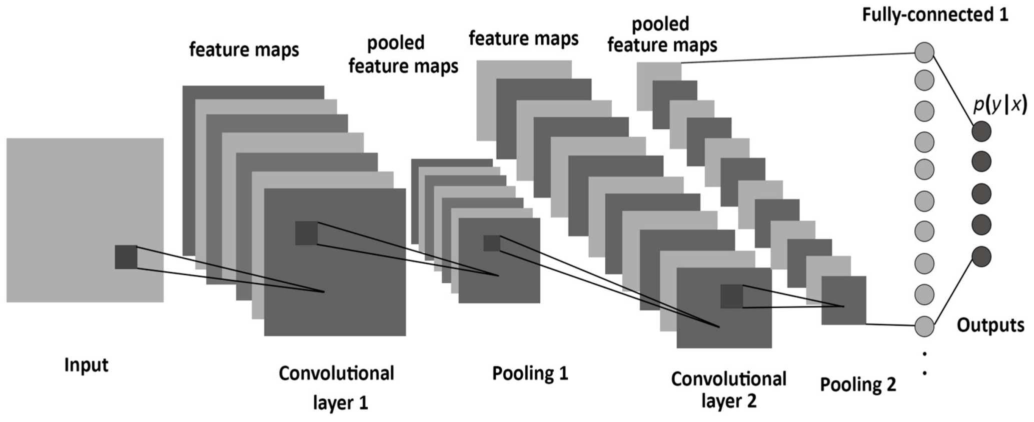


Figure 14: The structure of a CNN, consisting of convolutional, pooling, and fully-connected layers. Adapted from [59]

In a convolutional layer, units are organized in so called feature maps, as illustrated in figure 14. Usually, there are several of these maps in one layer that allow to detect multiple features of an image [57]. Each unit in such a feature map receives input from a small subregion of the previous layer, known as receptive field [60]. Therefore, each neuron only has to process information of a local patch. All units in a feature map are constrained to share the same set of weight values (also called filter bank), which drastically reduces the degrees of freedom of the model. Different feature maps in a convolutional layer use different filter banks [56]. Input values of a local patch are linearly combined using the corresponding filter bank and a bias, and subsequently the result is passed through a non-linearity [57]. Using a non-linear activation function is essential, since cascading linear systems, such as convolutions, only results in another linear system; however, data that should be modeled is typically non-linear [33]. In modern convolutional neural networks usually the ReLU (Rectified Linear Unit) activation function, which is described in detail in chapter 3.2.4, is used [61]. All units of a feature map are able to detect the same pattern, but at different locations of the image. The assessment of the activations of each unit corresponds to a convolution of the image pixels with a filter-kernel consisting of the learned weight values, hence the name of the network [57].

In figure 15, the mathematical principle of a convolution operation is illustrated. In order to calculate each resulting value of the convolution, the filter-kernel slides across the input image with a certain shift size, usually referred to as stride. For example, when the filter kernel is moved one pixel at a time, then the stride size is one. In practice, however, the convolution of CNNs is done with larger strides, which is equivalent to performing down-sampling of the image after a regular convolution [33].

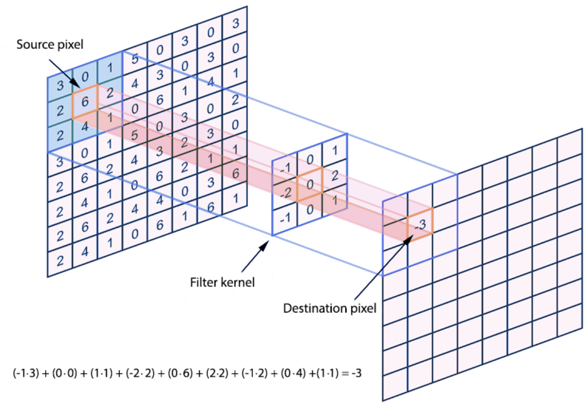


Figure 15: Mathematical principle of a convolution operation.

Pooling (subsampling) layers are used to merge semantically similar features into one [56]. The inputs of pooling layers are established by the outputs of previous convolutional layers. In pooling layers, each feature map is represented by a plane of units and each unit receives input from a local patch (receptive field) of the corresponding feature map of a convolutional layer [57]. Max pooling is the most commonly used type of pooling, which computes the maximum of a receptive field in a feature map, see figure 16. Pooling units are typically shifted by more than one row or column, leading to a dimensionality reduction of the representation and creating invariance to minor shifts and distortions [56]. Since pooling layers progressively reduce the spatial size of feature maps, the number of parameters is also reduced, leading to less computational effort involved in the network [33].

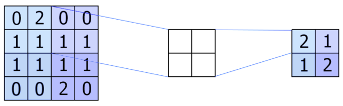


Figure 16: Max-Pooling.

An illustration of a convolutional layer and a pooling layer applied to an input image can be seen in figure 17.

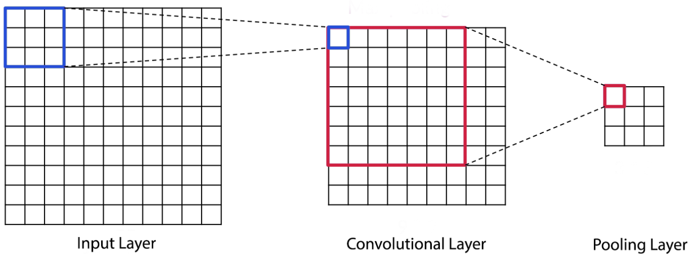


Figure 17: Convolutional layer and pooling layer applied to an input layer.

Several stages of convolution, non-linear activation functions and pooling layers are stacked, followed by more convolutional layers and finally one or two fully-connected layers [56]. Fully-connected layers are used to encode position-dependent information and global patterns [36]. A softmax activation function can be used at the output layer to provide class probabilities, see chapter 3.2.4.

3.2.4 Activation Functions

Activation functions are a very important part of artificial neural networks, since they introduce non-linearity to the model and thereby enable it to perform a non-linear mapping from input nodes to output nodes. This makes the network capable of performing highly complex tasks. An artificial neural network without an activation function would essentially be a linear regression model.

A linear activation function is typically used for output neurons in regression models, in order to provide the network with all real numbers; on the contrary, for output neurons of classification models, a softmax function (generalization of the logistic function where the output can be interpreted as a probability distribution over N different possible classes) is most commonly used [36].

- **Linear:**

$$f(x) = \alpha \cdot x \tag{3}$$

where α denotes the slope of the linear function. If α=1, the function is called identity function. The range of the linear function is between $(-\infty, \infty)$.

- **Softmax:**

$$f_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad \forall i \in 1..J \tag{4}$$

where $\mathbf{x}$ is a vector of dimension J. The softmax function squashes the outputs of each node to be in the range of (0, 1).

Regarding the activation functions of hidden neurons, a common choice is either the hyperbolic tangent function or the logistic (sigmoid) function, which both satisfy three desirable properties - continuous differentiability, monotony, non-linearity [36],[45].

1. **Continuous differentiability:** This property is crucial to compute gradients of error with respect to the connection weights and adjust the weights accordingly using gradient descent methods [36].
2. **Monotony:** This property is important because a non-monotonic activation function would introduce additional local minima in the parameter space, which hampers training [36].
3. **Non-linearity**: This property is necessary to introduce non-linearity into the network, which allows to model non-linear patterns in the training data [36].

The hyperbolic tangent and the logistic function can be written in the following form:

- **Hyperbolic Tangent:**

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

  where α is a shape parameter of the function. The range of the hyperbolic tangent function is between (-1, 1).

- **Logistic (Sigmoid):**

$$f(\mathbf{x}) = \frac{1}{1 + e^{-\alpha x}} \tag{6}$$

  where α is a shape parameter of the function. The range of the logistic function is between (0, 1).

Plots of the hyperbolic tangent function and the logistic (sigmoid) function are provided in figure 18 and figure 19, respectively.

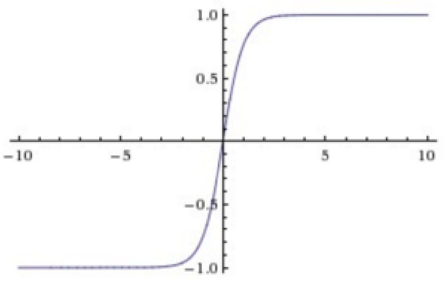


Figure 18: Hyperbolic tangent

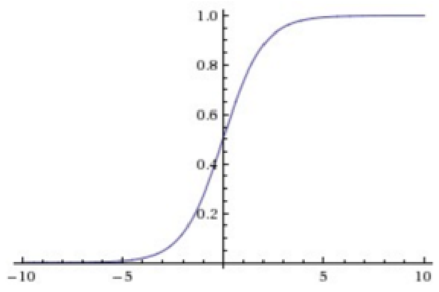


Figure 19: Logistic sigmoid

Both hyperbolic tangent and logistic sigmoid function suffer from the vanishing gradient problem when the function values become either too high or too low, since the corresponding derivative (gradient) becomes extremely small. This can impede the training process, since gradient descent optimization algorithms are used to adjust the connection weights. V. Nair and G. Hinton [62] proposed a so-called rectified linear unit (ReLU) for hidden neurons to solve this vanishing gradient problem. The functions takes a real-valued input and thresholds it at zero, see figure 20. However, the ReLU function is unbounded and not differentiable at x=0; therefore, in practice the gradient at x=0 is set to either 0 or 1 and concerning the unboundedness, a regularization technique can be used to limit the magnitude of the weights [33].

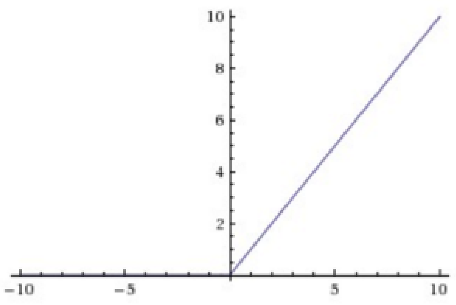


Figure 20: ReLU

- **ReLU:**

$$y(x) = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases} \tag{7}$$

3.2.5 Training Neural Networks

The aim of training a neural network is to adjust the trainable parameters θ, consisting of connection weights and biases, so that the network is able to model the given input data in the best possible way.

The learning paradigm can be distinguished between supervised learning (mapping an input to an output based on example pairs of input and desired output values), unsupervised learning (inferring a function which describes the organization of input data without giving any corresponding labels) and reinforcement learning (learn how software agents are supposed to behave in an environment in order to maximize a cumulative reward) [39]. Supervised learning is the most common form of machine learning and is typically used for pattern recognition and regression tasks. [56]. Furthermore, training algorithms for neural networks can be roughly divided into two main categories, which are gradient-based and non-gradient-based methods. Currently gradient-based techniques are most commonly used, because they typically converge faster [36].

Since the aim of the training process is to find parameters θ that minimize the difference between a desired pattern and the actual network output, the learning problem can be formulated as the minimization problem of an error function $E(\theta)$ [33]. The error function represents the distance between the prediction and the desired output and is chosen according to the problem definition [56]. For regression tasks a typical choice is the sum-of-squares error function (see formula 8), whereas for classification tasks the cross-entropy function is a common choice (see formula 9 for binary classification and formula 10 for multiclass classification) [57].

- **Sum-of-squares error function:**

$$E(\boldsymbol{\theta}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \boldsymbol{\theta}) - t_n\}^2 \tag{8}$$

  where N is the number of observations $x_1, ..., x_n$ with corresponding target values $t_1, ..., t_n$, and $y(x_n, \boldsymbol{\theta})$ denotes the network output for x_n.

- **Binary cross-entropy error function:**

$$E(\boldsymbol{\theta}) = -\sum_{n=1}^{N}\{t_n ln(y_n) + (1-t_n)ln(1-y_n)\} \tag{9}$$

where N is the number of observations $x_1, ..., x_n$ with corresponding target values $t_1, ..., t_n$, and y_n represents $y(x_n, \boldsymbol{\theta})$, which denotes the network output for x_n.

- **Generalized cross-entropy error function:**

$$E(\boldsymbol{\theta}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} ln(y_k(x_n, \boldsymbol{\theta})) \tag{10}$$

where K denotes the number of classes; the binary target variables $t_k \in \{0, 1\}$ have a 1-of-K coding scheme indicating the class, and the network outputs are interpreted as $y_k(x, \boldsymbol{\theta}) = p(t_k = 1|x)$.

The error function $E(\theta)$, which is a smooth continuous function of θ, can be seen as a surface sitting over the weight space (parameter space), as illustrated in figure 21. The smallest value of the error function is at a point in parameter space where the gradient vanishes, such that $\nabla E(\theta) = 0$; however, the error function usually has a strong non-linear dependence on the parameters, which results in many local minima in addition to the global minimum [57].

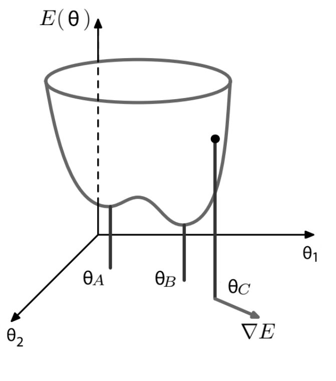


Figure 21: Error function E(θ). θ_A is a local minimum and θ_B is the global minimum. At any point θ_C , the local gradient of the error function is given by vector ∇E. Adapted from [57]

There is no analytic solution of the parameter set that minimizes the error function, due to it being highly non-linear and non-convex [33]. It may be possible for small network architectures to perform an exhaustive search for the ideal parameters which minimize E(θ), however for networks of reasonable size more sophisticated methods, like using gradient information, have to be used [36].

The simplest way to use gradient information is to perform weight updates according to a small step in the direction of the negative gradient of the error function, which is known as gradient descent optimization, since the parameter vector is moved in the direction of the steepest descent [57]. The gradient descent update can be formulated as follows:

$$\theta^{(\tau+1)} = \theta^{(\tau)} - \eta \nabla E(\theta^{(\tau)}) \tag{11}$$

where τ indicates the current iteration step, η denotes the learning rate (step size) and $\nabla E(\theta^{(\tau)})$ is the gradient of the error function at the current position.

The learning rate η is a hyper-parameter that controls the strength of parameter adjustment in each update step, with respect to the gradient of the error function. Typically, the optimal learning rate is close to the largest step size that does not cause divergence of the training criterion [63]. It is common practice to use a decaying learning rate, which means that the step size is rather large at the beginning and gradually decreases during the training process.

The error function E(θ) is defined with respect to a training set; therefore, in principle all training examples need to be processed in order to compute the true gradient of the error function in each step [57]. However, gradient descent optimization can vary with regard to the number of training examples used to compute ∇E(θ) [64]. Standard gradient descent, which uses the whole training set at once, is known as batch gradient descent and has the disadvantage that its convergence depends on the size of the training set [57]. Other common techniques are stochastic gradient descent (one training example at a time), and mini-batch gradient descent (several training examples at once), both providing the advantage that their convergence only depends on the number of update steps and the richness of the training distribution; furthermore, they converge much faster than batch gradient descent. [64].

In feedforward neural networks the gradient can be computed by using the so-called backpropagation algorithm [56]. The main idea of this algorithm, which is often simply called backprop, is to feed back error information from the output layer through all network layers [41].

Using backprop, the gradient of the error function E(θ) for a specific layer l in a network consisting of L layers can be estimated as follows:

$$\frac{\partial E}{\partial \boldsymbol{\theta}^{(l)}} = \frac{\partial E}{\partial \boldsymbol{a}^{(L)}} \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{a}^{(L-1)}} \cdots \frac{\partial \boldsymbol{a}^{(l+2)}}{\partial \boldsymbol{\theta}^{(l+1)}} \frac{\partial \boldsymbol{a}^{(l+1)}}{\partial \boldsymbol{\theta}^{(l)}} \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}} \frac{\partial \boldsymbol{z}^{(l)}}{\partial \boldsymbol{\theta}^{(l)}} \tag{12}$$

where $a^{(L)} =$ y, and $a^{(l)}$ and $z^{(l)}$ represent the activation function and pre-activation function respectively of layer l [33].

The chain rule of calculus can be used to calculate derivatives of functions composed of other functions with known derivatives; and the backpropagation algorithm calculates the chain rule highly efficiently by using a specific order of operations [41]. Specifically, a vector of errors $\delta^{(k)}$ is computed at each layer $k = l, l+1, ..., L-1$, which is then propagated backwards through the network [33].

Layer l receives an error vector $\delta^{(l+1)}$ from layer $l+1$ and updates it in the following way:

$$\delta^{(l)} = \frac{\partial \boldsymbol{a^{(l+1)}}}{\partial \boldsymbol{z}^{(l+1)}} \odot \left\{ \frac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} \cdot \delta^{(l+1)} \right\} = f'\left(\boldsymbol{z}^{(l)}\right) \odot \left\{ \left(\boldsymbol{\theta}^{(l+1)}\right)^T \cdot \delta^{(l+1)} \right) \tag{13}$$

where $\odot$ indicates an element-wise multiplication [33].

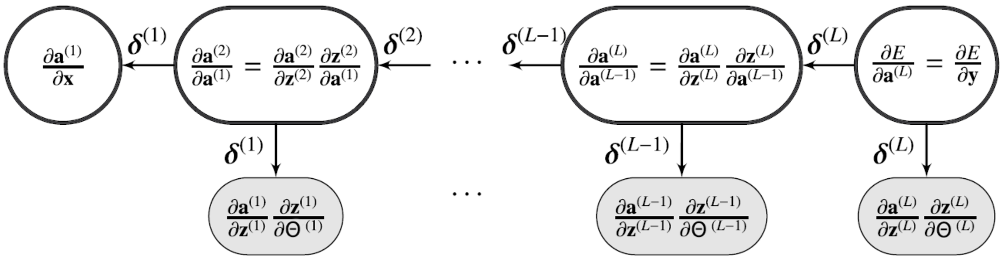
After the error vector is updated, the according gradient with respect to the parameters in layer l can be computed as follows:

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^{(l)}} = \delta^{(l)} \boldsymbol{a^{(l)}} \tag{14}$$

The updated error vector is then sent to layer $l-1$ until the gradient vectors of all layers are obtained [33]. Subsequently, the parameter set $\boldsymbol{\theta}$ can be updated as shown in formula 11, using:

$$\Delta E(\boldsymbol{\theta}) = \left[\frac{\partial E}{\partial \boldsymbol{\theta}^{(1)}} \cdots \frac{\partial E}{\partial \boldsymbol{\theta}^{(l)}} \cdots \frac{\partial E}{\partial \boldsymbol{\theta}^{(L)}}\right] \tag{15}$$

A graphical representation of the backpropagation algorithm in a network consisting of L layers can be seen in figure 22.



$\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{x}}$
$\boldsymbol{\delta}^{(1)}$
$\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{a}^{(1)}} = \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(1)}}$
$\boldsymbol{\delta}^{(2)}$
$\cdots$
$\boldsymbol{\delta}^{(L-1)}$
$\frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{a}^{(L-1)}} = \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{a}^{(L-1)}}$
$\boldsymbol{\delta}^{(L)}$
$\frac{\partial E}{\partial \mathbf{a}^{(L)}} = \frac{\partial E}{\partial \mathbf{y}}$
$\boldsymbol{\delta}^{(1)}$
$\boldsymbol{\delta}^{(L-1)}$
$\boldsymbol{\delta}^{(L)}$
$\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \Theta^{(1)}}$
$\cdots$
$\frac{\partial \mathbf{a}^{(L-1)}}{\partial \mathbf{z}^{(L-1)}} \frac{\partial \mathbf{z}^{(L-1)}}{\partial \Theta^{(L-1)}}$
$\frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}} \frac{\partial \mathbf{z}^{(L)}}{\partial \Theta^{(L)}}$

Figure 22: Graphical representation of the backpropagation algorithm for a network with L layers. Adapted from [33]

Using the backpropagation algorithm for convolutional neural networks is as simple as for regular deep neural networks, allowing all parameters in all filter banks to be adjusted [56].

When the model is entirely trained, the performance of the network has to be evaluated on a so-called test set, which represents unseen data. This is done in order to test the generalization ability of the network, which reflects the ability to produce correct outputs for new inputs that were never seen during the learning process [56]. Since deep neural networks are composed of multiple non-linear hidden layers, which makes them extremely powerful and flexible, they are prone to overfit training examples when limited training data is available; in this context overfitting refers to the fact that the network captures sampling noise which exists in training data but not in test data, even if it is drawn from the same distribution [65]. Overfitting results in a low training error, but leads to a rather high test error, since the network is too specialized to generalize beyond the training set [36]. Therefore, one of the most important aspects of training a neural network is to avoid overfitting, for example by using regularization methods.

3.2.6 Regularization

Deep neural networks are able to learn very complex relationships between inputs and outputs; however, when training data is limited, the network starts to model sampling noise and is not able to generalize from training samples to the entire domain of unseen data, which is known as overfitting [65]. Regularization is a very important concept that helps to prevent overfitting by controlling the complexity of a neural network [57].

