Segmentation in Skeletal Scintigraphy Images using Convolutional Neural Networks

LUND
UNIVERSITY

Konrad Gjertsson

Faculty of Engineering
Lund University

This dissertation is submitted for the degree of
Master of Science in Engineering, Computer Science and Engineering

Exini Diagnostics AB

June 2017

Acknowledgements

I would like to thank my supervisors Prof. Kalle Åström and Dr. Karl Sjöstrand for their continuous guidance and support throughout the project.

I would also like to thank Jens Richter and Dr. Kerstin Johnsson for their aid and collaboration with me during my thesis project, as well as Prof. Mattias Ohlsson for allowing me to participate in his much appreciated and informative deep learning journal club.

Furthermore, I would like to thank Exini Diagnostics AB and Progenics Pharmaceuticals Inc. for allowing me to use their resources when training the models constructed in the context of this project.

Abstract

In this work we have addressed the task of segmentation in skeletal scintigraphy images with deep learning models, where we research different approaches to convert convolutional neural networks designed for classification tasks to powerful pixel wise predictors. We explore different network architectures where two primary research paths have been followed. Firstly, an encoder-decoder architecture which aims to extract dense features in the first part of the network – which works as a feature encoder – and then up-sample these dense features to restore the original image resolution and perform pixel-wise predictions. This technique has shown great promise in other experiments of segmentation in medical images. This general architecture has been pitted against an entirely different approach, which works with expansions of the convolutional kernels, rather than sub-sampling through pooling layers, known as convolutions "atrous" or dilated convolutional kernels. While the atrous approach has been explored in different studies for the problem of semantic image segmentation for outdoor and indoor scenes with a large amount of classes it has yet to be tried in the medical imaging field. When compared to the encoder-decoder architecture we see that the convolutional neural networks atrous outperform the former in almost every way. We observe that the most promising atrous model generated a test error of 0.0659 on segmentations on the left scapula, which is a reduction of 50.67% on the test error as compared to the most powerful encoder-decoder model. The atrous model also managed to reduce the amount of parameters by a factor 100 and more than halve the required training time per epoch.

Table of contents

Chapter 1

Segmentation and Medical Image Analysis

Segmentation in medical images has been an ongoing field of research for a long period of time due to its high relevance and significance in clinical applications, which amongst other things, serve to provide physicians with solid, quantifiable bases for their judgments with regards to patient health and status of disease. Nuclear images in the medical field are often blurry and hard to decipher, even for human experts who have poor interobserver agreement when it comes to decisions based on medical imaging, where a quantifiable basis for the decisions can help to greatly decrease this variability [2]. This fact clearly illustrates the need for objective, reproducible and quantifiable analysis and measurements based on the data contained in medical images, to be used as a basis for judgments taken by physicians.

This master thesis has been done in collaboration with Exini Diagnostics AB, Lund and Progenics Pharmaceuticals Inc., New York. Exini is a medtech company which has a long history of segmentation based products for medical images. Their initial product focused on the heart and measures the blood perfusion in the left ventricle muscle. The initial segmentation technique was a simple method which analyzed the uptake of each slice in the image stack (3D-images are stored as stacks of 2D-images, also referred to as slices) where it tried to sample after the maximum uptake to generate polar plots, which only gave a rough estimation of the heart and was quite susceptible to noise from surrounding organs, which would also show up in the images. The importance of accurate and stable segmentations quickly became evident to be able to give indications of the health of the studied patients, which lead to the implementation of more advanced segmentation techniques such as active shape models [8, 30]. Similar techniques were, in addition to the region around the heart, employed to perform segmentations of the blood vessels in the cortex, where direct correlations could be seen between the size of the vessels and Alzheimer's disease. More

recently the focus at Exini has been on bone segmentation, and they have developed an automated measuring index for the state of metastases in the skeletal structure known as Bone Scan Index (BSI) [10]. BSI is derived from the ratio between the whole skeletal mass and the mass of the metastases, which is useful for measuring the progression of patients' metastatic disease over time. To calculate the index it is necessary to perform a whole body segmentation on the entire skeletal structure and compare its mass to that of the metastases. To this end it is essential to have an accurate segmentation of the entire skeletal structure as well as the metastases themselves. Traditionally, image registration with the Morphon method [47] has been used to perform the full body segmentations where each pixel is labeled as either a part of the skeleton or the background. More recent research indicate that deep learning and especially convolutional neural networks might be a better way to proceed and succeed where traditional methods have fallen short. Future research projects for Exini Diagnostics lies in the domain of 3D-imaging, and for such problems it is believed that deep learning will provide a powerful way to perform accurate segmentations and perform better than the traditional techniques today applied in 2D images. The aim of this thesis is to validate deep learning as a method for image segmentation of medical data, and more specifically the whole body segmentations of the skeleton utilized in the bone scan index, previously performed with the Morphon method. The goal is to recreate the 2D segmentation created by the Morphon method in skeletal scintigraphy images, as will be described in detail in Chapter 5. If proven successful, the herein performed research will serve as a stepping stone from where further experiments with data in both two and three dimensions can be performed.

The work in this thesis primarily enhances the structures described in [31, 3, 4, 49] with the ideas and building blocks introduced in [39, 16, 7, 17, 44], to take the models to new heights and customized them for the specific problem domain.

Chapter 2

Machine Learning

The ideas and concepts of beings such as artificially intelligent entities and thinking machines date back as far as to the 10th century BC, where ancient Chinese legends spoke of humanoid automatons in passages from Lie Zi's accounts [27]. China was however, far from the only ancient civilization who spoke of artificial agents, a concept which also appears in the legends of, amongst others: Jewish, Norse, Egyptian and Greek culture [14]. While the concept of artificially constructed intelligent agents itself may be thousands of years old, the modern field of science known as Artificial Intelligence had its birth as late as in the early 1950s with Alan Turing, who posed the now well-recognized question: 'Can machines think?' [43].

Since the time of Alan Turing, theoretical breakthroughs and advances in computational resources have taken the field of artificial intelligence far from its origins. The research that has since been conducted has led to the introduction of novel interdisciplinary research areas which have branched out from the traditional field of artificial intelligence: one of which goes by the name of machine learning. Machine learning has its roots in artificial intelligence together with pattern recognition and computational learning theory and is today considered to be an interdisciplinary field between computer science and mathematical statistics. Machine learning, as described by Arthur L. Samuel, can be said to be the art of giving computers the ability to perform certain tasks without explicitly being programmed to do so [33]. A key difference between artificial intelligence and machine learning is that in the latter, the learning is operational rather than cognition based. This means that, when conducting research in machine learning the goal is not necessarily the construction of artificial, intelligent agents but may instead be more localized tasks such as pattern recognition, classification, clustering, segmentation etc. A formal definition of the modern field of machine learning can be given by quoting Tom M. Mitchell [26]:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.

The history of machine learning, from its birth in the early 1950s to today has been, as with most scientific areas, a bumpy road. Many machine learning algorithms depend on access to large quantities of data to be an attractive option to conventional methods, or to even work at all. Historically this has been one of the major hurdles for machine learning models, but the explosion of data in recent years together with theoretical breakthroughs and a significant increase in the computational ability available to researchers have made machine learning alternatives preferable to traditional approaches for a wide range of problems. As the accessibility to and speed of the Internet have increased, the possibility to acquire data has improved enormously over the last few years. The world's total capacity to store data was in 1986 limited to 2.6 exabytes (one exabyte equals 10^{18} bytes), while in 2007 it was estimated to have increase to as much as 295 exabytes. We see no end to this exponential increase in data in the near future, as more and more types of applications are connected to the Internet, able to generate and process even more data. In 2013, the digital universe (the total amount of data generated or copied *annually*) was calculated to be 4.4 zetabytes (one zetabyte is equal to 10^{21} bytes). During the next decade the digital universe is predicted to increase by 40% every year to a point where we have a digital universe of 44 zetabytes by 2020 [12]. As the size of the available datasets continue to increase, so will the possibilities and challenges for machine learning algorithms and data scientists continue to evolve.

Today machine learning is applied to such a wide range of problems and comes in so many different shapes and forms that the algorithms and the tasks they attempt to solve have grown fundamentally different from each other, which has lead to a division of the field into three sub-areas: supervised learning, unsupervised learning and reinforcement learning. While the different areas are fundamentally different from each other, it is important to note that some machine learning algorithms – inter alia neural networks which will be covered extensively in this thesis – are relatively similar in shape and can be adapted to each of the different problems with just minor changes. It is only supervised learning that has been employed in the work in this thesis, but the other fields are also briefly covered to give the reader a broader understanding of machine learning and its applications.

2.1 Supervised Learning

Supervised learning is by far the most common form of machine learning with applications in fields ranging from image analysis to natural language processing to speech recognition

and many others, and is also the type of learning which has been employed in the work which has resulted this report. The task one often wishes to complete when performing supervised learning is to infer general ideas from limited, labeled, data samples and *generalize* the drawn conclusions to novel problem instances. This is one of the key concepts of machine learning, where it often is uninteresting to performing operations on and see trends in the *local* data. Take the fitting a line to a curve for instance. There are many ways to do this, where interpolation with a polynomial of a given degree is one of the simplest options available. See Figure 2.1 for an example of two different polynomial fits to a dataset of 2-dimensional data points.

Fig. 2.1 Curves constructed by polynomials of different orders fitted with least squares to a sample of randomly generated data points. The red curve shows a polynomial of the 10th order whilst the green curve shows a polynomial of the 2nd order. The true function is $8x - 10x^2 + x^3$ normalized between zero and one, plotted as the blue line. The functions have been fitted to data generated from this function with added Gaussian noise, visualized as orange dots.

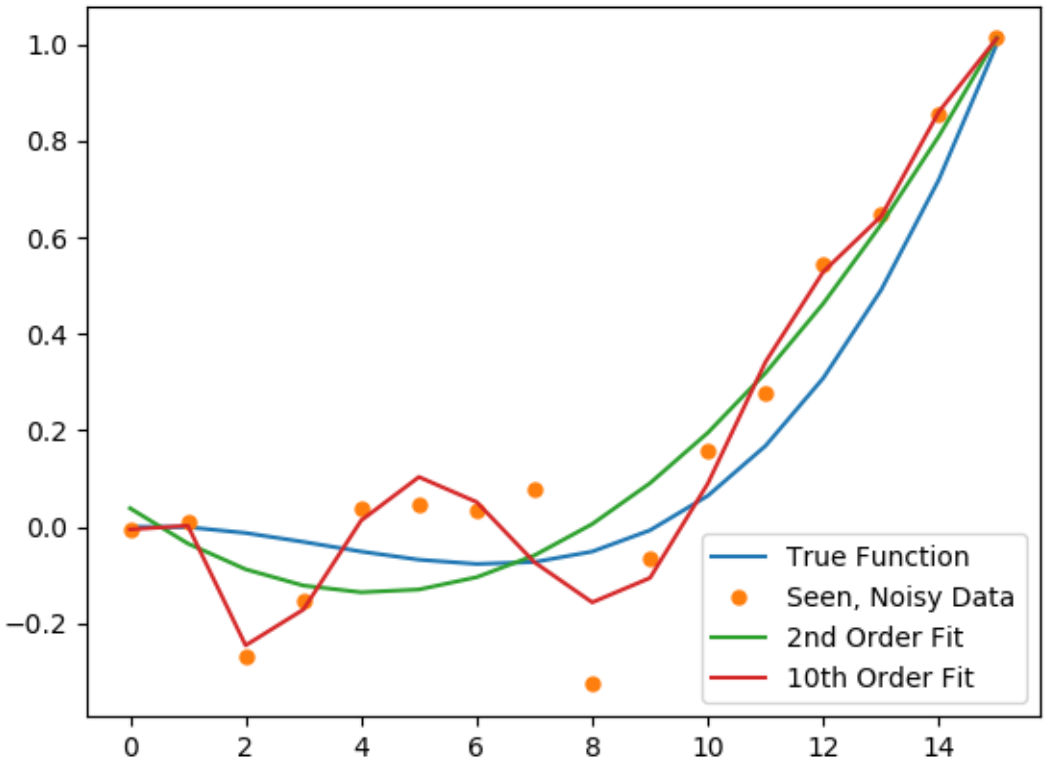


Figure 2.1 shows that the higher order polynomial performs a closer fit on the data points than the lower order polynomial. However, if the data contains noise, which all but every real-world problem suffers from, it is possible that the lower order polynomial actually fits the true generative function better than its higher order alternative. When working on machine learning problems the goal is never to get a good fit on the points that we can currently see, but rather to estimate the true generative function. In other words, we wish to construct a curve that will be able to approximate *new* data points as closely as possible. To the aim of a deeper discussion in this topic we introduce a few fundamental concepts:

- E: the error function, i.e. the measure of how much our approximated model differs from the true one
- E_{in}: the error on the points we use to construct our approximated function, i.e. the error on the points *in our data sample*
- E_{out}: the error on the points outside our known data, i.e. points that lie *outside of our known data sample*

The error function E can be anything from the mean squared error to complex domain specific functions. This function is referred to as the *loss* function or sometimes the *objective* function. The formulation of the objective function is one of the most important parts of constructing as successful machine learning model, which will be discussed in further detail in future sections. Figure 2.2 illustrates the difference between different models and their respective E_{in} and E_{out}. While it is clear that the higher order polynomial in Figure 2.2 has an $E_{in} = 0$, if one imagines that a new point would be added to the dataset it appears more likely that the lower order polynomial will in fact perform significantly better. Thus, its E_{out} is lower, which is all one truly cares about in this circumstance.

Figure 2.2b displays a problem commonly known as *overfitting*, which is one of the greatest challenges to overcome in machine learning. It is easy to accidentally push the algorithm too far when chasing better performance, but when the error drops on the dataset used for training/constructing the model, the error might in fact increase on the unseen samples. When the data is two-dimensional like the one we have seen in Figures 2.1 and 2.2 it is trivial to see when this phenomenon occurs. However, when the data is of a dimensionality in the order of thousands – or simply 4 or 5 – it is almost impossible to visualize the data in the same way which lead researchers to use other means to detect and take measures against overfitting. For an in depth description of overfitting and its countermeasures, see [1].

