TECHNISCHE UNIVERSITÄT WIEN

Diplomarbeit

Deep Learning for Fetal Brain Segmentation in MRI

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Mathematik

eingereicht von

Paul Weiser
Matrikelnummer: 01326310

ausgeführt im Computational Imaging Research Lab (Med. Uni.)
Eingereicht an der Fakultät für Mathematik und Geoinformation der
Technischen Universität Wien

Betreuung
Betreuer: Assoc.Prof. Dipl.-Ing. Dr. Georg Langs
Mitwirkung: Dipl.-Ing Ernst Schwartz

(Unterschrift Verfasser)

(Unterschrift Betreuer)

Wien, 20.10.2020

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Wien, 20.10.2020

Paul Weiser

Acknowledgments

I would never have reached this point in my life if I had not been supported by a large number of people. At this point I would like to express my gratitude. Ernst Schwartz and Georg Langs from the Computational Imaging Research Lab (CIR) at the Medical University of Vienna have guided me through this thesis. Without them this would not have been possible. I would also like to thank the entire team, which has warmly welcomed and supported me in many ways.
Special thanks goes to my parents. They have accompanied me through my whole life and have had an impact on me like nobody else. Without you, I would never have come this far, thank you!

Abstract

Medical research is a very diverse field and recently machine learning has become a part of it. One area in which it can be applied particularly well is the analysis of medical image data. The segmentation of this data is a difficult task in which a lot of research is done. In this thesis, a method for the automatic segmentation of magnetic resonance images (MRI) of fetal brains is presented. In comparison, the automatic segmentation of the adult brain is already well advanced and there are several interesting results. In this case, the quantity and quality of the data as well as the complex structure of the fetal brain are a major challenge for any automatic segmentation program.
A popular method for segmentation is deep learning. In this process, artificial neural networks are trained on pre-segmented data. With the experience gained further data can be evaluated independently. Convolution and the U-Net architecture are used to particularly improve the quality of the image analysis of neural networks.
In the course of this thesis, experiments are carried out to find a suitable structure for a neural network. Inspired by others, several techniques such as sequencing neural networks or hierarchical structures are evaluated and implemented. To improve the spatial information of the artificial neural network, spectral coordinates are applied and a topological loss function supports the identification of the cortex.
This techniques improve the automatic segmentation and lead to promising results. Especially the spectral coordinates and the topological loss function increase the performance of the network in the cortex.

Contents

Chapter 1

Introduction

Medical research has greatly benefited from the use of machine learning [40]. Many successes have been achieved. This thesis presents an approach for segmenting magnetic resonance images (MRI) of fetal brains. The following chapter introduces the challenges, which are waiting ahead, and presents the tasks that shall be fulfilled.

Although ultrasound is the dominant instrument for diagnosing fetuses, an MRI is often used for further examination [61]. MRIs offers numerous advantages, for example, it is not dependent on the position of the fetus or obesity. In addition, the use of different sequences and modalities allows varying areas to be focused on. They can be used to detect various pathologies, such as anomalies of the central nervous system, disorders of cortical formations and ventriculomegaly. In total, fetal MRI is an useful complementary instrument for the detection of abnormalities and the development of diagnoses [3].

To begin with, the challenges of fetal brain segmentation are discussed and what distinguishes it from adult brains. This shall serve as motivation for the further work. Then it is explained what the main goals of this work are. Additional data shall be generated and an approach for segmentation is to be created. Further, insight into how the objectives will be achieved is provided. The main contribution is the implementation of an artificial neural network tailored to the task of fetal brain segmentation. At last, a brief description of the structure is given by shortly summarizing all chapters.

1.1 Problem Statement

In recent years, automatic segmentation of MR brain images has improved significantly [18]. The reason for this is the successful application of deep learning in medical imaging [40]. Accurate methods for automatic segmentation of fetal brains are still being developed [41]. There are two reasons for this: First, the availability of data is much more limited compared to adult brain MRIs or non-medical data. The reason for this is the lower number of fetal than adult patients, and fetal movement can corrupt the data. Furthermore, medical data is difficult to acquire due to data protection reasons. To successfully apply deep learning algorithms a large amount of data is necessary [23]. Second, the variability of shapes in the brains of unborn children is far greater than of adults. Some regions are still developing or not clearly distinguishable. This makes a proper segmentation far more challenging.

1.2 Aim of the Work

The goal of this thesis is to implement automatic segmentation of fetal brain in-utero MRIs. Therefore, several deep learning models are tested and their performance is evaluated. Different architectures are implemented and their segmentation is compared. Furthermore, spectral coordinates are applied. They can be used to encode anatomical location of image elements. Their influence on the performance of the network will be investigated and evaluated.

1.3 Methodological Approach

To increase the number of manually segmented MRIs an atlas based software is used to pre-segment the images. These are then manually enhanced and used to train the neural network.
The basis of the algorithm is a convolutional artificial neural network in an U-Net architecture. Further techniques are implemented to improve its performance: several models are processed to a Sequence, so that each can improve the errors of the previous one. This improve the spatial information and leads to a more accurate segmentation. The model has an hierarchical structure, this allows to split up a problem in several sub-tasks, which are

easier to solve. In doing so, the fetal brain is first identified and then segmented. Spectral coordinates improve the structural understanding of the graph, and a topological loss function can respond to the specific characteristics of individual regions.
These techniques will be compared in experiments. The segmentation of each model is evaluated by various metrics, such as dice coefficient, mean surface distance or relative area of touch. The results will lead the way to a final neural network for automatic fetal brain segmentation.

1.4 Thesis Outline

This thesis is structured in the following way:

Chapter 1: Introduction. In this chapter the problem of fetal brain segmentation is described and the work of this thesis is motivated.

Chapter 2: Artificial Neural Networks. An overview of the theory of neural networks is given. First, the concepts are motivated and more simple and general topics are presented. Then, advanced structures are discussed.

Chapter 3: Spectral Coordinates. An introduction to the theory of spectral coordinates and diffusion maps is given. It is discussed what they reveal about the deeper structure of a graph and how they are linked.

Chapter 4: State-of-the-Art. In this chapter various publications are presented, which deal with automatic segmentation of fetal brains. When discussing them, special focus is put on the different deep learning techniques that are applied. Many of them, such as sequencing neural networks or spectral coordinates, are used later in this work.

Chapter 5: Methodology. The final architecture, which is used for the automatic segmentation of fetal brains, is presented in this chapter. The detailed structure of the neural network, the application of spectral coordinates and the topological loss function are discussed.

Chapter 6: Experiments and Results. In the course of this thesis different approaches and ideas are explored. The corresponding experiments and results are evaluated in this chapter. At the end, the final architecture is also tested extensively.

Chapter 7: Conclusion. In the final chapter of this thesis, a summary is given and ideas for future work are presented.

Chapter 2

Artificial Neural Networks

In recent years, artificial neural networks became strongly popular in a wide field of applications [30] [1]. Nowadays, they are also used in medical fields [40]. In this work an artificial neural network is applied to segment fetal brains from in-utero MRI acquisitions into distinct tissue types. Therefore, its theory is discussed in the following chapter, by reviewing existing work.
An artificial neural network is an algorithmic implementation of a mathematical model of the functioning of animal nervous systems, consisting of large numbers of interconnected brain cells called neurons. Each neuron receives a signal, processes and forwards it to further cells. Mathematically, an artificial neural network is a piece-wise linear function, which is used to approximate an unknown, hidden function. Non-linearities are achieved by using activation functions. The intuition and the basic structure is discussed in section 2.1. Activation functions are introduced in section 2.2.
In order to quantify the predictions, which are produced by the neural network, loss functions are used. A variety of these methods is presented in section 2.3. A network is trained by using stochastic gradient descent to minimize the loss function. This is done by computing the derivative (section 2.5) of the network and adapting its weights accordingly. There are several ways in which optimization is done. These are presented in section 2.4.
At the end of the chapter, more advanced techniques are discussed. Convolutional neural networks are typically used to process images. They are designed so that each layer recognizes more complex structures in an image. U-Nets are another way to design neural networks. The idea is to process the input and then restore it to achieve a segmentation of the original im-

age. These architectures are presented in sections 2.6 and 2.7. Recently, neural networks have become very deep. Therefore, training them can be quite unstable. Batch normalization is used to adapt the distribution of the data given to a layer. This prevents divergence during training and allows building deeper neural networks. Batch normalization is analyzed in section 2.8.

2.1 Basic Architecture of Artificial Neural Networks

The mathematical model of a artificial neural network is inspired by the structure of a biological neuron [13]. A simple neuron functions as follows: First, information is received as an electric impulse through the dendrites [43]. This stimulus is processed in the cell body. Each impulse has an unique influence on the output, based on which dendrite it came from and how strong it is. The resulting impulse is transmitted from the body through the axon and into its branches. The signal is then passed on to the dendrites on to further neurons. The impulse, that is passed on to other cells, differs depending on which arm of the axon the information is passed through. In the following we review information based on [23] [16] [13]
A basic mathematical model of a neural network is built in a similar way. First, the model receives information in form of a vector $\mathbf{x} \in \mathbb{R}^n$. The input is processed by multiplying it with another vector $\mathbf{w} \in \mathbb{R}^n$. A single scalar $b \in \mathbb{R}$ is added, and an activation function f is applied to the result. The output, which is given as $f(\mathbf{x} \cdot \mathbf{w} + b)$, is then passed on to the next mathematical neuron. In the case of the biological cell, each branch of the axon can transmit a different signal. In the mathematical model this can be realized by using several weight vectors $\mathbf{w_i} \in \mathbb{R}^n, i = 1, ...m$ and a bias vector $\mathbf{b} \in \mathbb{R}^n$. The result is then given by

$$y_i = f(\mathbf{w_i}\mathbf{x} + b_i), \ i = 1, ...m$$

The simplest neural network is the single-layer perceptron. Geometrically, it separates data points with a hyperplane and classifies new ones depending on which side of the plane they lie. The mathematical function which is defined by the perceptron algorithm is given as

$$y = \begin{cases} 1 & \text{if } \mathbf{x} \cdot \mathbf{w} + b > 0, \\ 0 & \text{else} \end{cases}$$

The hyperplane, which separates the data, is uniquely defined by the weights $\mathbf{w}$ and b. The normal vector of the plane is given by the weights. The bias b shifts the plane in the direction of $\mathbf{w}$.
The perceptron is a beautiful example but unfortunately very limited. It struggles with relatively simple problems, e.g., the exclusive-or problem. Mathematically, it is defined as a simple function from $\{0,1\}^2 \rightarrow \{0,1\}$. It's values are shown in table 2.1.

Input		Output
1	1	0
1	0	1
0	1	1
0	0	0

Table 2.1: Truth table for the exclusive-or problem

This is a problem that the peceptron algorithm is not able to solve. The four points $\{(0,0),(0,1),(1,0),(1,1)\}$ with corresponding labels $(0,1,1,0)$ cannot be separated by a straight line. Each straight line which divides them has always two differently labeled points on each side. Therefore, this problem is not solvable by using the perceptron algorithm.
Since the model already fails handling simple problems, the algorithm has to be modified to handle more complex tasks. The number of layers which are used in the neural network as to be adapted. It's number has to increase. The algorithm presented above is an example for a network with just one layer. Neural networks are usually structured in layers. The first layer is called the input layer and the last output layer. The ones in between are named hidden layers.
The fully connected or dense layer is the most common. It is structured in nodes, each is defined as the above mentioned neuron. All mathematical neurons of the previous layer are connected to each node of the current dense layer. The input of a single node is, therefore, a vector of the size of the previous layer.

Another issue to address are activation functions f. A wide range of methods is applied. They will be discussed later in more detail. For the time being, only the rectified linear unit (ReLU) is introduced. It is a simple and very common activation function:

$$f(x) = \max(0, x)$$

The major advantage of activation functions is that they allow the model to approximate non linear functions. This is an absolute necessity to handle more complex problems. Already a simple neural network with a single hidden layer containing only 3 nodes can solve the exclusive-or problem. The non-linear activation functions in the hidden layer allow the model to adapt to the situation. The model is given in [21].
Since neural networks with a single hidden layer are already a lot stronger than the classic perceptron algorithm, naturally, the following question arises: How strong are these networks really? Cybenko [16] has given a comprehensive answer to this problem. He proved that any continuous function in a compact space can be arbitrarily well approximated by a neural network with only one hidden layer and any continuous sigmoidal activation function.
In order to understand the statement, some expressions have to be clarified. Let I_n be the n-dimensional unit cube $[0,1]^n \subset \mathbb{R}^n$, and $C(I_n)$ be the space of real valued continuous functions on I_n. The supremum norm on $C(I_n)$ is denoted by $\|.\|$. Furthermore, the space of finite, signed and regular borel measures on I_n is given by $M(I_n)$.

Definition 2.1. *σ is called discriminatory if for a measure $\mu \in M(I_n)$*

$$\int_{I_n} \sigma(y^T x + \theta) d\mu(x) = 0$$

for all $y \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ implies that $\mu = 0$

Definition 2.2. *σ is called sigmoidal if*

$$\sigma(x) = \begin{cases} 1 & \text{as } x \to +\inf, \\ 0 & \text{as } x \to -\inf \end{cases}$$

Before the main theorem can be proven, some other results are needed. These won't be proven, but just stated. First, a corollary of the Hahn-Banach theorem is given. It states that if there is a subspace, which is not dense, then there exists a functional that vanishes on the subspace but not on the whole space. The second is the Riesz representation theorem. It will be used to find a measure that can represent the above functional.

Theorem 2.1 (Hahn-Banach [49]). *If $p : X \to \mathbb{R}$ is a sublinear function on a real vector space X, and $f : M \to \mathbb{R}$ is a linear functional on a linear subspace $M \subseteq X$ that is dominated by p on M, then there exists a linear extension $F : X \to \mathbb{R}$ of f to the whole space X that is dominated by p, i.e., there exists a linear functional F so that*

- $F(m) = f(m) \quad \forall m \in M,$
- $|F(x)| \leq p(x) \quad \forall x \in X.$

Corollary 2.1.1 ([49]). *Suppose S is a subspace of a locally convex space X, and $x_0 \in X$. If x_0 is not in the closure of S, then there exists a continuous linear functional L so that $L(x_0) = 1$ but $L(x) = 0$ for every $x \in S$.*

Theorem 2.2 (Riesz representation theorem [5]). *Let L be a continuous linear functional on $C(X)$, X compact and Hausdorff. Then there is a unique measure $\mu \in M(X)$ so that*

$$L(f) = \int_X f d\mu \quad \forall f \in C(X)$$

Now we have the tools to prove the theorem.

Theorem 2.3 ([16]). *Let σ be a continuous discriminatory function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \epsilon \text{ for all } x \in I_n.$$

Proof. Let $S \subset C(I_n)$ be the subset of all functions of the form $G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$. S is a linear subspace of $C(I_n)$. We want to prove that S is dense in $C(I_n)$: $\overline{S} = C(I_n)$.
Therefore, we assume that $\overline{S}$ is a real subset of $C(I_n)$: $\overline{S} \subset C(I_n)$. According to the corollary of the Hahn-Banach theorem, there exists a functional L on $C(I_n)$, such that $L \neq 0$ and $L(S) = 0$.
Due to the Riesz representation theorem, there is a signed measure $\mu \in M(I_n)$, so that

$$L(f) = \int_{I_n} f d\mu(x) \quad \forall f \in C(X).$$

L is zero on S. Since $\sigma(y^T x + \theta) \in S \quad \forall y \in \mathbb{R}^n, \forall \theta \in \mathbb{R}$,

$$\int_{I_n} \sigma(y^T x + \theta) d\mu(x) = 0 \quad \forall y \in \mathbb{R}^n, \forall \theta \in \mathbb{R}.$$

σ is discriminatory. Therefore, $\mu = 0$, contradicting $L \neq 0$. So S has to be dense in $C(I_n)$. □

It is shown that the statement holds for continuous, discriminatory and sigmoidal functions σ. This is quite unsatisfying, because being discriminatory is not a obvious property. Therefore, these functions are further explored, and it will be proven that the above statement is already fulfilled for continuous sigmoidal functions.

Theorem 2.4 ([16])**.** *Any bounded, measurable sigmoidal function, σ, is discriminatory. In particular, any continuous sigmoidal function is discriminatory.*

Proof. For all x, y, θ, ϕ we have

$$\sigma(\lambda(y^T x + \theta) + \phi) = \begin{cases} \to 1 & \text{for } y^T x + \theta > 0 \text{ as } \lambda \to \infty \\ \to 0 & \text{for } y^T x + \theta < 0 \text{ as } \lambda \to \infty \\ = \sigma(\phi) & \text{for } y^T x + \theta = 0 \text{ for all } \lambda \end{cases}$$

Therefore, the function $\sigma_\lambda(x) = \sigma(\lambda(y^T x + \theta) + \phi)$ converges pointwise and boundedly to

$$\gamma(x) = \begin{cases} = 1 & \text{for } y^T x + \theta > 0 \\ = 0 & \text{for } y^T x + \theta < 0 \\ = \sigma(\phi) & \text{for } y^T x + \theta = 0 \end{cases}$$

for $\lambda \to \infty$. Let $\Pi_{\gamma,\theta}$ be the hyperplane $\{x|y^T x + \theta = 0\}$ and $H_{\gamma,\theta}$ be the half-space $\{x|y^T x + \theta > 0\}$. By the Lebesgue bounded convergence theorem, the following holds

$$\begin{aligned} 0 = &\int_{I_n} \sigma_\lambda(x) d\mu(x) \\ = &\int_{I_n} \gamma(x) d\mu(x) \\ = &\sigma(\phi)\mu(\Pi_{\gamma,\theta}) + \mu(H_{\gamma,\theta}) \end{aligned}$$

for all ϕ, θ, y. Therefore, the μ is zero on all half-planes. It has to be shown that this implies that the measure μ itself is zero.
For fixed y and a bounded measurable function h, define the linear function F as

$$F(h) = \int_{I_n} h(y^T x) d\mu(x).$$

F is a bounded functional on $L^\infty(\mathbb{R})$, since μ is a finite singed measure.
If h is a indicator function on $[\theta, \infty)$, then

$$F(h) = \int_{I_n} h(y^T x) d\mu(x) = \mu(\Pi_{\gamma,-\theta}) + \mu(H_{\gamma,-\theta}) = 0.$$

Similarly, it can be shown that $F(h) = 0$ if h is the indicator function of the open interval (θ, ∞). Therefore, $F(h) = 0$ for any interval and hence for all simple functions. Since the set of simple functions is dense in $L^\infty(\mathbb{R})$[5], $F = 0$.
In particular, for $s(u) = \sin(m^T u)$ and $c(u) = \cos(m^T u)$ we get

$$\begin{aligned} F(s + ic) = &\int_{I_n} \cos(m^T x) + i\sin(m^T x) d\mu(x) \\ = &\int_{I_n} \exp(im^T x) d\mu(x) \\ = &0 \end{aligned}$$

for all m. The Fourier transform of μ is 0, and so μ is zero [49]. Therefore, σ is discriminatory. $\square$

Theorem 2.5 ([16])**.** *Let σ be a continuous sigmoidal function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \epsilon \text{ for all } x \in I_n.$$

Proof. This follows directly from theorems 2.3 and 2.4. □

Until now, the function f was always defined to be continuous. In practice, a neural network is often used to approximate a decision function. These are methods that are used for classification. They are defined below. The main statement does also hold for this class of functions.

Definition 2.3. *Let λ denote the Lebesgue measure in I_n. Let $P_1, P_2.....P_k$ be a partition of I_n into k disjoint, measurable subsets of I_n. Define the decision function, f, according to*

$$f(x) = j \text{ if and only if } x \in P_j.$$

Theorem 2.6 ([16])**.** *Let σ be a continuous sigmoidal function. Let f be a decision function for any finite measurable partition of I_n. For any $\epsilon > 0$, there is a finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j)$$

and a set $D \subseteq I_n$, so that $\lambda(D) \geq 1 - \epsilon$ and

$$|G(x) - f(x)| < \epsilon \text{ for all } x \in D.$$

Proof. By Lusin's theorem [50], there is a continuous function h and a set D with $\mu(D) > 1 - \epsilon$, so that $h(x) = f(x)$ for $x \in D$. Since h is continuous and by theorem 2.5, there is a summation G, of the above form, that satisfies $|G(x) - h(x)| < \epsilon \; \forall x \in I_n$. Therefore,

$$|G(x) - f(x)| = |G(x) - h(x)| < \epsilon \; \forall x \in D.$$

□

So far only sigmoidal activation functions have been discussed. The range of activation methods used in machine learning is much wider. There are similar statements proven for different functions [8] [46], but the above proofs give a broad overview of what is theoretically possible with neural networks.