- **Early stopping:** Very effective and easy regularization technique, where the error of a validation set is monitored and the training process is stopped as soon as the error starts to increase.
- **Weight decay:** In this technique an additional regularization term is added to the loss function (error function), such that the resulting loss function consists of the data loss and an additional weight regularization loss, which penalizes large weights. The aim of weight regularization is to prefer local minima which have a simpler solution. There are many different types of weight penalties, such as L1 and L2 regularization and soft weight sharing [65].
- **Dropout:** With this method neurons are randomly dropped (deactivated) with a certain probability during the training process, which helps to prevent undesirable dependence on the presence of specific neurons. Due to the short-termed and random deactivation of neurons, the method can be seen as training different networks at each iteration with their connection weights shared. Hence, a huge number of different network architectures is efficiently combined. During the test phase, all neurons are used, but the weights are scaled to maintain the same output range [33],[65].

3.2.7 Batch Normalization

Batch Normalization (BN) is a technique, which significantly speeds up convergence and is also capable of improving generalization. It tackles the problem of the so-called internal covariate shift, which arises from continuous change in the distribution of network activations during the training process and slows down training. To address this issue, BN normalizes the activations of a layer in order to ensure that they stay within a small interval. This normalization is performed with the running average of the mean-variance statistics of each mini-batch. [33]

3.3 Image Segmentation

Image segmentation is one of the most demanding problems in computer vision, particularly in the field of medical image analysis [66]. The term image segmentation refers to the process of partitioning an image into a set of non-overlapping, semantically meaningful segments, which display specific attributes [67]. Typically, different segments indicate different objects or parts of objects appearing in the given image. When the areas of interest do not completely cover the image, one can perform segmentation into foreground regions of interest and background regions of no interest [68]. A segmentation result can either be an image of assigned labels defining each segmented area or a set of contours representing the region boundaries [67].

An example of a simple segmentation task can be found in figure 23, where coins in an image are segmented. On the left side, an image which contains several coins is shown; on the right side, the corresponding segmentation mask separating foreground (white) and background (black) can be seen.

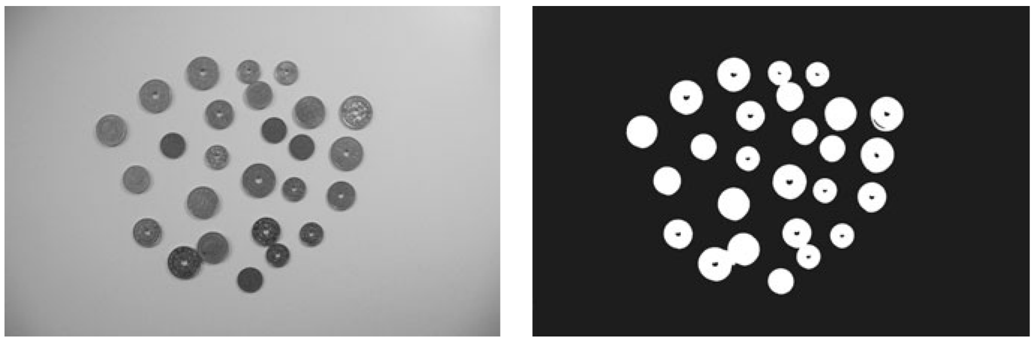


Figure 23: Segmentation of coins. Left: an image containing several coins. Right: corresponding segmentation mask indicating foreground (white) and background (black).

Image segmentation is considered the most important part of medical image processing, as it is used for many different applications, such as studying anatomical structures or localizing pathologies, and is often the first step in an image processing chain [69]. However, segmentation of medical images is in general not trivial, due to imaging imperfections, noise corruption and various image artifacts [67]. In the following sections, the most important image segmentation methods for medical image analysis are discussed.

3.3.1 Common Methods for Medical Image Segmentation

This section gives a brief overview of common methods for medical image segmentation; in particular it addresses segmentation methods with application to brain MR images and is adapted from [67].

- **Manual segmentation:**

  Manual segmentation is performed by human experts, such as physicians or radiologists, who segment and label 3D volumetric imaging data by hand in a slice-by-slice manner. This segmentation method is deemed to be very precise; however, manual image segmentation is a very demanding and time-consuming task, which, according to several intra- and interoperator variability studies, is highly prone to error. Furthermore, manual segmentation results are practically impossible to reproduce, even by a very experienced operator. Nonetheless, this technique is still intensively used for ground truth delineation and evaluation of automated segmentation methods.

- **Thresholding:**

  Thresholding is a segmentation method that uses the intensity histogram of an image to identify a threshold value τ, in order to separate different regions of the image. It represents the simplest form of image segmentation. There are various versions, such as single thresholding, local threshold, multi-thresholding or adaptive thresholding. This method is fast and computationally efficient, however, it does not incorporate spatial neighborhood information and is therefore sensitive to intensity inhomogeneities and noise.

- **Region growing:**

  This segmentation technique is a simple region-based method, which extracts a connected group of pixels with similar intensity values from an image. First, a seed point is set to a pixel that is known to belong to the region of interest, either manually or automatically. After that, the region growing algorithm checks every neighbouring pixel for its intensity value and if it satisfies a predefined uniformity or homogeneity criterion, the pixels are added to the growing region. The whole process is repeated until there is no more change of the resulting area. This method is highly sensitive to the initialization of the seed point.

- **Classification and Clustering:**

  Classification methods for image segmentation are pattern recognition techniques that use training data (sample images with known labels) to divide a feature space into several classes. This technique is representative of supervised learning. Image features can be intensity values, edge directions, texture or other image properties. Usually multi-dimensional feature sets are used. A disadvantage of classification methods is that manually segmented data is necessary as reference.

  Clustering techniques are similar to classification techniques, however they do not require reference data. Therefore, clustering is representative of unsupervised learning. This method groups a set of data samples, so that samples in the same group are as similar as possible (according to previously extracted features) and differ from samples in other groups. This method iteratively estimates the properties of each class and clusters data samples accordingly.

- **Atlas-based methods:**

  An atlas is a template consisting of several manually segmented images and can be seen as prior domain knowledge. If an atlas for a specific area of interest is available, unseen images of this area can be segmented without any additional cost. This technique is most commonly used for the segmentation of anatomical brain images. The concept of atlas-based methods is similar to classifier techniques, however, they are implemented in the spatial domain. Usually, image registration (alignment of atlas and image to be segmented) is required. Afterwards, the segmentation labels can be transformed to the target image.

- **Deformable models:**

  Deformable models can either be used in 2D (active contours) or in 3D (active surfaces). They use parametric curves or surfaces, which deform under the action of internal and external forces, to represent region boundaries. An initial region boundary has to be provided (for example by a rough manual segmentation), which is not too far from the desired boundary. External forces are a function of the features of the image and control the fit of the contour to the desired boundary. Internal forces control the surface regularity.

3.3.2 Deep Learning for Medical Image Segmentation

Deep learning approaches have demonstrated immense success in computer vision tasks in the past few years, outperforming many previous state-of-the-art techniques. Following the success of deep learning methods in other real-world domains, they are now also attracting increasing interest in the field of medical image segmentation, since they provide an objective, robust and fast automated segmentation technique. Deep learning algorithms have signi-ficantly benefited from the recent performance gains of graphics processing units (GPUs), which allow the training of very deep and complex models on large datasets. Regarding computer vision tasks, many of the latest advancements are due to the application of convolutional neural networks on modern GPUs [70]. Convolutional neural networks are typically used for image classification tasks, where for each input image a single class label is predicted, however, one of the most common tasks in medical image analysis is semantic segmentation [71]. Conventional CNNs, as discussed in section 3.2.3, are not capable of semantic segmentation, since the spatial information of an image is lost when the convolutional features are fed into the fully connected layers of the network architecture [70].

An elegant approach to overcome this limitation is the so-called fully convolutional neural network (FCN) proposed by J. Long et al. [72]. In FCNs the final fully-connected layers are replaced by de-convolutional layers, which perform a transposed convolution to upsample the low-resolution feature maps and thereby restore the original resolution of the input image while performing semantic segmentation [70]. A typical architecture of a FCN can be seen in figure 24.

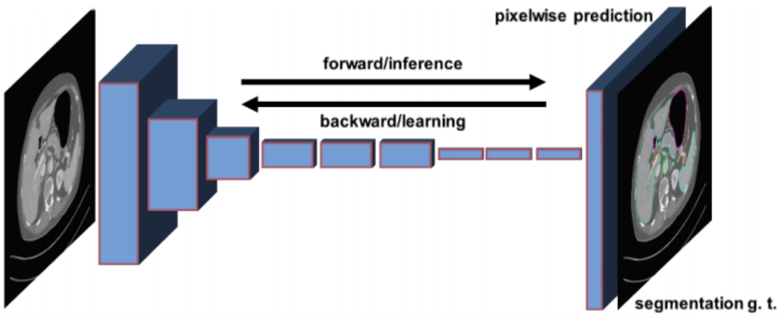


Figure 24: Typical architecture of a FCN. Adapted from [70].

A further advancement of the FCN is the U-Net architecture proposed by O. Ronneberger et al. [71], which is discussed in detail in section 4.

3.3.3 Evaluation of Segmentation Results

Evaluation metrics enable an objective quantitative comparison of different segmentation methods as well as of different parametrizations of a single method. Such evaluation metrics can for example be overlap-based or spatial distance-based. In either case, a ground truth is required, which can be compared to the outcome of the segmentation method. Clinical experts performed manual slice-by-slice segmentation on the relevant data of this work, in order to obtain the ground truth of each image.

In figure 25 an exemplary illustration of the accordance between a predicted segmentation mask and the corresponding ground truth can be seen, where TP indicates the number of true positives (correctly classified as positives by the segmentation method), FP represents the number of false positives (wrongly classified as positives by the segmentation method) and FN denotes the number of false negatives (wrongly classified as negative by the segmentation method). The number of true negatives (TP, correctly classified as negative) is not visible in the schematic illustration.

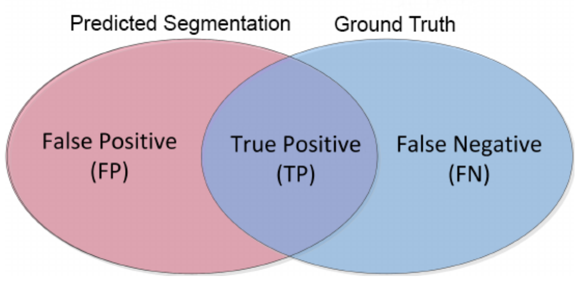


Figure 25: Predicted segmentation, ground truth and their union.

In the course of this thesis the Dice similarity coefficient (DSC), sensitivity, specificity and the Hausdorff distance have been calculated to evaluate the quality of the automatic segmentation results.

Overlap-based evaluation metrics:

- **Sensitivity:** represents the proportion of all positives that are correctly classified as such. This metric is also known as the true positive rate (TPR) or the recall.

$$Sensitivity = TPR = Recall = \frac{TP}{TP + FN} \tag{16}$$

- **Specificity:** represents the proportion of actual negatives that are correctly classified as such. This metric is also known as the true negative rate (TNR).

$$Specificity = TNR = \frac{TN}{TN + FP} \tag{17}$$

- **Dice similarity coefficient (DSC):** this metric is one of the most popular measures for evaluating segmentation performance and is commonly used for brain MRI data [67]. It is also known as Sørensen-Dice coefficient, Sørensen index or Dice's coefficient. Beyond the direct comparison between predicted segmentation and ground truth, the DSC is commonly used to measure reproducibility [73].

$$DSC = \frac{2 \mid A \cap B \mid}{\mid A \mid + \mid B \mid} = \frac{2TP}{2TP + FP + FN} \tag{18}$$

  where A and B denote the set of pixels classified as positives (foreground) by the ground truth and predicted segmentation, respectively. $\mid A \mid$ and $\mid B \mid$ denote the number of elements in A and B. The Dice similarity coefficient is in the range $(0, 1)$, where a value of 0 indicates that there is no overlap between the two segmentations and 1 indicates that both segmentations are identical.

Spatial distance-based evaluation metrics:

Evaluation metrics based on a spatial distance, where the spatial position of pixels is taken into consideration, are commonly used for image segmentation as dissimilarity measures and are advisable when the boundary delineation of the segmentation is important [74].

- **Hausdorff distance:** this metric measures the spatial distance between two point sets and is a commonly used dissimilarity measure for comparing medical image segmentations.

  The Hausdorff distance (HD) between two finite point sets A and B is given by

  $$HD(A, B) = max(h(A, B), h(B, A)) \tag{19}$$

  where $h(A, B)$ is called the directed Hausdorff distance defined as

  $$h(A, B) = \underset{a \in A}{max} \ \underset{b \in B}{min} \ \|a - b\| \tag{20}$$

  where $\|a - b\|$ is some norm, for example the Euclidean distance, which is given by

  $$\|a - b\| = \sqrt{\sum_{i=1}^{N} (a_i - b_i)^2} \tag{21}$$

3.4 Frameworks

This section gives a brief overview of valuable frameworks that were used for this thesis; namely MeVisLab, Tensorflow and Keras.

3.4.1 MeVisLab

The generation of synthetic MRI data was realized with the help of MeVisLab - a powerful modular framework that can be used for image processing research and development with a strong focus on biomedical imaging.

All explanations in this section are based on the "MeVisLab Reference Manual" [75] and the "Getting started tutorial" [76], provided by MeVis Medical Solutions AG. All figures in this section are obtained from the "Getting started tutorial" [76].

MeVisLab is a rapid prototyping and development platform for medical image processing and visualization. The basic entities are graphical representations of modules with their specific functions for image processing, image visualization, and image interaction. Image processing and interactive image manipulation can be achieved by building networks that are constructed of connected modules, which provide certain algorithms as well as a MeVisLab interface and can be interconnected to form architectures that represent algorithms on a higher abstraction level. MeVisLab provides three basic types of modules that are distinguished by color, see figure 26.

- **ML Modules (blue):** demand-driven processing of voxels.
- **Open Inventor Modules (green):** visual scene graphs (3D).
- **Macro Modules (brown):** combination of other module types, allowing implementation of hierarchies and scripted interaction.

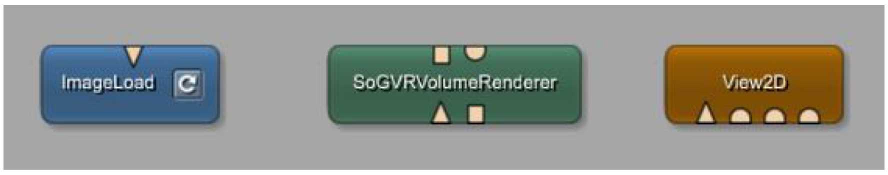


Figure 26: Comparison of different modules in MeVisLab. Blue: ML modules. Green: open inventor modules. Brown: macro modules

By combining several image analysis, visualization and interaction modules, complex image processing networks can be established.

Furthermore, it is possible to create individual applications based on macro modules. For this purpose, python scripting components can be added to implement dynamic functionality on the network level as well as the user interface level. In addition, it is possible to integrate new algorithms by using the modular, platform-independent C++ class library.

In MeVisLab there are three types of data connectors that are distinguished by their shape, see figure 27.

- **Triangle shape:** indicates transportation of ML images
- **Half-circle shape:** indicates transportation of inventor-scenes
- **Square shapes:** indicates transportation of pointers to data structures

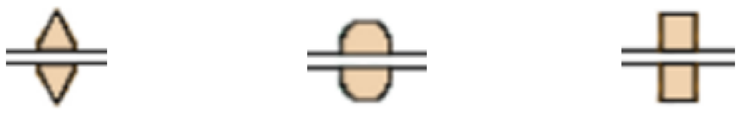


Figure 27: Module connectors. Left: connector that indicates transportation of ML images. Middle: connector that indicates transportation of inventor-scenes. Right: connector that indicates transportation of pointers to data structures.

A so-called data connection can be established by connecting modules with the connectors shown in figure 27, which causes the corresponding information to be transported from one module to another, however, it is only possible to build connections between connectors of the same type. Another way to connect modules is to use parameter connections, where any data field of a module can be connected to another one of the same type, which allows to join different types of modules.

3.4.2 TensorFlow

TensorFlow is a platform-independent open-source software library, which is widely used for machine learning applications and was developed by Google. It provides an interface for expressing machine learning algorithms as well as an implementation for executing them and is particularly useful for deep neural network models. This section gives a brief conceptual overview of the framework, based on the "TensorFlow whitepaper" [77] and the "TensorFlow developement guide" [78].

Several application programming interfaces (APIs) are available for tensorflow, where the lowest level API (TensorFlow Core) provides complete programming control and high level APIs help to manage datasets, estimators, training and inference.

In order to describe computations with regard to the dependencies between individual operations, TensorFlow uses a dataflow graph, which results in a low-level programming model. The nodes in dataflow graphs represent units of computation and edges represent data that is produced or utilized by a computation. The dataflow model is a typical model for parallel computing.

As the name suggests, tensors are the central data type of TensorFlow. A TensorFlow tensor can be seen as n-dimensional array or list consisting of base datatypes, having dynamic dimensions and a static type. A tensor object represents a computation that is partially defined and will finally produce a value; it is the main object to be manipulated when working with a low level TensorFlow API. A tensor has three properties, which are shape, rank and type. In addition, there are TensorFlow variables, which represent a tensor whose value can be changed by running different operations on it. Variables have to be initialized explicitly. They are used to hold and update parameters, since without having parameters, it would not be possible to train, update, save, restore or perform any other operation in a TensorFlow program. In contrast to tensors, variables exist outside a single session run.

TensorFlow Core programs consist of two discrete parts, where the first is to build a computational graph of tensor objects, specifying how each tensor is computed based on other tensors, and the second part is to run the computational graph within a session to eventually evaluate the nodes. A session comprises state and control of the TensorFlow runtime.

3.4.3 Keras

Keras is an open-source high-level deep learning API, which is written in Python. It is capable of running on top of TensorFlow, CNTK, or Theano and can be utilized for fast prototyping and advanced research. For this thesis, it was used in combination with TensorFlow. In the following a brief introduction is given, based on on the "Keras Documentation" [79].

The four main principles of Keras are:

- **Modularity:** Keras models can be seen as a sequence or a graph of modules that can be put together and are fully configurable with only few restrictions.
- **User friendliness:** Keras offers consistent and simple APIs, which are optimized for common use cases. Furthermore, it provides clear and actionable feedback for user errors.
- **Easy extensibility:** Custom modules are very easy to add as new functions or classes. It is possible to create innovative layers, loss functions, and to develop state-of-the-art models. This makes Keras suitable for advanced research, as it allows for total expressiveness.
- **Working with Python:** Models are written in native Python code, which is compact, easy to debug and simplifies extensibility, as there is no need for separate model configuration files.

The central data structure of Keras is a model, which can be seen as a way of organizing layers. The most straightforward type of model is the Sequential model, which simply represents a linear stack of layers. For more complex network architectures, such as multi-output models, directed acyclic graphs, or models with shared layers, the Keras functional API can be used, which allows to build arbitrary graphs of layers.

Once a model is defined, it can be compiled using the underlying framework of TensorFlow, in order to optimize the desired computation of the model with a specified optimizer and loss function. After that, the model has to be fit to training data, which is the actual training process of the model. Finally, the trained model can be used to make predictions on unseen data.

Keras can seamlessly execute on CPUs and GPUs, such as NVIDIA GPUs, Google TPUs and OpenCL-enabled GPUs, given the underlying frameworks. By default, Keras will use TensorFlow as its tensor manipulation library, however, it is also possible to configure the Keras backend.

4 Related Work

In this chapter, important publications related to the work in this thesis are discussed. Section 4.1 deals with other attempts of synthetic brain tumor generation, including general techniques to model 3D tumor growth as well as methods specifically developed for MRI data. In section 4.2, the U-Net (a convolutional neural network developed for biomedical image segmentation) is reviewed, since the segmentation network used in this thesis is based on the U-Net architecture. Finally, section 4.3 gives information of related work in the field of automatic brain tumor segmentation, with a focus on the MICCAI BraTS challenge for multimodal brain tumor segmentation.