Fig. 2.2 Illustration of curve fitting with polynomials of different orders. Figure 2.2a shows a fit of a 3rd order polynomial to the data points as the red curve plotted with the true function, shown as the green curve. Figure 2.2b sows a fit of a 9th order polynomial to the data points as the red curve plotted with the true function, shown as the green curve [37].

Figure 2.2a, 3rd order polynomial.

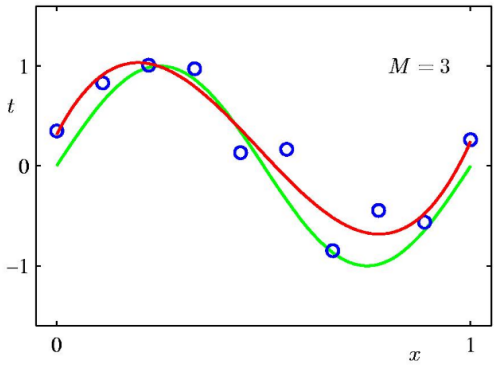
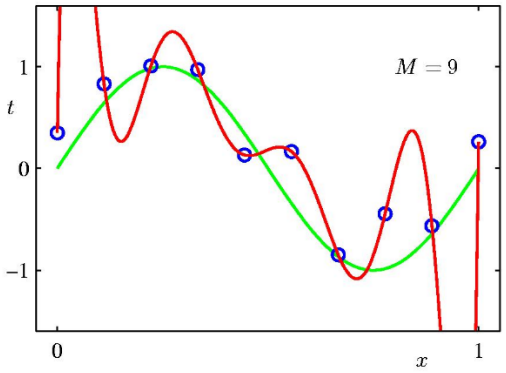


Figure 2.2b, 9th order polynomial.



A more formal definition of supervised learning is given as the art of inferring general functions from labeled data, and as has been previously mentioned, it is the availability of large scale datasets that has helped popularize machine learning. Today, there is a plethora of public datasets of different types, many of which are related to international challenges where teams from all over the world can compete with their algorithms to achieve as good a performance as possible. The aim of such competitions are usually to motivate researchers to focus on and improve a specific research topic or, if hosted by a company, to provide said company with a machine learning solution it currently lacks either the competence or the resources to construct internally. Of the fields mentioned in the beginning of this section, it is in the domain of image analysis that this thesis has been performed. Thus, most of the subsequent discussion and examples will concern image analysis in one way or another but please note that much of the discussion holds for other domains as well.

One form of supervised learning is classification: the prediction of a class label given a class instance. In image analysis, a public dataset that has been employed frequently in academic research is the MNIST dataset of handwritten digits [48]. The goal of the competition connected to the dataset is to construct a model that is able to tell, or *classify*, which number the handwritten image is meant to symbolize. This task have been used as a benchmarking tool for decades due to its simplicity: there are only 10 classes (numbers 0-9), the images are black and white and thus only consist of a single channel, the images are small and easily processed and there are lots of them. As can be seen in Figure 2.3 however, all numbers are far from easily discernible even to the human eye which makes this a suitable task for machine learning algorithms.

Fig. 2.3 Extract of characters taken from the MNIST database [18].

This dataset has historically been used to benchmark various classification models. These models look at thousands of images and try to predict the corresponding labels. Depending on whether the guess/prediction was right or wrong the models' parameters are updated correspondingly. We will go further into this process in the section which discusses the update method known as "Gradient Descent". The lowest recorded error rate E_{out} for the MNIST database of handwritten digits is 0.23% [6], which is an extremely good performance and a near perfect score. The machine learning community has since moved on to harder tasks, and one of the largest image analysis competitions currently worked on by scientists and engineers form all over the world – the current Olympic games of machine learning for image analysis tasks if one wills – is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) on the ImageNet database, where tasks include object localization, object detection, object detection from video, scene classification and scene parsing of millions of images and 1000 different classes [32].

2.2 Unsupervised Learning

Contrary to supervised learning, *un*supervised learning attempts to learn from unlabeled data. One of the most famous unsupervised learning algorithms is K-means clustering [23], which – as its name implies – is a clustering algorithm that divides the data into K different groups, which is applied to unlabeled data. See Figure 2.4a for an example of a given, unlabeled, dataset. If the task is to divide the data into three different groups then the resulting distinction made by the K-means algorithm is shown in Figure 2.4b.

Fig. 2.4 A case study of the unsupervised K-means algorithms where an unlabeled dataset is divided into three clusters. The left figure, Figure 2.4a, shows the unlabeled data and the right figure, Figure 2.4b, shows the corresponding clustering [29].

Figure 2.4a, the unlabeled data

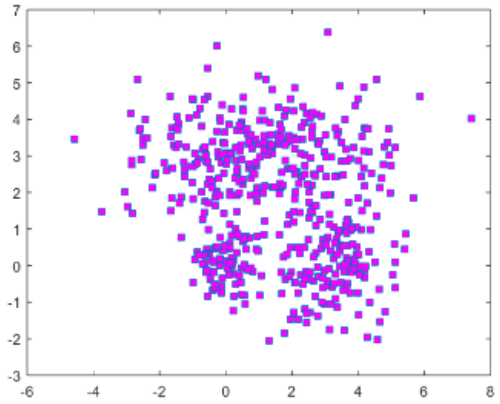
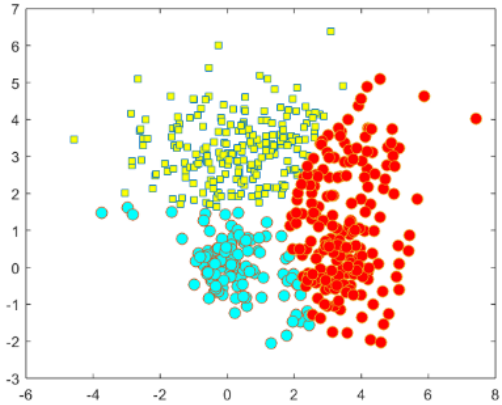


Figure 2.4b, the resulting labeling

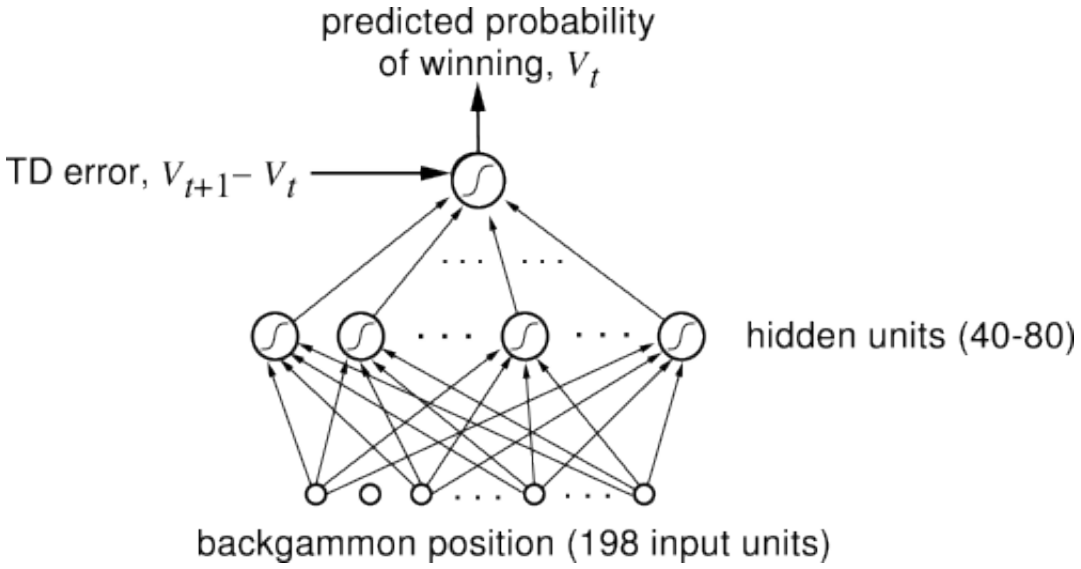


Compared to supervised learning the applications of unsupervised learning are very limited, mostly due to the fact that as of the writing of this thesis the machine learning community has failed to construct unsupervised models that are as efficient as those which work with labeled data. The great benefit of unsupervised learning however, which might come to be its saving grace in the future, is just that: it is unsupervised. Today there are large datasets that come without any labels and the construction of labels if often a costly enterprise, often requiring human experts to manually describe the data. In e.g. medical image segmentation this is an extremely tedious task that takes a very long time for human experts to complete. Significant breakthroughs in the field of unsupervised learning would have immense implications in wide range of fields. It is also closer to the way humans and animals learn from the world [21] which means it could be a step in the right direction of one of the original goals of Artificial Intelligence: the construction of a super-intelligent agent.

2.3 Reinforcement Learning

Reinforcement learning can be viewed as the step in between supervised and unsupervised learning and is sometimes referred to as semi-supervised learning. The tasks where reinforcement learning is applied are usually fundamentally different from the two previously mentioned areas of machine learning. Reinforcement learning is inspired by behaviorist psychology where an agent learns from its interaction with its surroundings as it tries to maximize a cumulative reward function. Reinforcement learning has been very successful in training agents to play different games on a superhuman level where a classical example is Tesauro's temporal difference approach to teaching an artificial neural network to play backgammon through reinforcement learning [41]. This network, in its simplest form, only looks at a binary representation of the current state of the board which serves as input to the first layer of the network. These binary inputs are then filtered through the artificial neural network which outputs a single value between zero and one. This value is interpreted as the probability of victory given the current state of the game. The network then looks at every possible alteration to the current state of the game (given that it is the networks turn to play) and picks the choice which maximizes its output value (i.e. the chance of victory). After a move is made, the parameters in the network are updated according to the backpropagation procedure – see Section 3.2 – but instead of a comparison of the prediction and the ground truth for each instance as is commonly done in supervised learning, a comparison is made between the current output of the network and the output in the previous time step. An exception is when the game is over, at which point the network either receives a large positive value if it was victorious or a large negative value otherwise. This process is performed over and over for thousands of iterations until the network has learned to perform well. This learning procedure has proven to be more successful than attempts to utilize pure supervised learning, where the network for each state of the board is given a correct choice to make which corresponds to choices made by human experts. Networks trained with the reinforcement learning approach eventually not only outperformed their supervised competitors, but also human masters. See Figure 2.5 for an illustration of such a network. The main difference between reinforcement learning and supervised learning is that the target of the backpropagation algorithm is a comparison between different states of the network rather than with a ground truth label. This means that the update procedure for the weights is de facto identical to a supervised learning problem, and the only difference is how the labels are constructed. For this reason, reinforcement learning can also be argued to exist as a sub-field of supervised learning. It can however, be useful to make the distinction between the two since the nature of problems they are applied to and the philosophy behind the algorithms are so different.

Fig. 2.5 A graphical illustration of an artificial neural network [22] constructed for the purpose of learning to efficiently play backgammon through reinforcement learning. A binary version of the current state of the board at a given point in time is fed to the 198 input neurons of the network. These inputs are then filtered through the network to generate a final output from a single artificial neuron, which is interpreted as the conditional probability of victory given the current state of the board. The output of the network is compared to the output in the previous time step to generate an error function which is used to update the internal weights of the model, see Section 3.1 for a more detailed explanation of the inner workings of artificial neural networks.



Another domain where reinforcement learning has proven to be useful is to find optimal maneuvers in controlled Markovian domains through the introduction of the Q-learning algorithm [45]. Reinforcement learning will not be covered to any further extent in this thesis but the interested reader is referred to the book on the subject by Richard Sutton, Reinforcement Learning: An introduction [38].

Chapter 3

Deep Learning

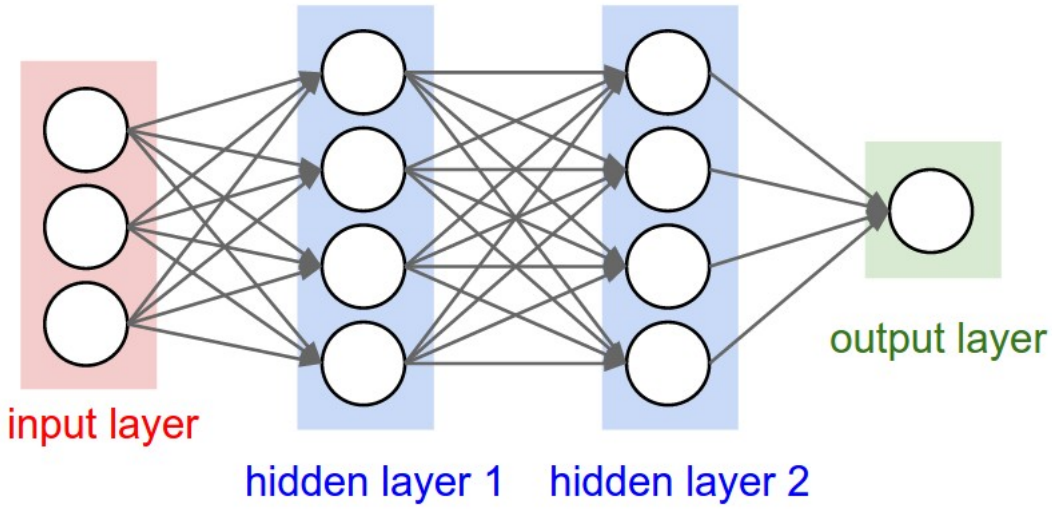
Seen from the outside, deep learning is simply an incredible family of techniques which has widened the horizon of possibilities in both academia the industry. For a scientist or an engineer however, the main shift in paradigm introduced with deep learning stems from feature extraction and the way they are processed by the models. Traditionally, engineers had to go through the tedious and time consuming process of finding hand-craft features, which would be fed into a shallow model which performs the machine learning task (e.g. some form of classification). This is an expensive process which often requires high domain specific knowledge, where not only the search for relevant feature candidates is a problem but also the selection of which features actually contribute to the model performance. Deep learning however, provides a way to automatically extract relevant features from the data during training which has lead to a complete shift in focus, where scientists now focus on model construction and optimization rather than feature engineering. The subsequent sections in this chapter covers some of the most important aspects of artificial neural networks and deep learning.