2.2 Activation functions

In this section, the focus is on activation functions. The main purpose of using these methods is the approximation of non-linear and more complex functions. There are several different activation methods, not all of them are covered in detail, but the most important ones are discussed thoroughly. The information reviewed in this section is taken from [23].
Sigmoid σ and the hyperbolic tangent are two classic activation functions. They are defined as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$
$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

These two functions are similar in the sense that their derivative is bell shaped, $\lim_{x\to\inf} f(x) = 1$, and $\lim_{x\to-\inf} f(x) = 0/-1$. The range of tanh is from -1 to 1 and the range of the sigmoid function is between 0 and 1.
Unfortunately, there are some downsides. They suffer from the vanishing gradient problem. The derivative of both functions is ranged between 0 and 1 and it decreases strongly for high absolute values of x. During training the gradient is propagated back through the network. If $|x|$ is large, the derivative of the activation function is small and the gradient of the network vanishes. This leads to a too early convergence of the training process. Another disadvantage is the expensive cost of computing the functions. Thus, the time consumed by training the network increases.
Motivated by these problems, another class of activation methods became popular. This group includes ReLU, which was already defined above, and Parametric ReLU (PReLU [62]). The latter is defined as

$$\text{PReLU}(x) = \begin{cases} x & \text{if } x > 0, \\ ax & \text{else} \end{cases}$$

These methods are less affected by the vanishing gradient problem than the functions discussed above and are also easier to compute. In total, neural networks with ReLUs instead of tanh units train several times faster (Krizhevsky, Sutskever, and Hinton [33]). Another activation function which outperforms ReLUs and PReLUs (Xu et al. [62]) is called Leaky ReLU (LReLU) and given as

$$\mathrm{LReLU}(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{else} \end{cases}$$

The last activation function, which is introduced here, is called scaled exponential linear unit (SELU). It was developed by Klambauer et al. [32] and is defined as follows:

$$\mathrm{SELU}(x) = \lambda \begin{cases} x & \text{if } x > 0, \\ \alpha e^{x} - \alpha & \text{else} \end{cases}$$

2.3 Losses and Metrics

To train a neural network, it is necessary to evaluate it's predictions. This is done via loss functions. In machine learning there are several approaches to assess the quality of a prediction. They vary depending on the problem and the method, which is applied to solve it. There are two main categories in which problems can be separated. The first is called regression. Predicting the price of a house or the age of a person are just two examples. Regression tasks always predict a value in a continuous spectrum. The price of a house is not discrete. It is not either €100,000 or €200,000, it varies continuously in a certain range. The same applies for the age of a person, which is not given in whole numbers, but alters continuously over the years. For the network to improve, it doesn't suffice to label a prediction as correct or incorrect, but it has to state how close the estimation was. Typical losses for regression tasks, which meet this requirement, are the L1 and L2-loss [44].
The L1-loss computes the mean absolute error between the predicted and the true value. It is given by

$$\mathrm{L1} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|,$$

where y_i is the true, $\hat{y}_i$ is the predicted value and n is the number of predictions. The L2-loss calculates the mean of the squared differences of the true and predicted values. This function gives heavier weights to larger errors than to smaller ones. This is due to the fact that the errors are squared. Therefore, the L2-loss is more sensitive regarding outliers. It is defined as

$$\mathrm{L2} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 .$$

These losses got their names from mathematical norms. The L1 loss is defined exactly as the L1 norm, the L2 norm is defined as the square-root of the L2 loss. The reason for this is that in machine learning it is important for derivatives to be simple. In comparison to the norm, the L2 loss is easy to differentiate.
Classification tasks are the second main category. Illustrative problems are the prediction of which product a user is likely to buy, segmenting or classifying an image. In all cases, there is a discrete set of values, which the network can predict. These can be the possible items in a store, the range of labels which can be assigned to a pixel, or the number of object which can be identified in an image. The difference to regression problems is, naively spoken, that there is only one possible correct label and the others are false. It is unnecessary to state how close a prediction is. If a user buys a car instead of the predicted boat, it is mathematically unclear how wrong the prediction is.
A softmax function [45] is often implemented in classification tasks between the last layer of the neural network and the loss function. It is defined as follows:

$$\mathrm{softmax}(\mathbf{y})_i = \frac{e^{y_i}}{\sum_{j=1}^{k} e^{y_j}},$$

The vector $\mathbf{y} = (y_1, ..., y_k)$ corresponds to the prediction of one subject, such as the segmentation of a single pixel or the prediction of which product

someone will buy. It has the form of a vector since each label is represented. The softmax function scales the values of each entry of the vector between 0 and 1, such that $\sum_{i=1}^{k} \sigma(\mathbf{y})_i = 1$. Therefore, the predictions are scaled, have the characteristics of probabilities and are comparable.
A popular loss for classification tasks, which is used in combination with softmax, is the categorical cross entropy [12]. If there are only two labels, it is also called binary cross entropy. It is defined as

$$H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

y_i is the true and $\hat{y}_i$ the predicted value. It is necessary that the predicted probabilities range between 0 and 1. This is due to the fact that the logarithmic function is not defined for negative values and larger numbers than 1 lead to a negative loss. Both cases result in problems during the optimization.
At last, the dice score [54] and the accuracy [2] is introduced. These metrics are rarely used for training but are important for evaluations. The dice score measures the overlap of two images. The dice score is 1, if the images are equal, and 0, if they have no pixels is common.

$$\text{Dice} = 2\frac{|X \cap Y|}{|X| + |Y|}$$

The dice score is often also used as loss function. Neuronal networks are trained by gradient descent optimization. Therefore, they, and also the loss function, must be differentiable. The dice score, as presented above, is not differentiable. In order to make the dice score differentiable, some changes must be made in the implementation. First, the values of the prediction are not saved as Boolean but as continuous number between 0 and 1. Second, the intersection operator is not differentiable. It is approximated by the element-wise product. These adaptions make the dice score differentiable, and therefore allow its use as an error function.
The accuracy is computed by dividing the number of correct predictions by the total number of predictions. As with the dice value, 1 indicates a correct prediction and 0 a poor one.

$$\text{Accuracy} = \frac{\text{Number of correct predicitons}}{\text{Number of total predicitons}}$$

The difference between these scores is that the accuracy does not take the false negative predictions into account.

2.4 Optimization Algorithms

To train a neural network, the error function is $f(\mathbf{x}, \mathbf{w})$ is minimized regarding the weights $\mathbf{w}$ of the network. Since the derivative of a function $f(w)$ indicates the direction of the slope, optimization is done by altering the weights in the negative direction of the derivative.

$$f(w - \epsilon f'(w)) < f(w) \text{ for small } \epsilon > 0$$

Most of the information review here is taken from [14] and [23]. The method of iteratively optimizing a differentiable function in order to find a minimum by following the negative derivative is called gradient descent. The algorithm is given in Alg. 1.

Algorithm 1: Gradient Descent [14]

```
Result: A local minimum x
learning_rate;
precision;
while While precision > dx do
    x += - leanring_rate * dx;
end
```

Once the algorithm has reached a value w so that $f'(w) = 0$, the derivative holds no further information in which direction the value of the function decreases. A point w such that $f'(w) = 0$ is called a critical point. There are three different types: local minima, local maxima and saddle points. A point w is called a local minimum, if there exists a sufficiently small neighborhood U of w such that $f(w)$ is smaller or equal to all values $f(w')\ \forall w' \in U$. Therefore, every infinitesimal movement leads to an increase of the value of the function.

Likewise, a local maximum has exactly the opposite characteristics. Points w, which fulfill $f'(w) = 0$, but are neither a local maximum nor minimum, are called saddle points. Furthermore, a point which has a lower/higher value than any other point is called global minimum/maximum. When training neural networks complicated functions are optimized, these might have a lot of local minima or flat planes with saddle points. This leads to complicated and delicate situations during the optimization and makes the procedure more challenging and difficult.
In machine learning multidimensional functions $f : \mathbb{R}^n \to \mathbb{R}$ are used. Since they receive a vector as input, partial derivatives $\frac{d}{dw_i} f(\mathbf{w})$ are used. These describe the variation of the function in relation to a single coordinate w_i. A point is called critical if $\frac{d}{dw_i} f(\mathbf{w}) = 0$ holds for all coordinates. In order to find the maximal gradient, not just along the coordinates, directional derivatives are used. These are defined as

$$\frac{d}{d\mathbf{v}} f(\mathbf{w}) = \frac{d}{d\mathbf{w}} f(\mathbf{w}) \cdot \mathbf{v}, \quad \|\mathbf{v}\| = 1$$

Here $\mathbf{v}$ is the direction in which the derivative is computed. This definition of the directional derivation allows to search for the greatest descent in every possible direction without explicitly calculating every derivative.
Now an illustrative example which demonstrates the gradient descent algorithm is given. The function used is

$$f(x, y) = \sin(x) + y^2.$$

Its directional derivations are given as

$$\frac{d}{dx} f(x, y) = \cos(x)$$
$$\frac{d}{dy} f(x, y) = 2y$$

The critical points are located at $y = 0, x = 2\pi n - \frac{\pi}{2}$. To demonstrate the algorithm, the function is visualized with a contour plot and the movement given by the gradient descent method is shown with a blue line. The algorithm starts at a random point, in this case $x = 1, y = 4$, computes the partial derivatives and updates the location accordingly. This process is repeated until the position of the point only changes slightly by each update, then the algorithm converges. The visualization is shown in figure 2.1.

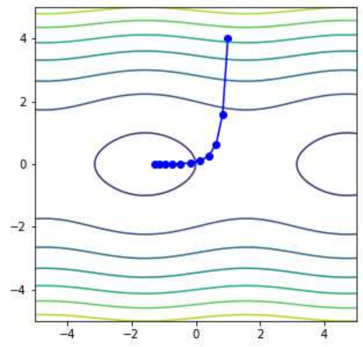

Figure 2.1: Contour plot of the function $f(x, y) = \sin(x) + y^2$. Gradient descent is applied and given by the blue line.

The problem of multiple local minimal and saddle points was already discussed above. This function does not only have several minima, as mentioned above, but also a number of saddle points, which are located at $x = 2\pi n + \frac{\pi}{2}, y = 0$. Here the choice of a clever starting point for the algorithm is essential. If a different starting point is chosen, e.g. $x = 2, y = 4$ instead of $x = 1, y = 4$, then the sequence converges towards another minimum (figure 2.2 left). In this specific case, this will not cause any problems, since both points are global minima and equally reasonable solutions. In many cases, different local minima are problematic, because they hinder the algorithm to find the best possible solution.
A critical situation, which can be observed in the right plot of figure 2.2, is the existence of saddle points. In such a case, the algorithm could not even find a minimum but converge at a plateau. Here the result is a point with value 1 instead of -1 thus far away from a satisfying solution.

A factor that has not yet been discussed is the learning rate. In the previous examples a learning rate of 0.3 was chosen and relatively good results where accomplished. This is not always the case. In the following, the consequences of a too high or too low learning rate are discussed. In figure 2.3 on the left, an example is given in which a higher learning rate of 0.7 is

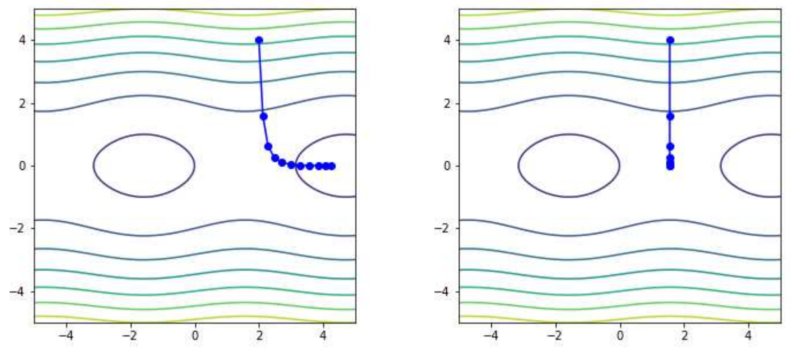

Figure 2.2: Contour plot with gradient descent which does not result in expected points. Left: Converges against different minimum. Right: Converges against saddle point.

chosen. It seems that due to the increased learning rate the step size of the algorithm is too large. Although the algorithm still converges and the local minimum is reached, one has to be careful. If the learning rate is chosen even higher and the steps become too big, the algorithm gives rather poor results and in many cases does not even converge.
In figure 2.3, an example is given in which a too low learning rate is applied. In this case, a rate of 0.1 is chosen. The step size gets smaller the closer it gets to the minimum. Unfortunately, it could not reach a reasonable result and the sequence converged to early.

This example shows that although the idea of gradient descent is easily understandable, its application in practice still presents many challenges. In the discussed example, the function is quite simple, and one can always understand what is happening since it is easy to visualize. Neural networks are a lot more complicated, and one can not understand a problem by just "looking" at it. Visualization might not be possible or sufficient to understand the situation. Therefore, training algorithms are an import part of deep learning. In the remaining part of this section, several learning algorithms will be discussed. They are motivated by the classic gradient descent algorithm but introduce some additional features that improve the training process.
The first modification being discussed is momentum. It is explained in detail by Bengio, Boulanger-Lewandowski, and Pascanu [7] and Sutskever [55]. Momentum is motivated by a simple physical processes. Consider an object

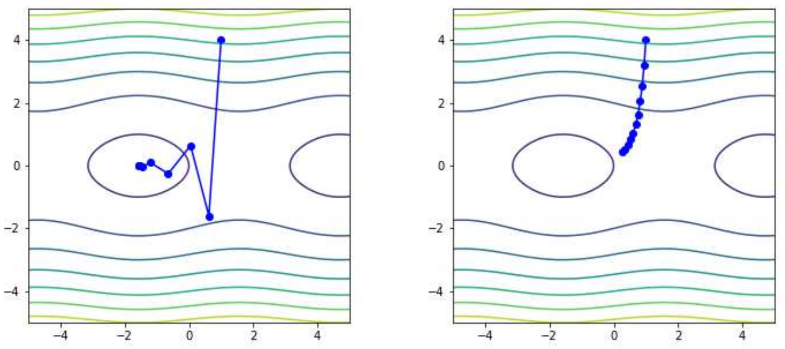
Figure 2.3: Contour plot with gradient descent which does not result in expected points. Left: Learning rate too high. Right: Learning rate too low.

which moves through space with constant speed and without any external force acting on it, then it's momentum is given as the product of it's mass and velocity. To change the speed of the object, a constant force must act on it over a certain period of time. The rate in which the speed of the object changes depends on it's momentum. The speed will adapt slowly if the object has a high momentum, whereas it will change quickly if the momentum is low.
To implement this idea in machine learning, a modification of the gradient descent algorithm is necessary. Instead of the usual update, a velocity vector is introduced. During every update it is multiplied by a constant μ, which ranges between 0 and 1, and added to the gradient. The algorithm then follows the resulting vector to find a local minimum. It is presented in Alg. 2. The idea here is to gradually decrease the influence of earlier gradients by the factor μ, while the current gradient is still taken into account.

Algorithm 2: Momentum [14]

```
Result: A local minimum x
learning_rate;
precision;
while While precision > dx do
    v = $\mu$ * v - learning_rate * dx;
    x += v;
end
```

To illustrate the idea of momentum, the previous example is used again. The algorithm is modified and momentum implemented. The result can be seen in figure 2.4.

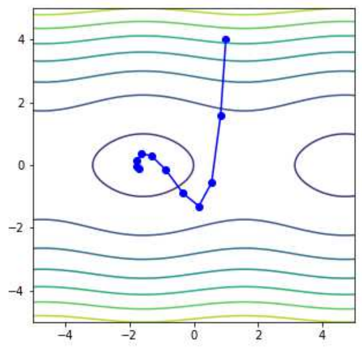

Figure 2.4: Contour plot with gradient descent using momentum.

In this specific case the adjustments have unfortunately not led to an improvement, but the idea is still very plausible. In deep learning the data might be quite noisy. Therefore, these optimization algorithms are normally not applied to smooth functions as has been now. Due to the noise, the gradient may not always return the best or fastest direction towards a minimum. Momentum adapts to this problem by including earlier gradients but weighting them differently. In total, the algorithm is less vulnerable to outliers and leads to better convergence.
The next optimization technique, which will be discussed, is called Rmsprop and was first introduced by Tieleman and Hinton [56]. A earlier, very similar version is called Adagrad and was developed by Duchi, Hazan, and Singer

[20]. Rmsprop is given in algorithm 3.

Algorithm 3: Rmsprop [14]

```
Result: A local minimum x
learning_rate;
precision;
decay_rate;
while While precision > dx do
    cache = decay_rate * cache + (1 - decay_rate) * dx**2 ;
    x += - learning_rate * dx / ($\sqrt{\text{cache}}$ + eps) ;
end
```

The main difference is the variable cache. It is given as the sum over all squared gradients and holds information of the geometry of the data, which was observed in earlier iterations. Furthermore, the variable decays over time. The reason for this is that the relevance of earlier information decreases over time and is, therefore, weighted lower. The variable cache is then used to scale the gradient dynamically. This is a major improvement, because so far the gradient has only been scaled by a constant value. In practice, the algorithm will slow down in geometries with large gradients and accelerate in rather flat areas. The algorithm is, therefore, more stable and resistant to noise.
The algorithm is again tested on the usual example, the result can be seen in figure 2.5

The direct comparison to the algorithms presented before is a little tricky. To achieve optimal results when implementing Rmsprop, the learning rate has to be adapted. The reason for this is that two variables scale the gradient. The learning rate has been increased from 0.3 to 0.7. This is the same value which was used in figure 2.3 as an example for a too high learning rate. When Rmsporp is compared to the original gradient descent algorithm, which was used in 2.1, several changes are immediately noticeable. First, Rmsprop does not make as large steps as gradient descent when the geometry creates large gradients. Second, when moving to flatter areas, the algorithm in 2.1 makes only tiny steps towards the minimum, whereas the step size of Rmsprop stays relatively constant. All in all, the algorithm leads to much more balanced step sizes and is, therefore, more robust against unexpected gradient changes.

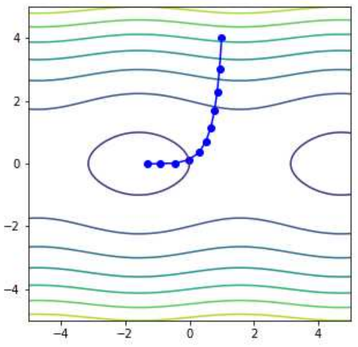

Figure 2.5: Contour plot with the Rmsprop algorithm.

The last optimization method is an algorithm called Adam. It was introduced by Kingma and Ba [31]. A simplified version can be seen in Alg. 4.

Algorithm 4: Adam [14]

```
Result: A local minimum x
learning_rate;
precision;
beta1;
beta2;
while While precision > dx do
    m = beta1*m + (1-beta1)*dx ;
    cache = beta2*cache + (1-beta2)*(dx**2) ;
    x += - learning_rate * m / ($\sqrt{cache}$ + eps) ;
end
```

This algorithm is basically a combination of two previously discussed methods. In Alg. 2 the idea of momentum was introduced. It is implemented in the Adam algorithm by the variable m. Furthermore, Rmsprop was discussed, it used the variable cache to scale the gradient. This is also implemented in the Adam algorithm. The variables beta1 and beta2 are used as decay rates for the scaling factors. In total, Adam chooses the scaling and direction of the algorithm dynamically based on earlier knowledge from the training process. The visualization of the algorithm can be seen in figure 4.

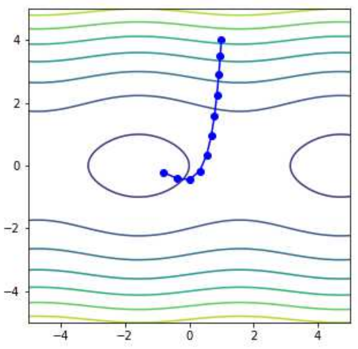

Figure 2.6: Contour plot with the Adam algorithm.

In this example, the learning rate is set to 0.7 and both decay rates beta1, beta2 to 0.5. One can clearly observe the influence of both. The momentum is visible by the fact that the sequence does not move straight towards the minimum but moves in a small arc. The dynamic scaling is noticeable, as with the algorithm above, by a more balanced step size. These two methods combined make the algorithm very robust and easy to apply.

2.5 Back-Propagation

During the training process, data is given to the neural network. It flows through the various hidden layers of the network and finally produces a prediction. This process is called forward propagation. At the end, the error function returns a loss depending on the quality of the prediction. During the computation of the derivative, information is passed backwards through the neural network, this is called back propagation (Rumelhart, Hinton, and Williams [51]). Most of the work discussed in this section is from [23]. To compute the derivative of the network regarding it's weights, the gradient of the loss function is calculated and passed backwards through the network. Mathematically, this is justified by the chain rule. Clearly, the derivative has to be computed regarding the weights of the network, since it represents a function which gets fitted to the data.
To understand back propagation, one must first study the chain rule. Let, therefore, $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be differentiable functions, then the

following holds:

$$(f \circ g)'(x) = f'(g(x))g'(x)$$

The derivative of the concatenation of f and g at x is the product of the derivative of f at $g(x)$ and the derivative of g at x. In practice, often higher dimensional functions, such as $f : \mathbb{R}^n \to \mathbb{R}^k$ and $g : \mathbb{R}^m \to \mathbb{R}^n$, are used. Then the derivatives are not given as scalars but as the Jacobian matrix. The formula above still holds, but the scalar multiplication is replaced by a matrix or vector product.
The chain rule is here very useful, because it reduces the computational workload. To obtain the derivative of a complex function, due to the chain rule, it is sufficient to decompose it in simpler functions and compute their gradient. The derivative of the complicated function is then given by the chain rule. To improve the understanding on this issue, as an example, the chain rule is applied on the sigmoid function σ, which is given as

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

The sigmoid function can be decomposed into the following simpler functions. Their derivatives can be determined a lot easier.

$$\begin{aligned} f(x) &= -x &&\to \frac{d}{dx}f(x) = -1 \\ g(x) &= e^x &&\to \frac{d}{dx}g(x) = e^x \\ h(x) &= 1 + x &&\to \frac{d}{dx}h(x) = 1 \\ k(x) &= \frac{1}{x} &&\to \frac{d}{dx}k(x) = -\frac{1}{x^2} \end{aligned}$$

The derivative of the sigmoid function can be computed by repeatedly applying the chain rule, then the following result is obtained:

$$\frac{d}{dx}\sigma(x) = \frac{e^{-x}}{(1 + e^{-x})^2}.$$

Before the algorithms of forward and backward propagation can be discussed, computational graphs must be introduced. They are used to represent functions. Like regular mathematical graphs, they are structured in vertices and edges. The vertices define variables. In general, these can be given as scalars, vectors, matrices or tensors, but in our example only scalars will be used. The edges of a computational graph represent mathematical operations. These are simple functions whose derivations are easy to determine. The graph is then structured according to the composition of the underlying, more complex function. If a variable y is computed by the variable x by using a single operation, then an edge is placed from vertex x to the vertex y. The computational graph of the sigmoid function is shown in figure 2.7.

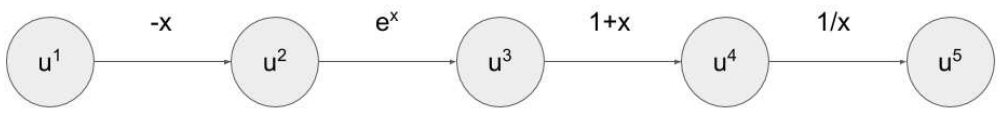

Figure 2.7: Computational graph of the sigmoid function.

Forward propagation is performed in the following way: The algorithm starts at the first node and progresses step by step through the graph. At each node the values of the parents $\mathbb{P}_i$ are used as input. The computation is done according to the label of the edge f^i, and the result is stored in the next node u^i. This process is repeated until all nodes are computed. The algorithm is given in Alg. 5 and introduced by Goodfellow, Bengio, and Courville [23].