4.1 Methods for Generating Synthetic Brain Tumors

M. Prastawa et al. [23] proposed a method for generating synthetic brain tumor MR images, which include most of the difficulties encountered in real MR data. They simulated a tumor mass effect by using a biomechanical model. Furthermore, they simulated the infiltration of brain tissue by tumor and edema using a reaction-diffusion process, which is guided by a modified diffusion tensor MR image. They also simulated tumor enhancement as present in contrast-enhanced T1 weighted MRI. More recently, Prastawa et al. [80] developed another method that combines physical and statistical modeling to generate synthetic multi-modal 3D brain MR images including tumor and edema, in combination with the anatomical ground truth. Their new method synthesizes the lesion in multi-modal MR imaging and diffusion tensor imaging by simulating a tumor mass effect, warping and destruction of white matter fibers, and infiltration of brain tissue by the tumor. In addition, they simulate the apperance of the tumor and brain tissue by synthesizing texture images from real MRI data. They underline that they do not attempt to simulate the full process of real tumor growth with their proposed method, instead their intention is to create a database of synthetic brain tumor MRI data, with similar challenges for segmentation as present in real tumors and to provide the corresponding anatomical ground truth. Rexilius et al. [81] proposed a framework for generating brain phantoms including tumors. They simulated the tumor mass effect by using a biomechanical finite element model. Their approach is to deform the phantom of a healthy subject and insert tumor structures from a diseased subject. Furthermore, they compute a model for edema by utilizing the distances to the tumor boundary and the white matter mask. Their framework considers contrast enhancement inside tumor structures, however, they do not simulate the enhancement of vessels and cerebrospinal fluid.

Clatz et al. [82] proposed a model to simulate the 3D growth of glioblastoma multiforme. They used the finite element method to simulate the invasion of the tumor in the brain tissue and its tumor mass effect, which is either modeled with a reaction-diffusion or a Gompertz equation (based on a linear elastic brain constitutive equation), depending on the considered tissue type. Furthermore, they propose a coupling equation, which takes the mechanical influence of tumor cells on the invaded tissues into account. They simulate the growth process by comparing the synthetic tumor growth with the real growth observed on two MR images of a patient acquired within six months.

4.2 U-Net

The segmentation concept of this thesis is based on the U-Net architecture, a convolutional neural network developed for biomedical image segmentation, proposed by O. Ronneberger et al. [71]. This section provides a review of the U-Net adapted from [71].

The network is built upon the so-called fully convolutional network, introduced by J. Long et al. [72], which is discussed in section 3.3.2. The architecture is modified and extended, so that the training process only requires very few example images and yet leads to more precise segmentation results. To this end, they supplemented the standard contracting path with a subsequent expanding neural network, where pooling layers are replaced by upsampling layers. As a result, these layers increase the output resolution. Furthermore, high resolution features from the contracting part of the network are combined with the upsampled output. Due to this additional information, a subsequent convolutional layer is able to learn a more accurate output. Another important modification in their network architecture is the large number of feature channels in the upsampling path, which enables the network to propagate context information to higher resolution layers. Therefore, the expanding part of the network is nearly symmetric to the contracting part, which leads to a u-shaped form.

No fully connected layers exist in the architecture and only the valid part of each convolution is utilized, which means that the segmentation mask only consists of pixels for which the full context is available in the input image. Hence, it is possible to perform a seamless segmentation of arbitrarily large images using an overlap-tile strategy, illustrated in figure 28.

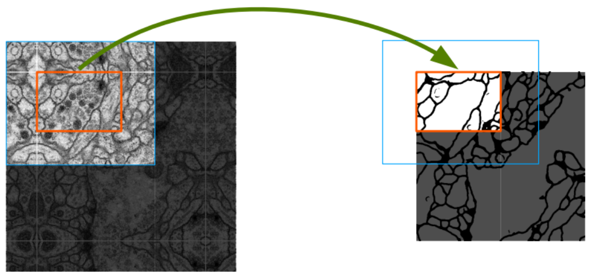


Figure 28: Overlap-tile strategy allowing seamless segmentation of arbitrary large images. Missing input data for predicting the segmentation in the red area is extrapolated by mirroring. Adapted from [71]

The overlap-tile strategy is essential to enable the application of the network to arbitrarily large images, without being limited by the GPU memory. In order to predict pixels at the border area of the image, the missing information is extrapolated by mirroring the input image.

Network Architecture

An illustration of the network can be seen in figure 29. As already mentioned, the architecture consists of a contracting path, which acts as an encoder, and an expanding path, which acts as a decoder.

The contracting path represents the architecture of a standard convolutional neural network, consisting of repeated 3x3 unpadded convolutions, each followed by a ReLU activation function and a 2x2 max-pooling layer with stride 2 for downsampling. The number of features are doubled at each downsampling step.

In the expanding path, each step comprises an upsampling of the feature map with a subsequent 2x2 up-convolution, which halves the number of feature channels, a concatenation with the according cropped feature map from the extensive path, and two 3x3 convolutions, each followed by a ReLU activation function. Due to the loss of border pixels in each convolution, cropping of the feature map is necessary.

A 1x1 convolution is used at the final network layer, in order to map each feature vector to the desired number of classes to be distinguished. Altogether, the U-Net architecture consists of 23 convolutional layers and is applicable to numerous biomedical segmentation tasks.

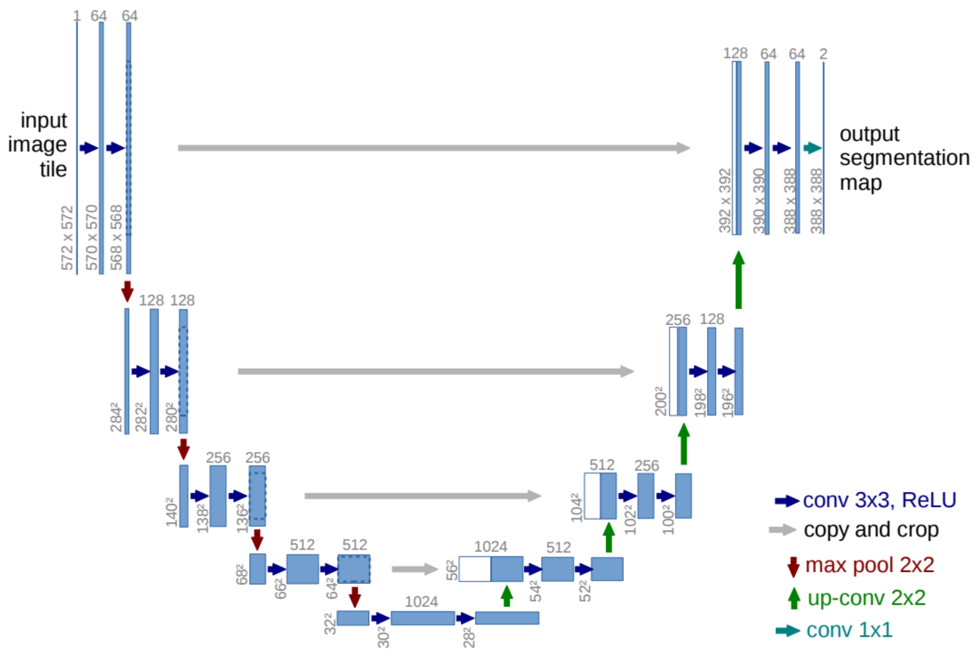


Figure 29: Example of a U-net architecture for 32x32 pixels in the lowest resolution, where each blue box represents a multi-channel feature map; the number of channels is indicated on top of the box and the x-y-size is denoted at the lower left edge of each box. White boxes indicate feature maps that were copied. Arrows denote different operations. Adapted from [71].

4.3 Multimodal Brain Tumor Segmentation Challenge

BraTS [3],[83] is a brain tumor segmentation challenge in conjunction with the MICCAI conference, which has a focus on evaluating state-of-the-art methods for the segmentation of brain tumors in multimodal MR images, and has been held annualy since 2012. Each year, a dataset of multi-institutional pre-operative MR images of gliomas and the underlying ground truth segmentation is provided by the organizers. An overview of the BraTS 2017 dataset is given in the following section.

4.3.1 Provided Material of BraTS 2017

The training dataset of the brain tumor segmentation challenge (BraTS) 2017 consisted of 75 low grade glioma, 210 high grade glioma and their corresponding manual ground truth segmentations, including the following labels:

- **1**: necrotic core and non-enhancing tumor
- **2**: peritumoral edema
- **4**: enhancing tumor
- **0**: everything else

Segmentation of the whole tumor (labels 1, 2, 4), tumor core (labels 1, 4) and enhancing tumor (label 4) was required in the challenge. In contrast, the aim of this thesis was to segment the necrotic core in combination with the enhancing tumor of glioblastomas multiforme (grade IV), which would not be possible with the ground truth labels of the BraTS 2017 dataset.

The BraTS 2017 validation set consisted of 46 cases of low grade glioma and high grade glioma, however, the grade was not revealed. A testing set was made available for participating teams during a time window of 48 hours. The provided datasets were preprocessed before distribution, including co-registration to the same anatomical template, interpolation to the same resolution and skull-stripping. Images had a dimension of 240x240x155. For each subject in the provided datasets, four different MR modalities were available, namely native T1, post-contrast T1-weighted, T2-weighted and T2 fluid attenuated inversion recovery (FLAIR).

4.3.2 Results of BraTS 2017

The first place for the segmentation task of BraTS 2017 was awarded to K. Kamnitsas et al. [84]. They introduced EMMA (Ensembles of Multiple Models and Architectures), which is an ensemble of widely varying CNNs, including the fully 3D multi-scale CNN DeepMedic [85], three 3D FCNs [72] and two 3D versions of the U-Net architecture [71].

Using their architecture, they were able to achieve a Dice similarity coefficient of 73.8% for the enhancing tumor, 90.1% for the whole tumor and 79.9% for the tumor core on the validation set; and 72.9% for the enhancing tumor, 88.6% for the whole tumor and 78.5% for the tumor core on the test set.

The results of all participating teams evaluated on the validation set can be found at: https://www.cbica.upenn.edu/BraTS17/lboardValidation.html

5 Methods

This chapter provides an overview of methods that were used to set the underlying ideas of this thesis into practice. In particular, section 5.1 discusses the data generation approach, which was used to create realistic looking MRI data containing synthetic brain tumors. Section 5.2 gives a detailed description of the deep learning methods and datasets that were used to perform automatic segmentation of glioblastomas in real MR images, where data preprocessing, the architecture of the segmentation network, implementation details and the training and testing process are discussed.

5.1 Synthetic Data Generation

Deep learning approaches require a large training dataset, in order to learn a task and generalize from training samples to the entire domain of unseen data. However, in the medical field there are usually only small datasets available. Furthermore, for most medical datasets no corresponding ground truth segmentation is available, since manual delineation is such a time-consuming task. Hence, one big challenge of using deep neural networks for medical image segmentation lies in augmenting the available dataset and train the network without overfitting the training examples. This issue can be overcome by using synthetic data. To this end, a method for generating MR images containing synthetic brain tumors along with the underlying ground truth segmentation is presented in this chapter. Parts of this section are adapted from my master project [86] and the corresponding publication about generating synthetic brain tumor MRI data for deep learning-based segmentation approaches [87].

5.1.1 Generation of Synthetic Glioblastomas

In general, it is much easier to receive and use data of healthy subjects than diseased ones, due to privacy concerns amongst other things. Therefore, the main idea of this data augmentation approach was to generate synthetic glioblastomas and insert them into brain MR images of healthy subjects. The resulting "hybrid" MRI slices could subsequently be used to train a deep neural network for automatic tumor segmentation. One advantage of inserting synthetic brain tumors into MR images is that the position, shape and size of each tumor is already exactly known; hence, it is very easy to automatically create the corresponding ground truth. The proposed method is simple but nevertheless allows to simulate all crucial characteristics of glioblastomas and facilitates the generation of a large synthetic dataset.

The precise modeling of tumor growth on cell level was beyond the scope of this work, since it is not necessarily needed for the segmentation approach at hand. With the proposed method, enhancing tumor and necrotic core of a glioblastoma (see chapter 2.3.1) as well as a tumor mass effect (see chapter 2.3.2) were simulated. In clinical practice of oncology and diagnostic medicine, typically only the enhancing tumor and the necrotic core are segmented, since these regions are of primary interest. Delineation of other structures is usually not done, since the segmentation of edema, for example, is extremely challenging and will not represent the truth very well [23].

5.1.1.1 Specifications

MR images can depict several different contrast characteristics. MRI scans can, for example, be T1-weighted, T2-weighted or post-contrast T1-weighted. The appearance of a tumor in an MR image varies widely, depending on the imaging modality that was used. For this thesis, the generated synthetic brain tumors should exhibit the basic features of glioblastomas in post-contrast T1-weighted brain MRI scans. A detailed description of those features can be found in chapter 2.3.1.

5.1.1.2 "GenerateTumor" Module

In order to easily generate realistic looking 3D glioblastomas, a custom ML module named "GenerateTumor" was created in MeVisLab, see figure 30.

Figure 30: Custom ML module "GenerateTumor".

This custom module allowed to easily generate synthetic brain tumors, which could subsequently be inserted into existing MRI data of healthy subjects using standard MeVisLab modules. The module's inventor-scene output connector provided the generated tumor as visual scene graph. The input connector was used to connect an existing 3D MR image, into which the tumor should later be inserted. However, the image was not further processed by this module; it was only required as reference of the image extend.

5.1.1.3 Tumor Generation Algorithm

In this section, a detailed description of the tumor generation algorithm, which was written in C++, is given. The tumor generation process was based on a special type of polyhedron, namely an icosahedron.

A polyhedron is a 3-dimensional, solid body that consists of several polygons, which are typically connected at their edges. An icosahedron is a 20-faced polyhedron. The regular icosahedron has 12 polyhedron vertices, 30 polyhedron edges and 20 equivalent equilateral polyhedron faces [88]. In figure 31 an illustration of a regular icosahedron can be seen.

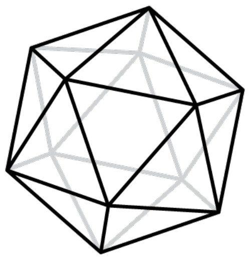


Figure 31: Regular Icosahedron.

In order to receive an approximation of a sphere with more polyhedron vertices, the icosahedron was recursively refined. This was done by dividing each of the existing faces into three equivalent, equilateral triangles and repeating this procedure five times, which resulted in a polyhedron with 2432 vertices, 4860 faces and 7294 edges. The algorithm used for refining the polyhedra was based on the work of J. Egger et al. [89].

The resulting polyhedra after each refining step can be seen in figure 32.

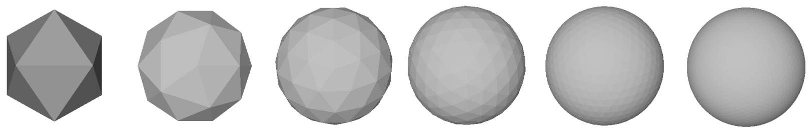


Figure 32: Polyhedra with 12, 32, 92, 272, 812 and 2432 surface points. Adapted from [89].

5.1.1.4 Displacement Algorithm for Vertices

In order to obtain the typical irregular shape of a glioblastoma, a special displacement algorithm was applied to the vertices of the refined polyhedron.

First, a vertex $V_{initial}$ was randomly selected according to a uniform distribution. After that, the distance between $V_{initial}$ and the center of the polyhedron was increased, which resulted in an elevation of the respective vertex, see figure 33. To provide a smooth deformation, the remaining vertices were also displaced accordingly, see figure 34. In order to determine the displacement strength for each vertex, which should decrease by distance to $V_{initial}$, all vertices were assigned to corresponding displacement levels k, according to the minimum number of edges that were required to reach $V_{initial}$. For example, vertices which were directly adjacent to $V_{initial}$ were assigned to displacement level k=1. Finally, the displacement of $V_{initial}$ and the remaining vertices was obtained by multiplying the x-, y- and z-coordinate of each vertex with factor $\epsilon(k)$, which was defined as follows:

$$\epsilon(k) = \begin{cases} 1 + d, & \text{for } k = 0 \\ 1 + d \cdot m, & \text{for } k = 1 \\ 1 + d \cdot m^{2k}, & \text{for } k > 1 \end{cases} \tag{22}$$

where k denotes the displacement level of a vertex, d indicates the initial strength of displacement, which is randomly chosen according to a uniform distribution on the interval [1, 2.5], and m is a decay factor of 0.97.

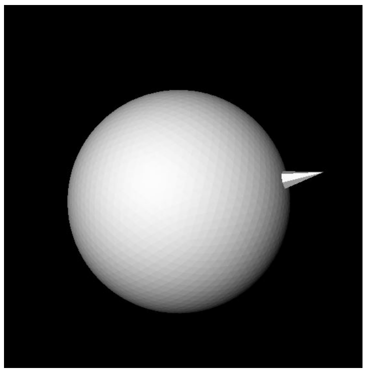


Figure 33: Elevation of a randomly chosen vertex $V_{initial}$

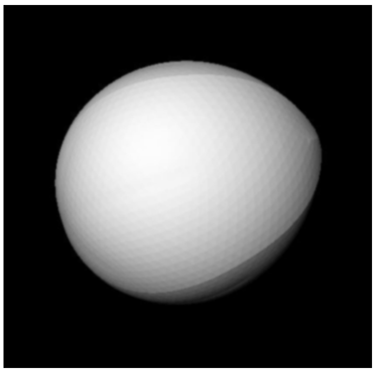


Figure 34: Subsequent displacement of remaining vertices.

Since the object in figure 34 still does not represent the shape of a tumor well, the process of randomly choosing a vertex, displacing it and subsequently displacing the remaining vertices, was repeated for altogether seven times.

Two examples of generated synthetic brain tumors can be seen in figure 35.

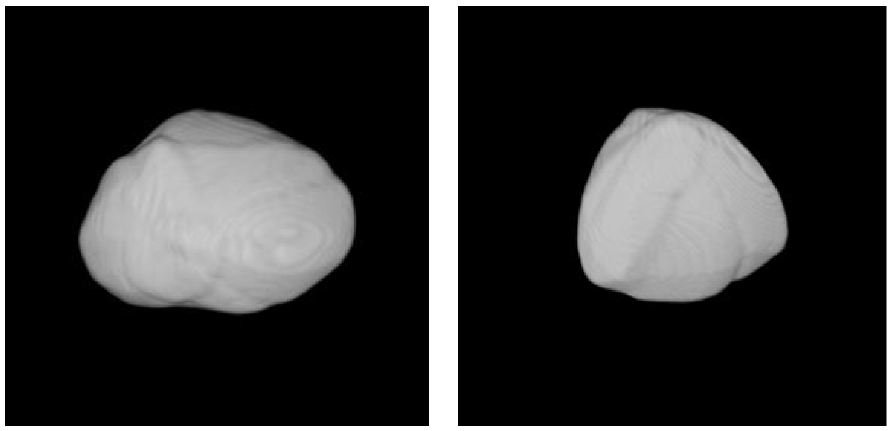


Figure 35: Examples of generated brain tumors using the proposed method.

Finally, the resulting synthetic tumors were inserted into healthy 3D brain MR images with the help of standard MeVisLab modules. To comply with the basic features of glioblastomas in post-contrast T1-weighted brain MRI data, the synthetic tumor had to be further processed, which was also done using standard MeVisLab modules. A detailed description of the implementation can be found in chapter 5.1.2.

5.1.1.5 Simulating the Tumor Mass Effect

The aim of the proposed method was not only to generate synthetic glioblastomas and insert them into healthy brain MRI data, but also to simulate mass effects, which are caused by the growing tumor.

A simplified tumor mass effect was simulated by deforming the brain area surrounding the synthetic glioblastoma, according to a dense vector field. To this end, the "ImageWarp" Module of MeVisLab was used, which is shown in figure 36.

Figure 36: MeVisLab "ImageWarp" module.

All explanations of this module are based on the "MeVisLab Documentation" [90].

The "ImageWarp" module has two inputs. An image volume is connected to input 0 and a vector field is connected to input 1, where the sizes of input 0 and input 1 must be equal. A warped image of the same type as the one at input 0 is provided at the output.

Backwards deformation is performed by the module. This means that vectors in the vector field are associated with voxels in the output image. A vector in the vector field shows the position of a voxel in the input image, where the value for the corresponding output voxel can be found. Usually, this position lies between the voxel grid of the input image, therefore interpolation (bi-linear (2D) or tri-linear (3D)) is required to find the value. The "ImageWarp" module loops over all voxels of the output image, finds the corresponding positions in the input image via the vector field and interpolates the values using linear interpolation.

Since the force that is applied to the brain by the tumor mass is an outward radial force that originates from the initial tumor region, the required dense vector field was constructed in the following way:

First the 3D euclidean distance transform of the synthetic tumor was calculated to receive a gradually decreasing deformation intensity from the tumor center to the border. Then a gradient filter was applied to the euclidian distance transform, leading to a dense vector field that can be utilized for the deformation process. A simplified visualization of the resulting gradient vectors can be seen in figure 37.

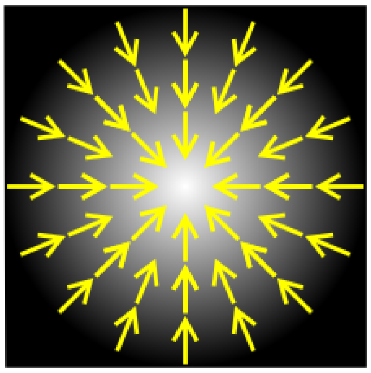


Figure 37: Vector field, where yellow arrows indicate the gradient direction.