3.1 Artificial Neural Networks

Like in so many other areas of science and fields of engineering, machine learning was born and raised from ideas inspired by nature and the animal biology. To give answer to Alan Turing's question "Can machines think?" and towards the ultimate goal of the construction of a super-intelligent agent, the attention of the scientists in the first half of the 20th century went to the structure and inner workings of the animal brain. The organizational structure of the neural network in a biological brain is of a complexity that is beyond the reach of both our ability to perform satisfactory measurements and our ability to model such structures in silico. However, careful examination of the different constituents in the biological axons and

their inter-axon communication have lead researchers to formulate a more organized version of the biological neural networks to create *artificial* equivalents, see Figure 3.1.

Fig. 3.1 A conceptual graph of a fully connected artificial neural network [19]. Each white ball represents a perceptron which processes its input, commonly a weighted sum over the output of all the data points in the previous layer, through a non-linear, differentiable activation function. The activation function in this kind of shallow neural networks is usually some form of sigmoidal function which transforms the input into a scalar between 0 and 1 or -1 and 1, which depicts the level at which the neuron fires. If the perceptron is an output neuron (which activation function almost exclusively outputs a value between 0 and 1) the output of the non-linear activation function can be interpreted as e.g. class probabilities in a classification problem. By intertwining many such perceptrons, an artificial neural network is created. This example demonstrates a fully connected feed forward artificial neural network with one input layer, two hidden layers and one output layer. It is common practice to have either one or two hidden layers in this kind of shallow network.



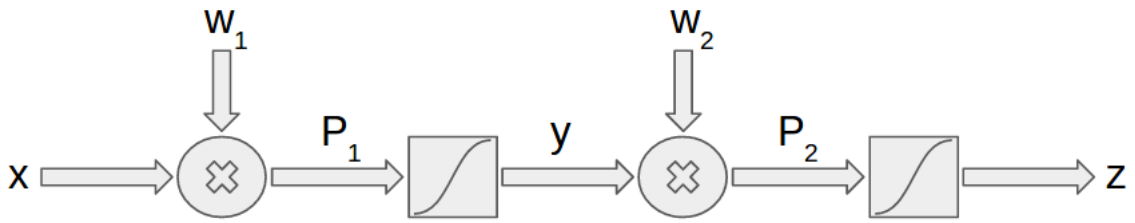
The core principle is that a large number of smaller entities, while by themselves unable to model complex functions nor understand high-level patterns, work together in a bottom-up approach to map complicated, non-linear functions and perform high-level tasks. Just like in biology, each of these entities – usually referred to as artificial neurons or perceptrons – receive an input based on the outputs of its neighboring perceptrons, and based on those values' determine its output signal. In biology, the neurons either fire with full force or not at all, i.e. they are binary. The artificial neurons pass their input value through a non-linear activation function which quite commonly approximates the delta function seen in nature by a smoother version in the interval zero to one, see Section 3.4.3 for a more detailed explanation of the activation functions used in artificial neural networks. When a multitude of such perceptrons collaborate to perform high level tasks, they form a multilayer perceptron/an

artificial neural network. The network described in this section is a shallow and fully connected artificial neural network. It is shallow since it only contains very few hidden layers and it is fully connected since each artificial neuron in a layer is connected to all neurons in the preceding layer and in the succeeding layer. In Section 3.4 we will discuss ways to further enhance the structure to better adapt it to our needs and move into the domain of deep learning.

3.2 Backpropagation

Backpropagation is the core methodology which has made artificial neural networks into the powerful machine learning model it is today. Backpropagation is a clever way to utilize the chain rule to propagate partial derivatives back through the network in such a way that no overlapping calculations need to be performed. The procedure is used in conjunction with an optimization method which decides how to properly update the parameters of the model based on the gradients calculated with backpropagation. The backpropagation algorithm is so fundamental to their success and so characteristic of artificial neural networks that some argue that a better name for the networks would be *differentiable* neural networks. The backpropagation algorithm is best demonstrated by example, for which reason we shall construct the simplest artificial neural network imaginable following the notation in [46], see Figure 3.2.

Fig. 3.2 An artificial neural network is defined as two or more perceptrons (or artificial neurons) that work together to approximate a function. Thus, the simplest possible artificial neural network consists of only two perceptrons [46]. This figure shows an example of such a network, where the input x is multiplied with the weight/parameter w_1 – which output is denoted as P_1 – and fed through a sigmoidal activation function to create the first neuron. The output from the activation function is multiplied with the weight w_2 and the output – denoted P_2 – is once again fed through a sigmoidal function to create the second artificial neuron. The output of the multilayer perceptron z is compared to the true label d in error function E.



In machine learning it is common practice to view the input as stationary and the parameters as the factor with which we can alter the performance of the model. For the toy example seen in Figure 3.2 this means that we have to calculate the partial derivatives of the error function E with respect to the weights w_1 and w_2. Before we are able to calculate these gradients there are two functions that have to be defined: the error function E and the activation function S. For simplicity we define the error function E as the mean squared error

$$E = \frac{1}{2}(d-z)^2, \tag{3.1}$$

where d denotes the true label and z the output of the network, i.e. its prediction. The activation function is also defined for its mathematical convenience. In nature, neurons either fire or they do not, i.e. they use a heaviside function which gives an output of one if its input exceeds a given threshold, otherwise zero. Such a function can not be incorporated directly into our artificial equivalent since it is not differentiable. Instead, what is used in an ANN is a smoothed version of the heaviside function commonly referred to as a sigmoidal function

$$S(\alpha) = \frac{1}{1+e^{-\alpha}}, \tag{3.2}$$

where α is the input value given to the artificial neuron which employs the sigmoidal function as its non-linear activation function. Given this set of functions we can find the gradient of the error with respect to the weights. Since our example is made up of two weights, the gradient with respect to the parameters is defined as $\bigtriangledown_w E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2})$. The gradients are found by the application of the chain rule propagated backwards through the network, which makes it natural to start with the calculation of w_2 rather than w_1 since it is closer to the output of the network. To find $\frac{\partial E}{\partial w_2}$, we first have to calculate $\frac{\partial E}{\partial z}$ since z is the only variable which has a direct dependence on E. Expansion with the chain rule gives $\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial w_2}$. To find the partial derivative of z with respect to w_2 we also have to consider P_2, for which reason we further expand the gradient with the chain rule $\frac{\partial z}{\partial w_2} = \frac{\partial z}{\partial P_2}\frac{\partial P_2}{\partial w_2}$. When put together the partial derivative of the error function E with respect to the weight w_2 is $\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial P_2}\frac{\partial P_2}{\partial w_2}$. This is all we need to calculate the gradient, and the partial derivative of E with respect to w_1 can be found in an similar fashion. The partial derivative of the error function with respect to the weights w_2 and w_1 are

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial P_2}\frac{\partial P_2}{\partial w_2}, \tag{3.3}$$

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial P_2}\frac{\partial P_2}{\partial y}\frac{\partial y}{\partial P_1}\frac{\partial P_1}{\partial w_1}. \tag{3.4}$$

Now that we know which partial derivatives we need to find in order to calculate the gradient, all that is left is to solve the equations. In our example, to find the gradient with respect to w_2 we calculate $\frac{\partial E}{\partial z} = \frac{\partial}{\partial z}(-\frac{1}{2}(d-z)^2) = d-z$ and $\frac{\partial P_2}{\partial w_2} = \frac{\partial}{\partial w_2}(yw_2) = y$. It is also necessary to calculate $\frac{\partial z}{\partial P_2}$ but this is a bit trickier since we have the additional dependence on α in the sigmoidal activation function. To find the derivative we calculate its partial derivative with respect to α and get

$$\begin{aligned}
&\frac{\partial S}{\partial \alpha} = \frac{\partial}{\partial \alpha}(1+e^{-\alpha})^{-1} = -(1+e^{-\alpha})^{-2}e^{-\alpha}(-1) = \frac{e^{-\alpha}}{1+e^{-\alpha}}\frac{1}{1+e^{-\alpha}} = \\
&= \frac{e^{-\alpha}+1-1}{1+e^{-\alpha}}\frac{1}{1+e^{-\alpha}} = \frac{1+e^{-\alpha}}{1+e^{-\alpha}}\frac{1}{1+e^{-\alpha}} = (1-\frac{1}{1+e^{-\alpha}})\frac{1}{1+e^{-\alpha}} = \\
&= (1-S)S. \qquad (3.5)
\end{aligned}$$

From equation (3.5) we see that the derivative of the function with respect to its input is simply the function times one minus the function itself, which means that it is extremely easy to differentiate and one of the reasons for which it has been used so frequently in artificial neural networks. When applied to our example, we get that $\frac{\partial z}{\partial P_2} = (1-z)z$ and when put together we get $\frac{\partial E}{\partial w_2} = (d-z)(1-z)zy$ and analogously $\frac{\partial E}{\partial w_1} = (d-z)(1-z)zw_2(1-y)yx$. The final formulation consists only of known values or values which have been calculated in previous steps of the backpropagation algorithm. When all gradients have been calculated with the backpropagation procedure, the optimization technique which backpropagation is used in conjunction with will decide how to best process said gradients to maximize the performance of the model.

3.3 Gradient Descent

Like most forms of deep learning, the optimization techniques used in this work is based on a gradient based optimization method known as gradient descent. Optimization techniques are in general employed to either maximize or minimize an approximated function $f(x)$ by alterations in x. It is common praxis to work on minimization tasks – i.e. minimize the error generated by the approximated function as compared to the true function – and thus reformulate maximization tasks to minimization by a change in sign such that the minimization is performed on $-f(x)$. The function we wish to minimize is the one we referred to as the objective function in Section 2.1.

The derivative of the function $f(x)$ gives the rate of change for the function at point x which gives an indication of how the function itself might change given a small change

in input like so: $f(x+\eta) \approx f(x) + \eta f'(x)$. Since gradient descent only depends on the first order gradient it is categorized as a first order optimization algorithm. The aim of the algorithm is to take the model from a bad position (a hill in minimization tasks) in the search space to as good a location as possible, i.e. a local optima (a valley in minimization tasks), see Figure 3.3 for a visualization of how a gradient descent-based algorithm might navigate a 3-dimensional landscape given different initial positions. At any given point in space, there are often more than one possible direction in which steps will lead to a decrease in the objective function, but the crux is to find the one which decreases the error the most. Given that the objective function $f(a)$ is defined and differentiable in the neighborhood of the point a, the function improves the most when a step η is taken in the direction of the negative gradient such that a_{t+1} is calculated as

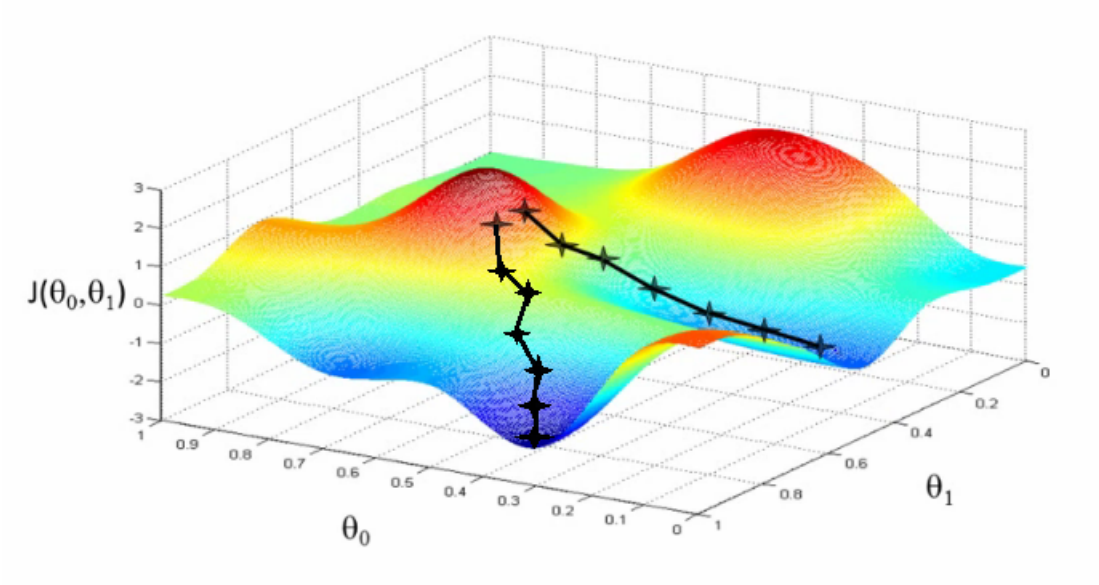
$$a_{t+1} = a_t - \eta \bigtriangledown_t f(a_t). \tag{3.6}$$

If equation (3.6) is true given stable conditions (i.e. η is small enough), it follows that $f(a_{t+1}) \leq f(a_t t)$, which means that we have taken a minimization step and therefore improved the objective function (and hopefully also the performance of the model) by following the negative gradient. The gradient descent algorithm simply performs an iterative, first order optimization process given an initial position θ_0 in the parameter space and then updates the weights to decrease the loss function based on the training data x, which means that θ_{t+1} is defined as

$$\theta_{t+1} = \theta_t - \eta_t \bigtriangledown_\theta f(x; \theta_t), t \geq 0. \tag{3.7}$$

The step size η is also referred to as the learning rate of the model, and is often thought of as the most important hyper-parameter for gradient descent-based optimization techniques and it is crucial that the learning rate is set correctly. Make it too low and the algorithm will take too long to converge, make it too large and you might walk over local minima located too close to your current position in which case the algorithm might never converge at all or miss optimal paths. Choosing a proper learning rate is a science in itself, and it is common to use an *adaptive* learning rate, where η is updated over time as the algorithm gets closer and closer to convergence. The gradient descent algorithm can be thought of as a blind baby making its way down a hill backwards by feeling the change in the ground around it, where the learning rate is the pace in which the baby crawls.