Algorithm 5: Forward Propagation [23]

```
for $i = 1, ..., n_i$ do
    $u^i \leftarrow x_i$;
end
for $i = n_i + 1, ..., n$ do
    $\mathbb{P}_i \leftarrow \{u^j | j \in Pa(u^i)\}$;
    $u^i \leftarrow f^i(\mathbb{P}_i)$;
end
return $u^n$
```

In algorithm 5, $x_i, i = 1, ..., n_i$ are the input values, which are given to the input nodes $u^i, i = 1, ..., n_i$. The last node u^n is the output, which is returned at the end of the algorithm.
To compute $\sigma(1)$, the forward propagation algorithm is applied on the computational graph of the sigmoid function. The result is shown figure 2.8

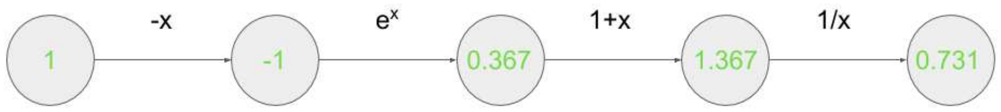

Figure 2.8: Computational graph of the sigmoid function after forward propagation is applied.

The back propagation algorithm proceeds backwards through the graph. It determines the derivative of every node. At first the derivative $\frac{du^n}{du^n}$ of the output node is computed, the algorithm follows the graph backwards and computes the gradient at every node $\frac{du^n}{du^j}$. This is done by applying the chain rule repeatedly.

$$\frac{du^n}{du^j} = \sum_{i:j \in Pa(u^i)} \frac{du^n}{du^i} \frac{du^i}{du^j}$$

The back propagation algorithm is shown in 6

Algorithm 6: Backward Propagation

```
gradTable;
gradTable[$u^n$] ← 1;
for $j$ = $n$-1,...,1 do
    gradTable[$u^j$] = $\sum_{i:j \in Pa(u^i)}$ gradTable[$u^i$] $\frac{du^i}{du^j}$;
end
return gradTable
```

There are a lot of algorithms, which can be used to compute derivatives. The advantage of back propagation is that it is relatively resource-saving.

Complex functions can be decomposed in simpler ones, then the computation of the gradient is done by calculating the derivatives of these easier functions. Due to the chain rule, the same sub-functions can be reused to compute the derivatives regarding different weights. Therefore, the same results can be reused several times and the whole process becomes less computationally expensive.
To explain back propagation in more detail, the sigmoid function is used as an example. With the notation from above, it can be written as $\sigma(x) = k \circ h \circ g \circ f(x)$. Its derivative is then given as

$$\sigma'(x) = k'(h \circ g \circ f(x)\ h'(g \circ f(x)))\ g'(f(x))\ f(x).$$

The notation shows that several expressions would have to be computed multiple times, if the derivation would have been determined in a naive way. In figure 2.9, the back propagation algorithm is applied on the computational graph of the sigmoid function.

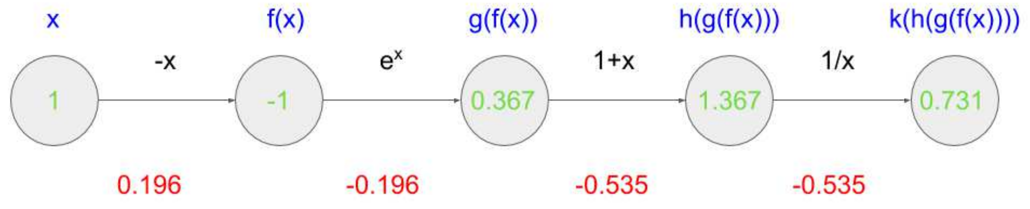

Figure 2.9: Computational graph of the sigmoid function after backward propagation is applied.

The blue expressions show the concatenations of the different functions. Their values are given by the numbers in the nodes. These numbers are used as input for the derivatives of the simple functions k', g', h', f'. To obtain the gradient of the composition of the sub-functions, the derivative of every simple function has to be multiplied with the derivative of the expression before. The intermediate results are given by the red numbers. The red value, which is furthest to the left, is the final result of the back propagation algorithm $\frac{d}{dx}\sigma(1) = 0.196$.

2.6 Convolutional Neural Networks

The convolution operation is applied in many areas. In physics, it is used in optics or signal processing. In mathematics, convolution is used to calculate the density function of the sum of random variables, to name just two examples. Convolution is also used in artificial intelligence, often in connection with image processing. Most of the work discussed in this section is from [23].
Mathematically, the convolution operation is defined for real valued integrable functions $f, g : \mathbb{R} \to \mathbb{R}$ as follows

$$f * g(t) = \int_{\mathbb{R}} f(x)g(t-x)dx$$

This is equivalent to

$$f * g(t) = \int_{\mathbb{R}} f(t-x)g(x)dx$$

If convolution is applied in machine learning, the function in the first position is typically the input, in this case a 2-dimensional image I. The second function is the kernel K, which has the same dimensions as the image. Furthermore, the functions, which are used here, are not continuous but discrete. Then the convolution is defined as

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

Often the input is not just a 2-dimensional image but possibly 3-dimensional. Then the last 2 dimensions describe the shape of the image, and the first is the number of channels. When images are used, there is typically a channel for each RGB value. Similarly, the output of the convolution also has several different channels, these are called feature maps. For each feature map a different kernel is applied. Whereas every kernel uses all the channels of the input to compute a feature map. The formula is then given by

$$(I * K)(k,i,j) = \sum_m \sum_n \sum_c I(c,m,n)K(k, i-m, j-n, c).$$

Here the indices c and k a linked to the channels. k is the index of the channels of the input image and c is the index of the feature maps.
When a dense layer in a neural network is implemented, every node from the previous layer is used to compute a node in the next layer. Here the situation is different, as only a small part of the neurons are used during the convolution. The reason for this is the design of the kernel, except for a small section, it is almost everywhere zero. In practice, the input can be an image with thousands of pixels, while the kernel has usually just a couple of non zero values. In a densely connected layer with n pixels and n weights per pixel, there are in total $n \times n$ weights and just as many operations have to by computed during the forward propagation. With a convolutional layer there are only k weights per node, where k is significantly smaller than n. Therefore, the occupied memory and the required computational operations are drastically reduced compared to classic neural networks. A visualization of the two layers can be seen in figure 2.10

Another characteristic of convolutional networks is parameter sharing. In regular neural networks each node has its own weights, so as mentioned above $n \times n$ weights. Due to parameter sharing each node uses the exact same weights, so a layer does only need k weights. While the computing time during the forward propagation does not decrease, since there are still $n \times k$ operations needed, the required memory space is significantly less.

Due to the shared weights, convolution is equivariant to translation. Mathematically, a function $f : \mathbb{R} \to \mathbb{R}$ is equivariant with respect to an operation $g : \mathbb{R} \to \mathbb{R}$ if the following holds

$$f(g(x)) = g(f(x)) \ \forall x \in \mathbb{R}.$$

This means that applying the operation g on x before the function f leads to the same result as applying g on $f(x)$. In this case, f is the convolution and g the translation of the input. Since all the weights are shared in a convolutional layer, translating does not change the outcome, but it shifts the output by the same amount as the input. Convolution is only equivariant to translation, other operations, such as scaling, flipping or rotating the input, will lead to different outputs.
In the following, variations of convolution are discussed. First, strides [23] are introduced. Applying convolution means sliding a window sized kernel over the image and computing the resulting product. The convolution that

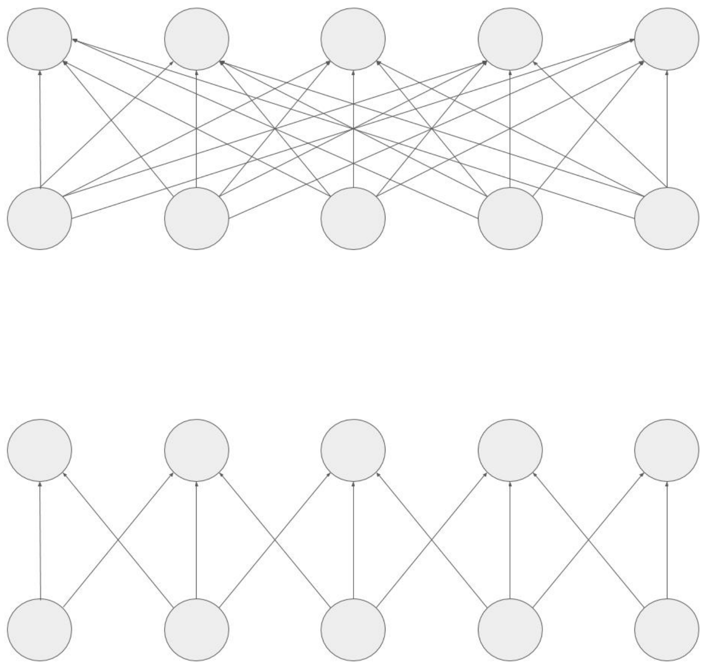

Figure 2.10: Comparison between a dense layer and a convolutional layer. Both layers have 5 nodes. The nodes are given by the circles and the connections/weights a represented by the arrows. Top: A dense layer. Each node is connected with all the previous nodes. Bottom: A convolutional layer with kernel size 3. Therefore, each node is only connected with 3 other nodes.

has been discussed until now has a stride of size 1. By increasing the number of sliding steps to s, convolution is only applied to every s-th node. This is called strided convolution. Of course, the step size can vary in each direction. Mathematically, this can be described as follows:

$$(I * K)(k, i, j) = \sum_m \sum_n \sum_c I(c, m, n) K(k, si - m, sj - n, c).$$

By multiplying the indices i, j with s the step size of the convolution increases, and the operation is only applied to every s-th node. A visualization

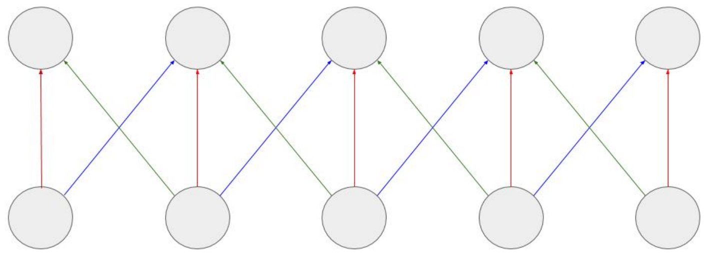

Figure 2.11: Parameter sharing. A convolutional layer is shown. The arrows with the same color represent shared weights. Every node uses the same 3 weights.

is given in figure 2.12. Strided convolution always leads to a loss of information, but the computational cost can already be reduced considerably with a small step size. Therefore, it is always difficult to find a reasonable stride size.

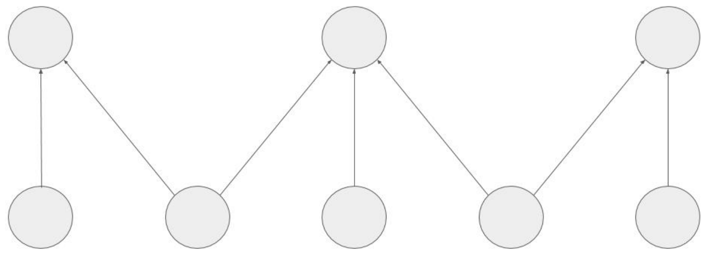

Figure 2.12: Strided convolution. The step size is chosen as 2. Therefore, convolution is only applied to every second node. The others are left out. This halves the output size.

The next variation within the convolutional operation is zero padding [23]. It describes the process of adding rows of zeros at the end of the input

to control shrinkage and avoid losing information at the boundary. Without zero padding the size of the input decreases and convolution can not be applied to every node. There are 3 main cases of zero padding.
The first one is called "valid". It is the case where no padding is applied. The size of the image shrinks after every convolution, depending on the stride and the size on the kernel. After a couple of operations a lot of information is lost. Therefore, the number of convolutions, which can be applied without zero padding, is limited.
The second case is called "same". Here just enough zeros are added to the boundary of the image to keep the size equal. Then the convolution is applied normally on the enlarged image. The nodes at the border are influenced by the zeros, since some values of the kernel are multiplied by zero.
The third variation is when zero-padding is used so that convolution is performed equally often at all nodes. In this case, the input image is enlarged a lot and usually the size of the output increases too. This case is called "full". In figure 2.13 a convolution operation with zero padding is shown.

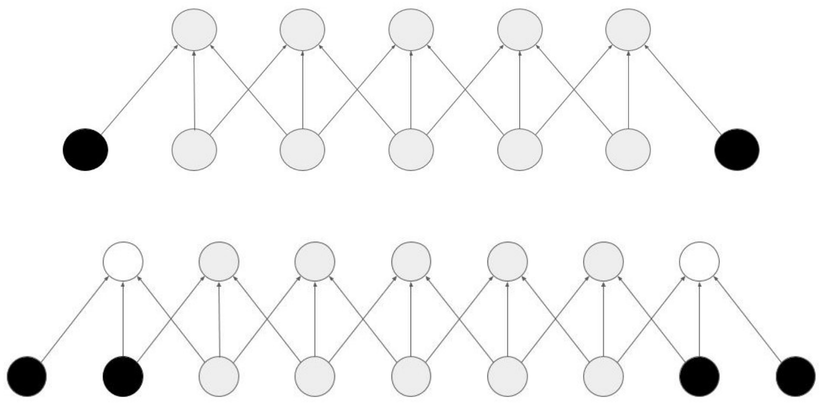

Figure 2.13: Zero-padding. Top: "Same" is applied. Therefore, nodes with value zero are added at the end. Bottom: Here "full" is used. Two nodes at each end are added, to make sure that convolution is applied equally often to all nodes.

At last, dilated convolution (Yu and Koltun [63]) is introduced. The difference to regular convolution is that the receptive field is enlarged, but the size of the kernel stays the same. In order to implement this, some nodes are skipped when computing the convolution.

$$(I * K)(k, i, j) = \sum_m \sum_n \sum_c I(c, dm, dn) K(k, i - m, j - n, c).$$

With regular convolution all the entries of the image and kernel are multiplied. In the formula above this is different, only every d-th entry of the input is considered. This increases the area of influence, without increasing the size of the kernel or the computing power needed to perform the operation. For $d = 1$ the dilated convolution is the same as the regular convolution, when $d > 1$ the receptive field increases. Figure 2.14 shows a visualization of the operation.
Dilation is mostly useful for images with higher resolution, when not every pixel is necessary to recognize important features. Nevertheless, information is lost when using dilation. It is, therefore, crucial to carefully select the dilation size.

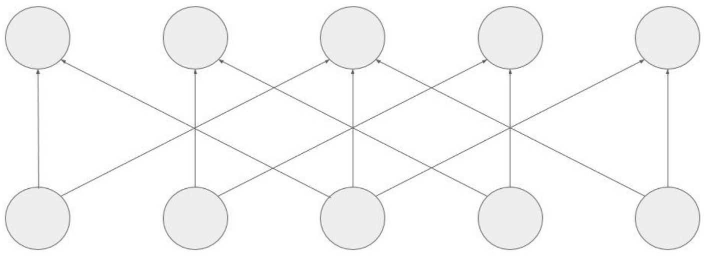
Figure 2.14: Dilated convolution. The dilation size d is set to 2. Therefore, only every second node is used for the convolution.

Now that several different versions of the convolution have been discussed, a different operation is introduced. It is called pooling and is often used after convolution. This operation uses a receptive field similar to convolution. While different nodes are weighted and added up during convolution, the pooling operator computes the mean or maximum. This is called max-pooling or average pooling.
The main effect of pooling is that it makes the output invariant to small translations of the input. If an object in the image is slightly shifted, the result will still be similar. As described in Goodfellow, Bengio, and Courville [23], sometimes it is only important to know the approximate position of an

object. To identify a face, for example, it is only important to know that one eye is on the left and one on the right side. It is not necessary to determine the exact location. An example of pooling is shown in figure 2.15

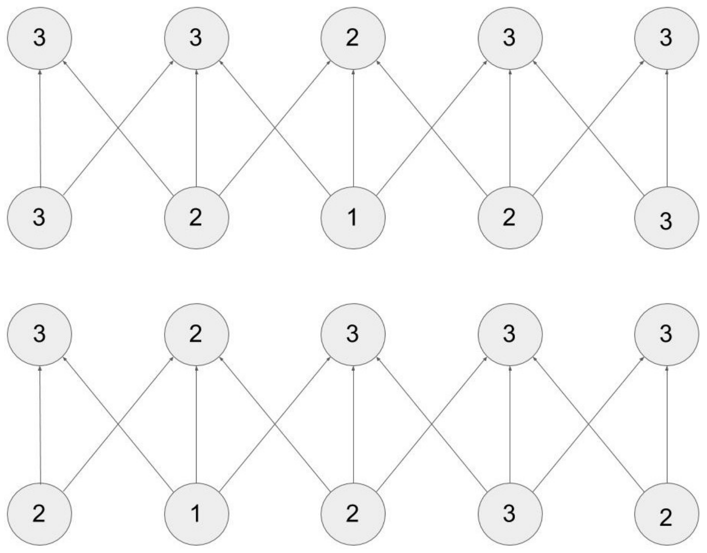

Figure 2.15: Max pooling. This image shows the max pooling operation. The lower layer is always the input. By applying pooling the upper layer is obtained. The input values in both cases are similar, but the bottom input is moved one node to the left. While all values in the input have changed only a few output values are different.

Maximum pooling has been found to work better than average pooling. In practice, convolutional layers tend to encode the spatial information of objects and patterns. Therefore, it's more revealing to consider the maximal than the average presence of different features (Chollet [9]).
Another type of pooling is called global pooling. In this case, the receptive field is the whole input. The entire feature map is then used to perform the operation. This type of pooling is used to aggressively summarize the presence of an object.
There is a common way in which convolutional blocks are built. First, convolution, then an activation function and finally pooling is applied. Neural networks often contain several of these blocks in series. This is motivated by the following idea: The first blocks are supposed to recognize simple

structures, such as lines, curves or edges. This information is then further processed by the following blocks. They identify more complex pattern like squares or circles. The last convolutional blocks can then finally locate the desired objects. These might be houses, wheels or even a brain tumor, depending on the problem. Therefore, every block in the network is responsible for detecting increasingly complex structures, which ultimately result in the identification of the targeted object.

2.7 U-Net

In this section, a modern architecture for convolutional neural networks is discussed, which was published by Ronneberger, Fischer, and Brox [48]. In recent years, this structure has become increasingly popular and is now state of the art. Until today the paper has been cited over 1200 times. The neural network proposed in the paper is called "U-Net". The name is inspired by the shape of the architecture, which strongly resembles the shape of a "U". The reason for this is that the model consists of an encoder and decoder part. In the encoder, the data is processed "downward" to encode the relevant features of an image. In the decoder, output of the encoder is processed "upward" to recreate the image in a appropriated way. The output of the decoder is then for example a segmentation of the image.
The encoder and a decoder are each made up of several convolutional blocks. The blocks in the decoder consist of two convolutional layers with relu activation followed by a pooling layer. The size of the input is halved after each block, while the number of channels is doubled. The output of the encoder is reduced from 572×572 to a size of 30×30 and the number of channels is increased to over 1000. The decoder is built the other way around. After each block the number of channels is halved and the size of the image is doubled. The convolutional blocks in the decoder have a similar structure as the blocks in the encoder. The main difference is that there is no max pooling. Instead, up-convolution is used. Here the up-convolution operator is implemented as up-sampling followed by a convolutional layer. Furthermore, skip connections are introduced. This means that the output of a convolutional block of the encoder is merged with the input of a block of the decoder. This improves the reconstruction on the original image, while the data is up-sampled.
Another way to decode an image is to use de-convolution/transposed convo-

lution (Zeiler et al. [64]) instead of up-convolution. The general idea arised from the desire to perform convolution in the opposite direction. To properly understand how the method works, let $y_1, ..., y_{k_0}$ be the input of the de-convolutional layer with k different channels. When regular convolution is applied to generate the image $y_1, ..., y_{k_0}$, then it is computed from feature maps $z_j,\ j = 1, ..., k_1$ and kernels $f_{i,j}$:

$$y_i = \sum_{j=1}^{k_1} z_j * f_{i,j} \tag{2.1}$$

During de-convolution y_i is given and used as an input to compute z_j from the kernel $f_{i,j}$. Equation 2.1 holds, but y_i is given and z_j unknown. Unfortunately, equation 2.1 is an under-determined system, so additional conditions have to be defined to obtain a unique solution for z_j. In [64] a regularization term on z_j is introduced, it favors sparsity on the feature maps. The final loss function is then given as

$$L = \frac{1}{2} \sum_{i=1}^{k_0} || \sum_{j=1}^{k_1} z_j * f_{i,j} - y_i||_2^2 + \sum_{j=1}^{k_1} |z_j|^p$$

The formula shows that convolution is not applied regularly but actually the other way round. Instead of the feature maps z_j, y_i is used as input and the feature maps are computed by using the kernels $f_{i,j}$.