As the outward radial force that is applied to the brain weakens by distance, another step was necessary: The euclidian distance transform was normalized and multiplied with a factor, which could be varied between 10 and 25 (depending on the desired strength of deformation). The result of this computation was then multiplied with the previously calculated gradient field. This caused the deformation to be stronger at the tumor center and approach zero at the border.

5.1.2 Implementation

The network built in MeVisLab consists of several ML modules (blue) and macro modules (brown). ML modules are page-based and provide a demand-driven processing of voxels, while macro modules were used to combine the functionality of standard MeVisLab modules. The complete network can be seen in figure 38.

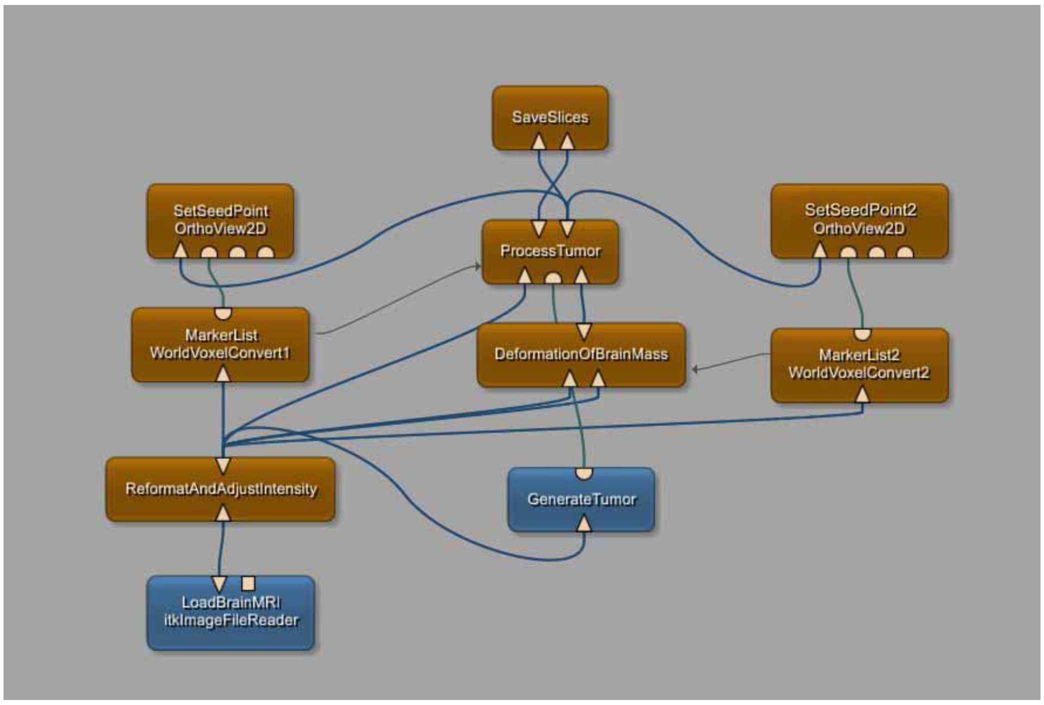

SaveSlices
SetSeedPoint OrthoView2D
ProcessTumor
SetSeedPoint2 OrthoView2D
MarkerList WorldVoxelConvert1
DeformationOfBrainMass
MarkerList2 WorldVoxelConvert2
ReformatAndAdjustIntensity
GenerateTumor
LoadBrainMRI itkImageFileReader

Figure 38: Complete MeVisLab Network.

The specific function of each module and the internal networks are described in detail in the following sections. All explanations of standard MeVisLab modules are based on the "MeVisLab Documentation" [90].

5.1.2.1 ML Modules

The blue ML modules that can be seen in figure 38 are the "LoadBrainMRI" module as well as the "GenerateTumor" module.

- **"LoadBrainMRI":** This module was used to load an existing MR image of a healthy subject.
- **"GenerateTumor":** This module was used to generate synthetic glioblastomas. At the ouput, the synthetic tumor is provided as visual scene graph (inventor scene). See chapters 5.1.1.2 and 5.1.1.3 for a more detailed description of the implemented algorithm.

5.1.2.2 "ReformatAndAdjustIntensity" Module

The "ReformatAndAdjustIntensity" module is a so-called macro module. This module groups and combines other module types to achieve a specific desired function.

The internal network of the "ReformatAndAdjustIntensity" module can be seen in figure 39.

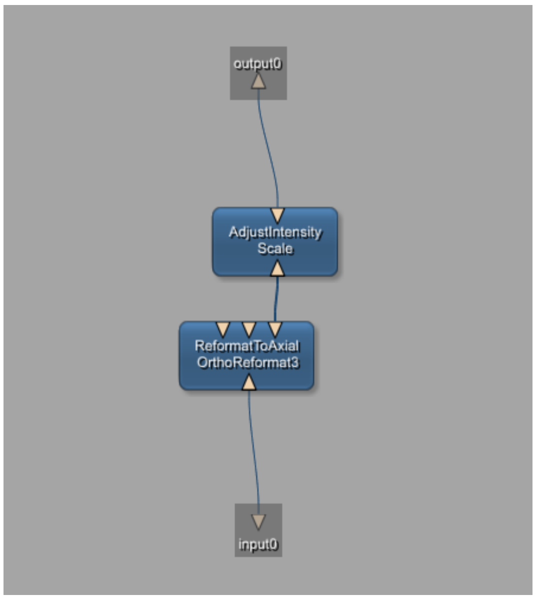


Figure 39: Internal network of "ReformatAndAdjustIntensity" module.

The aim of this module was to adapt the intensity values of the loaded MR image to a reference intensity range. This is done to simplify further image processing steps. The intensity was adapted by using the standard "Scale" module of MeVisLab, which scales the input image value to another interval.

There are three primary imaging planes that are utilized in MR imaging, namely axial, sagittal and coronal. To simplify further image processing steps and the insertion of a synthetic tumor, the module was also used to reformat the loaded MR image to axial view, which was achieved by applying the "OrthoReformat3" module.

5.1.2.3 "SetSeedPoint" Module

The "SetSeedPoint" module was used to manually set a seed-point at which the synthetic glioblastoma was subsequently inserted. This was done to ensure that the tumor was located within the brain region and could be achieved by simply clicking on the desired position in the MR image.

For this purpose, the standard MeVisLab "OrthoView2D" module - which provides a 2D view displaying the input image in three orthogonal viewing directions - was utilized, see figure 40.

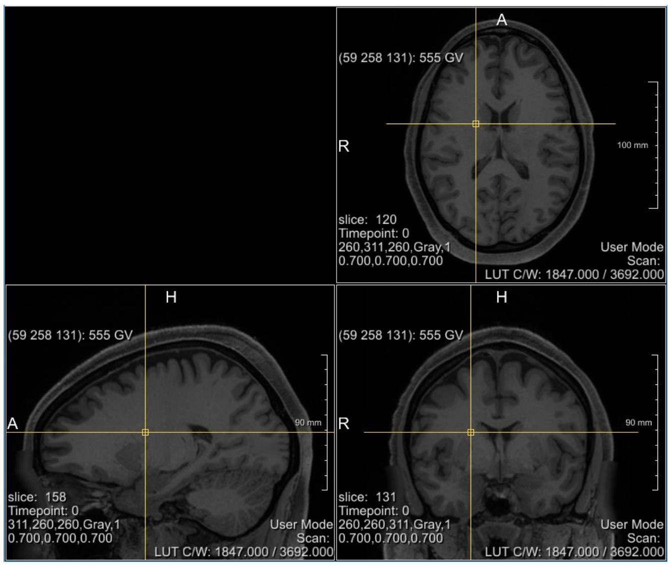


Figure 40: "OrthoView2D" module (three orthogonal viewing directions).

5.1.2.4 "MarkerList" Module

The internal network of the "MarkerList" Module can be seen in figure 41.

In order to save the selected seed-point, its x-, y- and z- coordinates were stored in a marker-list-container by using the MeVisLab "XMarkerListContainer" module.

The module "SoView2DMarkerEditor" allows interactive placement, editing and showing of markers.

Markers are stored in voxel coordinates, however, in order to use the position of the seed-point for inserting the synthetic tumor, the marker was required in world coordinates. Therefore, the coordinates were converted using the MeVisLab "WorldVoxelConvert" module.

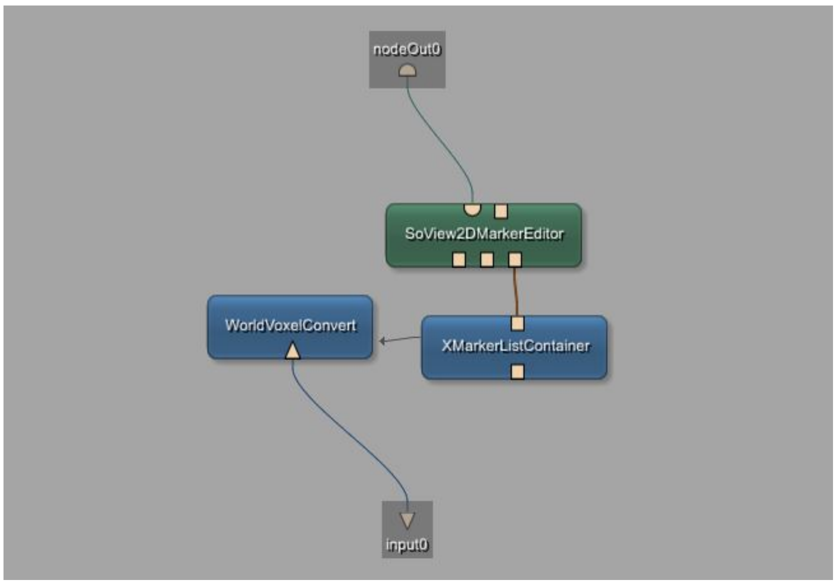


Figure 41: Internal network of "MarkerList" module.

5.1.2.5 "ProcessTumor" Module

The internal network of the "ProcessTumor" module can be seen in figure 42. It was used for further processing the generated tumor, inserting it into a provided MR image, and creating the corresponding ground truth.

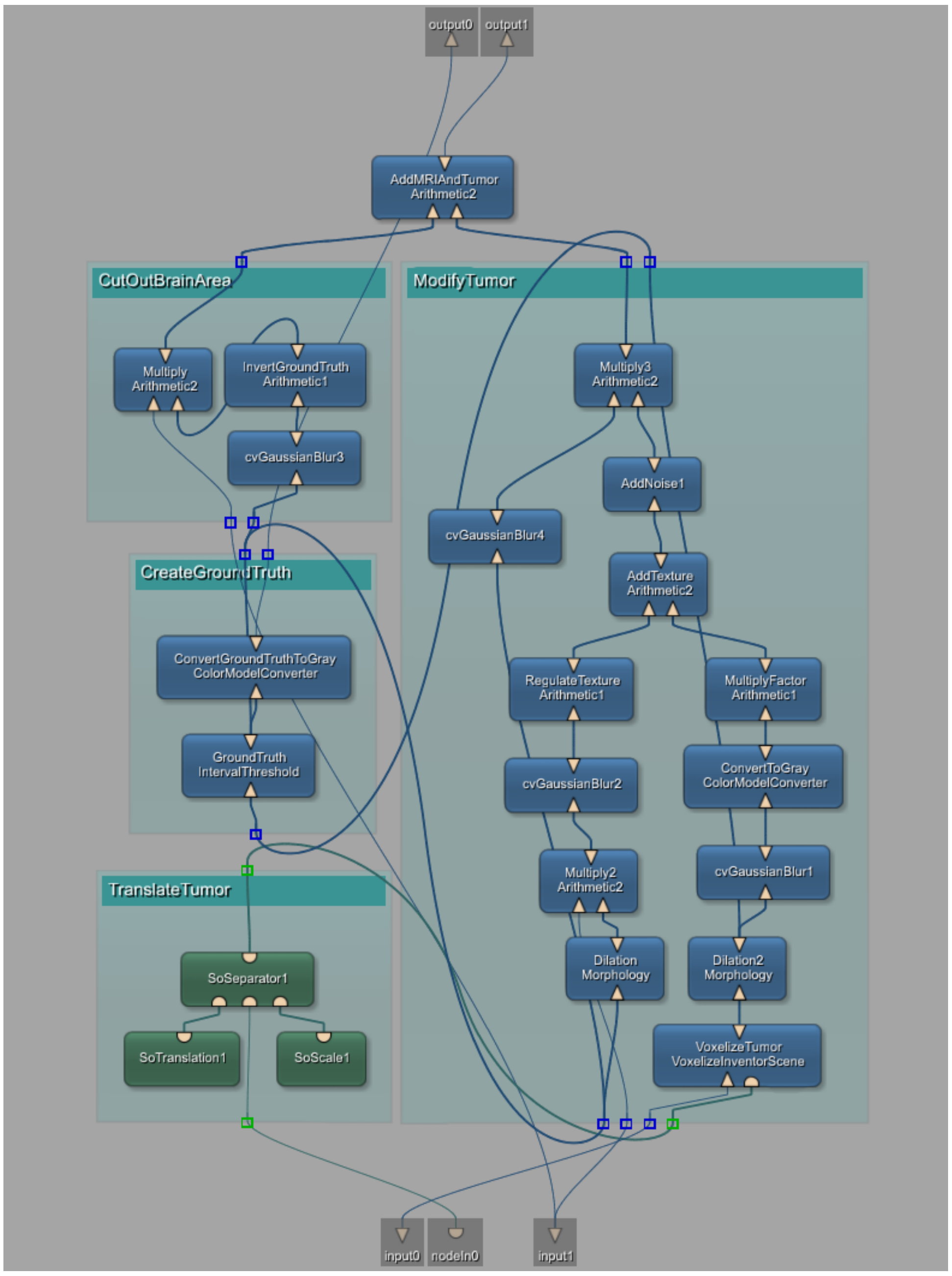


Figure 42: Internal network of "ProcessTumor" module.

Initially, the synthetic tumor had to be scaled and translated according to the selected seed point. This was done with the help of a "SoScale" and a "SoTranslate" module. By using a parameter connection between the "WorldVoxelConvert" module and the "SoTranslate" module, the parameters were synchronised with the converted coordinates of the selected seed-point, which were saved in the marker-list. After that, the visual scene graph of the tumor had to be voxelized (filled voxelisation). This was done with the MeVisLab "VoxelizeInventorScene" module, which allows a filled voxelization of closed surfaces, using a scan line algorithm.

As can be seen in figure 42, the "VoxelizeInventorScene" module was on the one hand used to further process the artificial glioblastoma and on the other hand to automatically create the ground truth.

The corresponding ground truth segmentation was created by using the filled voxelization of the synthetic glioblastoma and applying the "IntervalThreshold" module to it, which processes an image by filtering values that lie below/above a certain intensity value threshold. This threshold was set to 0.2. Voxels below this threshold were set to zero, whereas voxels above this threshold were set to one. This yields an exact ground truth of the synthetic tumor. Since the model of the ground truth was still in RGB format, the MeVisLab "ColorModelConverter" module was used to convert it to gray scale.

To depict the characteristics of a glioblastoma in post-contrast T1-weighted (T1Gd) MR images, a filled voxelization of the visual scene graph of the tumor was performed, where a dark gray fill color was used to represent necrosis. After that, a border was added to the tumor, where a light gray fill color was used to represent the contrast-enhanced surface. The according fill colors were based on the statistics of real glioblastomas. After the voxelization process, the voxelized tumor was slightly blurred, to smooth the edges between the fill color and the edge color. Then, the synthetic glioblastoma was converted to gray scale. In addition, the intensity of the tumor was increased by a factor of 1000, which could be slightly varied to manually modify the intensity of the synthetic glioblastoma. In order to add some texture to the tumor core, as it is seen in real T1Gd MRI data, the underlying structure of healthy brain tissue was exploited. The amount of texture that was added to the synthetic tumor was controlled by the "RegulateTexture" module. After that, gaussian noise ($\mu = 50$, $\sigma = 20$) was added in order to obtain a more realistic appearance of the lesion.

The resulting image was then multiplied with the previously calculated ground truth (which was blurred by the "GaussianBlur" module in the same extent as the tumor itself) so that only voxels within the tumor region were inserted into the MR image, see figure 43b.

In the top left corner of figure 42, one can see a group of several modules that were used to cut out the brain area of the MR images, exactly at the position where the tumor would later be inserted, see figure 43a. This was achieved by multiplying the MR image, with the inverted ground truth. It is important to note that the ground truth was blurred using the "GaussianBlur" module in the same extent as the tumor itself to ensure that the cut out area corresponded exactly to the position of the inserted tumor; additionally the edge between brain mass and tumor was smoothed.

Finally, the MR image with the cut out area and the completely processed synthetic glioblastoma were added using the "Arithmetic2" module, which performs arithmetic operations on two images.

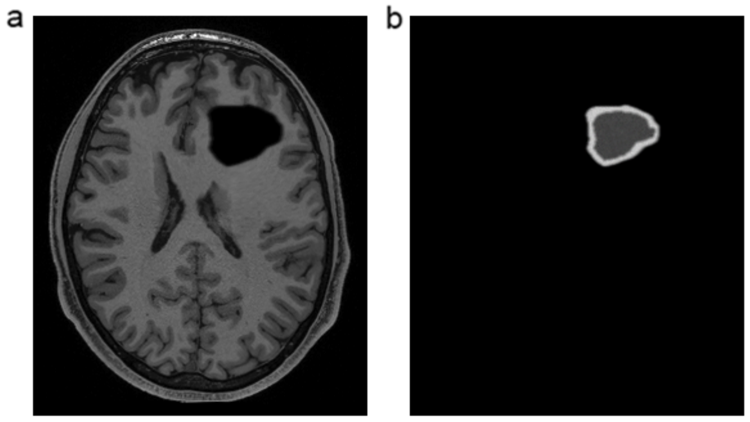


Figure 43: a.) Area at which tumor will be inserted, is cut out of MRI.
b.) Completely processed synthetic glioblastoma

5.1.2.6 "SetSeedPoint2" & "MarkerList2" Module

As already mentioned in chapter 5.1.1.4, a brain tumor causes displacement and compression of the surrounding healthy brain tissue. The "SetSeedPoint2" module was used to manually set a seed-point (by clicking on the desired position in the image) which indicated the center of brain tissue deformation. This strategy was necessary since the skull was not automatically detected by the algorithm. When the synthetic tumor was located close to the skull, the center of deformation could be further shifted towards the middle of the brain, leading to realistic looking results.

The function of the "MarkerList2" module equals exactly the function of the "MarkerList" module, which is described in chapter 5.1.2.4.

5.1.2.7 "DeformationOfBrainMass" Module

The internal network of the "DeformationOfBrainMass" module is illustrated in figure 44. It was used to simulate a tumor mass effect, which is discussed in chapter 2.3.2. To reduce computational effort, a simple sphere with variable radius ("SoSphere" module) was used instead of the visual scene graph of the actual synthetic tumor, in order to calculate the deformation.

Initially, the visual scene graph of the sphere had to be voxelized (filled voxelisation). This was achieved by the MeVisLab "VoxelizeInventorScene" module. After that, the euclidian distance transform of the voxelized sphere was computed by using the "EuclideanDistanceTransform" module. This was done to receive a gradually decreasing deformation from center to border. Then, the "itkGradientImageFilter" module was applied, which resulted in a dense vector field. Since the outward radial force that is applied to the brain by a growing tumor mass weakens by distance, the result of the euclidian distance transform was normalized and multiplied with a factor that could be varied between 10 and 25, depending on the desired strength of deformation.

Finally, the result was multiplied with the previously calculated gradient field. This yielded a stronger deformation at the center of the vector field which approached zero at the borders. The dense vector field was then used as input for the "ImageWarp" module, which deforms the MR image accordingly and is further explained in chapter 5.1.1.5.