Fig. 3.3 An example of a gradient descent optimization process with two different starting points in a 3-dimensional landscape [28]. The starting points are located on a hill with steep slopes, where the different starting points lead to two different local optima based on the gradients in their respective neighborhoods. The different positions in the search space after different iterations of the gradient descent algorithm are marked with X's, and the lines are the paths taken between the positions which are derived from the gradient at each such location.



3.3.1 Stochastic Gradient Descent

While the pure version of the gradient descent optimization technique as described in Section 3.3 sounds good in theory, it is in many machine learning problems not practical to work directly with the gradient based on all the available data since this would require the evaluation of the model on every single instance in the entire dataset at each point in time. Particularly when one moves into the domain of big data where we have access to hundreds of thousands of high-dimensional data points, which is often the case for deep learning problems, the time and computational expense is way above what is reasonable in many practical applications. In practice, what is done instead is that the gradient is estimated by the extraction of a small random sub-set of the entire dataset. Since this only gives an estimate of the gradient, this technique has been named *stochastic* gradient descent (SGD). See Algorithm 1 for an overview of the update procedure connected to the stochastic gradient descent optimization

technique for a standard machine learning task.

Algorithm 1: Stochastic gradient descent update procedure at training iteration k [15]

Require: Objective function L.
Require: Learning rate η_k.
Require: Initial parameter θ.
while *stopping criteria not met* **do**
  Sample a minibatch of m samples from the training set $\{x_0, ..., x_m\}$,
  with corresponding targets/labels y_i.
  Compute gradient estimate: $\hat{g} \leftarrow +\frac{1}{m} \bigtriangledown_\theta \sum_i L(f(x_i; \theta_{t-1}), y_i)$.
  Apply update: $\theta_t \leftarrow \theta_{t-1} - \eta_k \hat{g}$.
end

While algorithm 1 illustrates the standard version of the stochastic gradient descent procedure it is common to use various augmentations/alterations to the standard procedure to increase convergence rate and to tackle common difficulties in the search space (e.g. saddle points). It is common to add a momentum to the update rule which adds a dependence to the gradients of previous iterations, sometimes based on the Nesterov accelerated gradient update rule when iterating through the parameters and gradients [15].

3.3.2 Adam

While it is common to refer to the standard version of the stochastic gradient descent optimization algorithm in academic settings and while it does have practical applications, there exist a wide range of updated and highly specialized stochastic optimization techniques which outperforms the procedure described in Section 3.3.1, both in terms of convergence time and final error rate. One such technique is called *Adam* [20] and has been frequently used in contemporary neural network-based research and has proven to be a robust and fast way of navigating the parameter space. Adam consistently outperforms many of its competitors and works in such a way that is calculates individual adaptive learning rates based on the first and second moment of the gradient and is an extensions/combination of

two other popular optimization techniques: AdaGrad and RMSProp.

Algorithm 2: Adam: a stochastic optimization technique

Require: η: Stepsize.
Require: $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates.
Require: $f(\theta)$: Stochastic objective function with parameters θ.
Require: θ_0: Initial parameter vector.
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector).
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector).
$t \leftarrow 0$ (Initialize timestep).
while θ_t *not converged* **do**
  $t \leftarrow t+1$.
  $g_t \leftarrow \bigtriangledown_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. the stochastic objective at timestep t).
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (Update biased first momentum estimate).
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (Update biased second raw momentum estimate).
  $\hat{m}_t \leftarrow m_t / (1-\beta_1^t)$ (Compute the bias-corrected first moment estimate).
  $\hat{v}_t \leftarrow v_t / (1-\beta_2^t)$ (Compute the bias-corrected second raw moment estimate).
  $\theta_t \leftarrow \theta_{t-1} - \eta \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon)$.
end

3.4 Convolutional Neural Networks

As the original architecture of artificial neural networks seen in Section 3.1 drew inspiration from the synaptic connections and axons in the biological brain, so have the more recently developed convolutional neural networks (CNN) drawn inspiration from nature and the constituents of the mammalian brain. More specifically, the connectivity of the artificial neurons are constructed in a fashion similar to the organizational structure of the animal visual cortex. The CNN networks are ad hoc structures that have been highly optimized for the explicit assumption that the input data is image-based, be it a standard 2D image, 3D volumes or a steady stream of temporal images. This allows for an extremely optimized architecture that has managed to greatly reduce the memory requirements and computational cost of inference and training both, and which is able to efficiently capitalize on the spatial relationships between neighboring data points in images.

Traditional fully connected artificial neural networks have proven unsuitable for image processing tasks due to the extremely high dimensionality associated with images of even the most moderate of sizes. For example, imagine a black and white image – i.e. with only

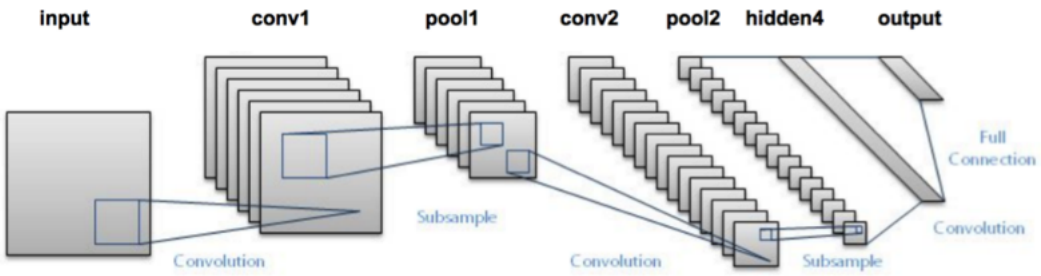
a single channel – with a resolution of 32×32. When gaged by the human eye, such an image appears very small. Should it be processed by a fully connected ANN however, the amount of parameters in the first layer only would be $32 \times 32 \times k$ where k is the number of neurons in the first layer. Each neuron in the input layer in such a network would require $32 \times 32 = 1024$ parameters. While this number may seem relatively manageable, it scales poorly and the number of parameters will rapidly diverge when the image resolution increases. Most competitions in the image analysis field provide images with a spatial resolution of around 512×512. For such a problem, where it is normal to work with RGB images, *each* artificial neuron in the input layer would need $512 \times 512 \times 3 = 786\,432$ parameters. Imagine the modest case of 100 input neurons in such a network. The number of parameters in the *first* layer of the network would number 78 643 200. This is an extraordinary amount of parameters and an extremely inefficient way to process images, and this phenomenon is known as the curse of dimensionality. Through clever processing, shared weights, a smart architecture and by the utilization of the spatial correlations between neighboring data points in images, we shall see that convolutional neural networks make it possible to reduce the number of weights to a mere fraction of what is needed for the traditional ANNs while at the same time taking the possibilities of computerized image analysis to soaring heights.

Convolutional neural networks drop the fully connected layers from the main body of their architecture, and in their stead introduce a mixture of convolutional and pooling layers. These two components are usually stacked in an appropriate fashion to extract relevant features from the images whilst reducing the spatial resolution of said images. The output of the convolutional and pooling part of the network is thereafter flattened into a feature vector and fed into a more conventional fully connected ANN which then produces the final predictive output. See Figure 3.4 for an example of the layout of a typical convolutional neural network [11]. The subsequent sections of this chapter will give a in-depth explanation of the most important components that together form a convolutional neural network.

3.4.1 Convolutional Layers

The convolutional layers are, as the name suggests, the core building block of convolutional neural networks. It is also the constituent that performs the heavy computational lifting in deep convolutional neural networks (DCNNs). What is referred to as a convolutional layer is in fact simply a set of convolutional filters, also referred to as kernels. Each kernel only processes a small part of its input tensor at each given point in space, e.g. 3×3 pixels/voxels, where the spatial size of this window is defined as the kernels *receptive field*. The kernel has as many weights as there are elements in its receptive field. The convolutional operation performed by the kernels calculates the dot product between its weights and the elements

Fig. 3.4 An example image of an architecture of a convolutional neural network used for classification [11]. The herein shown architecture is named LeNet, and was one of the first truly successful convolutional architectures. Today, LeNet is often used as an example network for educational purposes or as a solid starting point for further research. The LeNet architecture consists of two convolution-pooling pairs followed by a fully connected structure with one hidden layer and one output layer which outputs the final class predictions. Note that what is shown in the image are the output volumes or *feature maps* produced by the different operations within the network when image(s) are propagated through the network.



within its receptive field, which includes all the pixels/voxels in the entire depth of the input tensor within the given spatial range. For example, if the input image is colored and thus made up of three different channels, one for each RGB value, then a kernel with a receptive field of 5×5 sees $5 \times 5 \times 3$ data points at each given point in space, which also corresponds to the number of trainable parameters of each kernel. Each dot product produces a single output scalar and by convolving the filter over the input tensor we produce an output tensor – or feature map – with the same spatial resolution as the input tensor. Different techniques exist to deal with the edge case of when the kernel looks at the edges of the input tensor, where adding rows and columns of zeros around the image – referred to as zero padding – is a way to maintain the spatial resolution of the image after the convolution without performing any alteration of the data within the image. A different approach is to only let the filter look at valid pixels/voxels, i.e. lose information kept in the pixels at the borders of the images. This also results in an output tensor with a slightly lowered spatial size. It is uncommon for problems to have sensitive or relevant data in the edges of the images which means that in practice, the border-pixels are of little significance the choice of how to deal with them is arbitrary. This is of course a choice that could bear significance for certain rare problem instances in which case it might be beneficial to study domain specific solutions. Should the convolutional layer include more than one such kernel – which is the case in most examples of convolutional neural networks – the resulting feature maps are concatenated in the dimension where the channels live to form a $D+1$ tensor where D denotes the spatial

dimensionality of the data. Said tensor is then fed as the input to the subsequent layer, which outputs the tensors that are fed into the succeeding layer et cetera.

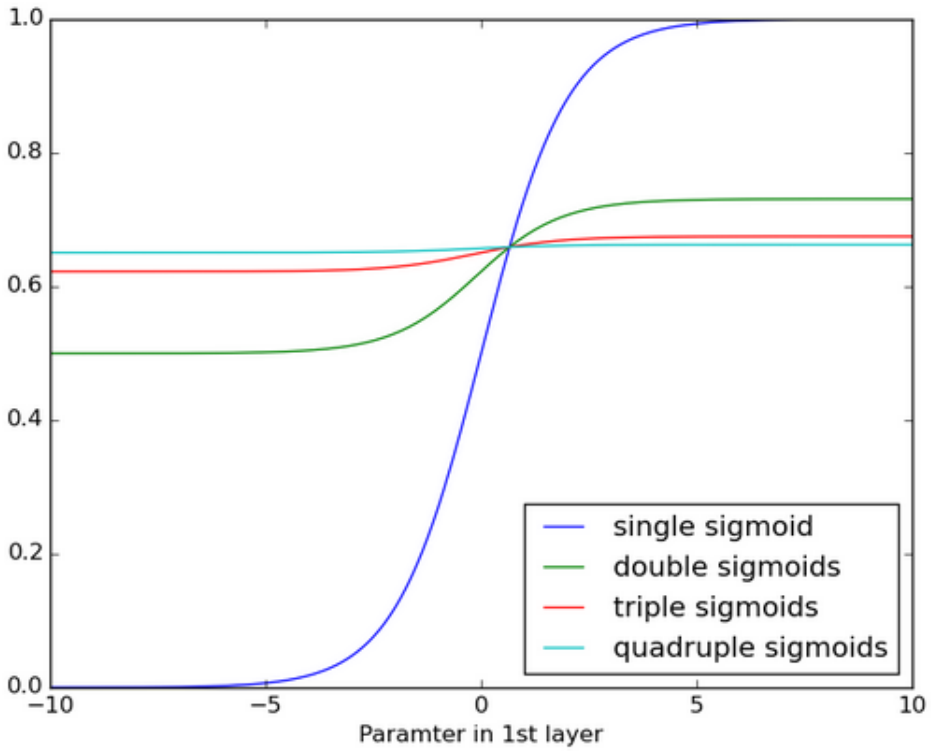
3.4.2 Pooling Layers

In between convolutional layers the modus operandi is to insert pooling layers which serve a dual purpose. Firstly to perform a reduction of the spatial resolution of the data to ease the computational cost, lower the number of parameters and thereby combat overfitting. Secondly, give the network a more abstract representation of the data to reduce the importance of the exact locations of the object in the input image and to allow the network to better understand and learn data on a higher abstraction level, which has proven to be extremely useful for many different tasks. While there is a plethora of different pooling operations, the most common form of pooling is max-pooling. A max-pooling layer looks at a window of the values in its defined receptive field and chooses the largest of those values as its output. The window is then moved s steps in a convolutional fashion, where s is the defined stride. Most commonly both the receptive field and stride is set to 2, which results in a reduction in the spatial resolution by a factor 4 for 2-dimensional data. Notice that this procedure is performed on all feature maps on the input tensor independently which means that the depth of the input tensor is unchanged after the pooling operation. Another common pooling operation is average pooling, where the average value is calculated from the values in the receptive field instead of just choosing the largest value.

3.4.3 Rectified Linear Unit

As discussed in Section 3.2, in traditional artificial neural networks such as the one described in Figure 3.1, the activation functions are usually some kind of logsitic sigmoid, see e.g. equation (3.2). This function outputs a smooth s-shaped curve in the interval between zero and one, see Figure 3.5. As already noted in Section 3.2, the logistic sigmoid has been chosen due to its mathematical convenience – it is very easy to differentiate – and since its output can be interpreted as the probability or importance of the given input data it is suitable for use in a wide range of tasks. The logistic sigmoid and other similar functions work very well in shallow networks or as the output of deeper networks, but are very problematic when the functions are performed again and again on the same data in a deeper structure. When the function is performed repeatedly, which happens when the data is propagated through the network, the values – and their gradients – will quickly go to zero and we loose the ability to train the network. This phenomenon is known as the vanishing gradient problem, see Figure 3.5.

Fig. 3.5 An illustration of the vanishing properties of the sigmoidal functions when applied repeatedly [9]. We see a drastic decay in the values of the function as its power increases.