2.8 Batch Normalization

Batch normalization (Ioffe and Szegedy [25]) was introduced to address the problem of varying distributions of the data handled by a single layer. The method normalizes the mean and standard deviation of the data, this allows the use of a higher learning rate and less careful initialization.
When a neural network is trained with stochastic gradient descent the following optimization is done.

$$\Theta = \text{argmin}_\Theta \frac{1}{N} \sum_{i=1}^{N} l(x_i, \Theta)$$

Here $x_1, ..., x_N$ is the training data, Θ are the weights of the network and $l(x_i, \Theta)$ is the loss depending on both. Instead of computing the loss of the whole training data, often only a smaller subset $x_1, ..., x_m$ of m samples is considered, called mini-batch. To optimize the loss of the whole training data, the gradient of the mini-batch is computed.

$$\frac{1}{m}\frac{dl(x_i, \Theta)}{d\Theta}$$

Since the computation over the whole data set is often very expensive, the gradient of the mini-batch is used as an approximation of the true gradient. A neural network is designed as a structure consisting of many layers. Let F_1, F_2 be two such layers. Then x is the input data and y the output.

$$y = F_2(F_1(x, \Theta_1)\Theta_2)$$

During the training of the network, the weights of each layer are updated simultaneously. Since modern neural networks have become very deep, small changes in the weights of early layers can lead to an enormous divergence in later results. Therefore, the distribution of the data in a layer changes after every update. This effect is called covariant shift [53]. It makes training the later layers difficult, because the weights have to adapted to the varying distribution. It is an advantage if the distribution of the data, which is given to a layers, does not change.
Lets further consider the sigmoid activation function:

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

The derivative of the sigmoid function is almost zero for large values of $|x|$. This means that the gradient will vanish for these values. Therefore, back propagation does not work properly. This leads to an early convergence of the training process, although a minimum might not be reached. The spreading of x near zero will prevent the gradient from vanishing. A normalization of the distribution can achieve this.
Let $\mathcal{B} = \{x_1, ..., x_m\}$ be a mini-batch of size m. During batch normalization the mean $\mu_{\mathcal{B}}$ and standard deviation $\sigma_{\mathcal{B}}$ are set to 0 and 1. They are defined as

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$$
$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

Ideally, the mean and standard deviation are computed with the whole training data, but here mini-batches are used to estimate the actual mean and std.
Then the data set is normalized by subtracting the mean and dividing it by the standard deviation.

$$\hat{x_i} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

The normalized data $\{\hat{x_1}, ..., \hat{x_m}\}$ is not applicable for all layers, since it would change the effect of certain functions. Let us take another look at the activation function sigmoid. In this case, normalizing the data would lead to a situation where mainly the linear part of the non-linear activation function is used. Therefore, a pair of parameters γ, β is trained to scale the normalized data. The final output of the batch normalization is then given as

$$y_i = \gamma \hat{x_i} + \beta$$

By training these parameters, batch normalization does not necessarily scale the data. When setting $\gamma = \sigma_{\mathcal{B}}$ and $\beta = \mu_{\mathcal{B}}$, batch normalization is equal to the identity function.
By adapting the distribution of each layer, small changes in early layer will not amplify later in the network. It is, therefore, less likely to converge in saturated regions of activation functions. This allows the usage of higher learning rates.
When training convolutional neural networks, the data is structured using many feature maps. Then batch normalization is applied to each map individually. In the case of a batch of size m which uses k feature maps with dimensions $p \times q$, when batch normalization is applied, the same feature map of every data set is taken into account. Therefore, batch normalization is applied to k data sets which have size $|m \times p \times q|$.

2.9 Discussion

Neural networks are, in many cases piece-wise linear, functions which are trained to approximate unknown decision functions. A main result is given in theorem 2.6. It says that under certain conditions every decision function can be approximated by an appropriately large network. Based on this result it makes sense to deal more intensively with the architecture and training of neuronal networks.
Several optimization algorithms are used to train neural networks. The Adam [31] algorithm is very popular. Its uniqueness is dsiccused in section 2.4. The scaling and direction of the weight adjustment is chosen dynamically, based on earlier knowledge of the data. The derivation is further computed with back-propagation. It allows a fast and resource-saving determination of the gradient.
There are several different architectures used in deep learning. In image segmentation convolutional U-Nets are state of the art [23]. Their structure allows them to recognize more complex structures and simultaneously maintain the information of the original image. Therefore, this structure is ideal for image segmentation tasks [15].

Chapter 3

Spectral Coordinates

Artificial neural networks are often patch based. When they are used for segmentation, they use a small window and are therefore limited to a local view. More large scale information is an advantage when segmenting biological structures. The anatomy of a brain, for example, differs greatly depending on its position. Therefore, information about the location of a patch improves the quality of the segmentation [59].

In this chapter spectral coordinates and diffusion maps are discussed. These are used to gain structural information about a graph [10]. They are therefore an ideal support for neural networks in the segmentation of anatomical structures. In the following sections, an introduction to their theory and an insight to the numerous applications is given.

In section 3.1, the graph Laplacian and its varieties are introduced. To improve the understanding, some of its characteristics, with a focus on eigenvalues and eigenvectors, are discussed and proven. At last, spectral coordinates are introduced, and several examples of graphs and their coordinates are given. Spectral clustering is a popular application. It is a method to partition the graph based on its structure. Spectral coordinates and clustering is presented in sections 3.2 and 3.3.

Diffusion maps are closely connected to spectral coordinates. Although they are not directly applied in this thesis, a thorough understanding of diffusion coordinates improves the knowledge of spectral coordinates. In particular, theorem 3.3 gives a strong justification for the use of diffusion and therefore spectral coordinates. The common features and subtle differences are discussed in section 3.4.

If information spreads from a single node through the graph, this process

can be described as diffusion. The diffusion metric describes how differently this process takes place with two different starting nodes. It gives insight to the deeper structure of a graph. The main result of this section is that the diffusion metric can be computed be the diffusion coordinates.
At last, in section 3.5, applications of spectral coordinates and diffusion maps are shown. Examples are given in which the coordinates are used to match brains and add spatial information to neural networks. Another fascinating application is the identification on functional brain regions.

3.1 The Graph Laplacian

This section is about the Graph Laplacian and its properties, the main information is given in [58]. Let, therefore, $G = (V, E)$ be an un-directed graph with vertices $V = \{v_1, ..., v_n\}$ and edges E. The graph is weighted. An edge $e_{i,j} \in E$ with corresponding vertices v_i, v_j is, therefore, assigned to a weight $w_{i,j} \geq 0$. The adjacency matrix W of a graph G is given as $W = (w_{i,j})_{i,j=1,...,n}$. If there is no edge between two vertices, the corresponding weight is set to zero $w_{i,j} = 0$.
Furthermore, the degree d_i of a vertex v_i is given as

$$d_i = \sum_{j=1}^{n} w_{i,j}.$$

The degree of v_i is defined as the sum of all weights, which are attached to v_i. It indicates how strong a vertex is connected to the graph. The degree matrix D is defined as the diagonal matrix formed by the degrees of the vertices. $D = \operatorname{diag}(d_1, ..., d_n)$.
The (un-normalized) Laplacian L of the graph is given as

$$L = D - W.$$

In the literature there is no uniform definition for this matrix. Often each author defines his individual graph Laplacian. In this work the definitions given by Von Luxburg [58] are used.
Two further graph Laplacian exits. These are known as normalized graph Laplacian and are defined as

$$\begin{aligned} L_{\text{sym}} &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \\ L_{\text{rw}} &= D^{-1} L = I - D^{-1} W \end{aligned}$$

L_{sym} is the symmetric version of the graph Laplacian and L_{rw} is often used in connection with random walk problems.
Let u and λ be an eigenvalue and eigenvector of the un-normalized graph Laplacian L. In the following equation the generalized eigenvalue problem $Lu = \lambda D^{-1}u$ of the symmetric Laplacian L_{sym} is discussed.

$$\begin{aligned} L_{\text{sym}} D^{\frac{1}{2}} u &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} D^{\frac{1}{2}} u \\ &= D^{-\frac{1}{2}} L u \\ &= \lambda D^{-\frac{1}{2}} u \\ &= \lambda D^{-1} D^{\frac{1}{2}} u \end{aligned}$$

One can see that the eigenvalue λ and vector $D^{\frac{1}{2}}u$ satisfy the generalized eigenvalue problem for L_{sym}. A similar property can be proven for the random walk Laplacian L_{rw}.

$$\begin{aligned} L_{\text{rw}} u &= D^{-1} L u \\ &= D^{-1} \lambda u \\ &= \lambda D^{-1} u \end{aligned}$$

Therefore, the eigenvector u and the eigenvalue λ of the un-normalized Laplacian L satisfy the generalized eigenvalue problem $L_{\text{rw}}u = \lambda D^{-1}u$.
In the following part a couple of basic properties of the graph Laplacian are discussed. The focus lies mainly on the un-normalized version, but similar results hold for the normalized Laplacian as well.

Theorem 3.1 ([58]). *Let G be a connected, un-directed and weighted graph with n vertices. Let further L be the un-normalized Laplacian of G, defined as $L = D - W$. Then the matrix L holds the following properties:*

1. *For every vector $f \in \mathbb{R}^n$*

$$f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{i,j} (f_i - f_j)^2$$

2. *L is symmetric and positive semi-definite.*
3. *The smallest eigenvalue of L is 0 and the corresponding eigenvector is the constant one vector $\mathbb{1}$.*

4. L has n non-negative, real valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$.

Proof. 1. By the definition of d_i

$$\begin{aligned} f^T L f =& f^T D f - f^T W f = \sum_{i=1}^{n} f_i^2 d_i - \sum_{i,j=1}^{n} f_i f_j w_{i,j} \\ =& \frac{1}{2}\left(\sum_{i=1}^{n} f_i^2 d_i - 2\sum_{i,j=1}^{n} f_i f_j w_{i,j} + \sum_{i=1}^{n} f_i^2 d_i\right) \\ =& \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}(f_i - f_j)^2 \end{aligned}$$

2. L is symmetric because the matrices W and D are. D is a diagonal matrix, hence symmetric. In an un-directed graph $w_{i,j} = w_{j,i}$ holds. Therefore, W is symmetric.
   From 1. follows that $f^T L f \geq 0 \; \forall f \in \mathbb{R}^n$. Therefore, L is positive semi-definite.

3. To proof this, the i-th entry of the of the vector $L\mathbb{1}$ is calculated:

$$(L\mathbb{1})_i = d_i - \sum_{j=1}^{n} w_{i,j} = 0$$

   Hence, $L\mathbb{1}$ is the zero vector and $\mathbb{1}$ is an eigenvector of L with eigenvalue 0.

4. Since L is real, symmetric and positive semi-definite, its eigenvalues are real and non-negative. It follows from 3. that the smallest eigenvalue is 0.

□

The main result of this theorem is the characterization of the eigenvalues of the Laplacian. It says that all eigenvalues are non-negative and the smallest eigenvalue is always 0. In the following proof the connection between the multiplicity of the smallest eigenvalue and the number of connected components is discussed.

Theorem 3.2 ([58])**.** *Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 in L equals the number of connected components $A_1, ..., A_k$ in the graph. The eigenspace of the eigenvalue 0 is spanned by the indicator vectors $\mathbb{1}_{A_1}, ..., \mathbb{1}_{A_k}$ of those components.*

Proof. If $k = 1$, there is only one connected component. If $f \in \mathbb{R}^n$ is an eigenvector of L with eigenvalue 0, then the following holds:

$$0 = f^T L f = \frac{1}{2} \sum_{i,j=1}^{n} w_{i,j}(f_i - f_j)^2$$

For the sum to be zero, all the terms $w_{i,j}(f_i - f_j)^2$ have to vanish. All weights are non-negative $w_{i,j} \geq 0$. If a weight is positive $w_{i,j} > 0$, the corresponding vertices v_i, v_j are connected by an edge. Then $f_i = f_j$ has to hold. Every two vertices in a connected component can be connected by a path. Since there is only one connected component in the graph, the only vectors that satisfy the requirement are constant vectors. Therefore, the constant one vector $\mathbb{1}$ spans the eigenspace of the eigenvalue 0. $\mathbb{1}$ is the indicator function of the graph.
Now consider the case $k > 1$. Let the indices be ordered so that the adjacency matrix is a block diagonal matrix. Then the Laplacian is also a block diagonal matrix.

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{pmatrix}$$

The spectra of a block diagonal matrix is the union of the spectra of each block matrix. The eigenvectors are the eigenvectors of each block matrix, where the indices of the other matrices are filled up with zeros. Each block matrix represents the Laplacian of a connected component. From the case $k = 1$, it is known that the eigenvector of the eigenvalue 0 is given by the indicator vector $\mathbb{1}_{A_i}$ of the component. Therefore, the eigenspace of the eigenvalue 0, regarding the matrix L, is spanned by the indicator vectors of the connected components $\mathbb{1}_{A_1}, ..., \mathbb{1}_{A_k}$. □

These proofs show that the Laplacian matrix holds a lot of information about the structure of a graph. Therefore, further investigation is needed to exploit the connections between the Laplacian and the structure of its graph.

3.2 Spectral Coordinates

In order to calculate the spectral coordinates, the graph Laplacian is utilized. Again, in the literature there is no agreement on which matrix is to be used. Wachinger, Reuter, and Klein [59] prefers the un-normalized graph Laplacian $L = D - W$, whereas in Lombaert et al. [39] and Lombaert, Sporring, and Siddiqi [38] $L_{\text{rw}} = D^{-1}L$ is used. Since the calculation is identical in both cases, the same notation as in the last section is used, and the spectral coordinates are computed with the un-normalized Laplacian L.
Now let G be a graph and L the corresponding Laplacian, then let further be $\lambda_1, ..., \lambda_n$ the eigenvalues and $u_1, ..., u_n$ the eigenvectors of L. If a k dimensional representation of our data is required, then the spectral coordinates for the vertex v_i are given as $u_{1,...,k}[i]$. These are the i-th entry of the first k eigenvectors, a vector of length k. The eigenvectors are ordered according to the size of the associated eigenvalues so that $\lambda_1 \leq, ..., \leq \lambda_n$. Only the non constant eigenvectors are included [59] [39], since the constant vector does not hold any information about the structure of the graph.
Now some examples of spectral coordinates of different graphs are presented. The first one is basically a line of vertices, two adjacent vertices are connected by an edge. Every vertex has, therefore, two edges, except from the first and last one. The graph Laplacian looks as follows:

$$L = \begin{pmatrix} 1 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & & \ddots & & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 1 \end{pmatrix}$$

When the eigenvectors of the Laplacian are computed, the first vector is constant. Therefore, it will not be displayed. The eigenvectors of the following 4 smallest eigenvalues are given in figure 3.1. The value of a coordinate of the eigenvector is represented by the color of the vertex. The spectral coordinates of a vertex are then given by the values of the corresponding entries in the eigenvectors.

The second example is a square. In this case the vertices are horizontally and vertically alined. There is an edge connecting every vertex to the one

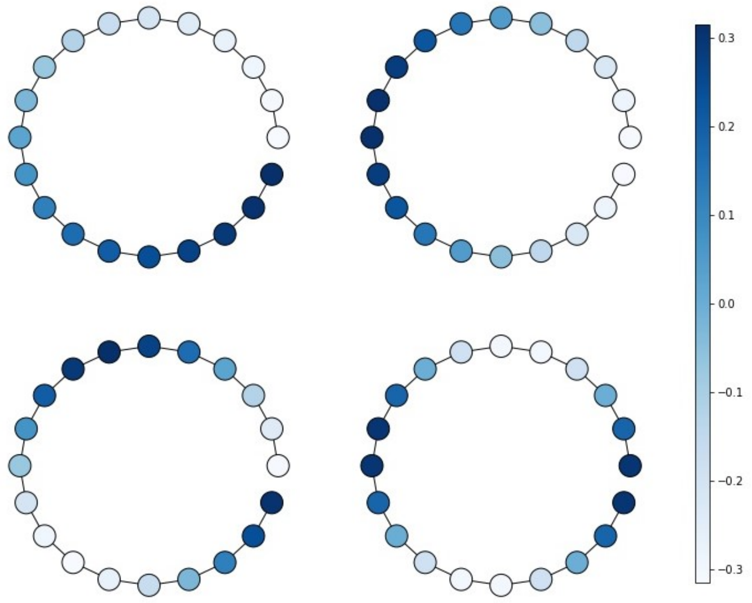

Figure 3.1: Spectral Coordinates. The first eigenvector is constant and not displayed. The second eigenvector is upper left, the third upper right, the forth bottom left and the fifth is at the bottom right

above, below, to the right and the left. The graph is relatively simple and the Laplacian looks similar to the one above. The difference is that on the diagonal, which before consisted only of 2s, there are also 3s and 4s, which represent the border and the inner points.

$$L = \begin{pmatrix} 2 & -1 & -1 & & & & & \\ -1 & 3 & -1 & -1 & & & & \\ & -1 & 4 & -1 & -1 & -1 & & \\ & & & \ddots & & & & \\ & & & & \ddots & & & \\ & & -1 & -1 & -1 & 4 & -1 & \\ & & & & -1 & -1 & 3 & -1 \\ & & & & & -1 & -1 & 2 \end{pmatrix}$$

After building the graph Laplacian, the eigenvectors can be computed. Since the graph has only one connected component, the first eigenvector is again constant. Therefore, it will not be displayed. The following four eigenvectors are shown in figure 3.2

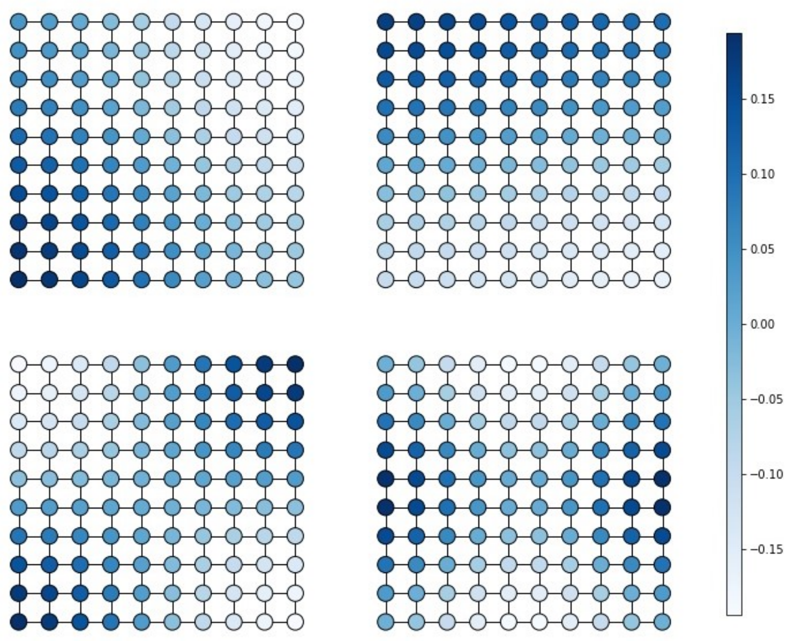

Figure 3.2: Spectral Coordinates. The first eigenvector is constant and not displayed. The second eigenvector is upper left, the third upper right, the forth bottom left and the fifth is at the bottom right

Spectral coordinates are often used to compare similar structures [39]. Since eigenvectors are used to compute the coordinates, there are a couple of things to consider. First, even normalized eigenvectors are not unique, since for every vector v also $-v$ fulfills the equation

$$L(-v) = \lambda(-v).$$

After ordering the eigenvectors according to the value of the eigenvalues, the sign of each vector has to be checked and possibly changed. Second, the multiplicity of eigenvalues can be a problem. If an eigenvalue has a geometric

multiplicity of more than 1, there are no uniquely determined eigenvectors, which span the eigenspace.
An example for a structure, in which such multiplicities occure, is a ring. The graph is similar to the first example, but the first and last vertex are also connected by an edge. The corresponding Laplacian is defined as

$$L = \begin{pmatrix} 2 & -1 & & & & & -1 \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & & \ddots & & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ -1 & & & & & -1 & 2 \end{pmatrix}$$

When the eigenvalues are computed and ordered, the second and third eigenvalue are equal. The eigenvalue has a geometric multiplicity of 2. The computed eigenvectors are given in figure 3.3. As discussed above, these vectors are not unique. Any vectors, which span the eigenspace, could be chosen as eigenvectors.

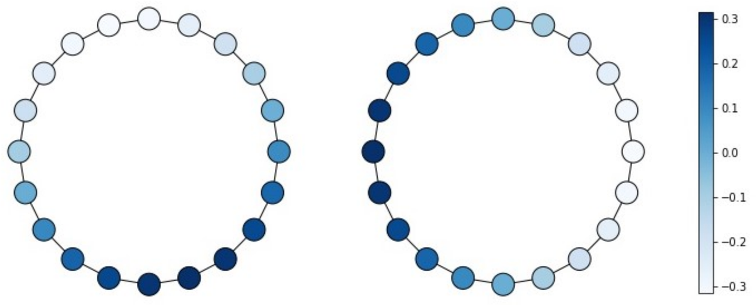

Figure 3.3: Spectral Coordinates. The first eigenvector is constant and not displayed. The second eigenvector is left, the third is right.

3.3 Spectral Clustering

A common use for spectral coordinates is spectral clustering. This method is not used in the later work, but it gives an insight on what information

is contained in the coordinates. Therefore, it justifies the use of spectral coordinates for a structural analysis of a graph. The work shown in this section, is from [58].
let n data points be given, an let the goal be to separate them in k disjoint clusters. At the beginning, their pairwise similarities $w_{i,j} = s(x_i, x_j)$ are measured to obtain the weights. They have to be real and positive. There are several ways to do that. One approach is to connect two data points, if the distance between them is smaller than a predefined value ϵ. This leads to a sparsely connected graph and might miss out information about points which are further away from each other. Another way to compute the weights is by fully connecting all the data points. This can be done by using a distance function, such as

$$s(x_i, x_j) = \exp(-\|x_i - x_j\| /2\sigma).$$

The parameter σ defines how quickly the function decreases when the distance increases. The last approach, which is mentioned, is used to compare fMRI data (Langs et al. [36]). Here every point is represented by a time series. To calculate meaningful weights, the correlation coefficient $< i, j >$ between two time series is calculated. The distance function is then given by

$$s(x_i, x_j) = \exp(- < i, j > /2\sigma)$$

In any case, the method on how to define weights between data points is crucial for the outcome of the spectral clustering and should be chosen wisely.
The un-normalized graph Laplacian L is computed with the weights and the adjacency matrix. The first k eigenvectors of L are written as a matrix $U \in \mathbb{R}^{n\times k}$. The columns of U are the eigenvectors, and the rows the spectral coordinates $y_i \in \mathbb{R}^k$ $i = 1, ..., n$. These are then clustered by a k-means algorithm in order to obtain k disjoint clusters. The algorithm is given in 7. Similar algorithms exist for the normalized Laplacians L_{sym} and L_{rw}, which give equal results.

Algorithm 7: Spectral Clustering as in [58]

```
Input: Adjacency matrix $W$ ;
Compute unnormalized Laplacian $L = D - W$;
Compute first $k$ eigenvectors $U \in \mathbb{R}^{n \times k}$;
For $i = 1, ..., n$ let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i$-th
  row of $U$;
Use $k$-means algorithm to group $y_1, ..., y_n$ in $k$ clusters $C_1, ..., C_k$;
return $C_1, ..., C_k$
```

Spectral coordinates are a useful way to group data points in disjoint clusters. Within these algorithms there are still difficulties, such as the required computing power or the question in how many clusters the data points should be arranged. Nevertheless, the results suggest that spectral coordinates are meaningful when it comes to describing the structure of a graph.