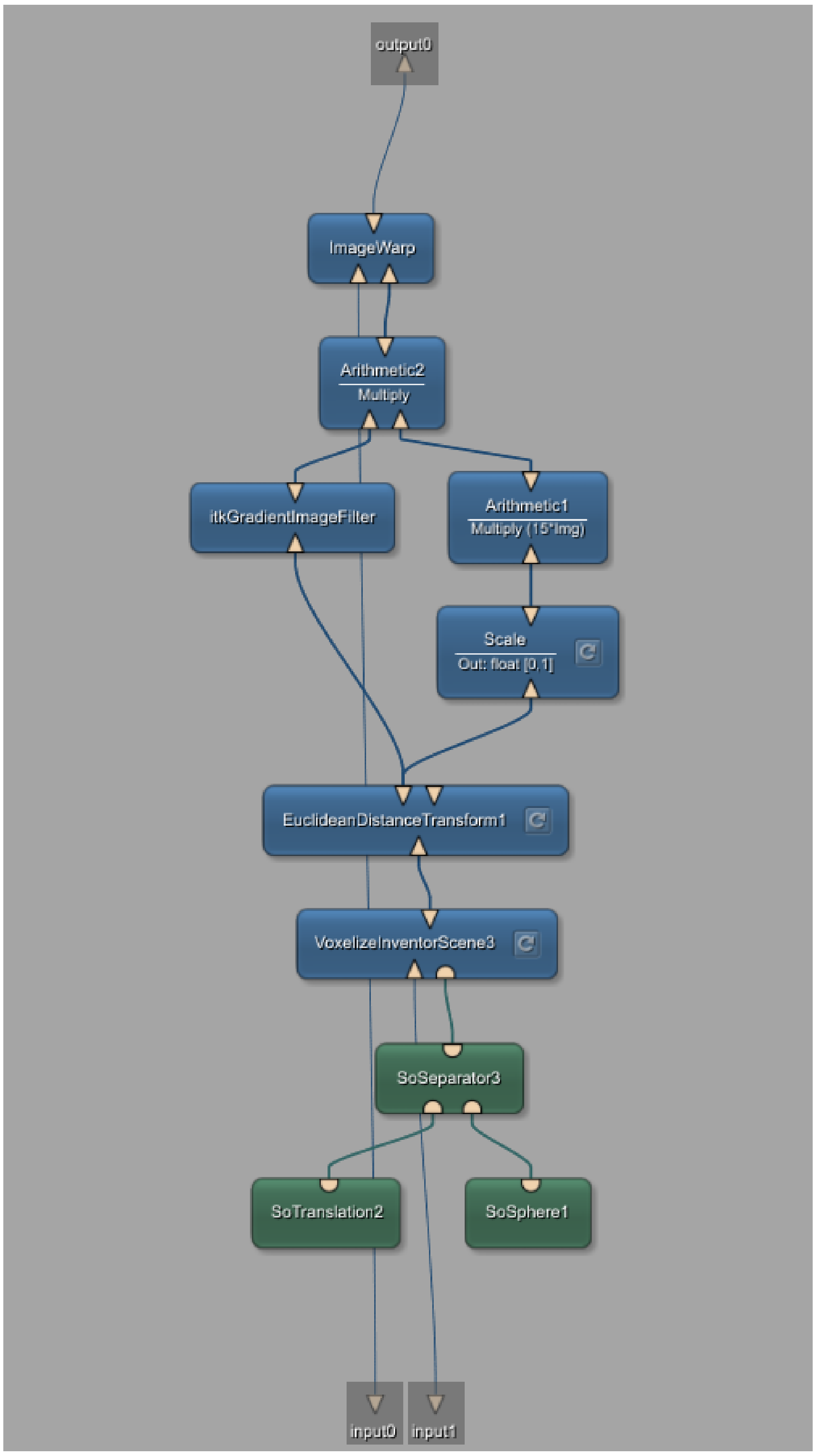


Figure 44: Internal network of "DeformationOfBrainMass" module.

5.1.2.8 "SaveSlices" Macro Module

The "SaveSlices" module is an adapted version of the "SaveAsSingleSlices" module from the work of B. Pfarrkirchner et al. [91].

MR images had to be connected to the first input of the module, while the corresponding ground truth image had to be connected to the second one.

The internal network of "SaveSlices" can be seen in figure 45. It was used to save the final brain MRI data containing the synthetic glioblastoma as well as the corresponding ground truth as single slices.

The MeVisLab "SubImage" module was utilized to extract sub-images from the input image (either the MR image or the segmentation mask). A python script was used to iterate over the single slices, which were saved to a selected file format by using the MeVisLab "ImageSave" module.

The panel of the "SaveSlices" module can be seen in figure 46. It allowed to select the path, where the single slice images and ground truth images were to be saved. Additionally, the file format could be chosen (TIFF or PNG).

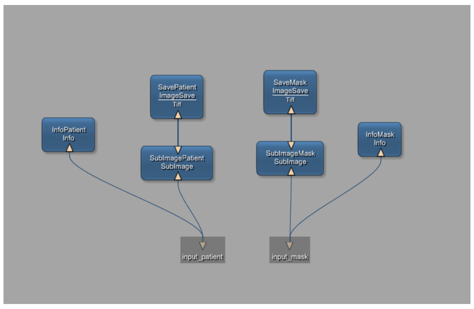
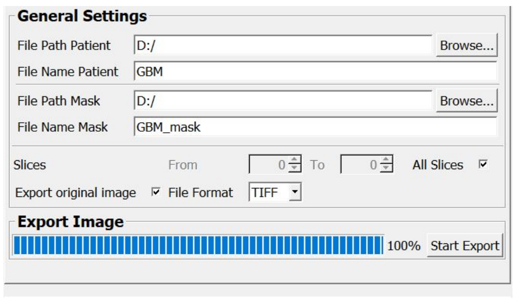


Figure 45: Internal network of "SaveSlices" module.



General Settings

File Path Patient	D:/	Browse...
File Name Patient	GBM	
File Path Mask	D:/	Browse...
File Name Mask	GBM_mask	

Slices From 0 To 0 All Slices ☑

Export original image ☑ File Format TIFF

Export Image

100% Start Export

Figure 46: Panel of "SaveSlices" module.

5.2 Brain Tumor Segmentation

This section gives a description of methods and datasets that were used to perform automatic segmentation of glioblastomas in MR images.

5.2.1 Available MRI Datasets

For this thesis, data was provided by the University Hospital of Giessen and Marburg in Germany and by the Human Connectome Project, WU-Minn Consortium (Principal Investigators: David Van Essen and Kamil Ugurbil; 1U54MH091657) funded by the 16 NIH Institutes and Centers that support the NIH Blueprint for Neuroscience Research; and by the McDonnell Center for Systems Neuroscience at Washington University.

5.2.1.1 Real Data

Post-contrast T1-weighted MR images of 14 patients suffering from glioblastoma multiforme were available for this work. Manual slice-by-slice segmentation has been performed by clinical experts to obtain the corresponding ground truth, where white foreground pixels (value 1) indicate the enhancing tumor as well as the necrotic tumor core; and black background pixels (value 0) represent everything else. Figure 47 illustrates four example MR images, along with their manually created ground truth segmentation.

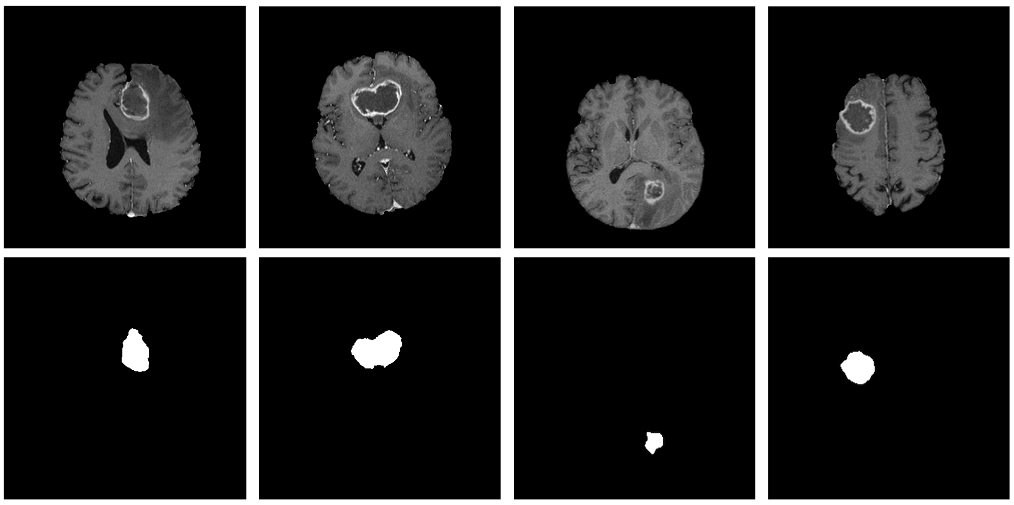


Figure 47: Excerpt from the available real MRI dataset, illustrating four MR images (top) along with their manually created ground truth (bottom).

A detailed overview of the dataset - including acquisition parameters - is given in table 1. The column 'Side Length of Voxels' indicates the in-plane resolution, whereas the column 'Slice Thickness' indicates the voxel depth. Dimensions of the whole 3D MR image are given by the columns 'Image Height', 'Image Width' and 'Number of Slices' respectively. Finally, the number of slices containing a tumor is indicated by the column 'Slices incl. Glioblastoma'.

Table 1: Overview of available real MRI data.

Case	**Side Length of Voxels**	**Slice Thickness**	**Image Height**	**Image Width**	**Number of Slices**	**Slices incl. Glioblastoma**
	mm	mm	voxel	voxel		
1	1.133	1.199	256	256	144	36
2	0.488	1.000	512	512	160	51
3	0.976	1.000	256	256	160	42
4	0.976	1.000	256	256	160	60
5	0.586	2.000	512	512	80	10
6	0.488	1.000	512	512	160	43
7	0.976	1.000	256	256	160	37
8	0.488	1.000	512	512	160	36
9	0.586	2.000	512	512	95	42
10	0.586	2.000	512	512	88	11
11	0.488	0.999	512	512	176	28
12	1.000	1.000	256	256	176	41
13	0.976	1.000	256	256	160	39
14	0.976	1.000	256	256	160	35

To sum up, there were 2039 real MRI slices available, out of which 511 slices contained glioblastomas.

These real MR images were used for evaluating the performance of the segmentation network. To this end, the dataset was split into validation and test data; specifically, the MRI data of ten patients was used as test set and the rest was used as validation set.

5.2.1.2 Synthetic Data

To supplement the previously discussed real MRI dataset, which was not large enough to successfully train a deep neural network, synthetic MR images were generated as described in section 5.1.

Artificial glioblastomas were inserted into 200 MR images of healthy subjects, of which 80 were post-contrast T1-weighted and 120 were native T1-weighted. These 3D MR images consisted of 260 image slices, where each slice was of dimension 260x310. The side length of each voxel as well as the slice thickness was 0.7mm.

Figure 48 illustrates four synthetic MR images, along with their automatically created ground truth. Again white foreground pixels (value 1) indicate the enhancing tumor as well as the necrotic tumor core; and black background pixels (value 0) represent everything else.

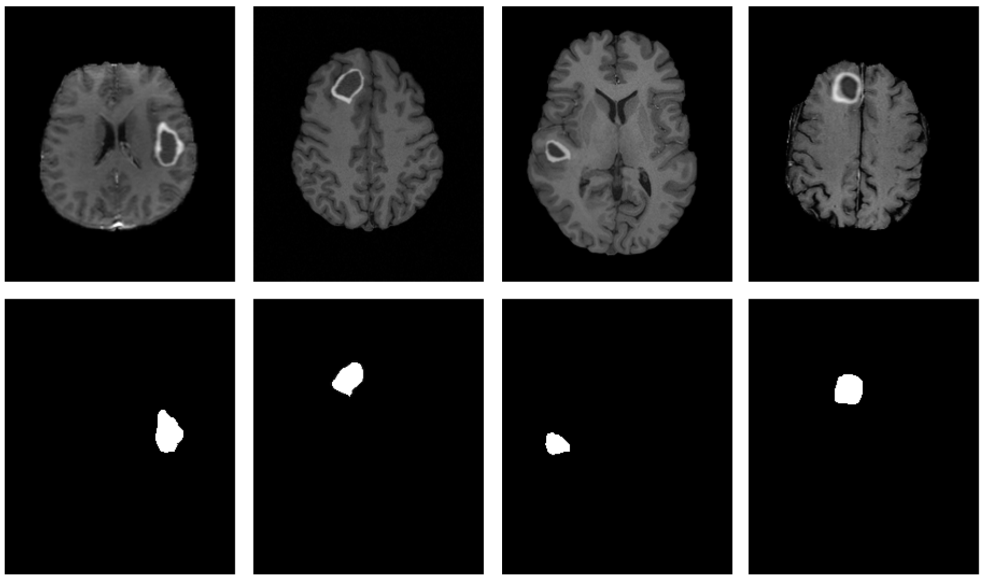


Figure 48: Excerpt from the synthetic MRI dataset, illustrating four MR images (top) along with their automatically created ground truth (bottom).

This synthetic MRI dataset containing 52000 slices, of which 13158 slices include an artificial glioblastoma, was used to train the segmentation network.

5.2.2 Data Preprocessing

Prior to training the segmentation network, several preprocessing steps were performed on the MRI data; including the removal of non-brain tissue (also called skull stripping), resizing of the MR images, contrast enhancement, as well as intensity normalization.

Skull Stripping:

Brain extraction, also known as skull stripping, is the first step in many brain image processing applications and refers to the removal of non-brain tissue, such as skull, eyes, fat, muscles and skin, from the 3D MRI volume of the whole head, see figure 49. These non-brain tissues could otherwise complicate further image processing and analysis steps.

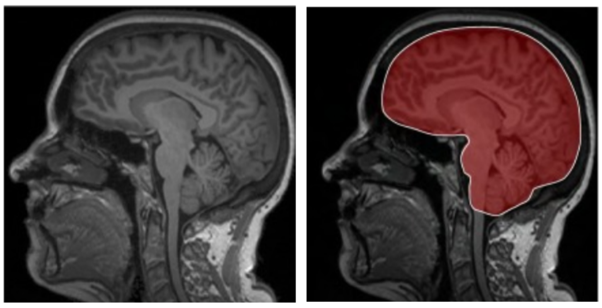


Figure 49: Illustration of the skull-stripping principle in sagittal view. Left: MRI of the whole head. Right: brain area to extract (indicated in red).

A common and accurate method for skull stripping in T1- and T2-weighted brain MR images, called brain surface extraction (BSE), was proposed by D.W. Shattuck et al. [92]. It is an edge-based technique that employs anisotropic diffusion filtering and operates using a 2D Marr-Hildreth edge detector, which applies low-pass filtering with a Gaussian kernel to the image and localizes zero crossings in the Laplacian of the filtered result. Edges between brain and skull are relatively well defined, however, non-brain structures such as the brain stem or optic nerves interrupt these boundaries. Therefore, these connections are broken using a morphological erosion operator. After the whole brain area is detected, a corresponding dilation operator is applied to reverse the effect of the previous erosion. Finally, a morphological closing operator is used to further improve the result.

For this thesis, brain extraction of all MR images was accomplished using the software tool BrainSuite, which provides the previously described brain surface extraction (BSE) method. BrainSuite is a useful image processing and analysis tool that offers a collection of open source software and allows largely automated processing of MR images of the human brain. It is produced and distributed as a collaborative project between Dr. David W. Shattucks research group at the Ahmanson-Lovelace Brain Mapping Center at the University of California, Los Angeles and Dr. Richard M. Leahys Biomedical Imaging Group at the University of Southern California.

Figure 50 illustrates a skull stripped MR image in axial view, sagittal view and coronal view.

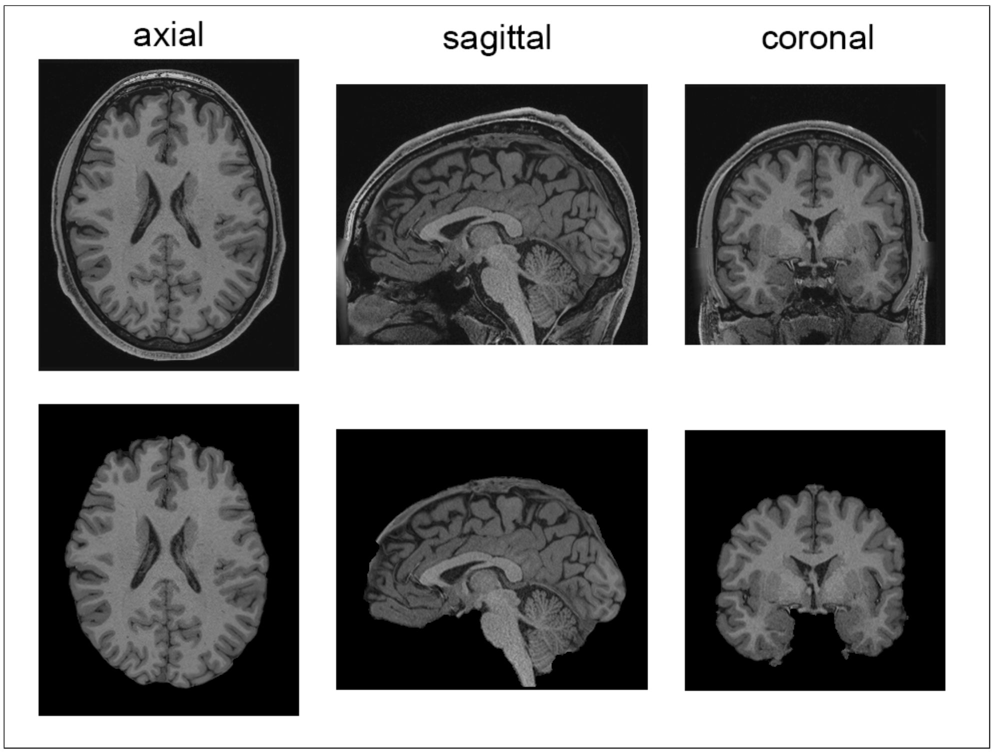


Figure 50: Illustration of MR image before (top) and after brain extraction (bottom). Left: axial view; middle: sagittal view, right: coronal view.

Resizing and Zero Padding:

As can be seen in section 5.2.1, the available MR images used for training and testing the segmentation network exhibit different dimensions. To introduce a standard, the scans were rescaled to 256x256x260, indicating the image width, image height, and number of image slices, respectively. This was accomplished by rescaling the image, such that the larger dimension (either image width or height) equalled 256 pixels. Subsequently, the smaller dimension was zero padded on both sides to also exhibit a length of 256 pixels.

This approach prevents distortion and translation of the underlying brain structure when resizing the image. It is important to note that all images were reformatted to axial view prior to rescaling.

An example of the rescaling approach and the subsequent zero padding for one slice can be seen in figure 51.

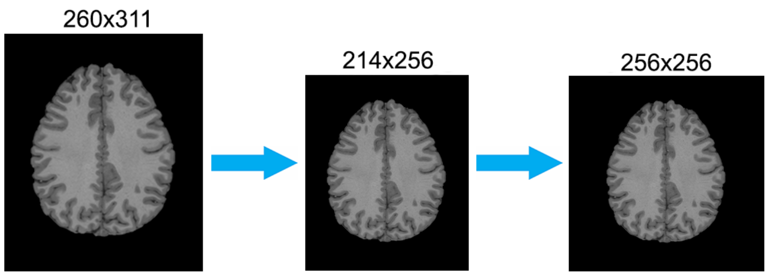


Figure 51: Illustration of rescaling approach for one slice. Left: original image; middle: image resizing, so that the larger dimension equals 256 pixels; right: zero pading at both sides of smaller dimension to obtain an image of 256x256.

Lanczos resampling was used to resize the available MRI volumes. This interpolation method uses a convolution kernel to interpolate pixel values of the input image in order to calculate the pixel values of the output image. The convolution kernel is defined as a sinc function, which is windowed by the central lobe of a second longer sinc function.

For MR image volumes with less than 260 slices, the number of slices was also adjusted accordingly.

Contrast Adjustment

In order to further enhance the contrast of the available MRI data and reduce the variation between post-contrast T1-weighted and native T1-weighted images, histogram equalization and gamma correction was performed.

The application of both methods on an example image is shown in figure 52.

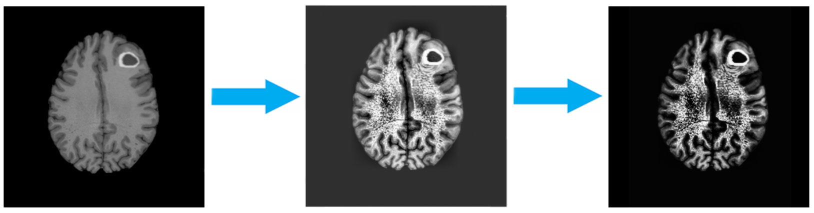


Figure 52: Illustration of the contrast adjustment on an example image.

Normalization

Since MRI intensities are not standardized, intensity normalization is an essential step in the preprocessing chain for automatic segmentation techniques. It allows for data from different MRI scanners, obtained with varying acquisition parameters and scanning protocols, to be processed by the same automatic algorithm.

To this end, min-max normalization was conducted on each single slice, which performs a linear transformation on the given data. Specifically, it maps intensity value i of image I to i' in the range $[min_{new}, max_{new}]$ by calculating:

$$i' = \frac{i - min_I}{max_I - min_I}(max_{new} - min_{new}) + min_{new} \tag{23}$$

where max_{new} and min_{new} are the new minimum and maximum values of I.