The vanishing gradient problem has historically been one of the main hurdles that researchers had to overcome to make deep artificial neural networks into a viable option. The introduction of the rectified linear unit (ReLU)

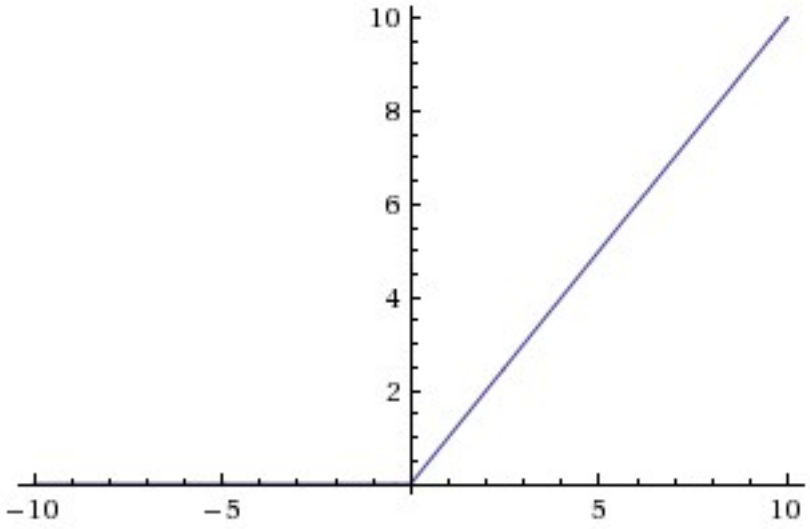
$$S(\alpha) = max(\alpha, 0) \tag{3.8}$$

as a replacement for the sigmoidal functions allow the data to pass through even extremely deep networks without the occurrence of the vanishing gradient problem due to its linear nature.

It is essential for the backpropagation algorithm that the network is differentiable, and thus requires there should be some form of non-linearity present. The rectified linear unit is an extremely simple function, but its non-linearity and linear relationship to a large portion of the input data makes it very apt for use in deep learning based techniques which employ very deep networks. The standard ReLU function, illustrated in Figure 3.6, has since its introduction been extended by a whole family of related activation functions, each with their

individual specialty. We will explore some of these options in chapter 6 when presenting the models developed in the context of this master thesis.

Fig. 3.6 The Rectified Linear Unit (ReLU) activation function [19] commonly used in deep neural network models. The mathematical formulation of the function can be seen in equation (3.8). Accordingly, the value of the function is zero when the X-axis (which depicts the range of input values) is less than zero the unperturbed input value when it is larger than zero. Notice that the function is not differentiable in the origo. This is a special case that can be solved in different ways depending on the problem at hand and the researchers preference. The derivative of the function is simply zero for inputs smaller than zero and one for inputs on the positive side of the X-axis.



3.4.4 Dropout

Another reason due to which researchers struggled to increase the depth of artificial neural networks for a long time was that they were extremely prone to overfitting. When the number of parameters increase (which is a natural consequence of increasing the depth of a network) so does the risk of overfitting, which meant that researchers had trouble increasing the depth of their networks for decades even though the theoretical advantages of deeper structures in artificial neural networks were evident from an early stage. Together with the ReLU activation function, the dropout regularization technique might be the most important breakthrough that has let the scientific community to move from shallower networks into the domain of deep learning. Like the rectified linear unit, the dropout technique is very simple. The technique only entails that for each layer where dropout is applied, each artificial neuron is excluded from the network for the duration of the current training iteration with

a probability p, which is a hyperparameter defined by the researcher. This means that the neurons cannot rely on the fact that other neurons will be present at each given point in time and thereby have to learn the actual pattern in the data and not learn non-existent patterns caused by neighboring artificial neurons with strong activations. Dropout can be seen as a way to train many different versions of the same network at the same time, which means that overfitting will be significantly reduced at the cost of longer convergence times.

Chapter 4

Related Work

As mentioned in Section 3, before the era of deep learning and automatic feature extraction, most machine learning algorithms – in the domain of supervised learning – heavily relied on efficient, hand-crafted features coupled with a flat/shallow classifier to perform their respective tasks. Models that showed great promise in the field of semantic segmentation in the past decades have inter alia been Boosting [35], [42], Random Forests [34] and Support Vector Machines [13]. The focus was primarily on feature engineering rather than model design. With the automatic feature extraction that comes with deep learning models the shift in focus has instead moved on from finding relevant features to elaborate model engineering resulting in large and complex models. This master thesis is based primarily on the work of four major research articles published in the context of image processing and deep learning: the U-net model for semantic segmentation of biomedical data [31], the GoogLeNet model introducing the inception module [39], the ResNet model introducing the residual learning building block for extremely deep convolutional networks [16] and finally the introduction of convolutional layers atrous for semantic image segmentation in deep convolutional neural networks [4].

4.1 The VGG Architecture

Simonyan & Zisserman [36] at the Visual Geometry Group (VGG) at Oxford University illustrated the correlation between the depth of convolutional neural networks and their performance. Their networks achieved the first place in the localization task in the ImageNet challenge 2014 and got second place in the classification task at the same competition (where Szegedy et al. won with their GoogLeNet model [39]). The architecture in [36] exclusively utilizes convolutions with a receptive field of 3x3, where experiments with networks of varying depths are performed. Simonyan & Zisserman found that the networks which had the

deepest architecture – i.e. the highest amount of layers – achieved the best performance. Their deepest model had 19 trainable layers, therefore denoted VGG19, and was the model which achieved the best performance. As usual when improving a certain technique in optimization, the authors found that an increase in depth eventually gave diminishing returns on the model performance. It is therefore the second deepest model constructed in [36] which has seen most use in practice, since it performs almost as well as VGG19 but is a smaller model with less parameters and is thus easier to manage. See Table 4.1 for an illustration of the VGG16 architecture.

Simonyan & Zisserman also showed that it is often preferable to apply convolutions with a small receptive field. Instead of the usage of a convolutional filter with a receptive field of 7×7 it is possible to use three stacked filters with a receptive field of 3×3. In such a structure the first layer does of course have a receptive field of 3×3, but while the second layer has a receptive field of 3×3, it has an effective receptive field of 5×5 due to the nature of the convolutional operation. The third and deepest layer has an effective receptive field of 7×7. By utilizing three convolutional layers with a smaller receptive field we not only make use of three non-linear activation functions which is good for the discriminative ability of the model, but we also reduce the amount of parameters in the model. With the assumption that both the number of input channels and output channels of three stacked convolutional layers with a receptive field of 3×3 is set to a constant C, the structure consists of $3(3^2C)^2 = 27C^2$ weights. A single convolutional layer with a receptive field of 7×7 would require $7^2C^2 = 49C^2$ weights.

4.2 U-net

The most remarkable achievements made by convolutional neural networks in recent years have been in classification, where convolutional and pooling layers have been stacked and glued together in different constellations, usually followed by a number of fully connected layers to make the architectures as efficient as possible and with as high a generalization ability as possible. The advances in network design have lead to a significant reduction in the error rates on notable image analysis tasks such as the ILSVRC classification challenge where significant improvements have been made on an annual basis the in last few years [36, 39, 16]. These classification networks were initially also used for semantic segmentation tasks, where the approach was to use the output of the fully connected layers to predict the class of each pixel based on its own corresponding value and surrounding region in the input image and thus produce a map of predictions corresponding to a semantically segmented mask. However, the groundbreaking results of [24] showed that it is in fact

Table 4.1 An illustration of the layers used in the VGG16 model [36]. Each layer shows the receptive field of the convolutional layers as well as the number of filters used in each layer, which are increased with a factor two after each pooling operation. The max-pooling layers use a stride of two. The final layers are fully connected, where the number of artificial neurons are illustrated for each layer. The last, fully connected, part of the network is often removed when adapting the network for segmentation tasks, as introduced in [24]. The final layer use a softmax activation function to generate the final prediction.

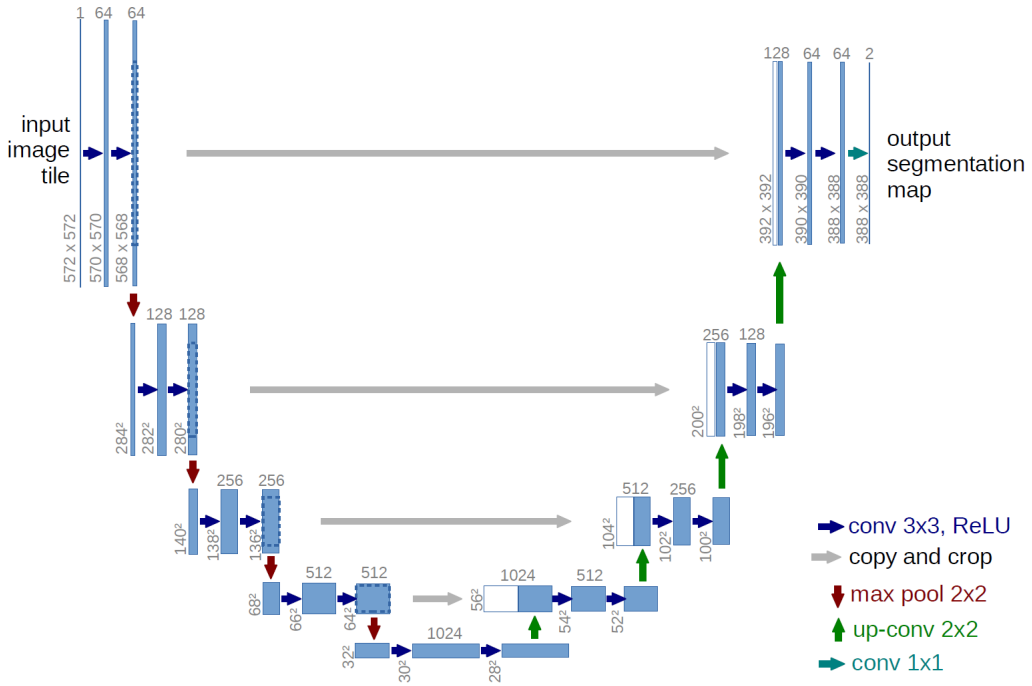
Conv3x3-64
Conv3x3-64
Max-Pooling
Conv3x3-128
Conv3x3-128
Max-Pooling
Conv3x3-256
Conv3x3-256
Conv3x3-256
Max-Pooling
Conv3x3-512
Conv3x3-512
Conv3x3-512
Max-Pooling
Conv3x3-512
Conv3x3-512
Conv3x3-512
Max-Pooling
FC-4096
FC-4096
FC-1000
Softmax

beneficial to remove the computationally expensive fully connected layers altogether and construct what is commonly referred to as a 'Fully Convolutional Network' (FCN). The fully convolutional networks make use of the same architectures as their classification-based cousins, but by dropping the fully connected layers only keep about 10% of the original number of parameters.

Following the branch of research initiated by [24], the aptly named U-net model [31] consists only of convolutional and pooling layers, i.e. no fully connected layers and the model has been fine-tuned for applications in the medical field, which generally suffers from a lack of data due to the high acquisition and labeling cost. The U-net model comprises of primarily two parts: (1) a feature extraction part which collects densely extracted features of

different resolutions and (2), a part which performs up-sampling to the original resolution and performs a segmentation based on the features from (1), see Figure 4.1 for a conceptual view of the U-net model and Figure 6.3 for an overview of the layers used in the implementation made for this thesis.

Fig. 4.1 A conceptual overview of the U-net model [31]. The blue arrows depict a convolutional operation with a 3×3 kernel, the gray arrows depict a copy and crop of feature maps, the red arrows a max pooling operation, the green arrows a transpose convolutional operation with a 2×2 kernel which doubles the spatial resolution and the teal arrows depict a convolutional operation with a 1×1 sized kernel with a softmax activation function which generates the final output.



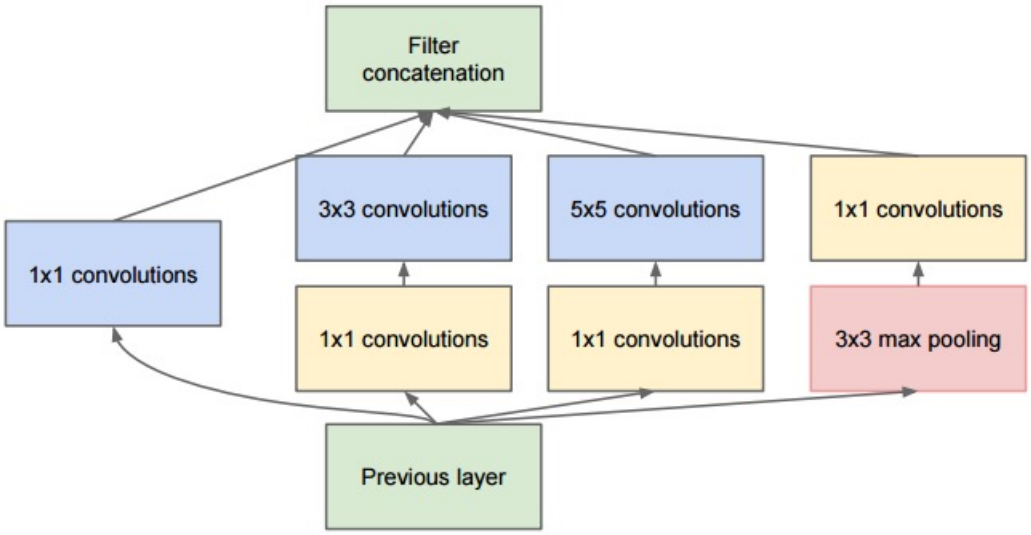
The feature extraction network follows the same architecture as the VGG16 network where convolutional and max pooling layers are stacked after each other in such a way that the number of feature maps is increased when we have a decrease in the spatial resolution and an increased network depth. The deepest layer does thus produce feature maps with a spatial resolution which is reduced by a factor 32 as compared to the input resolution. To convert these feature maps into a meaningful segmentation which maintains the original spatial resolution the VGG16 architecture is mirrored, with the output of the first network as input and so called *transpose* convolutions as replacements for the max pooling layers. These transpose convolutions – also called fractionally strided convolutions – increases the spatial resolution by a factor two by swapping the forward and backward passes of the convolutional operation. This architecture leads to a final output mask with the same resolution as the

input image. The final layer is a convolutional layer with a 1×1 kernel that uses a softmax activation function to reduce the 64 feature maps from the previous layer to K final feature maps which also serves as the final prediction/segmentation for the K classes present in the data. To further help this structure find a satisfactory prediction, the feature maps from the steps in the down-sampling part of the network that has the same spatial resolution as the corresponding part in the up-sampling network are concatenated to extract as much information as possible from each image, see Figure 4.1. This final trick helps the network to converge to a good local optima even with for very limited datasets.