3.4 Diffusion Maps

Diffusion maps, as discussed by De la Porte et al. [17], [42] and [35], also have a tight connection to spectral coordinates. Although their interchangeability is not entirely proven, discussing diffusion maps helps to strengthen the understanding of spectral coordinates.
The idea of diffusion maps is to measure the distance between two vertices v_i and v_j by not just computing the shortest path but by taking several paths into account. The diffusion distance decreases when there are multiple short paths from v_i to v_j. Therefore, vertices in the same high density region are closer than points that are only connected by a single short path.
As in spectral clustering, data points are given and an adjacency matrix W is defined. This can be done with similar methods as before, but in De la Porte et al. [17] they choose a more general approach. A diffusion kernel k is introduced. It has to satisfy the following two properties:

1. k is symmetric: $k(v_i, v_j) = k(v_j, v_i)$

2. k preserves positivity: $k(v_i, v_j) \geq 0$

These properties are in line with the requirements from above, since the weights of an undirected graph are symmetric, and the weights have to be real valued and non-negative. Instead of computing the graph Laplacian, the diffusion matrix is used. It is defined as

$$P = D^{-1}W$$

The matrix D is, again, defined as the diagonal matrix of the row sums of the adjacency matrix W.
The diffusion matrix P seems to be very different form the graph Laplacian, but there are some similarities. Therefore, the eigenvectors and eigenvalues of P and the normalized Laplacian L_{rw} are explored. Let u and λ be an eigenvector and eigenvalue of L_{rw}.

$$\begin{aligned}
Pu =& D^{-1}Wu \\
=& (I - I - D^{-1}W)u \\
=& (I - L_{\text{rw}})u \\
=& Iu - L_{\text{rw}}u \\
=& u - \lambda u \\
=& (1 - \lambda)u
\end{aligned}$$

The eigenvectors of L_{rw} are also the eigenvectors of P. The difference is that the corresponding eigenvalue is $1 - \lambda$ instead of λ. In spectral clustering, the focus always lies on the eigenvectors with the smallest eigenvalues. These hold the most information about the graph. In this case, the eigenvectors with the largest eigenvalues are used.
All the entries of P are positive, and the sum over each row equals 1. Therefore, the entries of P can be interpreted as transition probabilities. The entry $p_{i,j}$ can be seen as the probability of making a step from the vertex v_i to v_j. Similarly, the entry $p_{i,j}^t$ of the matrix P^t represents the probability to move from v_i to v_j in t steps.
The closer two points are to each other, or the more short paths there are from v_i to v_j, the higher is value of $p_{i,j}^t$. The matrices P^t reveal the underlying geometry of the graph, since an increasing number of paths are taken into account.
Next, the diffusion metric is defined. It benefits from the information of the diffusion matrix and is given as follows:

$$D_t(v_i, v_i)^2 = \sum_{v_k \in V} (p_{j,k}^t - p_{i,k}^t)^2 \frac{1}{D_{k,k}}.$$

V is the set of vertices of the graph G. $p_{i,k}^t$ is the probability of moving from v_i to v_k in t steps. $D_{k,k}$ are the row sums of the adjacency matrix W. The diffusion metric sums up all the weighted differences of the probabilities of moving from v_i or v_j to a vertex v_k. This means that the metric describes how differently the diffusion process of two vertices takes place. Two vertices which lie in the same high density area will diffuse similarly through the graph. In contrast, points in different areas with only sparse connections diffuse differently and, therefore, have a high diffusion value.
Let $u_1, ..., u_n$ be the eigenvectors of the diffusion matrix P. The main result is that the diffusion metric of two vertices $D_t(v_i, v_j)$ can be written as the difference between the spectral coordinates of P multiplied with the eigenvalues of the diffusion matrix P. Let $u_1, ..., u_n$ be the eigenvectors, $\lambda_1, ..., \lambda_n$ the eigenvalues of P and $u_i[j]$ be the j-th entry of i-th eigenvector.

$$D_t(v_i, v_j) = \left\| \begin{pmatrix} \lambda_1^t u_1[i] \\ \lambda_2^t u_2[i] \\ \vdots \\ \lambda_n^t u_n[i] \end{pmatrix} - \begin{pmatrix} \lambda_1^t u_1[j] \\ \lambda_2^t u_2[j] \\ \vdots \\ \lambda_n^t u_n[j] \end{pmatrix} \right\|$$

$\|\|$ is the euclidean norm. The most important characteristics of this statement are, first, that all eigenvectors are taken into account. The statement does not hold, if only the first k eigenvectors are used. The second characteristic is the use of eigenvalues. The i-th eigenvalue is multiplied with the corresponding eigenvector before computing the spectral coordinates.
Although the theorem above does only hold if all eigenvectors are used. In practice and in order to achieve dimensionality reduction, only k eigenvectors are chosen. Therefore, the eigenvectors with the most dominant eigenvalues are chosen [17], [42]. This ensures the best approximation of the diffusion distance. The algorithm to compute the diffusion distance is given in 8

Algorithm 8: Diffusion mapping algorithm as in [17]

```
Input: Adjacency matrix $W$ ;
Compute diffusion matrix $P = D^{-1}W$;
Compute first $k$ eigenvectors and multiply them with the
  corresponding eigenvalue $U = (\lambda_1^t u_1, ..., \lambda_k^t u_k)$;
For $i = 1, ..., n$ let $y_i \in \mathbb{R}^k$ be the vector corresponding to the $i$-th
  row of $U$;
return $y_1, ..., y_n$
```

The algorithm works as follows: First, instead of using the Laplacian, the diffusion matrix P is computed. Second, the eigenvectors u_i of P are multiplied with the corresponding eigenvalues λ_i. To compute the diffusion metric of P^t, the t-th power of the eigenvalue λ_i^t is taken instead of λ. To achieve dimensionality reduction, only the eigenvectors corresponding to the k largest eigenvalues are used. The vectors y_j are then defined as the j-th index of each vector $\lambda_i^t u_i[j], i = 1, ...k$. The euclidean distance between these vectors is equal to the diffusion distance between the vertices v_j.
The following statement proves that the diffusion distance can be computed with the vectors $y_i, ..., y_n$. This will be done in two steps. At first, the matrix $P' = D^{\frac{1}{2}} P D^{-\frac{1}{2}}$ is introduced. It will be verified that it has the same eigenvectors and similar eigenvalues as P. In the second prove, this will be utilized to achieve the desired result.

Lemma 3.2.1 ([17])**.** *Let G be a graph with vertices V and k be the diffusion kernel defined on G. Let further W be the symmetric $n \times n$ matrix such that $W[i, j] = k(v_i, v_j)$. D is defined as the diagonal matrix containing the row sums of W. Then the diffusion matrix P is given as*

$$P = D^{-1}K. \tag{3.1}$$

The matrix P' is defined as

$$P' = D^{\frac{1}{2}} P D^{-\frac{1}{2}}, \tag{3.2}$$

Then the following statements hold:

1. *P' is symmetric*

2. *P' has the same eigenvalues as P*

3. *The eigenvectors x'_k of P' are multiplied by $D^{-\frac{1}{2}}$ and $D^{\frac{1}{2}}$ to receive the left and right eigenvectors of P respectively.*

Proof. When 3.1 is substituted into 3.2, the following is obtained:

$$P' = D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

Since W and D are symmetric, so is P'. Therefore, there exits a orthonormal set of eigenvectors, so that

$$P' = S \Lambda S^T \tag{3.3}$$

Λ is a diagonal matrix, containing the eigenvalues of P'. S is a matrix with the eigenvectors as columns. Equation 3.2 can be rewritten as

$$P = D^{-\frac{1}{2}} P' D^{\frac{1}{2}} \tag{3.4}$$

and with equation 3.3

$$P = D^{-\frac{1}{2}} P' D^{\frac{1}{2}} = D^{-\frac{1}{2}} S \Lambda S^T D^{\frac{1}{2}}$$

is received. S is an orthogonal matrix.

$$\begin{aligned} P =& D^{-\frac{1}{2}} S \Lambda S^{-1} D^{\frac{1}{2}} \\ =& (D^{-\frac{1}{2}} S) \Lambda (D^{-\frac{1}{2}} S)^{-1} \\ =& Q \Lambda Q^{-1} \end{aligned}$$

This equation shows that the eigenvalues λ of P are also the eigenvalues of P'. Further, the right eigenvectors of P are the columns of

$$Q = D^{-\frac{1}{2}} S$$

and the left eigenvectors are the rows of

$$Q^{-1} = S^T D^{\frac{1}{2}}.$$

Therefore, the eigenvectors x'_k of P' can be written as the eigenvectors of P. The right eigenvectors u_k of P are

$$u_k = D^{-\frac{1}{2}} x'_k \tag{3.5}$$

and the left eigenvectors e_k are

$$e_k = D^{\frac{1}{2}} x'_k. \tag{3.6}$$

□

Since P' is symmetric, its eigendecomposition is given by

$$P'[i,j] = \sum_n \lambda_n x'_n[i] x'_n[j].$$

Similarly, the eigendecomposition of P'^t is given by

$$P'^t[i,j] = \sum_n \lambda_n^t x'_n[i] x'_n[j].$$

Applying 3.5 and 3.6 leads to the eigendecomposition of P^t.

$$P[i,j] = \sum_n \lambda_n^t u_n[i] e_n[j].$$

This result will be used in the following theorem.

Theorem 3.3 ([17])**.** *If the diffusion coordinates are chosen as the eigenvectors of P multiplied with their eigenvalues, then the diffusion distance between points in the data space is equal to the euclidean distance in the embedding space.*

Proof. Let y_i be the i-th diffusion coordinate of the matrix P^t such that

$$y_i = \begin{pmatrix} \lambda_1^t u_1[i] \\ \lambda_2^t u_2[i] \\ \vdots \\ \lambda_n^t u_n[i] \end{pmatrix}$$

What has to be proven is

$$\begin{aligned} D_t(x_i, x_j)^2 &= \sum_{v_k \in V} (p_{i,k}^t - p_{j,k}^t)^2 \frac{1}{D_{k,k}} \\ &= \|y_i - y_j\|^2 \end{aligned}$$

$p_t(x_i, x_j)$ are the elements of the diffusion matrix P. Using the eigendecomposition, which was discussed above, this becomes

$$\begin{aligned} D(x_i, x_j)^2 &= \sum_{v_k \in V} (p_{i,k}^t - p_{j,k}^t)^2 \frac{1}{D_{k,k}} \\ &= \sum_k \Big(\sum_n \lambda_n^t u_n[i] e_n[k] - \sum_n \lambda_n^t u_n[j] e_n[k]\Big)^2 \frac{1}{D_{k,k}} \\ &= \sum_k \Big(\sum_n \lambda_n^t e_n[k] (u_n[i] - u_n[j])\Big)^2 \frac{1}{D_{k,k}} \end{aligned}$$

Then the brackets are expanded, and the order of summation is exchanged.

$$
\begin{aligned}
&= \sum_k \sum_{n,m} \lambda_n^t \lambda_m^t e_n[k] e_m[k] (u_n[i] - u_n[j])(u_m[i] - u_m[j]) \frac{1}{D_{k,k}} \\
&= \sum_{n,m} \lambda_n^t \lambda_m^t (u_n[i] - u_n[j])(u_m[i] - u_m[j]) \sum_k e_n[k] e_m[k] \frac{1}{D_{k,k}} \\
&= \sum_{n,m} \lambda_n^t \lambda_m^t (u_n[i] - u_n[j])(u_m[i] - u_m[j]) \sum_k x'_n[k] D_{k,k}^{\frac{1}{2}} x'_m[k] D_{k,k}^{\frac{1}{2}} \frac{1}{D_{k,k}} \\
&= \sum_{n,m} \lambda_n^t \lambda_m^t (u_n[i] - u_n[j])(u_m[i] - u_m[j]) \sum_k x'_n[k] x'_m[k]
\end{aligned}
$$

The eigenvectors x'_n are orthonormal.

$$
\begin{aligned}
x_n'^T x'_m &= 0 \quad \text{for} \quad n \neq m \\
x_n'^T x'_m &= 1 \quad \text{for} \quad n = m
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
&= \sum_n \lambda_n^{2t} (u_n[i] - u_n[j])^2) \\
&= \|y_i - y_j\|^2
\end{aligned}
$$

□

After diffusion maps have been discussed in detail, the similarities and differences to spectral coordinates are analyzed. First, the eigenvectors of P are the eigenvectors of the normalized Laplacian L_{rw}. These are the eigenvectors of the generalized eigenvalue problem $Lv = \lambda Dv$. Second, let $\lambda_1, ..., \lambda_n$ be the eigenvalues of P. These are not the eigenvalues of L_{rw}. They have the be transformed as follows:

$$1 - \lambda_1, ..., 1 - \lambda_n.$$

Again, these are the eigenvalues of a generalized eigenvalue problem for L. The third difference are the diffusion coordinates themselves. Spectral coordinates are obtained by computing the first k eigenvectors $u_1, ..., u_k$. The diffusion coordinates of the matrix P are defined as follows:

$$\begin{pmatrix} \lambda_1^t u_1[i] \\ \lambda_2^t u_2[i] \\ \vdots \\ \lambda_k^t u_k[i] \end{pmatrix}$$

The eigenvectors are multiplied with to corresponding eigenvalues: $\lambda_1 u_1, ..., \lambda_k u_k$. Then they are formed into a matrix U, in which the columns are the eigenvectors multiplied with the eigenvalues. The rows of U are the diffusion coordinates. The i-th row of U corresponds to the i-th vertex in the graph.

3.5 Applications

In this section four recent papers are discussed, in which spectral coordinates and diffusion maps have been successfully used. The focus of the papers lies on medical applications of spectral coordinates and diffusion maps. The first paper, "DeepNAT: Deep Convolutional Neural Network for Segmenting Neuroanatomy" by Wachinger, Reuter, and Klein [59], introduces spectral coordinates to deep learning to segment brains. The second and third paper, "Fast Brain Matching with Spectral Correspondence" and "Diffeomorphic Spectral Matching of Cortical Surfaces" by Lombaert et al. [39] and Lombaert, Sporring, and Siddiqi [38], use spectral coordinates to develop an algorithm which matches human brains according to their cortical surface. The last paper, "Identifying Shared Brain Networks in Individuals by Decoupling Functional and Anatomical Variability" by Langs et al. [36], applies diffusion maps on fMRI data to explore the connectivity map of the human brain.
"DeepNat" is a 3 dimensional neural network which segments MR images of human brains. The network consists of two parts. At first, the MRI is segmented into foreground and background. The result is then used to compute the spectral coordinates. In the second part, another neural network, which receives the original MRI and the first three spectral coordinates as input, identifies 25 different regions of the human brain.
The spectral coordinates are computed by using the eigenvalues of the unnormalized Laplacian $L = D - W$. The weights in the adjacency matrix are set to 1 if two points are neighbored and 0 if not. It is argued that since neural networks are often patch based, they only have limited spatial information of the brain. Spectral Coordinates are isometry invariant, so they

do not change when the brain rotates or translates. The coordinates, given to the neural network, provide further regional context and improve the segmentation accuracy. In the paper the mean dice coefficient is improved from 0.819 using a state of the art method to 0.92 with DeepNat.
In "Fast Brain Matching with Spectral Correspondence" a method to align cortical surfaces based on spectral coordinates is introduced. First, the spectral coordinates are computed, whereby sign changes and the problem of algebraic multiplicity is handled. The weights are defined according to the distance of two vertices on the cortical surface.

$$W_{i,j} = \begin{cases} (\text{dist}(v_i, v_j) + \epsilon)^{-1} & \text{if } v_i \in N(v_j) \text{ and } i \neq j, \\ 0 & \text{else} \end{cases}$$

$\text{dist}(v_i, v_j)$ is a distance metric and ϵ a small value greater zero. The graph Laplacian used is given by

$$\begin{aligned} L =& D^{-1}(D - W) \\ =& I - D^{-1}W \end{aligned}$$

At last, a nonrigid transformation is used to align the spectral coordinates. The vertices are then matched with the nearest point of the spectral representation of the other brain.
The algorithm's results are similar to other state-of-the-art methods but a major time improvement is achieved.
The paper "Diffeomorphic Spectral Matching of Cortical Surfaces" also focuses on matching with spectral coordinates. In this case, the main outcome is not the decrease of computing power but an improvement of the quality of the matching. In order to match two brain surfaces an initial correspondence is applied. Therefore, the Laplacian graph includes both brains and connecting links from the initial correspondence. The weights are selected as before, and the graph Laplacian is defined as $L = D^{-1}(D - W)$. Then the eigenvectors of the Laplacian are computed, these have an entry for every vertex of both brains. The eigenvectors are split up and the spectral coordinates are assigned to each vertex of the graphs.
The difference to the matching algorithm from above is that multiplicity or sign ambiguity is no longer a problem. The eigenvectors of the two brain surfaces are computed from the same Laplacian.

The resulting spectral coordinates align perfectly. With these coordinates it is possible to define a diffeomorphic matching ψ between the surfaces. Let v_i be a vertex on the surface of the first brain S_1. The vertex on the second brain surface S_2 with the nearest spectral coordinates shall be called $\hat{v}_i$. The matching point $\psi(v_i)$ of v_i on S_2 is defined as

$$\psi(v_i) = \frac{\sum_{v_j \in N(\hat{v}_i)} w_{i,j} v_j}{\sum_{v_j \in N(\hat{v}_i)} w_{i,j}}$$

This defines a diffeomorphic matching between the brain surfaces S_1 and S_2. At last, the paper "Identifying Shared Brain Networks in Individuals by Decoupling Functional and Anatomical Variability" is discussed. Here fMRI's are used. Therefore, a vertex does no longer represent a voxel, but a whole time series. To define the adjacency matrix a correlation coefficient of two vertices $< v_i, v_j >$ is computed. Values below a threshold are set to zero. The weights are defined as

$$W_{i,j} = e^{-(<v_i, v_j>/\epsilon)}.$$

Finally the diffusion coordinates are computed. In the resulting diffusion space vertices have low euclidean distance, if their diffusion metric in the graph is high. The metric is defined as follows

$$D_t(v_j, v_i)^2 = \sum_{v_k \in V} |p^t_{j,k} - p^t_{i,k}|^2$$

Vertices that are close together in the diffusion space have a similar connection pattern in the graph. When the brain is analyzed one is interested in regions with similar connectivity, so clustering in the diffusion space is reasonable. This was done to define connectivity regions.

3.6 Discussion

In this chapter, the theory of spectral coordinates and diffusion maps has been discussed. The spectral coordinates are given as the coordinates of the eigenvectors of the graph Laplacian [10]. These vectors hold information about the structure of the graph e.g. the number of connected components. Therefore, they are used for partitioning graphs [58].

To compute the diffusion coordinates, the eigenvectors of the diffusion matrix are multiplied with their corresponding eigenvectors. The coordinates of the resulting vectors are the diffusion coordinates. They are closely linked to the diffusion metric, and therefore have a great significance for the structure of the graph [35].
Since these coordinates are so expressive, there is wide variety of applications. In this chapter deep learning, brain matching and analyzing connectivity patterns have been presented.

Chapter 4

State-of-the-Art

There are already highly developed programs in the field of segmentation, such as FreeSurfer [22] and SPM [6]. Since this thesis is focused on fetal brain segmentation, only a few outstanding examples of adult brain segmentation are discussed here. The goal of this section is to give an overview of related work that solves the problem of fetal brain segmentation. The papers presented all use deep learning. Nevertheless, there are significant differences. The most important characteristics of each work are highlighted. The papers shown have achieved excellent results and serve as inspiration for further work.

4.1 Brain Segmentation

4.1.1 Isointense Infant Brain MR Image Segmentation

Zeng and Zheng [65] introduce a 3-dimensional convolutional neural network which is built hierarchical and contains two U-Nets. Each network uses convolutional layers with a kernel size of $3\times3\times3$ and stride 1. Max pooling with kernel size of $2 \times 2 \times 2$ and stride 2 is applied. Finally, batch normalization and rectified linear units are used. Each U-Net contains an encoder and a decoder. A multimodal input is used, it consists of T1 and T2 MR images. Therefore, the first network contains two encoders, which are merged at the end and continue into a single decoder.
The hierarchical approach is used to obtain a probability map from the first U-Net. This is then computed into a distance map and given as input to-

gether with the original images to the second U-Net. It predicts the final segmentation.

4.1.2 Deep Convolutional Neural Network for Segmenting Neuroanatomy

Wachinger, Reuter, and Klein [59] also introduce a hierarchical approach for segmentation called DeepNat. Two 3-dimensional convolutional neural networks are used. First, the MR scan is segmented into foreground and background. In the second stage, 25 brain structures are identified. Both networks have an identical architecture. 3 convolutional layers are used. The kernel shapes are $7 \times 7 \times 7$, $5 \times 5 \times 5$ and $3 \times 3 \times 3$. Thus a reduction of the image size from $23 \times 23 \times 23$ to $3 \times 3 \times 3$ is achieved. ReLU as non-linear activation function, dropout and batch normalization is applied. The output and the spectral coordinates are concatenated to improve the spatial information. Fully connected layers are used before and after the merging. Finally, the segmentation is achieved.
Not only spectral coordinates are used. Experiments were also done with cartesian coordinates. The best results were achieved with a combination of both. The spectral coordinates are computed by constructing an adjacency matrix W. The entry $w_{i,j}$ is set to 1 if the voxels v_i and v_j are neighbored. Then the graph Laplacian is computed as follows

$$L = D - W$$

D is the diagonal matrix which has the sums of each row of W as entries. The i-th spectral coordinate is the vector containing the i-th index of the eigenvectors with the smallest eigenvalues of L.
After the neural networks a conditional random field (CRF) is applied to finalize the segmentation. DeepNat was tested on a public dataset and outperformed several state-of-the-art methods.