The new intensity value range was specified to be $[0, 1]$, hence, the equation can be simplified as follows:

$$i' = \frac{i - min_I}{max_I - min_I} \tag{24}$$

5.2.3 Architecture of the Segmentation Network

The network architecture used for the segmentation task of this thesis was based on the U-Net, which is discussed in section 4.2. The standard U-Net architecture, proposed by O. Ronneberger et al. [71] , was slightly modified to provide a better fit for the segmentation problem at hand. A detailed illustration of the employed network can be seen in figure 53. It comprises an initial contracting path (convolutional encoder part) and a subsequent expanding path (convolutional decoder part).

The contracting path comprises repeated application of convolution blocks followed by 2x2 max pooling operations for down-sampling. Convolution blocks in the contracting path consist of two 3x3 convolutions, each followed by a batch normalization layer, ReLU activation function and a dropout layer. The number of feature maps is doubled at each down-sampling step. The expanding path consists of repeating blocks, in which feature maps are up-sampled to expand the dimension and subsequently concatenated with the corresponding feature map of the contracting path. These so called 'skip-connections' allow to combine coarse- and fine-level features, in order to recover spatial information lost during down-sampling. In contrast to the standard U-Net architecture, no 2x2 convolution is applied after the up-sampling step. Each of these blocks is followed by a regular 3x3 convolution, a batch normalization layer, ReLU activation function and a dropout layer. Finally, a 1x1 convolution is applied at the last layer of the network followed by a sigmoid activation function, in order to obtain a pixel-wise segmentation mask of the same size as the input image.

Besides omitting the 2x2 convolution in the expanding path of the network and the additional application of batch normalization and dropout after each regular 3x3 convolution, one important modification is that 'same padding' was used for convolutions instead of 'valid padding'. This yielded a segmentation mask, which exhibits the same size as the input image. In contrast, with 'valid padding' (as used in the standard U-Net), the output image would be smaller than the input by a constant border width.

The network architecture consists of 23 convolutional layers and was designed for gray scale input images of dimension 256x256, as can be seen in figure 53. In the lowest resolution feature maps exhibit a size of 16x16, with a maximum number of 256 feature channels.

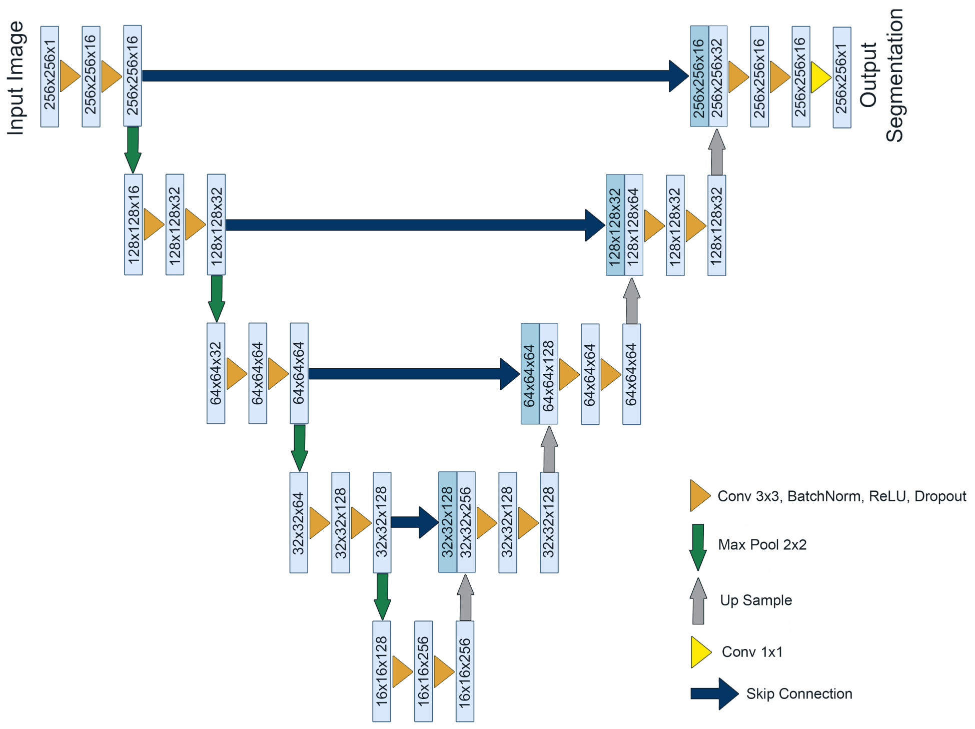


Figure 53: Illustration of the network architecture used for the segmentation task of this thesis. Each blue box represents a multi-channel feature map; dark blue boxes correspond to copied feature maps. The x-y dimensions of each feature map and the corresponding number of feature channels are denoted within each box.

5.2.3.1 Double Convolution Layers

Another modification of the standard U-Net architecture was the application of double convolution layers, which means that there were two subsequent 3x3 convolutions before each activation to increase the receptive field. According to G. Kim [93], applying two successive 3x3 convolutions is equivalent to having one 5x5 convolution, but at the same time requires less computational effort by a factor of 25/9.

However, preliminary evaluations showed that this modification did not significantly improve the segmentation performance of the model for the task at hand, while increasing the computation time. Therefore, this modification was not employed in the final architecture used for this thesis.

5.2.4 Implementation Details

The implementation was realized using *Python 3.6*, the open-source machine learning framework *TensorFlow 1.6* and the high-level API *Keras 2.1.5*. Further information regarding the functionality of *TensorFlow* and *Keras* can be found in section 3.4.

An overview of the content and function of each project file is given below.

- **architecture.py:** In this file, the previously described model architecture was defined using *Keras*. Furthermore, multi-GPU training and loading of pre-trained weights was managed.

- **data_generator.py:** This custom-written data generator enabled real time data feeding and preprocessing during the training and testing phase and is described in detail in the following section.

- **losses.py:** Standard and custom loss functions, which are further discussed in the following section, were defined in this file.

- **metrics.py:** Evaluation metrics to compute similarity measures between network predictions and ground truth, which were used for validating the performance of the segmentation network, were implemented in this file. These metrics included, the Dice similarity coefficient, the Hausdorff distance, as well as sensitivity and specificity.

- **segmentation.py:** This script was used for both training and testing the deep neural network model defined in 'architecture.py'. The according training and testing procedures are discussed in detail in section 5.2.5 and 5.2.6, respectively. Custom callbacks for saving model weights and log-values were also implemented in this file.

5.2.5 Training the Segmentation Network

In order to optimize the trainable parameters of the segmentation network, a supervised learning approach was employed. Therefore, the generated synthetic input images along with their corresponding ground truth segmentations were used to train the model. A detailed description of the training procedure is provided in the following sub-sections.

5.2.5.1 Hardware

Multi-GPU training was performed on a GPU-Cluster equipped with eight NVIDIA® Tesla® K20Xm GPUs, each having 6GB GDDR5 graphics memory. During the training phase data was prepared and provided in parallel by an Intel® Xeon® CPU E5-2630 v2 with 6 cores.

5.2.5.2 Loss Function

As already discussed in section 3.2.5, the learning problem of a neural network can be formulated as the minimization problem of a loss function, which represents the distance between the network output and the desired output and is chosen according to the present task.

A common choice for segmentation networks is the cross entropy loss. In particular, as the task of this thesis is to segment an image into foreground and background pixels, the binary cross entropy loss is suitable to perform a pixel-wise binary classification, resulting in an output which can be interpreted as a probability value between zero and one. The binary cross-entropy loss is defined as follows:

$$L_{CE} = -\sum_{n=1}^{N} \{t_n ln(p_n) + (1 - t_n) ln(1 - p_n)\} \tag{25}$$

where N is the number of pixels of the predicted segmentation image $p_n \in P$ and the corresponding ground truth image $t_n \in T$.

When the predicted output value of the model diverges from the actual value, the cross entropy loss increases. An ideal model would produce a cross entropy loss of zero.

It is important to note that in medical images the region of interest (foreground) is usually quite small, as it was the case for MR images processed in this thesis. This leads to a significant class imbalance in the data. Hence, when using the conventional cross-entropy loss, the learning process is likely to get trapped in local minima, which hampers training. The data used in this work, for example, exhibited around 100 times more background pixels than foreground pixels. This results in a model with a strong bias towards background pixels, whereas foreground regions are often not detected well.

To overcome this issue, a loss based on the Dice similarity coefficient (section 3.3.3) was utilized, similar to the one proposed by F. Milletari et al. [94].

The employed Dice loss is defined as follows:

$$L_{Dice} = 1 - \frac{2 \cdot \sum_{n=1}^{N} p_n t_n}{\sum_{n=1}^{N} p_n + \sum_{n=1}^{N} t_n} \tag{26}$$

where N is the number of pixels of the predicted segmentation image $p_n \in P$ and the corresponding ground truth image $t_n \in T$.

Since the aim was to maximize the Dice coefficient, the loss was defined as $1 - DSC$, in order to obtain a minimization problem, which could be solved by gradient descent optimization. The definition of the Dice coefficient in this loss is equivalent to the classic definition discussed in section 3.3.3, however, no threshold was applied to the network output, which made the Dice loss differentiable and allowed for direct implementation in an optimization algorithm based on backpropagation.

Preliminary tests showed that better segmentation results could be achieved on the validation set, when using the Dice coefficient loss instead of the cross entropy loss. Furthermore, even slightly better results could be obtained on the validation set, when cross entropy loss and Dice coefficient loss were combined. The employed combined loss, which was eventually used for the segmentation task of this thesis, is defined as follows:

$$L = L_{CE} + L_{Dice} \tag{27}$$

5.2.5.3 Optimizer

The aim of the training process was to tune the internal parameters of the network so as to minimize a loss function. Hence, the choice of a suitable optimizer was essential to enable proper learning and fast convergence.

In order to select a suitable optimizer for the task at hand, the performance of several gradient descent optimization algorithms has been evaluated in preliminary tests. An introduction to gradient descent optimization can be found in section 3.2.5.

A short overview of the investigated optimizers, adapted from [95], is given below, followed by a direct comparison of the corresponding convergence curves, which were acquired in preliminary evaluations.

- **SGD (Stochastic Gradient Descent)**:

  Stochastic gradient descent computes the gradient of the loss function with respect to the trainable parameters for each training example. In practice the term SGD is also used, when mini-batch gradient descent is performed, as it was in this work. Mini-batch gradient descent performs an update for every mini-batch of all training examples and is commonly employed when training a neural network. However, SGD does not perform well in valleys of the loss function, where the surface is much steeper in one dimension than in another, which is usually the case around local minima. In such settings, SGD starts to oscillate and only makes small progress in direction of the local minimum.

- **RMSprop (Root Mean Square Propagation):**

  RMSprop is an adaptive learning rate method which restricts the previously mentioned oscillations. It divides the learning rate by an exponentially decaying average of squared gradients, which has the effect of balancing the step size. Therefore, RMSprop usually converges faster than SGD. Furthermore, it is quite memory-efficient since it just stores rolling averages.

- **Adam (Adaptive Moment Estimation):**

  Adam is another adaptive learning rate method, which computes adaptive learning rates for each network parameter. However, in contrast to RMS prop, which only stores an exponentially decaying average of past squared gradients, Adam also stores an exponentially decaying average of past gradients. Therefore, Adam can be seen as a combination of RMSprop (exponentially decaying average of past squared gradients) and momentum (exponentially decaying average of past gradients). This optimization method is computationally efficient and suitable for large datasets and networks with many parameters.

- **Nadam:**

  Nadam combines Adam and Nesterov accelerated gradient (NAG). NAG introduces some kind of foresight, which means that the gradient is not calculated with respect to the current network parameters but instead with the approximated future position of the parameters.

Figure 54 shows a comparison of the smoothed convergence curves for the previously described gradient descent-based optimization algorithms, applied to the present segmentation task. It is clearly visible that Adam and Nadam outperformed both SGD and RMSprop. Since Nadam converged even slightly quicker than Adam, it was the optimizer of choice for this thesis.

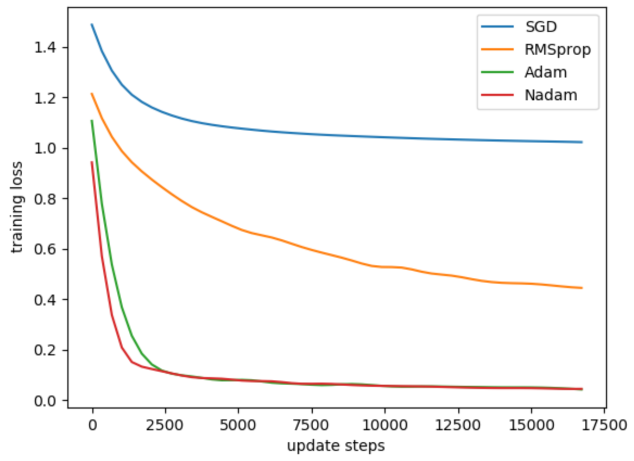


Figure 54: Comparison of the smoothed convergence curves for different optimizers. Blue: SGD, orange: RMSprop, green: Adam, red: Nadam.

5.2.5.4 Real-time Data Feeding

Typically, the entire dataset for training (or testing) a neural network is loaded at once, which is both memory and time-consuming for a large amount of available data. For very large datasets, such as the one used in this thesis, even the best configuration will not allow to load all data at once, due to limited memory space. Therefore, an efficient solution to this issue had to be found.

A convenient way to cope with this problem, was to generate the dataset on multiple cores in real time and directly feed it to the segmentation network. To this end, a custom Python class named *DataGenerator* was implemented, to enable real-time data feeding.

The implemented generator allowed to create batches of arbitrary size directly from source files, which means that there was no need to separately store data in a NumPy array. Hence, to uniquely identify samples of the dataset, the local paths of all MR images and the corresponding ground truth segmentations of a specified directory were stored in lists. These lists were sorted to guarantee that each image and its associated segmentation mask were stored at the same position and therefore have the same 'ID'.

Generators are iterators that do not store all data in memory but generate it on the fly. Typically, the keyword 'yield' is used instead of 'return' when working with data generators, which ensures that exactly one batch is generated and fed to the network each time the generator is called by the environment. During data generation, the dataset was divided into N partitions, where N = number of samples / batch size. For each batch, specific images and their corresponding segmentation masks were loaded from the previously described lists in a predefined exploration order. This exploration order was shuffled at each new epoch, in order to provide a random exploration scheme.

Since the implementation allows for multiprocessing, data generation did not become the bottleneck in the training process. Using this method, data could be generated in parallel by the CPU and then directly be fed to the GPU.

5.2.5.5 Training Process

Training of the previously presented segmentation network was performed in the file segmentation.py. Before the training process was started, the number of epochs, the desired batch size and the number of GPUs to be employed, had to be specified. Furthermore, a batch size of 64 was used and the number of epochs was set to 40, where each epoch represented an iteration over the entire dataset. The employed model was trained from scratch, using eight GPUs for efficient multi-GPU training. Pre-trained weights were only loaded for testing the final segmentation performance of the network after the entire training process, which is further discussed in section 5.2.6.

The previously generated synthetic 2D MR images of size 256x256 along with their corresponding ground truth segmentations (see section 5.1) were used in axial view to train the network. The current segmentation performance was validated after each epoch using a validation dataset consisting of 1040 MR images of real patients suffering from glioblastoma multiforme. To this end, separate data generators were created for training and validation data. Since training and testing are both implemented in the file segmentation.py, the variable '*training*' either had to be set to 'TRUE' to train the model or to 'FALSE' to initiate the final model evaluation.

The employed deep learning model was defined in architecture.py. Before it could be trained, multi-GPU training had to be enabled using the *Keras* method *utils.multi_gpu_model*(), which replicates a model on different GPUs. In particular, this method implements single-machine multi-GPU data parallelism. This means that the model's input was divided into sub-batches and a copy of the model was applied to each sub-batch. Every model copy was then executed on a dedicated GPU. Finally, the results were merged back into one batch of original size on the CPU. Thus, this method allowed quasi-linear speed-up. Furthermore, the learning process had to be configured before training the model, which was done via the *model.compile*() method of *Keras*. This method receives three arguments: the optimizer, the loss function and a list of evaluation metrics. As already mentioned in previous sections, a combined loss consisting of cross-entropy loss and Dice loss was employed and Nadam was used as optimizer. The Dice similarity coefficient as well as sensitivity and specificity (implemented in metrics.py) were utilized as evaluation metrics during the training phase.

Several custom $Keras$ callback functions were implemented in segmentation.py to accomplish specific tasks at given stages of the training phase. Such functions can be used to get a view on internal states and statistics of the model during training. In the case of this work, they were used to save model weights whenever the validation loss decreased at the end of an epoch and to store log-values of the training process for later illustration.

Finally, the learning process of the network was started using the $Keras$ method $model.fit\_generator()$, which trained the model on data generated batch-by-batch by the implemented data generator described in section 5.2.5.4. For better efficiency, the data generator operated in parallel. A list of the previously defined callback functions and the data generators for both training and validation data were passed as arguments to the $model.fit\_generator()$ method. The relevant methods of the implemented callbacks were then called at the according stage of training. The initial learning rate was set to 0.001 and dropout was applied as specified in section 5.2.3, using a dropout rate of 0.3.

In each update step, a pixel wise error was computed using a sigmoid activation function over the final layer of the network combined with the employed loss function. Network weights were then adapted accordingly. During the entire training phase, 32480 weight updates were performed.

During each epoch, a running average of the training loss and evaluation metrics was printed to enable convenient monitoring of the learning process. When the validation loss decreased, the current model was saved at the end of an epoch using the $Keras$ method $model.save\_weights()$, which saved the weight parameters as a HDF5 file.

In deep neural networks with many convolutional layers, a good weight-initialization is essential for successful training. Therefore, weights were initialized using a Glorot uniform initializer, also known as Xavier uniform initializer, which draws samples from a uniform distribution within the range $[-a, a]$, where a is defined as $\sqrt{6/(u_{in} + u_{out})}$, where u_{in} denotes the number of input units in the weight tensor and u_{out} denotes the number of output units in the weight tensor.

5.2.6 Testing the Segmentation Network

In order to provide an unbiased evaluation of the final model, the segmentation network was tested on unseen data consisting of 1040 real MR images of patients suffering from glioblastoma multiforme. The test dataset was only used to asses the generalization ability of the fully trained model.

5.2.6.1 Hardware

Testing was conducted on a single NVIDIA® Tesla® K20Xm GPUs, having 6GB GDDR5 graphics memory. During the testing phase data was prepared and provided in parallel by an Intel® Xeon® CPU E5-2630 v2 with 6 cores.

5.2.6.2 Evaluation Process

Testing the previously trained segmentation network was also conducted in the file segmentation.py. In order to evaluate the final model, the variable '*training*' had to be set to FALSE. It is important to note that dropout was deactivated during the testing phase.

For the testing process, pre-trained weights were loaded by the *Keras* method *model.load_weights*(), which expects a path to previously stored weights of the model and loads these weights from the specified HDF5 file.

The evaluation of the entirely trained segmentation network was performed using the *Keras* method *model.predict*(), which generates output predictions for given input samples. The method expects the input data as a NumPy array and returns a NumPy array of predictions. Therefore, the test images and their corresponding ground truth segmentations had to be stored in NumPy arrays prior to the testing process. Furthermore, the batch size had to be specified, which was again set to 64.

Using the predicted network output along with its corresponding ground truth, specific evaluation metrics were computed, namely the Dice similarity coefficient, the sensitivity, the specificity, and the Hausdorff distance, which are discussed in section 3.3.3.

The corresponding methods for computing these evaluation metrics were implemented in the file metrics.py. Each of these methods received the predicted segmentation of the network as well as the ground truth segmentation and calculated a certain similarity measure. Since the output of the sigmoid activation function was in the range of (0, 1), a threshold of 0.5 was applied to the network output, in order to obtain a binary prediction.

As proposed by G. Csurka et al. [96], measuring per-image scores for segmentation tasks is important, since evaluations over the whole test dataset do not allow to distinguish models that perform moderately on all images from models that perform very well on some images and very bad on others; whereas plotting histograms of per-image scores allows such a differentiation. Furthermore, they argue that the use of per-image scores enables comparison to a specified threshold, which can be useful in real applications, where a minimum level of quality is usually expected.

In order to perform an additional per-image evaluation, Dice coefficient, sensitivity, specificity and Hausdorff distance were computed for each individual test image and corresponding histograms were created to represent the according score distribution.

6 Results

This chapter presents the final results of this thesis. In section 6.1, examples of the synthetic training dataset are illustrated, which were created using the proposed data generation method (discussed in section 5.1). The actual segmentation results, obtained with the presented deep convolutional neural network, are provided in section 6.2. The segmentation performance of the model was assessed for the entire test dataset and additionally per-image evaluation was performed. Furthermore, several test images along with their predicted segmentation mask and ground truth are illustrated.

6.1 Synthetic MRI Data

In this section, results of the proposed method for generating MRI data containing synthetic glioblastoma multiforme and a simulated tumor mass effect are provided. In total, a training dataset consisting of 52000 images was generated with the presented approach. All images illustrated in this section are already skull stripped, which is discussed in section 5.2.2.