4.3 GoogLeNet

Most convolutional architectures, for segmentation and classification both, follows the same pattern of linearly stacked convolutional and pooling layers. The performance of a network has often been considered to be directly dependent on its depth and researchers have pushed the number of stacked layers to its limit to create deeper and deeper networks. Compare e.g. the AlexNet [21] published in 2012 using 8 layers – which at the time was state-of-the-art in multiple image analysis categories – with the networks published by the VGG just two years later which showed a significant increase in performance with their VGG16 and VGG19 networks. The GoogLeNet architecture however, was the first to branch out and leave the standard architectures behind. Instead of stacking convolutional and pooling layers in the traditional sense to increase the depth of the networks, the authors in [39] worked to increase the *width* of the network by the introduction of the *inception module*, see Figure 4.2. Instead of using the standard dense convolutional layers the idea is to split up the calculations into different sub-layers with kernels of varying size. The architecture was inspired by the strong mathematical arguments for the utilization of sparse matrix operations for image processing, whilst being able to utilize the extremely optimized software and hardware which today exist for dense matrix operations (which renders a sparse architecture far too inefficient for most practical applications). The inception module is the result of an attempt to simulate sparse operations with dense building blocks, and the result was a network which contained less parameters than its predecessors while it increased the state-of-the-art performance for multiple image analysis tasks. The herein described inception module is today referred to as InceptionV1. The architecture has been further experimented with and as of this writing the latest version is the InceptionV3 module [40].

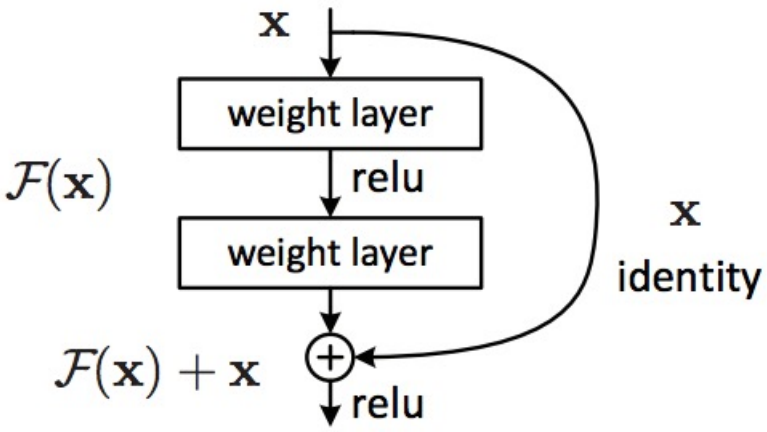
Fig. 4.2 The inception module as introduced in [39]. Convolutions with kernels of size 1x1, 3×3 and 5×5 are performed in parallel where the resulting feature maps are concatenated to form the input to the next layer. A max pooling layer is also added to keep the beneficial effects of such layers as observed in previous convolutional architectures. The pooling layer does however have a stride of one which results in feature maps with the same spatial resolution as the input. Also note that additional convolutional layers with 1×1 kernels with a single feature map as output is included in the module. These are present to drastically reduce the number of parameters needed to perform the convolutional operations with larger kernels. Should these be omitted the number of parameters and consequently the training time and computational cost would quickly explode for deeper networks.



4.4 ResNet

The chase for deeper networks did not stop with VGG19 or GoogLeNet however. The authors of [16] experimented with even deeper architectures and further increased the state-of-the art on the ILSVRC-2015 challenge with an ensemble of convolutional networks consisting of as many as hundreds of layers each. [16] found in their initial experiments that when deeper networks start to converge, a process called *degradation* occurs, which means that when they start to converge, the accuracy gets saturated and then drastically drops. This phenomenon has been concluded to be unrelated to overfitting and is thus a new obstacle to overcome when one utilizes extremely deep architectures with 30 or more layers. Unconstrained, this means that the deeper networks yields a lower performance compared to their shallower counterparts. To address the degradation problem, [16] introduced the so-called *residual building block*, see Figure 4.3.

Fig. 4.3 Let $H(x)$ denote the underlying desired mapping given the input variable x. Instead of optimizing for $H(x)$ directly, we let the weighted layers fit the another mapping: $F(x) := H(x) - x$. We then recast the mapping to the residual form of: $F(x) + x$. The hypothesis in [16] was that this is an easier formulation of the optimization problem which will lead to a more stable training of extremely deep convolutional neural networks.



The residual building block achieved its purpose in that it mitigates the degradation process and allowed He et al. [16] to construct extremely deep networks that outperformed the shallower networks of the past. Their submission to the ILSVRC-2015 consisted of network with an increase in depth by a factor 8 as compared to the VGG networks. While the depth of the networks employed in the research connected to this thesis is not of the same scale as in [16], the residual building block as been incorporated to increase stability and performance. Should the networks benefit from the addition of the residual building block it will serve to validate their use in shallower versions of deep neural networks.

4.5 Atrous Convolutions

As has been previously mentioned, the main hurdle to overcome when one adapts classification network architectures to semantic segmentation is to combat the decrease in resolution that comes with the pooling operations. The pooling layers play a crucial role in object detection and registration and is vital for location invariance. In addition to these benefits, they also increase the receptive field of the kernels in the deeper parts of the network which is essential for understanding the input, which makes them indispensable. This is true for classification and segmentation tasks both, even though they result in a drastically reduced spatial resolution. In semantic segmentation one wishes to perform pixel wise predictions,

often in the original image resolution, which makes the pooling operations a major problem. The U-net architecture overcomes this challenge by the addition of a second decoder network which up-samples the features extracted from the first part of the network to produce an output mask in the original spatial resolution. While this is has been shown to be a reliable technique that has significantly improved the state-of-the-art, the design results in large networks which are computationally expensive. The architecture introduced in [3] and further optimized in [4] instead employs a technique called *dilated* convolutions – or convolutions *atrous* which translates to "with holes in" from French – which allows the construction of networks which retain the spatial resolution of the feature maps as the images are propagated through the network. The principal idea of the approach is to introduce holes in spatial dimensions of convolutional kernels to significantly increase the receptive field of the filters and maintain the spatial resolution of the feature maps while keeping the effective kernel size unchanged. Table 4.2 shows an example of a normal convolutional kernel with a receptive field of size 3×3 where the dilation rate is equal to 1. Table 4.3 illustrates the same convolutional filter with a dilation rate of two.

Table 4.2 An example of a 3×3 convolutional kernel with a dilation rate of one. Notice that the values in the kernel are set arbitrarily for illustrative purposes and bear no particular significance.

9	5	6
8	7	3
1	8	4

Table 4.3 The same convolutional kernel as described in table 4.2 but with a dilation rate of two, where each element from table 4.2 is highlighted. This means that the receptive field of the kernel is increased and only every other element in the input map is convolved with a non-zero element in the kernel, which is true for each spatial dimension. The sparsity of the filter is thus increased with a factor two for each spatial dimension of the input data.

0	0	0	0	0	0	0
0	**9**	0	**5**	0	**6**	0
0	0	0	0	0	0	0
0	**8**	0	**7**	0	**3**	0
0	0	0	0	0	0	0
0	**1**	0	**8**	0	**4**	0
0	0	0	0	0	0	0

The authors in [3, 4] use a mix of dilated networks and max-pooling operations in the VGG16 architecture to find a balance between performance and efficiency in their very

deep neural network architectures. Since the resolution of the images are kept as they are filtered through the network, this network design puts an additional load on the memory usage during training and inference. Large networks such as VGG16 are hard to train with purely full resolution images. It is possible however, to employ a smaller, "fully dilated network" for a large number of segmentation tasks that do not require such a large network as VGG16. In such smaller networks it is possible to completely skip the down-sampling steps commonly performed through max-pooling and maintain the original spatial resolution of the input images, as also done independently in [49]. This architecture has been included in the experiments as an alternative to the U-net based models, and we shall see that they produce competitive results while drastically reduce the complexity of the models.

Chapter 5

Experimental Setup

This chapter will cover the experimental setup used in the different scientific experiments conducted in the context of this thesis. The implementation details will be covered as well as the computational platform which has been employed for the training of the convolutional neural networks. Finally, a description of the data used in training and testing will be given.

5.1 Implementation Details

At the time of the writing of this thesis, one of the most popular deep learning libraries is the open source Keras library for Python [5]. Keras is a high-level wrapper library around either the TensorFlow or Theano libraries which are also based on the Python programming language. Keras is an easy to use option compared to many of its lower level competitors and was constructed to reduce time between scientific ideas and actual implementation. In the research conducted in the context of this thesis Keras has been used with TensorFlow as the back end library.

5.2 Computational Platform

Today, deep neural networks are almost exclusively trained on graphical processing units (GPUs) instead of the more traditional approach where training is performed on one or more CPUs. A GPU has extremely optimized parallelization capabilities, and since the nature of neural networks makes them well suited for parallelized training, modern GPUs can decrease training times significantly.

Due to a number of different reasons, performing the training of the DCNN models on local machines has been shown to be an inefficient strategy. For one, the size of the models

are of a scale that makes them hard to fit on all but the most powerful of GPUs. Secondly, when a training session is active on the local GPU the entire machine is locked down due to the large computational resource needed for the training which effectively locks all other activity on the machine.

Initial experiments were conducted on the GPU on a local machine, but when it became evident that the solution was unsatisfactory we moved to a cloud based solution where Amazon Web Services (AWS) was utilized. AWS is a highly customizable platform that supports training of DNNs through a smorgasbord of different libraries. In the context of this thesis we have used p2 xlarge instances for training which come equipped with powerful Nvidia Tesla K80 GPUs. In addition to presenting a solution to the aforementioned issues, the cloud based solution also provides the opportunity to run multiple experiments simultaneously on different server instances which has significantly increased the progress speed of the research through e.g. hyperparameter optimization.

5.3 Data Description

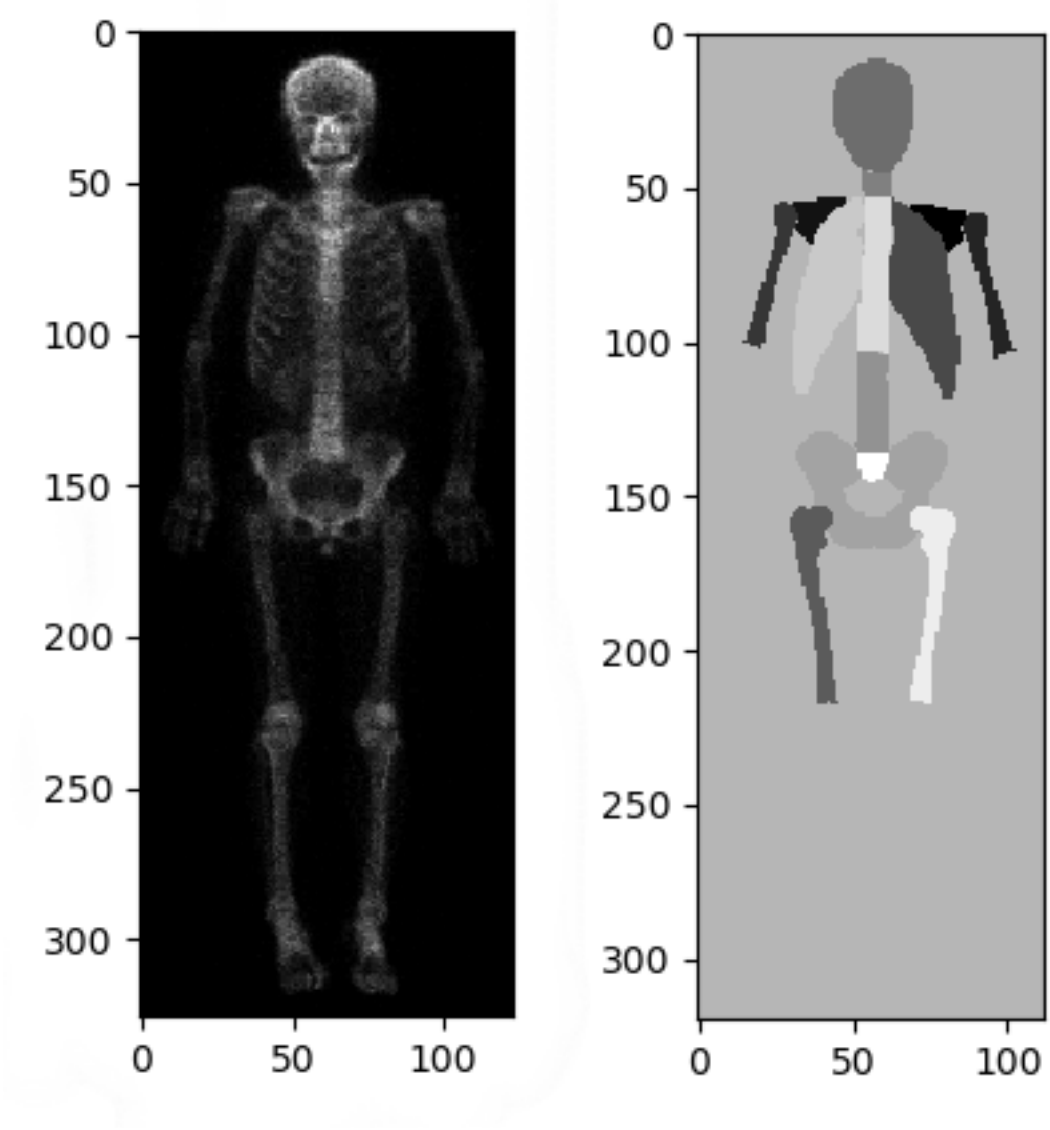
The data used in the herein described experiments consist of 2345 skeletal scintigraphy images used for training and 298 additional images used for testing. Each image comes with a corresponding ground truth segmentation where 14 different body parts and the background are distinguished by labels in the range of 0 to 14, see Figure 5.1 for an example of a bone scan and its corresponding ground truth segmentation. The labels used as the ground truth when training the convolutional neural networks have been constructed by image registration using the Morphon method via the commercially available Exini bone^{BSI} software.