4.2 Fetal Brain Segmentation

4.2.1 A Hyper-Densely Connected CNN for Multi-Modal Image Segmentation

Dolz et al. [19] introduce a 3-dimensional fully connected convolutional neural network called "HyperDenseNet". The model is composed of 11 consecutive convolutional layers, the last 4 are fully convolutional. A stride of 1 and no pooling is used, so the input shape decreases from $25\times25\times25$ to $9\times9\times9$. The "HyperDenseNet" is, therefore, not a U-Net. The main difference between this approach and others is that each layer is not just connected to the previous layer but to all preceding layers. Let x_l be the l-th layer of the CNN. Typically, it is computed by a non-linear function H_l which takes the previous layer as input:

$$x_l = H_l(x_{l-1})$$

In this case, the non-linear function H_l does not just take the previous layer as input but all the preceding layers.

$$x_l = H_l(x_{l-1}, x_{l-2}, ..., x_1)$$

If a multimodal input, here T1 and T2 MR scans are used, each modality has it's own convolutional path. Each path is densely connected with not only itself but the other one as well. Let $x_1^1, ..., x_n^1$ be the layers of the first path and $x_1^2, ..., x_n^2$ the layers of the second. Again, the layers x_l^1 and x_l^2 are computed by non-linear functions H_l^1 and H_l^2 which take the previous layers as input.

$$x_l^1 = H_l^1(x_{l-1}^1, x_{l-1}^2, ..., x_1^1, x_1^2)$$
$$x_l^2 = H_l^2(x_{l-1}^2, x_{l-1}^1, ..., x_1^2, x_1^1)$$

The paths are merged at the end to generate the segmentation.
HyperDenseNet was evaluated on human and fetal brain and achieved great results in both cases.

4.2.2 Auto-Context Convolutional Neural Network

Salehi, Erdogmus, and Gholipour [52] propose two different convolutional neural networks for human brain extraction. The first approach is a voxelwise

CNN architecture. It is used to predict each voxel of a 3-dimensional MR scan separately. The second is a fully convolutional network with a U-Net architecture. Furthermore, an auto-context algorithm [57] is proposed and applied on the CNNs
The first network consists of 9 pathways. They are split up in 3 sagittal, 3 coronal and 3 axial ways. From each plane 3 different sizes 15×15, 25×25 and 51×51 are cropped and given to separate paths. In each pathway the input is convolved several times until 256 feature maps with size 1×1 remain. These are concatenated with the other pathways from the same plane. After applying 1×1 convolution, the feature maps are merged with the remaining pathways. Finally, 1×1 convolution and softmax is applied to obtain the final prediction.
Due to the patch size of 51×51 relatively large kernels with a size of up to 7×7 are used. Furthermore, each convolutional block consisted of a convolutional layer, a ReLU activation function and batch normalization.
In the second approach a U-Net architecture is used. In the encoder 3×3 convolution, a ReLU activation function and 2×2 max pooling with a stride of 2 is used. In the decoder 2×2 upsampling followed by two convolutional layers is applied. Skip connection are implemented to concatenate the feature maps of the decoder with the corresponding feature maps of the encoder. To generate the final segmentation 1×1 convolution is used.
In the auto-context algorithm a series of classifiers is designed and trained so that the probability map of a previous classifier is used as input for the next one. In this case, the CNN presented above is used as classifier. The probability map computed by the CNN with the softmax function can be written as follows:

$$p(y_i = l | X(N_i)) = \frac{e^{f_l}}{\sum_{j=1}^{c} e^{f_j(N_i)}}$$

Here X is an input image and $X(N_i)$ is the patch used to compute a prediction for the voxel y_i. c is the number of labels, $l = 1, ..., c$. f_l is the output of the CNN regarding the label l. The cost function is defined as

$$H = -\sum_i \log p(y_i = \text{true label} | X(N_i)) \tag{4.1}$$

Without the auto-context algorithm the segmentation is directly computed from the probability map $p(X)$. Now it is used as additional features for the next classifier. The concrete algorithm is given in 9

Algorithm 9: Auto-context algorithm as in [52]

```
Input: Training data set $\{(X_i, Y_i), i = 1, ..., n\}$ ;
Construct uniform distribution $p^0(X_i)$ ;
while $I > \epsilon$ do
    Set up training sets $\{((X_i, p^{t-1}(X_i)), Y_i)\}$;
    Train a CNN architecture described above;
    Compute probability maps $p^t(X_i), i = 1, ..., n$ ;
    Calculate $H_t$ using 4.1;
    $I = |H_{t-1} - H_t|$ ;
end
return Final probability maps $\{p^T(X_i), i = 1, ..., n\}$;
```

Salehi, Erdogmus, and Gholipour [52] prove in their paper that the auto-context algorithm converges. The training is stopped, as soon as the variation of the loss functions falls below a certain threshold ϵ.
The architectures are evaluated on on the basis of two publicly available data sets and surpass the latest deep learning methods.

4.2.3 Automatic Segmentation of the Intracranial Volume in Fetal MR Images

Khalili et al. [29] introduce a convolutional neural network for voxelwise segmentation of fetal brains. The architecture uses 2-dimensional MR scans as input. When predicting a label for a voxel, three differently sized patches are cut out and used to compute the label. The patches have the sizes 151×151, 101×101 and 51×51. These are give to three different pathways, each applying 3 strided convolutions followed by batch normalization. At the end of each pathways a fully connected layer is used. Then the three pathways are merged by concatenating the layers. This is followed by another fully connected layer, which produces the final prediction. After every fully connected layer dropout is applied.
To avoid bias, the minority class is oversampled, so the network is trained on an equal amount of positive and negative cases (brain and non-brain).

4.2.4 Fetal Cortical Plate Segmentation using 2D U-Net with Plane Aggregation

Hong et al. [24] introduce a 2-dimensional convolution neural network with a U-Net style architecture. The network is designed to segment the cortical plates and the corresponding internal region.
The encoder and decoder each consist of 4 blocks which are separated by max pooling or deconvolution. Each block consists of 4 convolutional layers with a kernel size of 3×3. These are followed by an ELU activation function and batch normalization. The final layer is a 1×1 convolution followed by a softmax function.
They argue that the dice coefficient as loss function is unfavorable regarding small structures, because small missclassifications lead to a large decrease in the score. Therefore, a different loss function is introduced, where the logarithm is applied on the dice score. It is defined as

$$L(I, \hat{I}) = -\frac{1}{N} \sum_{i=1}^{N} \ln(\text{Dice}(I_i, \hat{I}_i)^{0.3})$$

Here $I = I_1, ..., I_N$ are the true labels of the images, and $\hat{I} = \hat{I}_1, ..., \hat{I_N}$ are the corresponding predictions of the network.

4.2.5 Automatic Brain Tissue Segmentation in Fetal MRI using Convolutional Neural Networks

Khalili et al. [28] introduce a pipeline to segment fetal brains. The problem is split up in two subtasks. The first part is the identification of the fetal brain in the MR image, which is then automatically extracted from the image. During the second part, the brain tissue is segmented into seven classes. Both subtasks are accomplished by a convolutional neural network with a similar 2-dimensional U-Net structure.
The design of the networks used to identify the brain and to segment it is the same. The U-Net architecture consists of the contracting and an expanding path. The contracting part consists of 4 convolutional blocks in which 3×3 convolution and the non-linear activation function ReLU are applied. 2×2 max pooling is used to downsample the feature maps. In the expanding path up-sampling followed by 2×2 convolution is applied. It halves the number of channels and expands the feature maps. The results are concatenated with

the corresponding features from the contracting path and convolved with a kernel of size 3×3. Then ReLu is applied. In the final layer 1×1 convolution is used to obtain the final segmentation. Additionally, batch normalization is used after every convolutional layer.
Intensity inhomogeneity augmentation is applied to increase the robustness of the network towards these artifacts. They are simulated by a combination of linear gradients and random offsets.

$$Z = I \times ((X + x_0)^2) + (Y + y_0)^2)$$

I is the image before the artifacts are applied. X and Y are 2-dimensional matrices with integer values form 0 to the size of the image. x_0 and y_0 are the offsets. These are chosen randomly out of a certain range. The gradient pattern is randomly rotated between 0 and 360°. These inhomogeneity artifact make the network robust against regions with varying intensity.
The proposed method performed accurate segmentation. Furthermore, it was shown that the introduced method made the neural network robust against occurring artifacts.

4.2.6 A deep learning approach to segmentation of the developing cortex in fetal brain MRI with minimal manual labeling

Anonymous [4] trained a convolutionl neural network to segment fetal brains in two regions, namely cortex and non-cortex. The model used was earlier proposed by Kamnitsas et al. [27] with the name ”DeepMedic”. It was used for brain lesion detection.
To augment the data 11 different combinations of Gaussian distributions are used. The network was trained for 65 epochs and the learning rate was halved at predefined epochs.
The network architecture is built up of three pathways. Each one processing input at a different resolution. Pathways with higher resolution encode local, detailed information. In contrast, paths with lower resolution are more focused in contextual information such as the recognition of structures. The first path processes the regular image, the data for the second and third paths are downsampled by 3 and 5 times, respectively. Each path consists of 8 convolutional layers with a kernel of size 3×3. The output features of

the pathways are concatenated and processed by a final classification layer. The method produces good results. It should be emphasized that the segmentation of the network remained very stable over various stages of cortical development. It seems to be relatively age-invariant, since it was tested on fetal brains in a wide range of gestational age. The results show that the network is invariant regarding intensity changes around the brain. This is important due to the alternating structure of the developing brain.

4.2.7 Automated 2D Fetal Brain Segmentation of MR Images Using a Deep U-Net

Rampun et al. [47] introduce a network based on the U-Net. It takes 2-dimensional MR images as input and segments them (brain vs. non-brain). Although the architecture is very similar to the classic U-Net a few adaptions have been made. A U-Net has a decoder and an encoder, each one is built up of convolutional blocks. Instead of only using 4, this architecture uses 7 blocks. This enables the network to capture more coarse and finer contextual information.
The cross-entropy loss function is very popular in machine learning. It is improved by using a combination of a binary cross-entropy and dice coefficient as loss function. If I is the true label of the image, and $\hat{I}$ the prediction by the neural network, then the loss is given by

$$L(I,\hat{I}) = -\frac{1}{N}\sum_{n=1}^{N}(\frac{1}{2}I_n\hat{I}_n + \frac{2I_n\hat{I}_n}{I_n+\hat{I}_n})$$

The network employs an activation function called "ELU" [11], this guarantees better learning generalization. The function is defined a follows

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

To avoid overfitting batch normalization and dropout layers are added to the convolutional blocks.
The experimental results have shown that the proposed method has achieved significantly better results than the original U-Net and its variants. The networks was tested in different scenarios including low contrast, obscure brain regions, diffuse brain boundaries and overlapping boundaries. The proposed changes lead to competitive results across these various challenges.

4.3 Discussion

After discussing several papers, in the following the most important features are going to be elaborated. A special focus lies on characteristics which make the papers unique and outstanding compared to the broad field of brain imaging. Several of the features, which are discussed now, will be used in the next part of the thesis.
To improve the spatial information different pathways are used. These process several images either in varying resolution/size or from different axis (coronal, sagittal and transversal). Images with higher resolution/smaller size are used to process local and more detailed information. The pathways which are given lower resolution/ larger size identify main structures and generate more contextual information. When 2-dimensional input patches are used, the network lacks spatial information about the missing axis. Supplying it with patches of all 3 dimensions improves the final segmentation.
Most of the network architectures are designed as a U-Net. This structure has great success in image recognition and has, therefore, influenced the medical image analysis. The U-Net architecture is state-of-the-art and used in many popular papers.
In recent years spectral coordinates and diffusion maps have gained popularity. They are successfully applied in various applications such as deep learning and brain segmentation. They are invariant to rotation and translation. Therefore, they can improve the spatial information of the neural network. Especially in fetal brain segmentation, where brains are difficult to align due to fetal movement. In addition, neural networks predict often based on smaller patches, so a lot of global information is lost. Therefore, spectral coordinates can have a positive influence on the segmentation.
A hierarchical architecture is sometimes favorable compared to a single neural network. This gives the possibility of separate problems in simpler sub-tasks. The typical implementation consists of first recognizing the fetal brain in the original image, then cropping the image and processing it through a second U-Net to segment several brain regions. Often the hierarchical structure is used to compute further information such as distance maps or spectral coordinates.
Another interesting feature is the auto-context algorithm. Several networks are aligned in a series and the probability map generated by a previous network is given as input feature to the next one. This is repeated until the average loss converges. The auto-context algorithm can dramatically im-

proved the performance of a single network
In the papers discussed above, many different loss functions such as cross-entropy, dice score and logarithmic dice are presented. In deep learning loss function are a crucial part of success and have to be carefully chosen [26].
A lot of this ideas are used in the following chapter. Three 2-dimensional networks are going to be used as a composition, where each receives a dimension (coronal, sagittal and transversal) as input. These artificial neural networks will be designed as U-Nets. Furthermore, a hierarchical architecture will be applied, by putting two compositions in series. The first will identify the brain and the second segment several regions. This concept will be extended by the use of spectral coordinates. These are computed with the mask of the brain, and given as additional information to the next composition. Furthermore, several compositions will be put in a sequence to improve the mask. This is motivated by the auto-context algorithm.
These methods give very good results, but they are not perfect. Limitations are caused by the quality and quantity of the data. Deep learning methods depend on the data-sets which they are trained on. These problems will also influence the results of this thesis.

Chapter 5

Methodology

This work is devoted the segmentation of the fetal brain. Therefore, 3-dimensional MR images of fetal brains were collected and manually segmented. By performing and evaluating various experiments, a final model has been developed. The architecture of this artificial neural network and the design of the corresponding experiments are discussed in this chapter.

Preprocessing

To successfully work with deep learning methods, data must first be provided and processed. In this case 3-dimensional MR images of fetal brains are available. This data was processed in several ways. The movement of the fetus during the imaging often leads to corrupted images, this problem had to be corrected. Further was the maternal and non-brain tissue removed and the volume centered. The data is then split up in training and validation data. The first set is used for training, while the model is evaluated on the other one. This assures a independent testing procedure.

Segmentation with a U-Net Architecture

The method, which is the final result of several experiments, consists of 3 components. First, a sequence of U-Net compositions is the main structure of the model. Each composition is built up of 3 2-dimensional U-nets. 4 compositions are put in a sequence to identify the fetal brain. The mask is used to compute further information, which is then given, together with the

original image, to a final composition. The output is a segmentation of the brain in 7 regions. This architecture is explained in section 5.1.
The second component is the application of spectral coordinates. They are computed with masks, which are produced by the sequence of U-Net compositions. The mathematical theory is discussed in chapter 3. In section 5.2 the calculation is explained in detail. The third component is a loss function to improve topological consistency. Earlier models often had problems with segmenting the thin cortex correctly. The custom loss function penalizes holes in the cortex. It is presented in section 5.3.

Comparing alternative approaches

To get the architecture, which is described in the sections above, various experiments were done. First of all, naturally the question arises if a 2 or 3-dimensional should be used. A 2-dimensional input has the advantage of larger size, while one dimension remains unseen. This approach is used in the final model. The contrary applies for the 3-dimensional case, all dimensions are used, although the input size is smaller. With this approach the main model is compared. In section 5.4 an experiment set-up is explained, that analyzes this difference. In section 5.5 the influence the composition of several models is explored. If 2-dimensional U-Nets are used to the produce a segmentation, they lack the information of the third dimension. This disadvantage can be minimized by putting models in a sequence. In the main model a sequence of length 4 is used. With the experiment the choice of the length shall be confirmed. A hierarchical approach, as used in the main model, has the advantage that a task can be divided in smaller ones. This allows several models to by trained, each specializing on a single sub-task. This approach is compared to a single composition, which takes the MRIs as input and segments them immediately. The experiment to evaluate this procedure is described in section 5.6. Section 5.7 is dedicated to the exploitation of spectral coordinates. The theory is discussed in chapter 3. The question arises how spectral coordinates can best be integrated into an artificial neural network. The experiment, which is described here, compares the normal spectral coordinates, which are used in the main model, to re-scaled ones and not using them at all.
In the final chapter, the different evaluation metrics are discussed. Special focus lies on the dice coefficient, the mean surface area and the area of touch.

5.1 Cascaded U-Nets for 3D Segmentation

The model is based on several neural networks. The architecture consists of 3 parts. The first one is an U-Net structure, which segments foreground from background. In the second the spectral coordinates S_i are computed, and at last, an U-Net ensemble segments the image I. The first structure (figure 5.1, left) is given by 4 ensembles of neural networks. Each ensemble is made up of 3 2-dimensional U-Nets, which have been trained on coronal, sagittal and transversal slices. Each U-Net produces a probability map $P_i^{x,y}, i = 1, ..., \ x \in \{c, s, t\}, \ y \in m, s$. The letter c, s, t represent the slice (coronal, sagittal and transversal) that is used, m, s differs between the probability map of the mask or the segmentation of the 7 regions.The number i is the output of the i-th model. The output of the first ensemble $P_1^{c,m}, P_1^{s,m}, P_1^{t,m}$ is merged together by taking the average $A_1^m = \text{average}(P_1^{s,m}, P_1^{c,m}, P_1^{t,m})$. This result is concatenated with the original image (I, A_1^m) and used as input for the next ensemble. This procedure is repeated for all the following U-Net ensembles. After the 4th ensemble the resulting outputs A_4^m are used to compute the first 3 spectral coordinates S_1, S_2, S_3 of the obtained mask M. Finally, another ensemble of U-Nets (figure 5.1, right), which receives the original image, the mask and the spectral coordinates as input (I, M, S_1, S_2, S_3), computes the segmentation of the original image P.

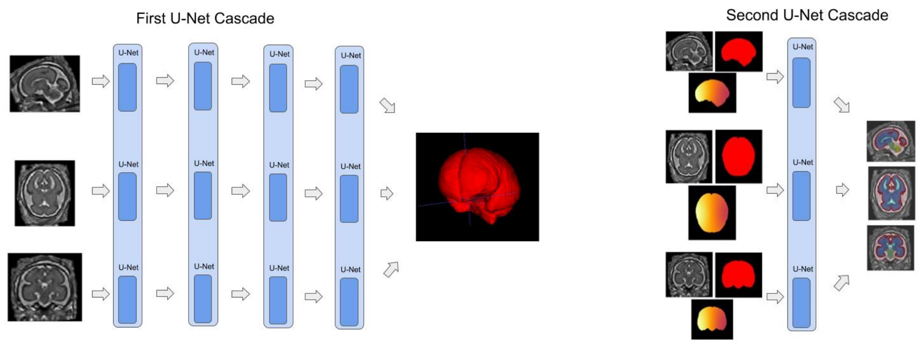

Figure 5.1: Left: First U-Net Cascade, which computes the masks. Right: Second U-Net Cascade, which segments 7 regions, with the use of spectral coordinates.

The hierarchical structure offers the possibility to solve simplified tasks at each level. First, the fetal brain is identified. Second, further information is computed and used to segment the brain into several regions. Therefore,

in each level a simple sub-task is solved, and the final segmentation is easier to achieve.
2-dimensional U-Nets are used to solve a 3-dimensional task. Each network has a lack of spatial information, this typically leads to problems. This issue has been resolved, by using coronal, sagittal and transversal planes and combining the predictions of the single U-Nets.
As proposed by the auto-context algorithm, several U-Net ensembles are put in series. Each ensemble uses the probability map generated by the previous one as additional feature map. Thus a single network can use 3-dimensional spatial information, although only a 2-dimensional input is used. A further advantage is that later networks can focus on improving the errors of previous ones. This results in an overall improved segmentation.
The U-Nets used for the identification of the brain and the segmentation of the different brain regions are built in a similar way. The only difference is the number of feature maps in the last layer. This is due to the different number of regions which the U-Nets identify.
As usual the U-Nets have an encoder and decoder, which are built up of 5 convolutional blocks. The input shape is 128×128. Therefore, the slices of the original MRI are cropped and only patches are given to the network. The number of feature maps in the first convolutional block is 16. This number doubles in every block, while the shape of the feature maps halves. Each block in the encoder uses three convolutional layers, which are followed by batch normalization and the activation function ReLU. Furthermore, the feature maps of the first and third convolutional layer are summed up. The downsampling between the blocks is performed by a strided convolution with stride 2.
The decoder also consists of 5 convolutional blocks. Unlike with the encoder, the number of feature maps halves, while the size doubles in each block. In each block 3 convolutions are applied, which are again followed by batch normalization and the ReLU activation function. The difference is that the feature maps of the first convolutional layer of a certain block in the decoder are summed up with the output of the corresponding block of the encoder. Between the blocks up-sampling is applied.
The output of the last three blocks of the decoder are merged together to receive the final segmentation. This is done by applying 1×1 convolution on each output. After convolving the third block it is up-sampled and summed up with the convolved output of the fourth block. The result is then up-sampled again, summed up with the last feature map of the fifth block and

finally softmax is applied to obtain the final result.

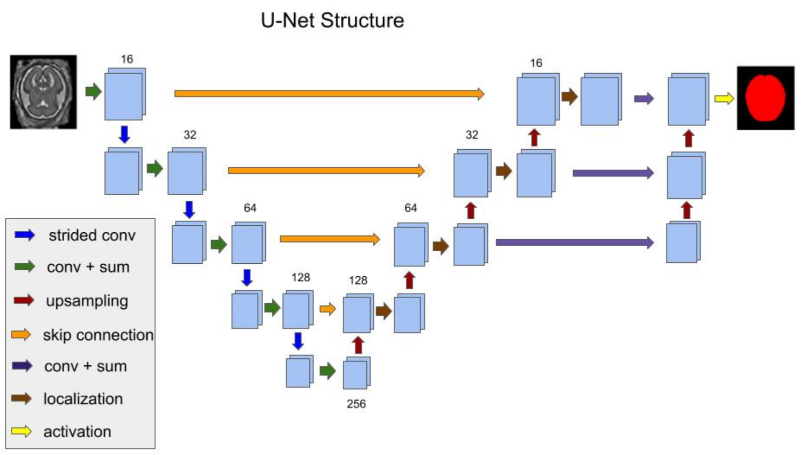

Figure 5.2: Architecture of a single U-Net

The U-Net structure is very popular and often applied for image recognition tasks such as in medical imaging. The network is inspired by this architecture, but some improvements have still been made. Several layers were added in the end to compute the segmentation. This allows the network to include information, which was gathered earlier, for the final result. Strided convolution is used instead of max-pooling. Therefore, not only maximum values are taken, but the image is further processed. Finally, the detailed design of the convolutional blocks has been modified. The most noticeable change is that the output of the first and third convolutional layer are added up. This allows the network to combine the initial image with the processed one, and therefore improve already processed results.