6.1.1 Simulation of the Tumor Mass Effect

Figure 55a shows an image of a healthy brain without a simulated tumor mass effect, whereas figure 55b illustrates the same MR image including a simulated tumor mass effect in the upper right area of the brain.

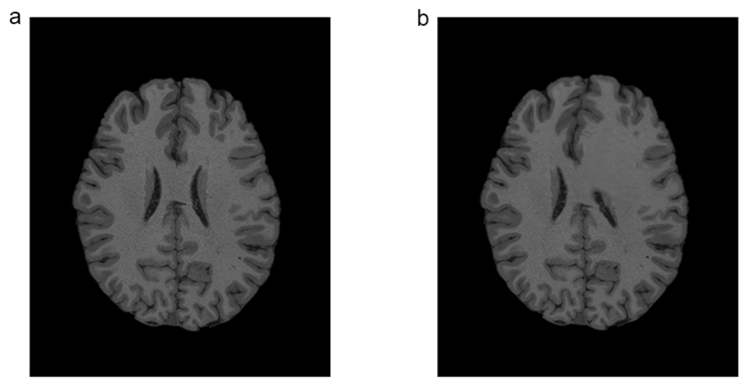


Figure 55: a.) healthy brain without tumor mass effect; b.) the same MR image demonstrating a simulated tumor mass effect in the upper right area.

6.1.2 Generated Synthetic Glioblastomas

Figure 56a illustrates a representative selection of synthetic MR images containing artificially generated glioblastomas as well as a simulated tumor mass effect, along with their automatically created ground truth. For comparison purposes, six real MR images of patients suffering from glioblastoma multiforme and their manually obtained ground truth segmentations are shown in figure 56b.

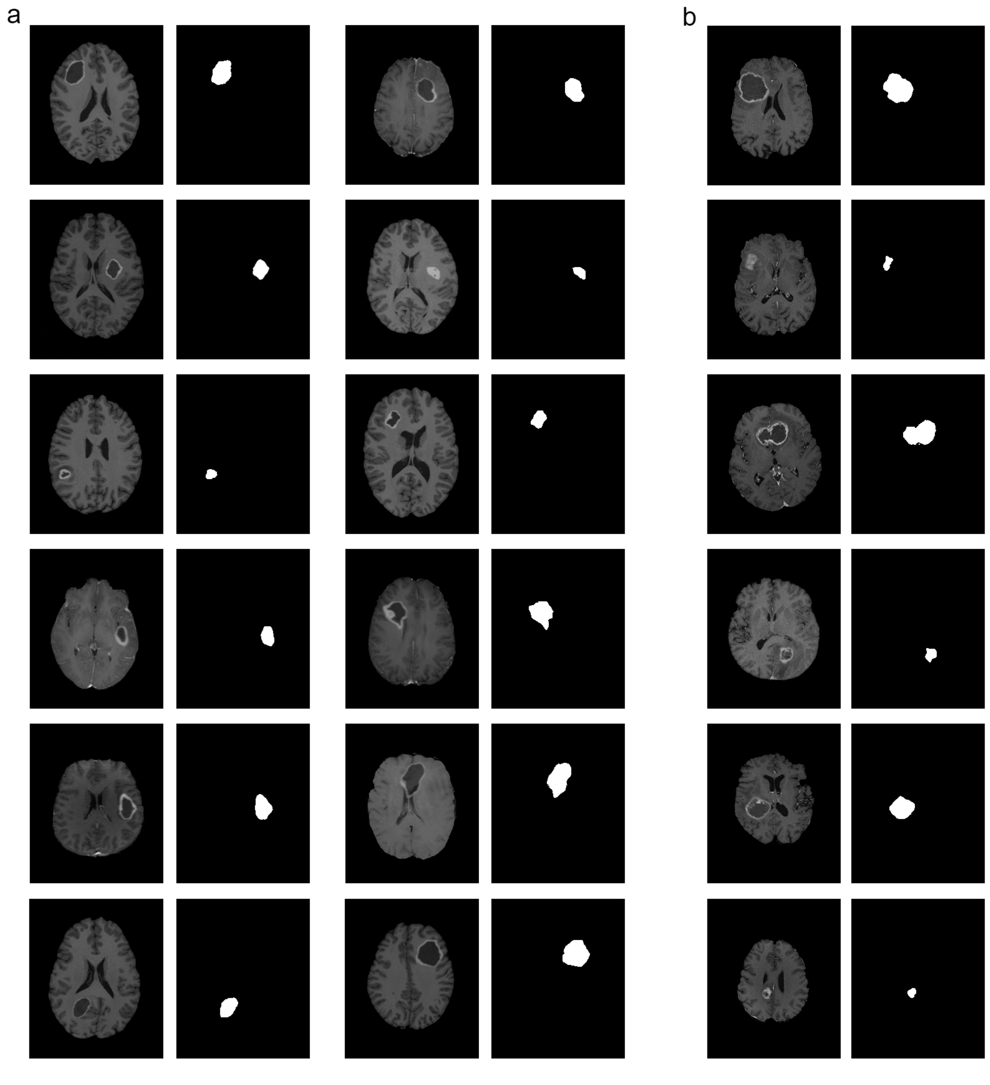


Figure 56: a.) synthetic MR images containing artificially generated glioblastomas as well as a simulated tumor mass effect, along with their automatically created ground truth; b.) real MR images of patients suffering from glioblastoma multiforme with manually obtained ground truth.

6.2 Segmentation Results

This section contains a concise overview of the segmentation results achieved with the proposed method. Results of per-image evaluation - as previously discussed - are provided in section 6.2.1.

The presented segmentation network was trained with 52000 synthetic 2D MR images and tested on 2600 real 2D MR images. Training was performed for 40 epochs, which corresponds to 32480 individual update steps. Figure 57 illustrates the smoothed convergence curves of the training loss and the training Dice coefficient, respectively. On average, the model took 10 hours to train with multi-GPU utilization.

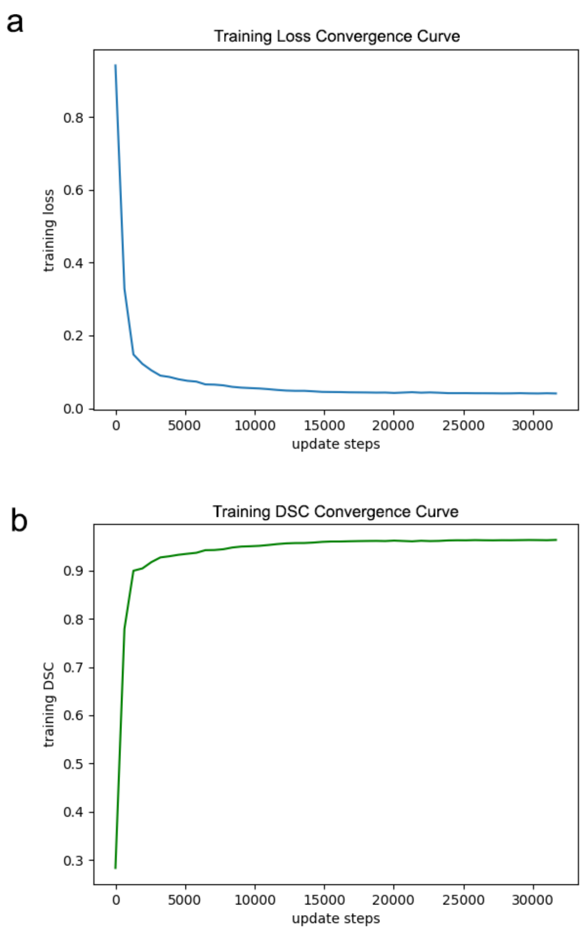


Figure 57: Smoothed convergence curves of training loss (a) and training Dice coefficient (b).

The final segmentation performance of the fully trained network was evaluated by computing the Dice similarity coefficient (DSC), the Hausdorff distance (HD), as well as sensitivity and specificity of the network output for the given test dataset. The according evaluation results are presented in table 2.

Table 2: Evaluation of segmentation results for the presented segmentation network, which was trained with 52000 synthetic MR images and tested on 2600 real MR images. Dice similarity coefficient (DSC), Hausdorff distance (HD), sensitivity and specificity were computed as evaluation metrics.

Training Steps	**Test DSC**	**Test HD**	**Test Sensitivity**	**Test Specificity**
	%	pixels	%	%
32480	82.449	9.002	73.447	99.987

All results in table 2 are obtained over the entire test dataset. Additionally, per-image evaluation is provided in the following section.

6.2.1 Per-image Evaluation

As discussed in section 5.2.6, per-image evaluation is essential in order to be able to better assess the results achieved with the proposed method. Otherwise, it would, for example, not be possible to distinguish between a model that performs moderately on all test images and a model that performs very well on some test images and very bad on others.

To this end, Dice similarity coefficient, sensitivity, specificity and Hausdorff distance were computed for each test image independently and histograms were created to visualize the per-image evaluation score distribution of the model. The corresponding histograms can be seen in figure 58.

In figure 59 qualitative segmentation results are provided, which show a representative selection of test images illustrating the predicted segmentation along with the actual ground truth segmentation.

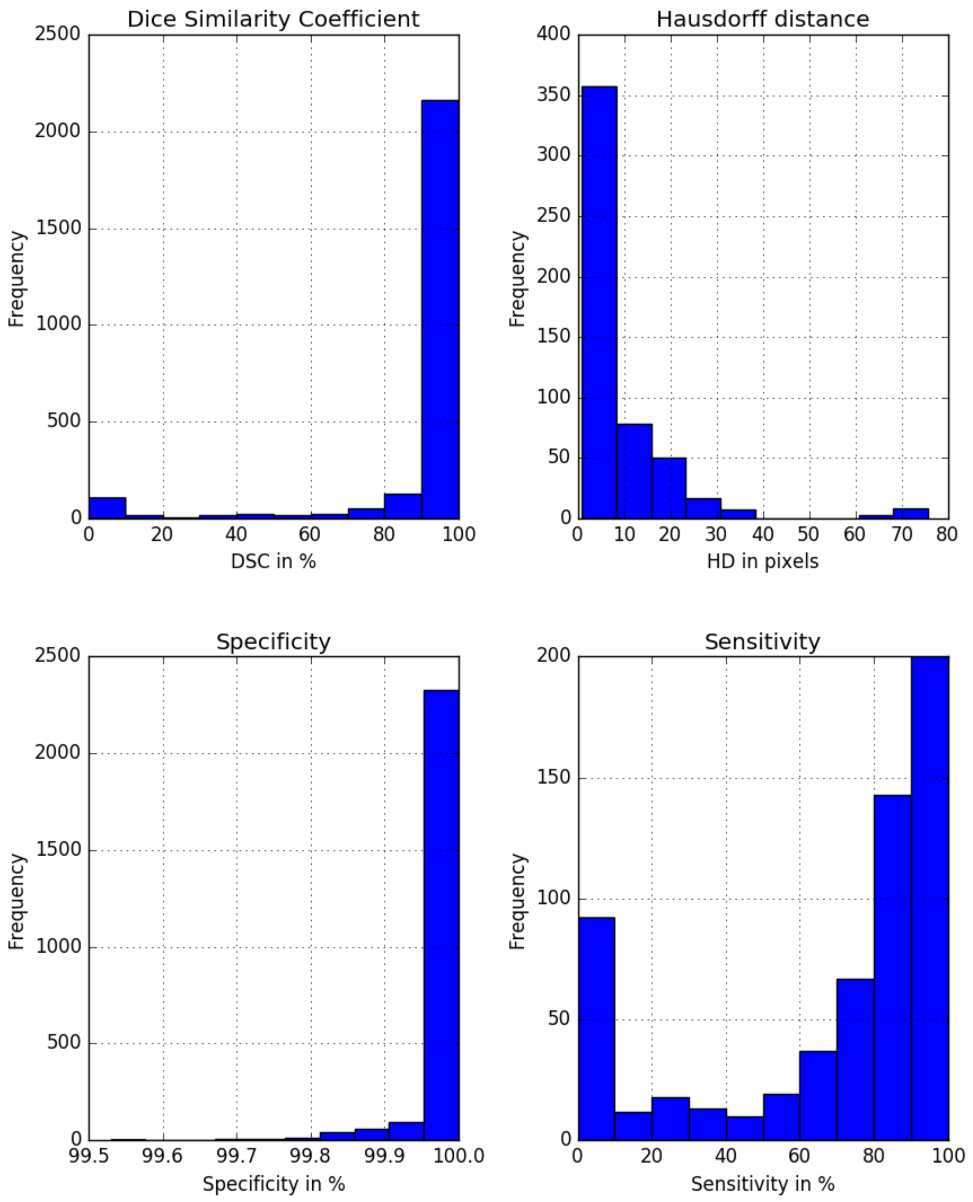


Figure 58: Histograms of per-image evaluation scores. Top left: histogram of Dice similarity coefficient; top right: histogram of Hausdorff distance; bottom left: histogram of specificity; bottom right: histogram of sensitivity.

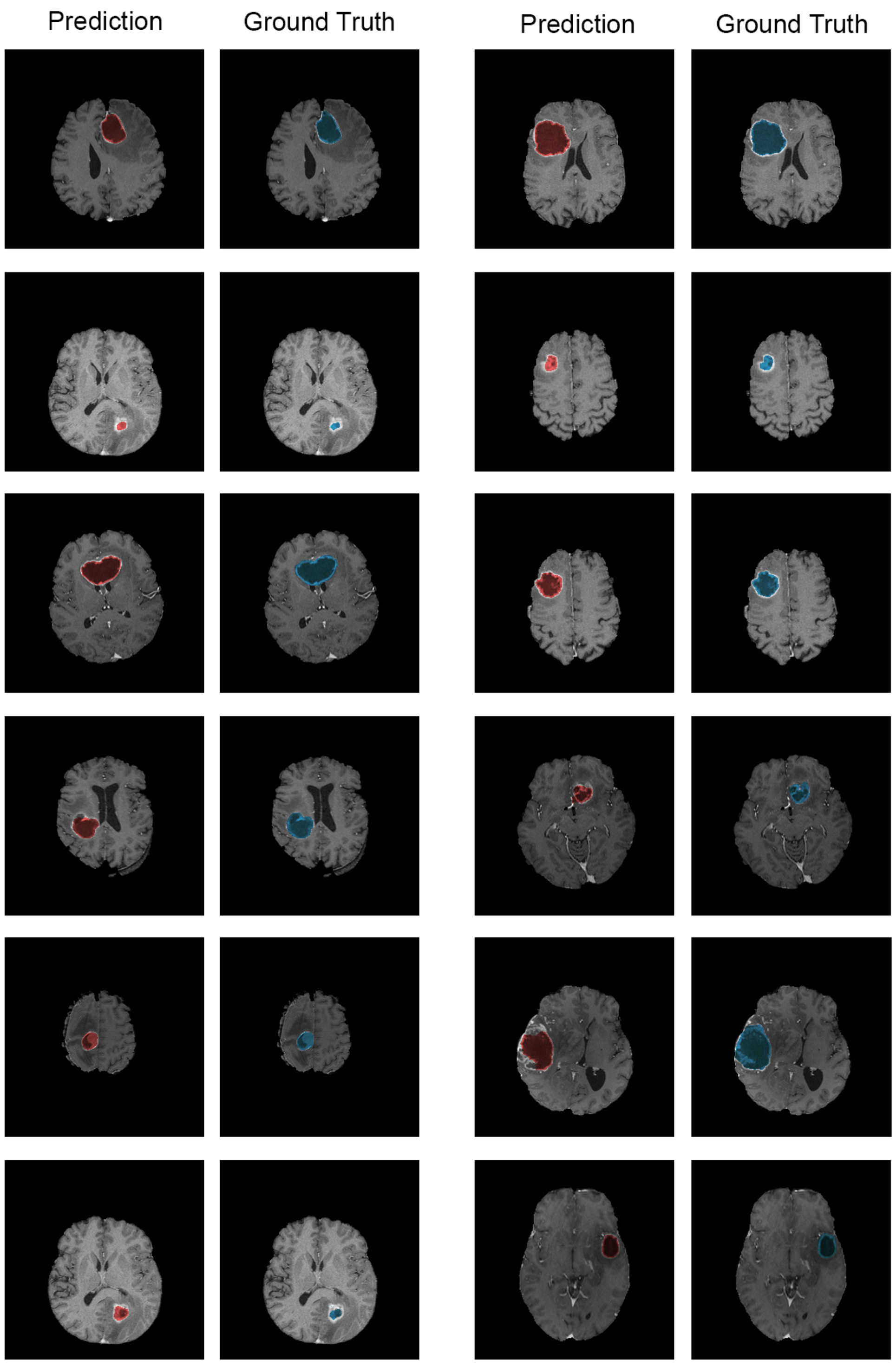


Figure 59: Selection of test images illustrating the predicted segmentation (red) along with the actual ground truth segmentation (blue).

7 Discussion and Future Outlook

In the proposed work, an approach for fully automatic brain tumor segmentation was presented, using a deep convolutional neural network trained with synthetically generated data. In particular, this work focused on the segmentation of glioblastoma multiforme in MRI data. A method for generating brain MR images containing synthetic glioblastoma multiforme and a modified U-Net architecture for automatic segmentation of the lesion were presented. The deep convolutional neural network was trained solely with synthetic MRI data and was subsequently tested on real MR images of patients suffering from glioblastoma multiforme.

7.1 Synthetic Data Generation

Since deep neural networks require a large training dataset, in order to learn a task and generalize from training samples to the entire domain of unseen data, MR images containing synthetic brain tumors along with the underlying ground truth segmentation were generated with the help of MeVisLab and the method presented in section 5.1.

In general, it is much easier to receive and use data of healthy subjects than diseased ones, due to privacy concerns amongst other things. With the proposed method, it was possible to generate realistic-looking brain tumors and insert them into healthy brain MR images. Furthermore, it was possible to simulate a tumor mass effect by using image deformation. The corresponding ground truth was created automatically in a very precise way. Finally, the resulting "hybrid" MR images could be used to train a deep convolutional neural network for automatic tumor segmentation.

In clinical practice of oncology and diagnostic medicine, typically only the enhancing tumor and necrotic core are segmented, since these regions are of primary interest. Delineation of other structures is usually not performed, since the segmentation of the surrounding edema, for example, is extremely challenging and will therefore not represent the truth very well [19]. Due to this reason, the data generation approach was restricted to simulating the enhancing tumor and necrotic core.

One advantage of inserting synthetic brain tumors into MR images is that the position, shape and size of the lesion is already exactly known; hence, it is very easy to automatically create a precise ground truth segmentation. Another advantage of the proposed method is that different artificial tumors can be inserted into the same image, thereby maximizing the size of the resulting dataset. In this work, an easy application for data generation, including one-click seed point selection and saving individual 2D image slices via MeVisLab, was introduced. Generating artificial brain tumors, creating the corresponding ground truth and simulating the tumor mass effect was all done in one step, which made it possible to generate a large synthetic MRI dataset for training a deep neural network.

Related work in the field of generating artificial brain tumors is often focusing on more complex approaches (biomechanical models, reaction-diffusion processes guided by diffusion tensors etc.), which to some extent try to simulate tumor growth on cell level [23],[80],[82]. The aim of the presented tumor generation technique was to empirically generate sufficiently realistic brain MR images containing synthetic glioblastomas and, furthermore, to automatically create the corresponding ground truth segmentation, which could subsequently be used to train a deep neural network. The precise modeling of brain tumor growth on cell level was beyond the scope of this work. The proposed method is simple but nevertheless allows to simulate all crucial characteristics of a glioblastoma multiforme, such as the enhancing tumor and necrotic core, as visible in post-contrast T1-weighted brain MR images.

The synthetic MR images were examined by a neurosurgeon with many years of experience in the diagnosis and treatment of brain tumors and were judged to be very realistic and consistent with the MR images of true glioblastomas, as they are seen in clinical routine. Although the synthetic data was considered to be not distinguishable from real MR data of glioblastomas by an experienced neurosurgeon, blind testing should be conducted in a future study to ensure the quality of the synthetic images.

A further adaptation of the proposed approach could be to simulate the effect of Gadolinium on brain tissue in non-contrast enhanced T1-weighted MR images. Another extension would be to simulate enhancing areas inside the necrotic core, in order to provide a larger range of reasonable tumor appearances. Furthermore, the proposed method may also be applied to other areas of clinical oncology and diagnostic medicine, for example, the simulation of liver tumors [97].

7.2 Brain Tumor Segmentation

The network architecture used for the segmentation task of this thesis was based on the well-known U-Net architecture proposed by O. Ronneberger et al. [71]. The original architecture was slightly modified to provide a better fit for the segmentation problem at hand.

The employed network comprised an initial contracting path (convolutional encoder) and a subsequent expanding path (convolutional decoder). The main modifications included omitting the 2x2 convolution in the expanding path of the network and the additional application of batch normalization and dropout layers after each regular 3x3 convolution.