The images have been down-sampled to half of the original resolution to speed up the scientific process. The models constructed in this thesis are therefore not necessarily usable in a practical context but should be viewed as prototypes for possible real-world applications, a proof of concept. In order to properly evaluate the models the training data has been split into a training set and a validation set. After the split we have the following number of images in the different sub-sets:

- *Training set*: 1759 images
- *Validation set*: 586 images
- *Test set*: 298 images

The training set and validation set has been used frequently during training to construct and evaluate the different models, whereas the test set has only been used at the end of the

Fig. 5.1 The leftmost image shows an example image chosen from the 2345 scintigraphy images. The rightmost image shows the corresponding ground truth labels for 14 different body parts and the background. The example image has the dimensions $326 \times 123 \times 1$ while the mask has $320 \times 112 \times 1$. This difference is due to the fact that the networks output masks which are slightly smaller than its inputs. This discrepancy was removed in the later models but the resolution of the images was retained to keep the continuity of the experiments and to be able to compare the different models as fairly as possible. The labels have been cut off to only include the upper arms and legs such that their lower equivalents have been removed. The decision to remove certain body parts was made since they have little relevance to the experiments and to keep the number of classes moderately small.



work to evaluate the models. No additional alterations have been made to the models post test set evaluation. This means that the models have been trained on the training set and evaluated

on the validation set during the research process. The validation set is an estimation of the true error and has been used as a base for hyperparameter selection and used to compare different model architectures. The performance gained on the test set is a closer estimation of the true performance since no model engineering has been done on the dataset.

Chapter 6

Method

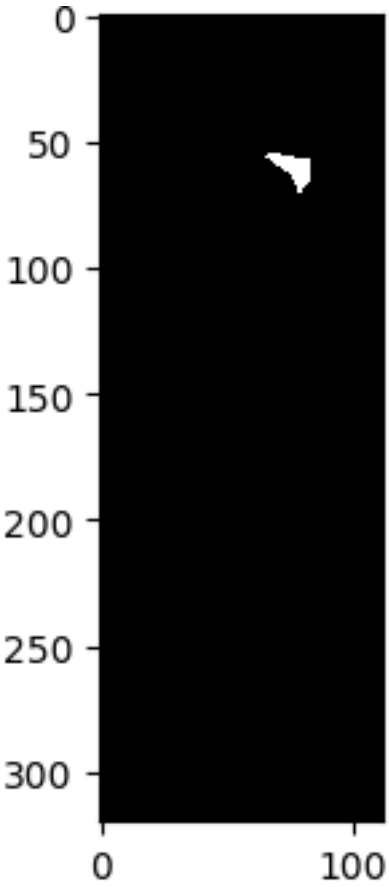
This chapter covers the structure of and describes the philosophy behind the architectures used in the herein performed experiments. There are primarily two architectures that have been investigated in the context of this project: the U-net structure introduced in [31] which has shown remarkable performance in medical applications given only very little training data, and the convolutional neural networks atrous introduced in [3] and further improved in [4]. The U-net structures have been further extended with ideas introduced in [39] and [16] to further increase performance and reduce training time. The atrous approach provides an interesting architecture that is works on fundamentally different principles than those behind U-net design.

While the raw labels contain 15 different classes (including the background), the segmentation of all 15 classes at once was found to be a problem which is significantly harder than the segmentation of just one class at a time in a one versus all scenario. The latter problem has therefore been the main focus of this thesis, even though attempts to perform multi-class – or semantic – segmentation have been performed. In all of the subsequent discussion, the problem at hand is the one versus all segmentation problem where class zero has been extracted from the labels and are depicted with ones while the rest of the image consists of zeros, see Figure 6.1 for an example of such a label. Class zero depicts the patients' left scapula.

6.1 U-net

The full U-net model is by far the heaviest model constructed in the herein performed experiments, with approximately 34 million trainable parameters. The U-net architecture has been replicated exactly as depicted in [31] with two alterations: (1) batch normalization has been introduced after each convolutional operation and (2) dropout has been added to the

Fig. 6.1 A random sample from the validation set which shows class zero – the patients left scapula – v.s. the background. The scapula is deemed to be in the middle range when it comes to the complexity of the problem due to its size and shape, as compared to the other body parts. Results gained on this class can not transferred directly to other classes/body parts but provides a good way to compare the different algorithms and gives a rough indication of the quality of the segmentation of the other classes.



deepest part of the network as well as after each concatenation step in the decoder network, see Figure 6.3 for a conceptual graph of the implemented version of the U-net model. Batch normalization [17] is a technique introduced to prevent internal covariate shift as the data is filtered through a network, and the technique has been shown to reduce training time, prevent overfitting and increase performance in DCNNs. Batch normalization is today an easily implemented and commonly used technique to perform an overall enhancement to deep neural networks and to make them more robust.

For segmentation tasks it is common to use some form of cross entropy as the loss function when training the neural networks. In the problem described in this thesis however, these functions provided a segmentation which was significantly inferior to other tested functions. The problem contains two classes, where the background is extremely dominant

Table 6.1 An illustration of the some of the hyperparameters and other relevant information used in the training of the U-net based models. From left to right the different columns in the table illustrate the model used, the utilized optimizer, the learning rate, the loss function, the batch size, the number millions of parameters used in the model, the number of seconds it took to complete each epoch and the number of epochs used during training. Do note that while the same number of epochs have been used for training in both the standard U-net and the URI-net, and while URI-net has a higher training time per epoch it did converge significantly faster than the standard U-net. Should early stopping be used, the URI-net would have a net-training time which is lower than the U-net even if each epoch takes more time to complete. Apart from the learning rate, the standard hyperparameters used in the Keras deep learning library have been used for the Adam optimizer.

Model	**Opt.**	**LR**	**Loss**	**Batch**	**Params**	**S/Epoch**	**Epochs**
U-net	Adam	$\frac{1}{8} * 10^{-5}$	-log(Dice)	16	34	512	200
URI-net	Adam	$\frac{1}{16} * 10^{-5}$	-log(Dice)	16	21	726	200

and makes up about 99.50% of the images. It is believed that if one weights the classes to even out this unbalance, the cross entropy function would perform better. However, instead of exploring this option we have employed a version of the Sørensen–Dice coefficient, also referred to as dice score or dice coefficient

$$D(X,Y) = -log\frac{2|X \cap Y|}{|X| + |Y|} = -log\frac{2\sum_i x_i y_i}{\sum_i x_i + \sum_i y_i}, \tag{6.1}$$

as the loss function for the convolutional networks. Notice that the negative natural logarithm have been taken of the index and used as a loss function rather than the raw coefficient.

In equation (6.1) X denotes the generated predictions and Y the corresponding ground truth labels. Notice that during the training phase, while Y is a binary matrix, X contains the predicted beliefs at each pixel which can be any value between zero and one. This applies to all models presented and discussed in the subsequent chapters. The nature of the function is such that the predictions will be driven to its edges, either zero or one, to generate as high a dice score as possible. Through empirical experiments we have observed that this loss function is suitable for such a task as described in this thesis and works very well even when the classes are heavily unbalanced. See Table 6.1 for an overview of the hyperparameters and settings used when training the U-net model.

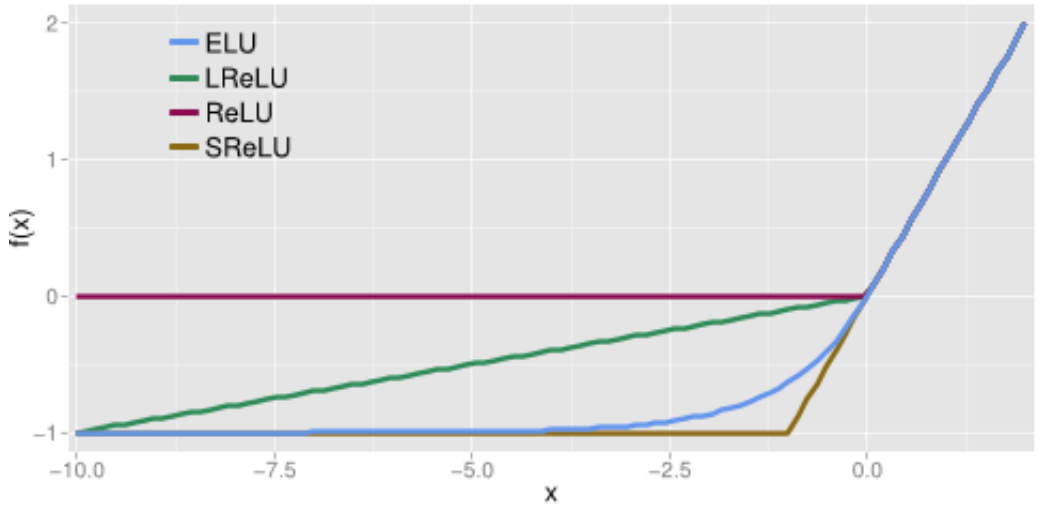
6.2 URI-Net

While the U-net model provides a solid baseline for both architecture design and segmentation performance it is too heavy a model and does not perform well enough to be completely satisfactory. The large amount of parameters make it prone to overfitting and lead to a high computational cost during training. By the introduction of a number of alterations to the original architecture we hope to achieve a significant reduction in model complexity, an increase in performance and a decreased training time. The alterations are primarily inspired by the work in [44] and [25].

The first alteration is the introduction of the inception module [39], as described in Section 4.3. Instead of two convolutional layers with the same number of feature maps in each step in the U-net "staircase", each step has one inception module followed by on of two options: either a convolutional layer with a stride of two used for down-sampling if in the encoder network or a transpose convolutional layer used for up-sampling in the decoder network. The convolutional layers with stride replace the max-pooling layers and thus, the only max-pooling operations in the entire network are those found in the inception modules which has a stride of one. The second major alteration is the introduction of residual connections [16] instead of the normal skip connections used in [31]. Contrary to the original design o the building block, only one convolutional layers is used in the residual building block in the skip connections rather than two, as in [16]. These two major alterations to the original U-net architecture have significantly changed the structure of the model, where the resulting architecture has been named the URI-net due to its design which is based on the U-net model, the residual building block and the inception module. In addition to these major changes, the activation function after each of the convolutional operations have been changed to the exponential linear unit (ELU) [7] instead of the standard rectified linear unit as described in Section 3.4.3. The ELU activation function is a smoothed version of the rectified linear unit as can be seen in Figure 6.2. The principle behind the ELU function is that it pushes the mean activation value closer to zero by the inclusion of negative values. In theory the switch in activation function from ReLU to ELU will speed up learning and increase the performance of the models. All the aforementioned alterations to the standard U-net model resulted in a substantial decrease in the number of parameters used in the model. See Table 6.1 for a list of the hyperparemeters and settings used in the URI-model as well as a comparison to the standard U-net structure.

The raw labels, as described in section 5.3, contain 15 classes. While models have been trained to perform a semantic segmentation of all 15 classes at once, the technique which yielded the highest accuracy was to construct 15 different networks which were trained independently on each class. Figure 6.5 shows an illustration of the final model with 15

Fig. 6.2 A comparison of the exponential linear unit and a family of rectified linear unit activation functions [7]. The ELU function is drawn with a blue line while the standard ReLU is drawn with a dark purple color.



smaller nets as constituents which perform a complete segmentation of the entire image with 15 different classes. Each network consists of approximately 20 million parameters, which means that the complete models is made up of about 300 million parameters. This ensemble model which is able to perform multi-class segmentations on the entire body has been named the multi-class URI-net, or mcURI-net. This experiment has only been carried out with the URI-net since it showed the greatest promise before the test performance was revealed and the time constraints of the project prevented the production of additional such networks.

6.3 Atrous Convolutions

In a separate branch of research we have looked at convolutional neural network that employ dilations in the convolutional filters. While the U-net architecture splits the model into an encoder and a decoder part, the dilated convolutional networks maintain a simpler and more lightweight architecture and instead perform alterations to the convolutional kernels. The dilated filters allow the removal of pooling operations in the network while retaining the increase in receptive field that would have come naturally with e.g. max-pooling. The experiments conducted with dilated convolutions have largely been inspired by [3, 4], where they still use some max-pooling layers to make a trade-off between accuracy and efficiency. When the pooling operations are cut from the model the images are propagated throughout the network in their original spatial resolution which can be extremely memory heavy for deeper networks. The trade off is necessary since the problem the authors of [3, 4] have

taken on is extremely difficult and only the largest of models can achieve satisfactory results. The herein described problem is believed to be significantly easier, and thus we do not have to make the same trade off. Instead we can employ what we refer to as fully dilated networks, that only include an increased dilation rate as the network depth increase and no decrease in the spatial resolution what so ever, similarly to the experiments conducted in [49]. The architectures that have been studied follow the theme of the VGG nets [36] where the number of feature maps increase with the depth of the network.. Experiments have been carried out with different architectures, see Table 6.2 for an overview of the different network architectures employed in the described experiments. As with the VGG networks, architectures with different depths have been examined. To be able to easily distinguish and hold a discussion around the different networks they have been named "FDN-N", where FDN stands for "fully dilated network" and N for the number of convolutional layers used in the network.

The main differences from the standard VGG architecture is that the FDN networks have significantly less feature maps and instead of performing sub-sampling of the data in the max-pooling layers, their stride is set to 1 and the convolutional layers after each max-pooling layer has increased their dilation rate with a factor two. I.e. after the first max-pooling layer the dilation rate is set to two and after the second max-pooling layer the dilation rate is set to four etc. All networks described in Table 6.2 have used a learning rate of 4×10^{-5} with the Adam optimizer where all additional hyperparameters have been set to the standard values as specified in the Keras deep learning library. The negative logarithm of the dice score seen in equation (6.1) has one again been used as the loss function during training. Each network has been trained for 100 epochs each, which has been observed to provide a sufficient number of iterations for the networks to converge.