5.2 Spectral Coordinates

Spectral Coordinates have been used in the past to solve various problems such as shape description (Wachinger et al. [60]), brain matching (Lombaert, Sporring, and Siddiqi [38]) as well as for brain segmentation (Wachinger, Reuter, and Klein [59]). A patch-based approach with neural networks, especially a 2-dimensional model on 3-dimensional images, as it is proposed here, is limited by its spatial context. Since spectral coordinates have achieved

good results in the past, we suggest to use them here as well.
In section 5.1 a U-Net cascade is used to generate a mask of the fetal brain. With this mask the spectral coordinates are computed. First a graph G is created by setting each voxel of the mask as a vertex v_i in the graph. The vertices v_i, v_j of voxels, which are neighbored in the mask, are connected by an edge e_i, j in the graph. The generated graph in un-directed. Let further be V the set of vertices and E the set of edges.
For every two adjacent vertices v_i, v_j the corresponding edge $e_{i,j}$ is assigned the weight $\omega_{i,i} = 1$. The weights are represented as a matrix $W = (\omega_{i,j})_{i,j=1,...,n}$, the weight matrix. The weights of non-adjacent vertices is set to 0. The degree d_i of a vertex v_i is defined as

$$d_i = \sum_{j=1}^{n} \omega_{i,j}.$$

The degree matrix is defined as a diagonal matrix with the values $d_1, ..., d_n$ in the diagonal. Finally, the un-normalized graph Laplacian is given as

$$L = D - W.$$

In order to receive the spectral coordinates, the eigenvectors of the graph Laplacian are computed. The eigenvectors $u_1, ..., u_n$ and their corresponding eigenvalues $\lambda_1, ..., \lambda_n$ are ordered such that $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$. Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the first k eigenvectors $u_1, ..., u_k$ as columns. Then the vector $y_i \in \mathbb{R}^k$ corresponding to the i-th row in U is an approximation of the spectral coordinates of the corresponding i-th voxel.
If the graph G is connected, the smallest eigenvalue $\lambda_1 = 0$, and the corresponding eigenvector is the constant one vector (Von Luxburg [58]). Furthermore, if G is connected, then $\lambda_2 > 0$ (Chung and Graham [10]). The first three eigenvectors with eigenvalue > 0 are used.

5.3 Topological Loss Function

Due to its thin shape, the cortex is sometimes not continuously segmented by the network if standard loss functions such as categorical cross entropy (section 2.3) are used, i.e. it has holes. In order to solve this, a topological loss function is proposed.
The output shape of the model is $128 \times 128 \times 7$, since the brain is segmented in

7 regions. Due to the softmax function at the end of the U-Net, each of the 7 layers represent the probability of a certain region. The cortex separates two regions, namely liquor and white matter. The topological loss is computed in two steps. First, the layers, which correspond to the regions liquor p_l and white matter p_{wm}, are detached from the output P. Second, these layers are convoluted with a kernel, whose values are all set to 1. At last, the results are multiplied with the respective other layer and added up. The topological loss $L^t(P)$ is given as

$$\text{Loss} = p_{wm} * (p_l \odot \mathbb{1}) + p_l * (p_{wm} \odot \mathbb{1})$$

The output is added to the categorical cross entropy loss and returned as final loss $L^f = L^c + L^t$.

5.4 Construction of 2D and 3D ANNs

To determine whether a 2-dimensional or 3-dimensional input is more advantageous, different models with varying input size are compared. The input dimensions, which are used, are of size $32 \times 32 \times 32$, 64×64 and 128×128. In the main model, which is discussed above, an input size of 128×128 is used.
In section 5.1 the single U-Net structure is explained. It takes a 2-dimensional image with size 128×128 as input. It is processed by the network and the output is a segmentation of size 128×128. For this experiment the model is slightly modified. Instead of using a 2-dimensions, a 3-dimensional input is used.
The first and second models, which are tested, have a input size of $32 \times 32 \times 32$ and 64×64. The corresponding output sizes are $32 \times 32 \times 32 \times 7$ and $64 \times 64 \times 7$. Expect for the dimensionality everything remains unchanged compared to the model in section 5.1.
At last, a 2-dimensional model is tested. It is made of three different U-Nets. Each takes a 2-dimensional slice of size 128×128 as input. The U-Nets are trained separately on coronal, sagittal and transversal images. To predict a 3-dimensional MR image, every 2-dimensional slice is processed by the U-Nets separately and produces probability maps $P_1^{s,s}, P_1^{c,s}, P_1^{t,s}$. After each network has predicted the whole image, they are merged together by taking the average A_1^s of all three probability maps. A similar procedure is described in section 5.1.

5.5 The Design of Sequences of ANNs

In the previous section and in section 5.1 a composition of 2-dimensional models is used to predict a 3-dimensional image. The downside of this procedure is that each model lacks spatial information of the remaining dimension. In order to address this problem, several of these composition are put in series. By comparing the following models, the effect of a sequence of 1 to 5 U-Nets is analyzed. In the main model a sequence of length 4 is used.
The first part is a composition of three 2-dimensional U-Nets. Each U-Net is trained on coronal, sagittal or transversal slices of the original 3-dimensional MR image. The probability maps $P_1^{s,s}, P_1^{c,s}, P_1^{t,s}$ of the U-Nets is merged by taking the mean $A_1^s = \text{average}(P_1^{s,s}, P_1^{c,s}, P_1^{t,s})$.
For the second part a similar composition of three neural networks is used. Again, each takes as input coronal, sagittal or transversal slices of the original MR image. The difference is that this time the image slices are merged with the probability maps, which were produced by the first composition of U-Nets (I, A_1^s). So instead of $128 \times 128 \times 1$ the input size increases to $128 \times 128 \times 2$. The output of the three networks are again probability maps $P_2^{s,s}, P_2^{c,s}, P_2^{t,s}$ of the segmentation. The final result is again generated by taking the mean of the predictions $A_2^s = \text{average}(P_2^{s,s}, P_2^{c,s}, P_2^{t,s})$.
This process is repeated five times. In summary, five compositions of three U-Nets are connected into a sequence. Each of the compositions, except the first one, takes the original image concatenated with the probability map, which was predicted by the previous one, as input. This is used to generate another probability map, which is then passed on to the next part in the sequence.

5.6 Architecture of a Hierarchical Approach

In this section a method for an experiment is described which evaluates an hierarchical approach for segmentation, which is also used in the main model and described in section 5.1 . A similar idea was presented by Khalili et al. [28]. First a U-Net was used to identify the fetal brain, and then a second one segmented it. A difficult task was divided into several smaller ones. Therefore, the hierarchical approach is compared to the regular composition of U-Nets, which takes the MRI as input and segments seven regions is one step.

The first model is again the composition of three U-Nets, which is explained in detail in section 5.1 . Each U-Net takes a coronal, sagittal or transversal slice as input, and the outputs $P_1^{s,s}, P_1^{c,s}, P_1^{t,s}$ are merged together by taking the mean A_1^s. The input of the composition is the original 3-dimensional MR image, and the output is a segmentation into seven regions.
The second model is hierarchically designed. Two separate compositions of three U-Nets are used. The first composition takes the original MR image I as input and produces probability maps $P_1^{s,m}, P_1^{c,m}, P_1^{t,m}$ of foreground and background. A voxel is labeled as foreground if it belongs to one of the following regions: cortex, cerebellum, white matter, brainstem, ventricles, liquor. The mask M is generated my taking the maximum of the probability map A_1^m. It is concatenated with the original image (I, M) and passed on to the second composition. Then probability maps of the segmentation are generated $P_2^{s,s}, P_2^{c,s}, P_2^{t,s}$, which are merged together by computing the average A_2^s. Segmentation of the brain is then given by the maximum. This approach is also used in 5.1.

5.7 Exploiting Spectral Coordinates

Spectral coordinates are applied in several papers and improve the understanding of the structure of a graph. They have already been discussed in detail in chapter 3 and section 5.2. By using spectral coordinates the network receives additional spatial information. In this experiment three different ways to apply spectral coordinates are compared: None, regular coordinates S and re-scaled spectral coordinates. In the main model the un-scaled spectral coordinates are applied.
Variations of spectral coordinates are compared. Nevertheless, the main architecture remains unchanged. The model is built in a hierarchical way and consists of 3 parts: First, a composition of 3 U-Nets receives the original MR image I and produces a probability map A_1^m of foreground and background. Second, the mask M is used to compute the spectral coordinates S or its variations. The image I, the mask M of the first composition and the spectral coordinates S are merged together (I, M, S) and passed on to the second composition and used to segment the fetal brain into seven regions.
In the first configuration only the probability map, which was produced by the first composition, concatenated with the MRI is used as an input. In this case, no spectral coordinates are used. This setup is necessary to determine

the influence of spectral coordinates on the segmentation.
In the second case, the probability map is used to compute a mask of the brain. This is done by setting the value of a voxel to 1 if its probability is greater than 0.5, and to 0 if it is lower. The generated mask defines a graph: Every voxel is a vertex, and to neighboring vertices are connected by an edge. This graph is used to compute the spectral coordinates. The first three coordinates are concatenated with the MRI and the mask. This is passed on to the second composition.
The last configuration, that is investigated, is a re-scaling of the spectral coordinates. Normally, all eigenvectors are normalized. Therefore, the maximal values of the spectral coordinates depend on the size of the mask. Larger brains produce graphs with more vertices. The normalized vector has therefore smaller values. A younger fetus with a tinier brain leads to lager values in the spectral coordinates. Therefore, the eigenvectors are all re-scaled with the maximum lying between -1 and 1.

5.8 Evaluation Measures

The experiments are evaluated by several metrics. This gives the opportunity to analyze the results in more diverse ways. The most important metric, which is used, is the dice coefficient. It is computed for every region separately (background, cortex, cerebellum, white matter, brainstem, ventricles and liquor). This allows each region to be assessed individually and appropriate changes to be made.
Furthermore, it is also necessary to assess the segmentation as a whole. Therefore, the dice coefficient, accuracy, false positive and false negative rate of the entire image is computed. Let S and $\bar{S}$ be the manual and automatic segmentation of a fetal brain, then the dice coefficient [54] D is given by

$$D = 2\frac{S \cap \bar{S}}{|S| + |\bar{S}|}$$

For the other metrics the true positive TP, true negative TN, false positive FP and false negative FN labeled voxels are required. These values are used, if there are two regions, e.g. foreground and background. The true positive voxels are the labels where the model correctly predicted the positive class, in this case foreground. Similarly, the false positive voxels are mistakenly labeled as the positive class. The cases true negative and false negative are defined accordingly. Let further P, N be the total number of positive and total number of negative voxels. The positive voxels are given by $P = TP + FN$. Likewise, the negative voxels can be written as $N = FP + TN$. Then the accuracy ACC, the false positive rate FPR and the false negative rate FNR are defined as

$$ACC = \frac{TP + TN}{P + N}, \qquad FPR = \frac{FP}{N}, \qquad FNR = \frac{FN}{P}$$

Another metric that is used is the mean surface distance of the cortex. This region is particularly difficult to segment as it topologically resembles a thin line. The mean surface distance measures, broadly spoken, the deviation of the cortex of the prediction from the original image. In detail, only the voxels in the manual segmentation and the prediction, which are labeled as "cortex" and have a neighboring voxel of label "liquor", are considered. After removing the remaining voxels, the images are overlaid to compute the shortest distance from every predicted voxel to the nearest manually segmented voxel. The mean of all these distances results in the mean surface distance.

The metric described above only considers the outer surface of the cortex. To achieve a better understanding, another surface distance is computed. It includes the complete surface area of the cortex and is labeled as "whole cortex". Instead of only taking the voxels in the cortex, which are neighbored to liquor, all the voxels in the border of the cortex are included. In summary, every voxel, which is labeled as "cortex" and has a neighboring voxel with a different label, is used.

As discussed above, the cortex is fairly challenging to segment. Sometimes holes have appeared in the predicted cortex. Of course, this should not happen. To evaluate this mistake, the area of touch between the neighboring regions of the cortex is computed. Therefore, the number of voxels labeled as "white matter", which have a neighboring voxel named "liquor", is determined. This was done for the manually segmented C_m and the automatically segmented images C_a. Furthermore, the difference of these two values is computed. Due to the varying sizes of fetal brains, it is necessary to scale the obtained value. This is done by dividing it by the surface area of the cortex S_c.

$$\text{Relative area of touch} = \frac{C_a - C_m}{S_c}$$

5.9 Discussion

In this chapter the methodological approach is explained. A composition of several convolutional neural networks is used. First, foreground and background is segmented. In the second step, 7 brain regions are identified. Each U-Net takes 2-dimensional images, the results are put together to receive a 3-dimensional output.
Spectral coordinates are computed from the mask. These are passed on to later networks to improve their spatial information. A topological loss function is introduced, to reduce the holes of the segmentation of the cortex.
The set-up of several experiments is discussed. These results of these experiments will give a justification the architecture of the final model.

Chapter 6

Experiments and Results

In this chapter the experiments, which were done in the course of this work, are presented. The characteristics of the data are discussed in section 6.1. The following chapters deal with the design and evaluation of the various experiments.
The first experiments deal with the development of the final architecture. At the end, it will be tested and evaluated in combination with spectral coordinates and a custom loss function.

6.1 Data

The study includes 341 T2-weighted MR scans. The image size is $210 \times 238 \times 200$ and the voxel spacing is $0.5 \times 0.5 \times 0.5$ mm^3. 102 MR scans have been manually segmented into 7 regions, which are cortex, cerebellum, white matter, brainstem, ventricles, liquor and background. Out of all the 102 cases, 33 are diseased cases and the remaining are normal brains. The other 239 MR scans have been segmented in foreground, which includes all the labels for brain regions, and background. An overview is given in 6.1.

The images are already registered and centered by an in-house developed software. The MR scans were segmented in two stages. Before manually enhanced, some of the images were pre-segmented by an atlas-based software. Later, the remaining images were pre-segmented by an U-Net, which was trained on the first images, and then manually improved.
The first structure of U-Nets was trained on the 239 MR scans, which are only segmented in foreground an background. Subsequently, masks were predicted

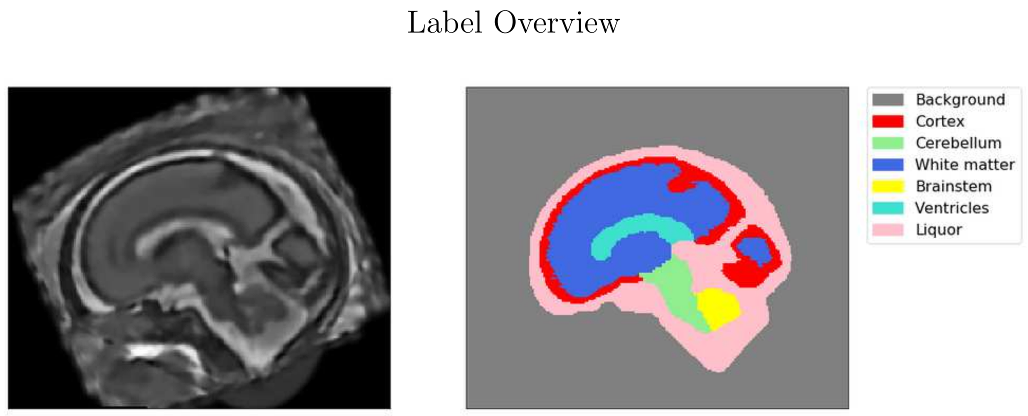

Figure 6.1: Overview of all segmented labels.

for the remaining 102 images and used to compute spectral coordinates. After being concatenated with the original images and predicted masks, these were passed on to the second U-Net structure. It was trained and validated by a 5-fold cross validation scheme of the 102 images.
In order to train the models, Adam ([31]) optimizer with an initial learning rate of $5 * 10^{-5}$, which dropped to half every 10 epochs, was applied. A patience of 15 epochs, and a dropout of 0.3 was used during the training.

6.2 Comparing 2D and 3D ANNs

In this section the results of the comparison of different dimensional inputs are presented and discussed. The method is described in detail in section 5.4. To evaluate the results the overall dice score (section 5.8) of all 7 region is measured.
The three models used input sizes $32 \times 32 \times 32$, $64 \times 64 \times 64$ and 128×128, respectively. The third model produces the best results, with a dice coefficient of 0.944 compared to 0.911 and 0.930 of the first and second one. The first model performs significantly worse than the second.
Due to the small input size of $32 \times 32 \times 32$ of the first model, the calculation time is reduced dramatically, and a fast training and prediction time is the result. A input of this size also has drawbacks. The small size leads to less spatial information, so the segmentation accuracy decreases. The dice coefficient is 0.911. Compared to the first model, the second network has signif-

icantly more spatial information. Therefore, the quality of the segmentation increases to a dice of 0.930. A disadvantage of this model is the increased training time. An eight-fold increase in input size leads to a considerable rise in computing effort. Therefore, the time needed for training and predicting increases a lot. The idea of the third construction is to further increase the input size from 32/64 to 128 without losing spatial information of the remaining dimension. By using 2-dimensional networks this was achieved. The computational resources needed to train the networks is less than the second model. Further, little spatial information was lost, as an U-Net was used for each dimension. All this led to a comparably good result and a dice coefficient of 0.944.

6.3 Comparing Different Sequences of ANNs

In this section the results of the experiments with a sequence of 1-5 models is discussed. This structure is used in the first cascade of the final model. The U-Nets are put in series, and the current model uses the predicted probability map of the previous one as input. The method is presented in detail in section 5.5. The evaluation is performed with the dice coefficient, which is discussed in section 5.8. It is measured in all 7 regions and compared to the other models. The data is presented in section 6.1.
When comparing the 5 different models, an increase in the dice score can be observed after each model in the sequence. The results are shown in table 6.1. Although an improvement takes place, it is not linear. There is a flattening after the fourth model. The fifth model does not lead to any noticeable improvement.

	Model 1	Model 2	Model 3	Model 4	Model 5
Dice	0.946	0.964	0.982	0.986	0.985

Table 6.1: Dice score of Sequence of models of length 5.

In each composition the predictions of each dimension are merged into a single output. Nevertheless, there is lack of spatial information. The advantage of the longer sequence is that the output of the previous composition is given as input to the next one. Therefore, each U-Net of the next composition uses spatial information produced by U-Nets that were trained on different dimensions. During the first models of the sequence the dice coefficient rises considerably, but after the fourth one a flattening can be seen. The reason for this is that the spatial information, given by the probability map of the previous composition, does not increase every time. At a certain point a saturation develops.

6.4 Evaluating a Hierarchical Approach

In this section the results of the experiment regarding the hierarchical approach are evaluated and discussed. Two models are compared. The first one is the regular U-Net composition. In the second one, the hierarchical approach is applied. The method is discussed in 5.6. The results are evaluated with the dice coefficient.
The hierarchical approach has clearly improved the segmentation and the dice coefficient has risen considerably from 0.9493 to 0.9788. Dividing the process into smaller tasks has the advantage that each network can focus on simpler problems and specialize in a certain task. Therefore, the hierarchical approach makes the overall problem easier to solve, this leads to an improved segmentation.

6.5 Comparing Different Ways of Exploiting Spectral Coordinates

In this section the experiment with different implementations of spectral coordinates is discussed. Therefore 3 variations are presented: None, regular spectral coordinates and re-scaled ones. The experiment set-up is presented in section 5.7. To evaluate the results the dice coefficient is used. The data is described in section 6.1.
When comparing the 3 models the dice coefficient has remained more or less constant and has sometimes fallen slightly. When examining the mean surface distance (m.s.d.), a clear drop can be seen when using spectral coordinates. Unfortunately, the re-scaling of the eigenvectors did not lead to a further improvement of the segmentation. The results are presented in table 6.2.

In summary, the dice of all three models was fairly equal. The reason for this is that the overall segmentation did not drastically alter. On the

	None	Regular	Re-scaled
Dice	0.9792	0.9716	0.9742
M.s.d	0.9133	0.9549	0.9465

Table 6.2: Dice score and mean surface distance of models with different applications of spectral coordinate

other hand, changes in the segmentation of the cortex are visible. These can be measured with the mean surface distance, which has decreased with use of spectral coordinates. Re-scaling the eigenvectors did not improve the segmentation, since the artificial neural network has already been able to use the regular coordinates efficiently.

6.6 Evaluation of Spectral Coordinates and Topological Loss

In this section the model which is explain in sections 5.1-5.3 is extensively tested. Therefore, the model from 5.1 is used. It is evaluated as a stand alone, with spectral coordinates and the topological loss function.
The results are evaluated with the dice score of each region, the accuracy, false positive and false negative rate. Additionally the mean surface distance of the whole and the outer cortex is computed. To assess the topological loss function the relative area of touch is computed. All these evaluation methods are presented in detail in section 5.8.
In all models a similar hierarchical structure is applied, which is explained in sections 5.1. They differ regarding the use of spectral coordinates (section 5.2) and the topological loss function (section 5.3). Concerning the first one, to which is referred as "Classic U-Net", it uses the first U-Net structure to generate masks, but does not compute spectral coordinates. The input to the second U-Net is the concatenation of the original image and the given mask. Moreover, it does not use the topological loss function. The second model which is called "Spectral Coordinates", computes the spectral coordinates. They are concatenated with both, the image and the mask, and passed on to the second structure. However, once again, the topological loss function is not in use. The third set-up is referred to as "Topological loss function". It is built in a similar way as the second one, but the custom loss function is

applied.

6.6.1 Result 1: Segmentation Accuracy

In table 6.3, the dice coefficients of all brain regions are given. The corresponding regions are illustrated in figure 6.1. Table 6.4 includes the overall dice coefficient, the accuracy as well as the false positive and false negative rate.
Except for the background the dice coefficient of each region ranges between 80-90%. The models perform best on the region liquor, and the cortex is the most challenging region. The dice coefficient is fairly constant in all three models. The region where the models have performed the most variously is the ventricles. Here a drop of 0.0054 occurred when using spectral coordinates. This has improved with the use of the custom loss function, but still a difference of 0.0031 remains. In table 6.4, the overall dice and accuracy declines by 0.0013 and 0.0003, while the false positive and false negative rate increases by up to 0.0002 and 0.0014.
In figure 6.2, a plot for each proposed model is presented, showing a box plot of every region. In the last figure a box plot is used to demonstrate the development of the dice coefficient of the cortex over all models.
In summary, the performance of the models decreases or remains fairly constant in these evaluations. Since the changes are small, this outcome motivates to take a closer look at the results. In further investigations, the mean surface distance as well as the relative area of touch will be discussed.