Another important adaptation is that 'same padding' was used for convolutions instead of 'valid padding', which resulted in a segmentation mask that exhibited the same size as the input image. In contrast, with 'valid padding' (as used in the standard U-Net) the output image would be smaller than the input by a constant border width.

In total, the utilized network architecture consisted of 23 convolutional layers and was designed for gray scale input images of dimension 256x256. In the lowest resolution, feature maps exhibited a size of 16x16, with a maximum number of 256 feature channels.

Prior to the actual training and evaluation process of the neural network, several preprocessing steps were performed on the employed MRI data, including the removal of non-brain tissue, resizing of MR images, contrast enhancement, as well as intensity normalization.

The presented segmentation network was trained with 52000 synthetically generated 2D MR images and tested on 2600 real 2D MR images. Training was performed for 40 epochs, which corresponds to 32480 individual update steps. On average, the model took 10 hours to train with multi-GPU utilization. A combined loss function, comprising the sum of cross-entropy loss and Dice loss, together with the Nadam optimizer was used for the learning process of the network.

Assessing the segmentation results given in section 6.2, one can see that it was possible to achieve a Dice coefficient of 82.449%, a Hausdorff distance of 9.002, a sensitivity of 73.447% and a specificity of 99.987% on the test dataset. These results prove that it is possible to train a deep convolutional neural network for brain tumor segmentation solely on synthetic data.

Analyzing the histograms in figure 58, which illustrate the per image evaluation of the model, it can be seen that the segmentation of most test images resulted in a Dice coefficient between 90% and 100%, a Hausdorff distance between 0 and 10 pixels, a sensitivity between 90% and 100%, and a specificity between 90% and 100%.

However, there is also a not negligible number of images from the test set that resulted in a Dice coefficient between 0% and 10% and a sensitivity between 0% and 10%, which indicates that certain tumor structures could not be identified as such or were only partially detected by the model.

Inspecting the qualitative segmentation results, it became apparent that most of these incorrectly segmented images only contain a very small area of tumor, with low tissue contrast. Furthermore, it could be observed that tumor structures that exhibit a large and diffuse enhancing area inside the necrotic core were also more difficult to segment for the model. Additionally, a couple of false positives were produced for brains with very large lateral ventricles, which also reflects in a higher Hausdorff distance. The very high specificity indicates that the model is very good at segmenting background. However, due to the class imbalance of foreground and background in the employed MR images, this metric is naturally quite high.

The representative selection of qualitative segmentation results of test images, shown in figure 59, illustrates that for MR images of tumors with good soft tissue contrast, a defined enhancing margin, and a nearly homogeneous necrotic core, the predicted segmentation result is quite precise. For some individual cases, the prediction produced by the model seems to follow the outline of the tumor even better than the manually obtained ground truth segmentation. In contrast, however, tumor structures that demonstrate a more diffuse enhancing margin and an inhomogeneous necrotic core, seem to be more difficult to segment for the model and were therefore sometimes only partially detected.

The results obtained in this work are not directly comparable to the results of the MICCAI BraTS challenge, where both high grade and low grade glioma are segmented in multimodal MRI data and segmentation of the whole tumor (non-enhancing tumor, enhancing tumor, necrotic core, peritumoral edema), tumor core (non-enhancing tumor, enhancing tumor, necrotic core) and the enhancing tumor alone is required. In contrast, the aim of this thesis was to segment the enhancing tumor together with the necrotic core in post-contrast T1-weighted MR images. The segmentation task at hand is probably most relatable to segmenting the tumor core in the MICCAI BraTS challenge.

For this thesis, the segmentation network was trained and tested using 2D images, since clinical experts are typically only interested in single MRI slices and not in 3D volumes. Furthermore, training a 3D network architecture requires even more data and is always associated with more computational effort. However, future work could still include a modification of the presented network to enable 3D training with a focus on efficient processing of the image volumes, since this approach could further improve the results by exploiting information of the whole volume content. Moreover, segmentation predictions of different network architectures could be combined for a more robust performance.

Another interesting extension of the proposed approach would be to additionally perform unsupervised domain adaptation with adversarial networks, as proposed by K. Kamnitsas et al. [98]. This may result in a segmentation method that is more invariant to differences between synthetic training and real test data. In particular, domain-invariant features could potentially be obtained by learning to counter a separate adversarial network, which aims to classify the domain of the given input data by observing the activations of the actual segmentation network.

List of Figures

References

[1] S. Bauer et al.; *A survey of MRI-based medical image analysis for brain tumor studies.* Physics in medicine and biology 58(13), 97-129, 2013. 1, 9

[2] A. Işın et al.; *Review of MRI-based Brain Tumor Image Segmentation Using Deep Learning Methods.* Procedia Computer Science, 102, 317-324, 2016. 1

[3] B.H. Menze et al.; *The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS).* IEEE Transactions on Medical Imaging 34(10), 1993-2024, 2015. 1, 5, 47

[4] D. Shen, G. Wu, H-I. Suk; *Deep Learning in Medical Image Analysis.* Annual review of biomedical engineering 19:221-248, 2017. 1

[5] G.P. Mazzara et al.; *Brain tumor target volume determination for radiation treatment planning through automated mri segmentation.* International Journal of Radiation Oncology* Biology* Physics 59(1), 300312, 2004. 1

[6] K. Kamnitsas et al.; *DeepMedic for Brain Tumor Segmentation.*. Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries, pp.138-149, 2016. 1

[7] I. Bankman; *Handbook of medical image processing and analysis.* Academic Press, 2008. 1

[8] S. Standring, H. Gray; *Gray's Anatomy: The Anatomical Basis of Clinical Practice.* Edinburgh: Churchill Livingstone/Elsevier, 2008. 3, 4

[9] B. Forstmann et al.; *An Introduction to Human Brain Anatomy.* An Introduction to Model-Based Cognitive Neuroscience, 71-99, 2015. 3, 4

[10] Australian Academy of Science; *Getting our head around the brain.* https://www.science.org.au/curious/people-medicine/brain. Retrieved: July 6, 2018. 3, 4, 97

[11] S. Cha; *Update on Brain Tumor Imaging: From Anatomy to Physiology.* An Introduction to American Journal of Neuroradiology, 27/3, 475-487, 2006. 4

[12] R. Daneman, A. Prat; *The Blood-Brain Barrier.* Perspectives in Biology 7.1, 2015. 4

[13] P. Argente-Arizn et al.; *Glial cells and energy balance.* J Mol Endocrinol, 58, p59-71, 2017. 5

[14] D.N. Louis et al.; *WHO Classification of Tumours of the Central Nervous System.*Lyon: International Agency for Research on Cancer (IARC) Press; 2007. 5

[15] L. McKinsey et al.; *Genetics of adult glioma.* Cancer Genetics, Volume 205/12, Pages 613-621, 2012, 5, 6

[16] J. Egger et al.; *GBM Volumetry Using the 3D Slicer Medical Image Computing Platform.* Scientific Reports 3, 2013. 5, 6

[17] J.H. Rees et al.; *Glioblastoma multiforme: radiologic-pathologic correlation.* Radiographics, 1413-38, 1996. 5, 6, 7

[18] H. Ohgaki et al.; *The definition of primary and secondary glioblastoma.* Clin Cancer Res., 19(4), 764-772, 2013. 6

[19] L. Dirven et al.; *Health-related quality of life in high-grade glioma patients.* Chin J Cancer, 33(1): 4045, 2014. 6

[20] M. Thurston, F. Gaillard et al.; *Glioblastoma.* https://radiopaedia.org/articles/glioblastoma. Retrieved: April 15, 2018. 7

[21] J.G. Smirniotopoulos et al.; *Patterns of Contrast Enhancement in the Brain and Meninges.* RadioGraphics, 27/2, 525-551, 2007. 7, 97

[22] T. Hines; *Mathematically Modeling the Mass-Effect of invasive Brain Tumors.* Preprint Arizona State University, 2010. 8

[23] M. Prastawa, E. Bullitt, G. Gerig; *Synthetic Ground Truth for Validation of Brain Tumor MRI Segmentation.* Medical Image Computing and Computer-Assisted Intervention, 2008. 8, 44, 50, 92

[24] J.Egger, *GBM Datasets.* https://www.researchgate.net/publication/319188348 _GBM_Datasets. Retrieved: March 20, 2018. 8, 97

[25] D. Leung et al.; *Role of MRI in Primary Brain Tumor Evaluation.* Journal of the National Comprehensive Cancer Network 12/11, 1561-1568, 2014. 9

[26] R. Brown et al.; *Magnetic Resonance Imaging: Physical Principles and Sequence Design.* 2014. 9, 10, 11, 12, 13, 14

[27] M. T. Vlaardingerbroek, J. A. den Boer; *Magnetic Resonance Imaging. Theory and Practice.* Springer Verlag: Germany, 1996. 9, 10, 11

[28] D. McRobbie et al.; *MRI from Picture to Proton.* Cambridge: Cambridge University Press, 2009. 9, 10, 13, 14

[29] M. Jacobs; *MR imaging: brief overview and emerging applications.* RadioGraphics 27, 121330, 2017. 10, 11

[30] A. Einstein; *Zur Quantentheorie der Strahlung.* Physikalische Zeitschrift, Vol. 18, 1917. 11

[31] M. Rogosnitzky, S. Branch; *Gadolinium-based contrast agent toxicity: a review of known and proposed mechanisms.* Biometals, 29, 365376, 2016. 14

[32] H.S. Thomsen, S.K. Morcos, P. Dawson; *Is there a causal relation between the administration of gadolinium based contrast media and the development of nephrogenic systemic fibrosis (NSF)?.* Clin Radiol. 61/11, 905-6, 2006. 14

[33] S.K. Zhou, H. Greenspan, D. Shen; *Deep Learning for Medical Image Analysis.* Academic Press, 2017. 15, 16, 17, 22, 23, 26, 27, 29, 30, 31, 32, 97

[34] J. Zou, Y. Han, S. So; *Overview of artificial neural networks, in Artificial Neural Networks (Methods in Molecular Biology).* vol. 458, D. J. Livingstone, Ed. Totowa, NJ, USA: Humana Press, 2009. 15, 17, 21

[35] J.M. Zurada; *Introduction to Artificial Neural System.* West Publishing Company, St. Paul, MN, 1999. 15

[36] M. Lai; *Deep Learning for Medical Image Segmentation.* arXiv preprint arXiv:1505.02000, 2015. 15, 19, 23, 24, 25, 27, 29, 31

[37] S. Agatonovic-Kustrin, R. Beresford; *Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research.* Journal of Pharmaceutical and Biomedical Analysis, 22/5, 717-727, 2000. 15

[38] I. Basheer, M.N. Hajmeer; *Artificial Neural Networks: Fundamentals, Computing, Design, and Application.* Journal of microbiological methods, 43, 3-31, 2001. 15

[39] A. Krenker, A. Kos, J. Beŝter; *Introduction to the artificial neural networks.* INTECH Open Access Publisher, 2011. 15, 16, 27

[40] K. Willems; *Keras Tutorial: Deep Learning in Python.* https://www.datacamp.com/community/tutorials/deep-learning-python, Retrieved: July 17, 2018. 15, 97

[41] I. Goodfellow, Y. Bengio, A. Courville; *Deep Learning.* MIT Press, http://www.deeplearningbook.org, 2016. 16, 21, 30

[42] F. Rosenblatt; *The Perceptron–a perceiving and recognizing automaton.* Report 85-460-1, Cornell Aeronautical Laboratory, 1957. 16

[43] W. McCulloch, W. Pitts; *A logical calculus of the ideas immanent in nervous activity.* In: Bulletin of Mathematical Biophysics, 5, 115-133, 1943. 16

[44] D. Hebb; *The Organization of Behavior. A Neuropsychiological Theory* John Wiley and Sons Inc., 1949. 16

[45] A. Zilouchian; *Fundamentals of neural networks. In: Jamshidi M, editor. Intelligent control systems using soft computing methodologies.* CRC Press, 2001. 16, 17, 18, 25, 97

[46] K. Willems, N. Stinchcombe, H. White; *Multilayer feedforward networks are universal approximators.* Neural networks, 2(5), 359366, 1989. 17, 19

[47] J. Schmidhuber; *Deep Learning in Neural Networks: An Overview.* Neural Networks, Volume 61, Pages 85-117, 2015. 19

[48] V. Nair, G.E. Hinton; *Learning while searching in constraint-satisfaction problems.* ICML'10 University of California, Computer Science Department, Cognitive Systems Laboratory, 1986. 19

[49] J. Schmidhuber; *Deep Learning.* Scholarpedia, 10(11), 32832, 2015. 19

[50] I. Aizenberg et al.; *Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications.* Neurons: Theory, Learning and Applications, 2000. 19

[51] M. Leshno; *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function.* Neural networks, 6(6), 861867, 1993. 19

[52] G. Schwarz; *Estimating the dimension of a model.* The annals of statistics, 6(2), 461464, 1978. 19

[53] J. Hochreiter; *Untersuchungen zu dynamischen neuronalen Netzen.* Masters thesis, Institut für Informatik, Technische Universitat München, 1991. 19

[54] X. Glorot et al.; *Deep sparse rectifier networks.* Proceedings of the 14th International Conference on Artificial Intelligence and Statistics, 15, 315323, 2011. 19

[55] Y. Bengio, A. Courvill, P. Vincent; *Representation Learning: A Review and New Perspectives.* IEEE Trans. PAMI, special issue Learning Deep Architectures. 35, 17981828, 2013. 20

[56] Y. LeCun, Y. Bengio, G. Hinton; *Deep learning.* Nature 521, 436444, 2015. 20, 22, 23, 27, 30, 31

[57] C.M. Bishop; *Pattern Recognition and Machine Learning.* Springer, 2006. 21, 22, 23, 27, 28, 29, 32, 97

[58] B. Kayalıbay, G. Jensen, P. van der Smagt; *CNN-based Segmentation of Medical Imaging Data.* arXiv:1701.03056v2, 2017. 21

[59] S. Albelwi, A. Mahmood; *A Framework for Designing the Architectures of Deep Convolutional Neural Networks.* Entropy, 19(6), 2017. 21, 97

[60] X. LeCun, Y. Bengio; *Convolutional networks for images, speech, and time series.* The handbook of brain theory and neural net- works, 3361(10), 1995. 22

[61] A. Krizhevsky, I. Sutskever, G. E Hinton; *Imagenet classification with deep convolutional neural networks.* Advances in Neural Information Processing Systems. 1, 10971105, 2012. 22

[62] V. Nair, G.E. Hinton; *Rectified Linear Units Improve Restricted Boltzmann Machines.* ICML'10 Proceedings of the 27th International Conference on International Conference on Machine Learning, 807-8014, 2010. 26

[63] Y. Bengio, O. Delalleau; *On the expressive power of deep architectures.* In ALT2011, 2011. 29

[64] Y. Bengio; *Practical recommendations for gradient-based training of deep architectures.* eprint arXiv:1206.5533, 2012. 29

[65] N. Srivastava et al.; *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research, 15, 1929-1958, 2014. 31, 32

[66] A. Elnakib et al.; *Medical Image Segmentation: A Brief Survey.* Multi Modality State-of-the-Art Medical Image Segmentation and Registration Methodologies: Volume II, Springer New York, 1-39, 2011. 33

[67] I. Despotović, B. Goossens, W. Philips; *MRI Segmentation of the Human Brain: Challenges, Methods, and Applications.* Computational and Mathematical Methods in Medicine, 2015. 33, 34, 38

[68] N. Srivastava et al.; *Computer Vision.* Prentice Hall PTR, 2001. 33

[69] A. Norouzi et al.; *Medical Image Segmentation Methods, Algorithms, and Applications.* IETE Technical Review, 31(3), 2014. 33

[70] H.R. Roth et al.; *Deep learning and its application to medical image segmentation.* Medical Imaging Technology, Volume 36(2), 63-71, 2018. 36, 97

[71] O. Ronneberger et al.; *U-Net: Convolutional Networks for Biomedical Image Segmentation.* Medical Image Computing and Computer-Assisted Intervention, 2015. 36, 45, 46, 47, 48, 73, 93, 98

[72] J. Long, E. Shelhamer, T. Darrell; *Fully convolutional networks for semantic segmentation.* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3431- 3440, 2015. 36, 45, 48

[73] A.A. Taha, A. Hanbury; *Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool.* BMC Medical Imaging, 2015. 38

[74] A. Fenster, B. Chiu; *Evaluation of Segmentation algorithms for Medical Imaging.* Conf Proc IEEE Eng Med Biol Soc., 7, 7186-9, 2005. 39

[75] Unknown Author; *MeVisLab Reference Manual.* http://mevislabdownloads.mevis.de/ docs/2.5/MeVisLab/Resources/ Documentation/Publish/SDK/MeVisLabManual/, Retrieved: April 25, 2018. 40

[76] Unknown Author; *Getting started tutorial.* https://mevislabdownloads.mevis.de/docs/current/MeVisLab/ Resources/Documentation/Publish/SDK/GettingStarted/index.html, Retrieved: April 25, 2018. 40

[77] M. Abadi, et al.; *TensorFlow: Large-scale machine learning on heterogeneous systems.*, 2015. Software available from tensorflow.org 42

[78] Unknown Author; *TensorFlow developement guide.* https://www.tensorflow.org/versions/r1.1/get_started/get_started. Retrieved: July 21, 2018. 42

[79] Unknown Author; *Keras: The Python Deep Learning library.* Keras Documentation. https://keras.io/. Retrieved: July 21, 2018. 43

[80] M. Prastawa, E. Bullittb, G. Geriga; *Simulation of Brain Tumors in MR Images for Evaluation of Segmentation Efficacy.* Medical Image Analysis, 2009. 44, 92

[81] J. Rexilius, et al.; *A Framework for the Generation of Realistic Brain Tumor Phantoms and Applications.* In Medical Image Computing and Computer-Assisted Intervention, Springer Berlin Heidelberg, 2004. 44

[82] O. Clatz et al.; *Realistic Simulation of the 3D Growth of Brain Tumors in MR Images Coupling Diffusion with Biomechanical Deformation.* IEEE Trans Med Imaging, 2005. 45, 92

[83] Unknown Author; *Multimodal Brain Tumor Segmentation Challenge 2017.* https://www.med.upenn.edu/sbia/brats2017.html Retrieved: March 25, 2018. 47

[84] K. Kamnitsas et al.; *Ensembles of Multiple Models and Architectures for Robust Brain Tumour Segmentation.* in Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries, 450-462, Springer International Publishing, 2018. 48

[85] K. Kamnitsas et al.; *Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation.* Med. Image Analysis, 36, 6178, 2017. 48

[86] L. Lindner; *Artificial Brain Tumor Data Generation for Deep Learning Based Segmentation Approaches.* Project Report, 2017. 49

[87] L. Lindner et al.; *TuMore: generation of synthetic brain tumor MRI data for deep learning based segmentation approaches.* Proceedings Volume 10579, Medical Imaging 2018: Imaging Informatics for Healthcare, Research, and Applications; 105791C, 2018. 49

[88] E. W. Weisstein; *Icosahedron.* http://mathworld.wolfram.com/ Icosahedron.html. Retrieved: November 15, 2017 51

[89] J. Egger, Z. Mostarkic, S. Grosskopf, B. Freisleben; *A Fast Vessel Centerline Extraction Algorithm for Catheter Simulation.* Twentieth IEEE International Symposium on Computer-Based Medical Systems, 2007. 51, 98

[90] *MeVisLab Documentation.* https://www.mevislab.de/developer/ documentation/, Retrieved: March 17, 2017. 54, 56

[91] B. Pfarrkirchner et al.; *Lower jawbone data generation for deep learning tools under MeVisLab.* Conference: SPIE Medical Imaging, 2018. 65

[92] D.W. Shattuck, R.M. Leahy;*BrainSuite: an automated cortical surface identification tool.* Med Image Analysis, 6(2), 129-42, 2002. 69

[93] G. Kim; *Brain Tumor Segmentation Using Deep Fully Convolutional Neural Networks.* in Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries, Springer International Publishing, 344-357, 2018. 74

[94] F. Milletari et al.; *V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation.* Neural Networks for Volumetric Medical Image Segmentation, 565-571, 2016. 77

[95] S. Ruder et al.; *An overview of gradient descent optimization algorithms.* 2016 78

[96] G. Csurka et al.; *What is a good evaluation measure for semantic segmentation?* in BMVC, volume 27, page 2013. Citeseer, 2013. 84

[97] A. Hann et al.; *Algorithm guided outlining of 105 pancreatic cancer liver metastases in Ultrasound.* Scientific Reports 7, 2017. 92

[98] K. Kamnitsas et al.; *Unsupervised Domain Adaptation in Brain Lesion Segmentation with Adversarial Networks.* in Information Processing in Medical Imaging, Springer International Publishing, 597-609, 2017. 95