	Dice			
	Background	Cortex	Cerebellum	White matter
Classic U-Net	0.9954 (0.002)	0.8125 (0.075)	0.8601 (0.062)	0.9093 (0.048)
Spectral coordinates	0.9953 (0.003)	0.8076 (0.074)	0.8555 (0.056)	0.9077 (0.044)
Topological loss	0.9953 (0.003)	0.8099 (0.072)	0.8601 (0.052)	0.9097 (0.043)
	Brainstem	Ventricles	Liquor	
Classic U-Net	0.8534 (0.099)	0.8541 (0.065)	0.8879 (0.056)	
Spectral coordinates	0.8485 (0.11)	0.8487 (0.062)	0.8846 (0.055)	
Topological loss	0.8536 (0.109)	0.8510 (0.064)	0.8860 (0.056)	

Table 6.3: Dice score for automatic segmentation of fetal brains for different approaches and separate regions

	Accuracy (Freesurfer)			
	Dice	Accuracy	False positive	False negative
Classic U-Net	0.9625 (0.023)	0.9937 (0.005)	0.0039 (0.003)	0.0332 (0.037)
Spectral coordinates	0.9612 (0.024)	0.9934 (0.004)	0.0041 (0.003)	0.0344 (0.038)
Topological loss	0.9614 (0.025)	0.9935 (0.003)	0.0040 (0.003)	0.0346 (0.040)

Table 6.4: Dice score, accuracy, false positive rate and false negative rate for automatic segmentation of fetal brains. The region measured was cortex + white matter + ventricles + brainstem + cerebellum vs. background + liquor

Dice Score of all regions produces be the 3 models

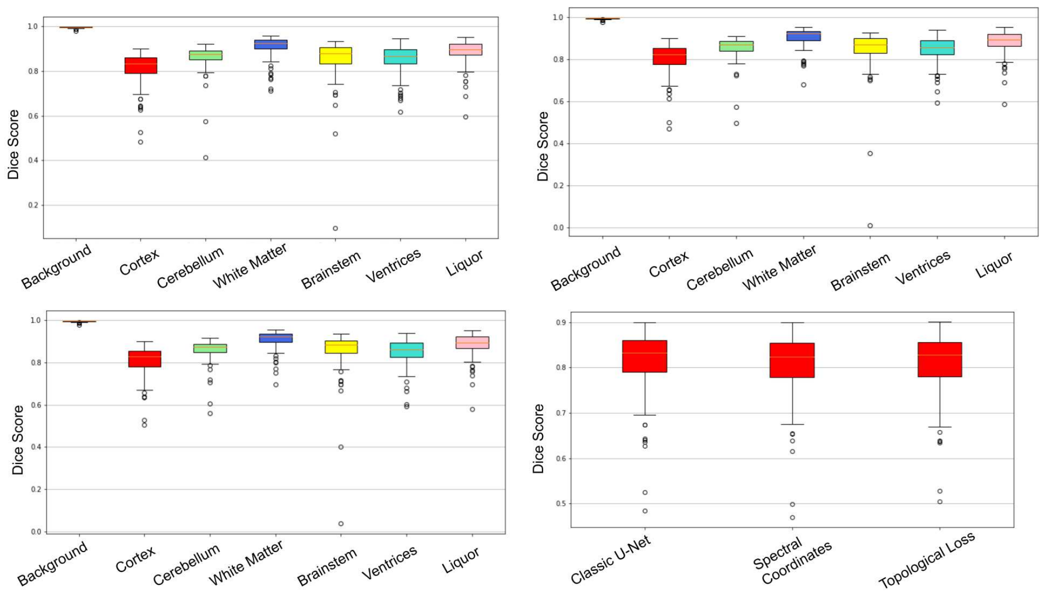

Figure 6.2: First three plots are the dice score of each brain region predicted with all three architectures: Classic U-Net is upper left, spectral coordinates upper right, topological loss is bottom left. The dice score of just the cortex of all architectures is shown at the bottom left.

Table 6.5 shows the surface distance of all three models. The first results present the surface distance of the outer cortex, while the second show the surface distance of the whole cortex. Each of the proposed models improves the mean and standard deviation compared to the previous one. The performance of the mean surface distance on the outer cortex improves from 0.8595 to 0.8311. On the whole cortex the mean decreases from 0.9205 to 0.8590. In figure 6.3, a box plot of the mean surface distances of all segmentations is shown.

	Mean surface distance		M.s.d. (whole cortex)	
	Mean	Standard deviation	Mean	Standard deviation
Classic U-Net	0.8595	1.3452	0.9205	1.3289
Spectral coordinates	0.8504	1.2830	0.8725	1.1622
Topological loss	0.8311	1.2169	0.8590	1.1200

Table 6.5: Surface error for ground truth vs. automatic segmentation for all three architectures.

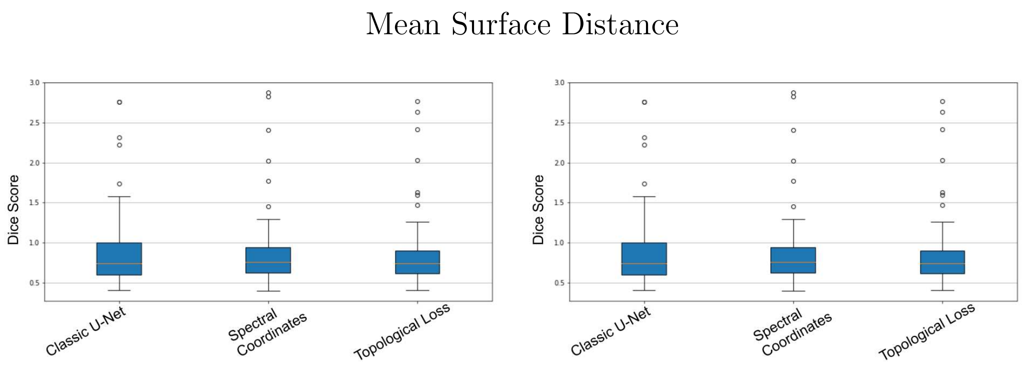

Figure 6.3: Surface distance predicted with all three architectures

Figure 6.4 shows the diversity of the network's predictions. Examples of good, bad and average results are presented. Each column shows a sagittal and coronal slice of a brain which has been segmented by the proposed model. It uses spectral coordinates and the topological loss. The images are sorted by dice. The two segmentations on the left have a good dice score, while the images in the middle show an average and those on the right a bad dice score. One problem of the model is to display the continuity of the cortex, this can be seen in the middle images. The models performed worst when the data was corrupted or the fetuses were very young. This is shown in the right images.

The performance of the proposed model is compared to a network by Hong et al. [24]. It is referred to as "CPS Net". This model was trained

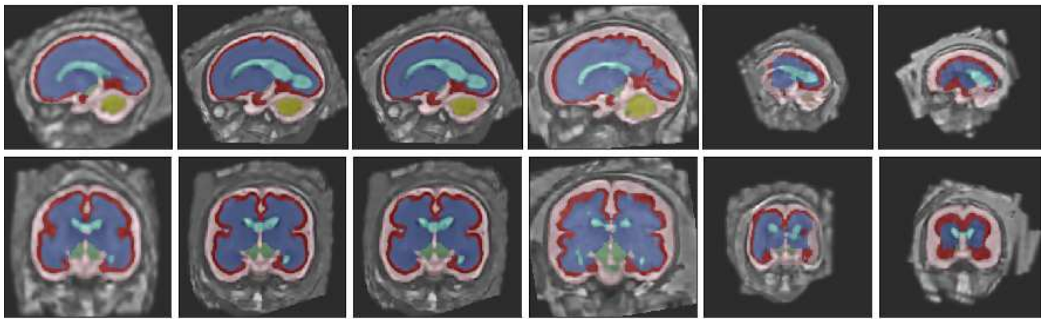
Figure 6.4: Figure: Sagittal/Coronal slices: 2 good cases, 2 average cases, 2 bad cases (Dice wise)

to identify the cortex and the internal region of the cortex. The internal region corresponds to the regions white matter and ventricles. In table 6.6.1, the dice score of each region is given. The model proposed in this paper performed better by more than 18% in the cortex and more than 10 % in the internal region.

	Dice Score		
	Background	Cortex	Internal region of cortex
CPS Net	0.9879	0.6279	0.8242
Topological loss	0.9967	0.8099	0.9293

6.6.2 Result 2: Topological Consistency of Annotations

As shown in table 6.6, the area of touch is reduced by both models. Using spectral coordinates has decreased the area of touch from 0.0459 down to 0.04402. The proposed topological loss function could reduce the area of touch further down to 0.0347.
The columns in figure 6.6 titled C_a/S_c and C_m/S_c show the area of white matter, which is neighbored to liquor voxels, divided by the outer surface area of the cortex. The first column represents the prediction and the second the manually segmented mask. While the latter is obviously the same in all cases, there is a reduction in the first column.

	Relative area of touch			
	Mean	Standard deviation	C_a/S_c	C_m/S_c
Classic U-Net	0.0459	0.0390	0.1083	0.0715
Spectral Coordinates	0.04402	0.0363	0.1080	0.0715
Topological loss	0.0412	0.0347	0.1030	0.0715

Table 6.6: Area of touch between Brian and Liquor. 3 different architectures were evaluated

In figure 6.5, images of segmented brains are shown. Both the two left and the two right pictures show the same brain. The first and third image from the left show brains which are segmented without the proposed loss function. In the second and fourth image the topological loss function is applied. The red area represent the cortex and the blue area the white matter, which lies underneath and should not be visible. As the model has difficulties with the segmentation of the cortex and sometimes creates holes, the topological loss function was introduced. Due to its use, the holes have been shrunk and closed. Therefore, the segmentation of the cortex has been improved.

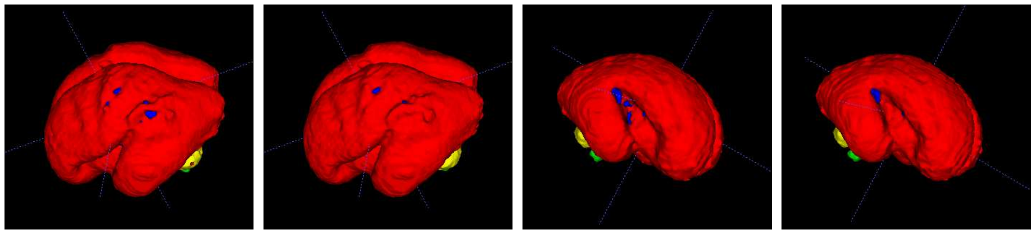

Figure 6.5: 3D rendering of 2 example cases, demonstrating how a hole is closed by the topological loss

6.7 Replication on Diseased Cases

It is further examined whether the results are replicable for diseased cases. Therefore, the second U-Net structure is retrained on only the alive cases and the dice for each region is evaluated. Furthermore, the mean surface distance is computed on a validation set of alive cases, which were not used for training, as well as the diseased cases. The results are provided in table 6.7.

The dice score drastically reduces in all regions, especially in the ventricles. More precisely, the dice score reduces from 84.8% to nearly half to about 48.74%. Concerning the other regions, the dice score only reduces by up to 10%. Furthermore, there is a major increase in the mean surface distance for the outer and the whole cortex. Regarding the outer cortex, the score nearly doubles and increases from 0.7221 to 1.4059. The mean surface distance of the whole cortex rises from 0.8598 to 1.4550. There is a clear increase of the standard deviation in both cases.

	Dice			
	Background	Cortex	Cerebellum	White matter
Regular	0.9956	0.8185	0.8627	0.9088
Diseased	0.9948	0.7220	0.7471	0.8339
	Dice			
	Brainstem	Ventricles	Liquor	
Regular	0.8555	0.8480	0.9007	
Diseased	0.8284	0.4874	0.8059	
	Mean surface distance		M.s.d. (whole cortex)	
	Mean	SD	Mean	SD
Regular	0.7221	1.4814	0.8598	1.1025
Diseased	1.4059	1.8258	1.4550	1.8584

Table 6.7: Replication on diseased cases: Dice score and surface error for ground truth vs. automatic segmentation for all compartments and different approaches

6.8 Discussion

In this chapter several experiments have been discussed, which evaluate the segmentation performance of neural networks on MR images of fetal brains. The effect of several techniques, such as sequencing neural networks, hierarchical structures, spectral coordinates and topological loss functions on the segmentation has been evaluated and the results have been discussed. These were used to develop a neural network, which delivers solid results.

Chapter 7

Conclusion

In this chapter the key points of this thesis are revisited and the most relevant results are highlighted. Finally, a short overview of possible further work is given.

7.1 Summary

Several things stand in the way of establishing an automatic segmentation of fetal brains. First, the quantity of fetal and healthy MRI data is very limited, compared to adults, and the manual segmentation is not only time consuming but also very complex. Therefore, extending the database at the beginning was necessary. Second, fetal brains are much more diverse than adult ones. Therefore, intense experiments and advanced techniques were necessary to establish an automatic segmentation.
In detail, the main results were the following: It was shown that a composition of 2-dimensional U-nets delivered better results than a single 3-dimensional one. A hierarchical approach allowed to separate the segmentation issue in several sub-tasks, which were easier to solve. Compared to an approach, where the segmentation is created from a single composition, the hierarchical model provided better results. The influence of spectral coordinates and a custom topological loss function were evaluated, and it was shown that the quality of the segmentation was increased.
All these experiments have led to a final architecture, which is used for automatic segmentation of fetal brains.

7.2 Future Work

In this thesis a relatively large amount of data was used to train the neural networks. Nevertheless, this is still a tiny quantity compared to other, especially non-medical, applications [37] [34]. In order to further improve the automatic segmentation, gathering and manually segmenting new data will be necessary.
A functioning automated segmentation process gives rise to the broad field of diagnosing brain related diseases [61]. Neural networks could be applied to segment and then classify brain MRIs accordingly and can assist physicians with challenging diagnoses. This opens the door to more medical research.

Bibliography

[1] Oludare Isaac Abiodun et al. "Comprehensive review of artificial neural network applications to pattern recognition". In: *IEEE Access* 7 (2019), pp. 158820–158846.

[2] *Accuracy and precision*. Oct. 2020. URL: `https://en.wikipedia.org/wiki/Accuracy_and_precision`.

[3] Michael Aertsen et al. "Fetal MRI for dummies: what the fetal medicine specialist should know about acquisitions and sequences". In: *Prenatal Diagnosis* 40.1 (2020), pp. 6–17.

[4] Anonymous. "A deep learning approach to segmentation of the developing cortex in fetal brain {MRI} with minimal manual labeling". In: *Submitted to Medical Imaging with Deep Learning*. under review. 2020. URL: `https://openreview.net/forum?id=VtVIlHSc0`.

[5] Robert B Ash. *Real Analysis and Probability: Probability and Mathematical Statistics: a Series of Monographs and Textbooks*. Academic press, 2014.

[6] John Ashburner and Karl J Friston. "Unified segmentation". In: *Neuroimage* 26.3 (2005), pp. 839–851.

[7] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. "Advances in optimizing recurrent networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 8624–8628.

[8] Leon Brown, B Schreiber, and B Alan Taylor. "Spectral synthesis and the Pompeiu problem". In: *Annales de l'institut Fourier*. Vol. 23. 3. 1973, pp. 125–154.

[9] Francois Chollet. *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

[10] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. 92. American Mathematical Soc., 1997.

[11] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[12] *Cross entropy*. Oct. 2020. URL: `https://en.wikipedia.org/wiki/Cross_entropy`.

[13] *CS231n Convolutional Neural Networks for Visual Recognition*. Sept. 2020. URL: `https://cs231n.github.io/neural-networks-1/`.

[14] *CS231n Convolutional Neural Networks for Visual Recognition*. Oct. 2020. URL: `https://cs231n.github.io/neural-networks-3/`.

[15] *CS231n Convolutional Neural Networks for Visual Recognition*. Oct. 2020. URL: `https://cs231n.github.io/convolutional-networks/`.

[16] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[17] J De la Porte et al. "An introduction to diffusion maps". In: *Proceedings of the 19th Symposium of the Pattern Recognition Association of South Africa (PRASA 2008), Cape Town, South Africa*. 2008, pp. 15–25.

[18] Ivana Despotović, Bart Goossens, and Wilfried Philips. "MRI segmentation of the human brain: challenges, methods, and applications". In: *Computational and mathematical methods in medicine* 2015 (2015).

[19] Jose Dolz et al. "HyperDense-Net: a hyper-densely connected CNN for multi-modal image segmentation". In: *IEEE transactions on medical imaging* 38.5 (2018), pp. 1116–1126.

[20] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of machine learning research* 12.Jul (2011), pp. 2121–2159.

[21] *Feedforward neural network*. Sept. 2020. URL: `https://en.wikipedia.org/wiki/Feedforward_neural_network`.

[22] Bruce Fischl. "FreeSurfer". In: *Neuroimage* 62.2 (2012), pp. 774–781.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[24] Jinwoo Hong et al. "Fetal Cortical Plate Segmentation using 2D U-Net with Plane Aggregation". In: ().

[25] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[26] Katarzyna Janocha and Wojciech Marian Czarnecki. "On loss functions for deep neural networks in classification". In: *arXiv preprint arXiv:1702.05659* (2017).

[27] Konstantinos Kamnitsas et al. "Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation". In: *Medical image analysis* 36 (2017), pp. 61–78.

[28] Nadieh Khalili et al. "Automatic brain tissue segmentation in fetal MRI using convolutional neural networks". In: *Magnetic resonance imaging* 64 (2019), pp. 77–89.

[29] Nadieh Khalili et al. "Automatic segmentation of the intracranial volume in fetal MR images". In: *Fetal, Infant and Ophthalmic Medical Image Analysis*. Springer, 2017, pp. 42–51.

[30] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[31] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[32] Günter Klambauer et al. "Self-normalizing neural networks". In: *Advances in neural information processing systems*. 2017, pp. 971–980.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[34] Alina Kuznetsova et al. "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale". In: *arXiv preprint arXiv:1811.00982* (2018).

[35] Stéphane S Lafon. "Diffusion maps and geometric harmonics." In: (2004).

[36] Georg Langs et al. “Identifying shared brain networks in individuals by decoupling functional and anatomical variability”. In: *Cerebral Cortex* 26.10 (2016), pp. 4004–4014.

[37] Yann LeCun, Corinna Cortes, and CJ Burges. “MNIST handwritten digit database”. In: (2010).

[38] Herve Lombaert, Jon Sporring, and Kaleem Siddiqi. “Diffeomorphic spectral matching of cortical surfaces”. In: *International Conference on Information Processing in Medical Imaging*. Springer. 2013, pp. 376–389.

[39] Herve Lombaert et al. “Fast brain matching with spectral correspondence”. In: *Biennial International Conference on Information Processing in Medical Imaging*. Springer. 2011, pp. 660–673.

[40] Alexander Selvikvåg Lundervold and Arvid Lundervold. “An overview of deep learning in medical imaging focusing on MRI”. In: *Zeitschrift für Medizinische Physik* 29.2 (2019), pp. 102–127.

[41] Antonios Makropoulos, Serena J Counsell, and Daniel Rueckert. “A review on automatic fetal and neonatal brain MRI segmentation”. In: *NeuroImage* 170 (2018), pp. 231–248.

[42] Boaz Nadler et al. “Diffusion maps, spectral clustering and eigenfunctions of Fokker-Planck operators”. In: *Advances in neural information processing systems*. 2006, pp. 955–962.

[43] *Neuron*. Oct. 2020. URL: `https://en.wikipedia.org/wiki/Neuron`.

[44] Andrew Y Ng. “Feature selection, L 1 vs. L 2 regularization, and rotational invariance”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 78.

[45] Chigozie Nwankpa et al. “Activation functions: Comparison of trends in practice and research for deep learning”. In: *arXiv preprint arXiv:1811.03378* (2018).

[46] G Palm. “On representation and approximation of nonlinear systems”. In: *Biological Cybernetics* 34.1 (1979), pp. 49–52.

[47] Andrik Rampun et al. “Automated 2D Fetal Brain Segmentation of MR Images Using a Deep U-Net”. In: *Asian Conference on Pattern Recognition*. Springer. 2019, pp. 373–386.

[48] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[49] Walter Rudin. *Functional analysis*. 1973.

[50] Walter Rudin. *Real and complex analysis*. Tata McGraw-hill education, 2006.

[51] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[52] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. “Auto-context convolutional neural network (auto-net) for brain extraction in magnetic resonance imaging”. In: *IEEE transactions on medical imaging* 36.11 (2017), pp. 2319–2330.

[53] Hidetoshi Shimodaira. “Improving predictive inference under covariate shift by weighting the log-likelihood function”. In: *Journal of statistical planning and inference* 90.2 (2000), pp. 227–244.

[54] *Sørensen–Dice coefficient*. Oct. 2020. URL: `https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient`.

[55] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, Ontario, Canada, 2013.

[56] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[57] Zhuowen Tu and Xiang Bai. “Auto-context and its application to high-level vision tasks and 3d brain image segmentation”. In: *IEEE transactions on pattern analysis and machine intelligence* 32.10 (2009), pp. 1744–1757.

[58] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[59] Christian Wachinger, Martin Reuter, and Tassilo Klein. “DeepNAT: Deep convolutional neural network for segmenting neuroanatomy”. In: *NeuroImage* 170 (2018), pp. 434–445.

[60] Christian Wachinger et al. “BrainPrint: A discriminative characterization of brain morphology”. In: *NeuroImage* 109 (2015), pp. 232–248.

[61] Christian Weisstanner et al. “MRI of the Fetal Brain”. In: *Clinical neuroradiology* 25.2 (2015), pp. 189–196.

[62] Bing Xu et al. “Empirical Evaluation of Rectified Activations in Convolutional Network”. In: *CoRR* abs/1505.00853 (2015). arXiv: `1505.00853`. URL: `http://arxiv.org/abs/1505.00853`.

[63] Fisher Yu and Vladlen Koltun. “Multi-scale context aggregation by dilated convolutions”. In: *arXiv preprint arXiv:1511.07122* (2015).

[64] Matthew D Zeiler et al. “Deconvolutional networks”. In: *2010 IEEE Computer Society Conference on computer vision and pattern recognition*. IEEE. 2010, pp. 2528–2535.

[65] Guodong Zeng and Guoyan Zheng. “Multi-stream 3D FCN with multi-scale deep supervision for multi-modality isointense infant brain MR image segmentation”. In: *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. IEEE. 2018, pp. 136–140.