In addition to the different architectures, experiments have also been carried out with different activation functions and with the introduction of batch normalization. Experiments of this nature has been carried out on FDN-11 due to its good trade-off between accuracy and training time as well as on FDN-16 to measure their impact on deeper/heavier networks.

Table 6.2 A compilation of the different architectures used in the experiments which include the convolutional neural networks atrous. Each column shows the layer by layer architecture associated with each network. The receptive field is given for each convolutional layer as well as the number of resulting feature maps, e.g. Conv3 $\times$ 3-64 indicates that the layer has 64 filters with a receptive field of 3×3 and thus produces 64 feature maps. In addition to the architecture of the networks, the number of parameters for each model and the training time for each epoch in seconds is given in the second to last and last row correspondingly.

FDN-5	**FDN-11**	**FDN-13**	**FDN-16**
Conv3 $\times$ 3-64	Conv3 $\times$ 3-8	Conv3 $\times$ 3-8	Conv3 $\times$ 3-8
Conv3 $\times$ 3-64	Conv3 $\times$ 3-8	Conv3 $\times$ 3-8	Conv3 $\times$ 3-8
Max-Pooling	**Max-Pooling**	**Max-Pooling**	**Max-Pooling**
Conv3 $\times$ 3-128	Conv3 $\times$ 3-16	Conv3 $\times$ 3-16	Conv3 $\times$ 3-16
Conv3 $\times$ 3-128	Conv3 $\times$ 3-16	Conv3 $\times$ 3-16	Conv3 $\times$ 3-16
Max-Pooling	**Max-Pooling**	**Max-Pooling**	**Max-Pooling**
Conv1 $\times$ 1-1	Conv3 $\times$ 3-32	Conv3 $\times$ 3-32	Conv3 $\times$ 3-32
	Conv3 $\times$ 3-32	Conv3 $\times$ 3-32	Conv3 $\times$ 3-32
	Max-Pooling	Conv3 $\times$ 3-32	Conv3 $\times$ 3-32
	Conv3 $\times$ 3-64	**Max-Pooling**	**Max-Pooling**
	Conv3 $\times$ 3-64	Conv3 $\times$ 3-64	Conv3 $\times$ 3-64
	Max-Pooling	Conv3 $\times$ 3-64	Conv3 $\times$ 3-64
	Conv3 $\times$ 3-128	Conv3 $\times$ 3-64	Conv3 $\times$ 3-64
	Conv3 $\times$ 3-128	**Max-Pooling**	**Max-Pooling**
	Max-Pooling	Conv3 $\times$ 3-128	Conv3 $\times$ 3-128
	Conv1 $\times$ 1-1	Conv3 $\times$ 3-128	Conv3 $\times$ 3-128
		Conv3 $\times$ 3-128	Conv3 $\times$ 3-128
		Max-Pooling	**Max-Pooling**
		Conv1 $\times$ 1-1	Conv3 $\times$ 3-256
			Conv3 $\times$ 3-256
			Conv3 $\times$ 3-256
			Max-Pooling
			Conv1 $\times$ 1-1
259,137.0.	295,033.0	488,793.0	1,964,249.0
129	208	314	1034

Fig. 6.3 Visualization of the U-net model which has been employed in this thesis. Notice that in addition to the standard U-net structure, batch normalization and dropout layers have been added to correspondingly make the model more robust and decrease overfitting.

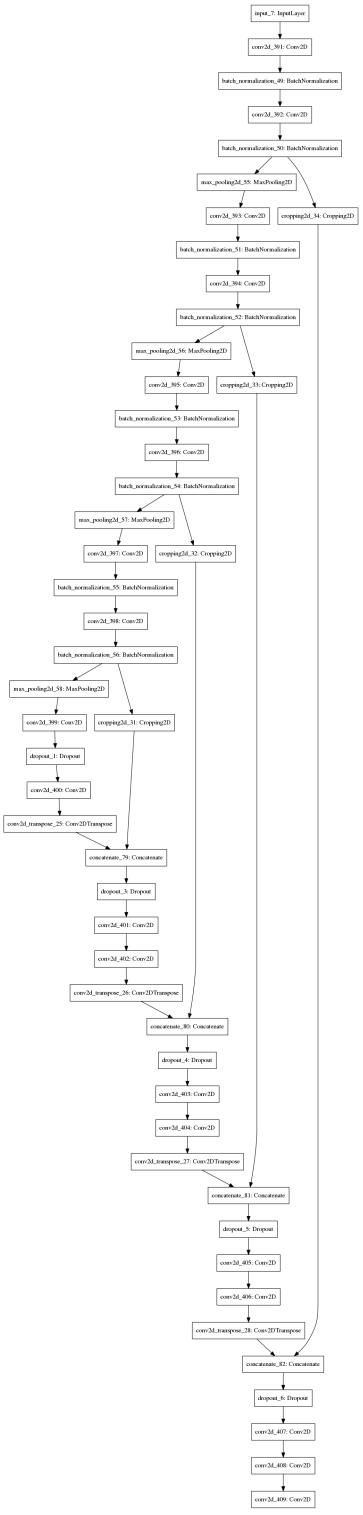


Fig. 6.4 A graphical illustration of the modified U-net structure named URI-net. The main modifications that distinguish the model from the standard U-net architecture is the replacement of the standard convolutional layers with inception modules [39] and the introduction of residual blocks [16] instead of standard skip connections. Other modifications are the use of ELU [7] instead of ReLU, convolution with a stride of two instead of sub-sampling by max-pooling and a general decrease in the number of feature maps.

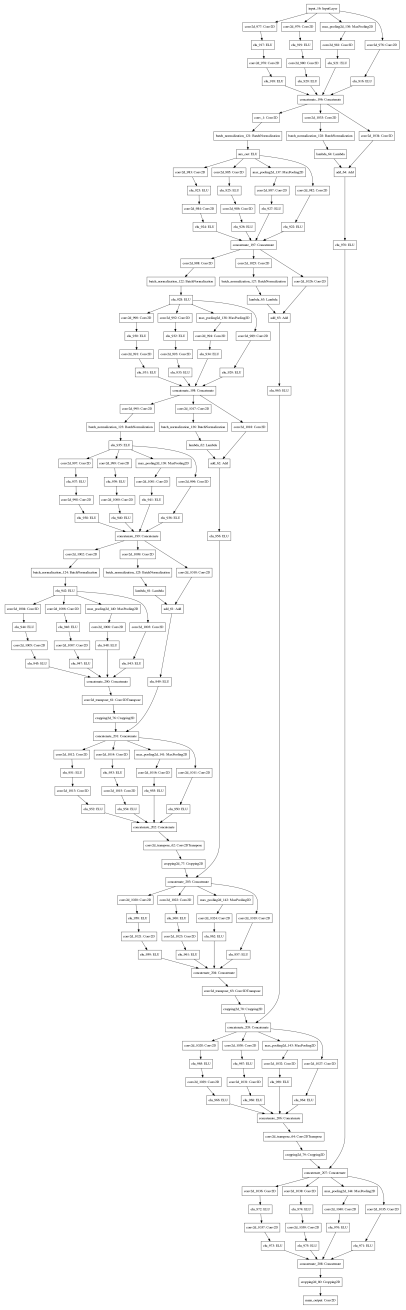
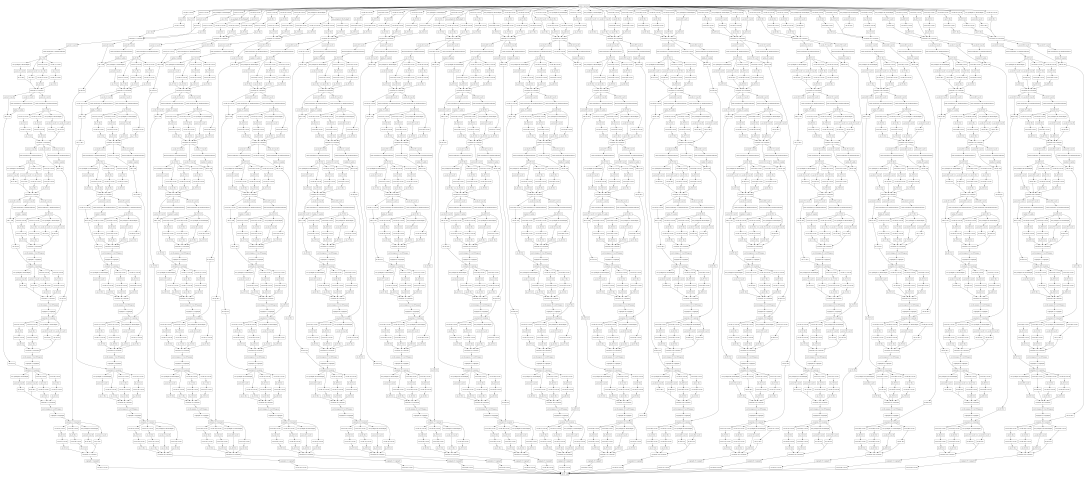


Fig. 6.5 15 different URI-nets have been merged to create a model which is able to perform a segmentation on whole input images where each individual class is identified, located and given a pixel wise segmentation. The model is made up of 15 different networks which have been trained on one of each of the 15 different classes. Each network contains about 20 million parameters which adds up to a total of circa 300 million parameters for the entire, multi-class URI-net model. The constituent networks perform their segmentations independently and their resulting prediction maps are concatenated in a $row * column * N$ volume where N represents the number of classes which is 15 in this case. The index of the network which has given the highest predictive value for a certain pixel is chosen as that pixel's class, i.e. an argmax operation is performed on the output volume of the 15 different networks to perform the final segmentation. The multi class model is denoted as a mcURI-net.



Chapter 7

Results

Table 7.1 shows a summary of the performance of all models produced in the context of this thesis on the training set, validation set and test set on the one versus all problem where the left scapula has been distinguished from the rest of the images.

Table 7.1 A comparison of the different produced models. A family of convolutional neural networks atrous is compared to the U-net based models as well as a dilation-less version of the FDN-11 network denoted DLN-11 (dilation-less network). The standard settings for the dilated networks is to use the ReLU activation function and no batch normalization. Some networks have been trained with ELU and some with batch normalization, in which cases an e or a b have been added to the model name. The first column shows the final validation loss for the different models, while the second, third and fourth columns show the training, validation and test error correspondingly where each row consists of one model. The best value achieved in the different data sets has been highlighted. FDN-16eb achieved the highest training dice score, FDN-16e the highest validation score and FDN-13 the best test score.

Network	Validation loss	Training Dice	Validation Dice	Test Dice
DLN-11	0.1672	0.8661	0.8466	0.8212
FDN-5	0.3026	0.7421	0.7417	0.5572
FDN-11	0.0566	0.9472	0.9459	0.9219
FDN-11e	0.0520	0.9523	0.9528	0.6039
FDN-11be	0.0559	0.9648	0.9501	0.9220
FDN-13	0.0490	0.9570	0.9526	**0.9341**
FDN-16	0.0466	0.9682	0.9546	0.9105
FDN-16e	**0.0438**	0.9842	**0.9572**	0.9286
FDN-16be	0.0533	**0.9938**	0.9481	0.9203
U-Net	0.0933	0.9523	0.9110	0.8591
URI-Net	0.0516	0.9881	0.9498	0.8289

Table 7.2 shows the performance of the mcURI-net on the training set, validation set and test set, where each class/body part has been segmented individually by different URI-nets. The table also illustrates the mean values and standard deviations of the scores generated on the different data sets. Figure 7.1 illustrates an example segmentation made from a randomly selected image in the validation set, and Figure 7.2 shows a segmentation of a randomly selected image from the test set. Like in Figure 7.1, the full body segmentation in Figure 7.2 has been produced with the mcURI-net, where each class has been segmented independently by different URI-nets. Finally, three different segmentations performed on samples from the test set by the FDN-13 model can be found in Figure 7.3.

In addition to the experiments carried out on a single of the 15 available classes, attempts to perform a semantic segmentation on all classes at once with a single network have been carried out. Figure 7.4 gives qualitative results from a few different network configurations of the multi class segmentation performed with the URI-net and the standard U-net on the validation set.

Table 7.2 The performance of the URI-net model on the 15 different classes for the different data sets. Each row depicts a class, where the first column shows the class name and the second, third and fourth columns the corresponding training score, validation score and test score. The last two rows show the mean dice score for each data set as well as the standard deviation of the dice scores in each set.

Class	**Training Dice**	**Validation Dice**	**Test Dice**
Left Scapula	0.9881	0.9498	0.8289
Right Scapula	0.9833	0.9511	0.8106
Left Humerus	0.9795	0.9390	0.7963
Right Humerus	0.9810	0.9423	0.8038
Left Costae	0.9899	0.9706	0.8824
Right Femur	0.9801	0.9444	0.6979
Cranium	0.9869	0.9818	0.8238
Cervical Vertebrae	0.9738	0.9311	0.7468
Lumbar Vertebrae	0.9799	0.9461	0.8110
Pelvis	0.9752	0.9568	0.7793
Background	0.9889	0.9883	0.9430
Right Costae	0.9820	0.9659	0.8738
Thoracic Vertebrae	0.9770	0.9567	0.8498
Left Femur	0.9596	0.9382	0.6710
Sacrum	0.9722	0.9138	0.7066
Mean score	0.9798	0.9517	0.8016
Standard Deviation	0.0075	0.0187	0.0711

Fig. 7.1 An example segmentation with mcURI-net on a randomly selected image from the validation set. The leftmost image shows the ground truth, i.e. the labels which have been produced by traditional image registration using morhpons. The image in the middle is the resulting prediction as produced by the mcURI-net where the mean dice score for the validation set was 0.9517. The rightmost image shows the input image and and an overlay of the corresponding predictions as constructed by the mcURI-net. The color of each pixel does of course indicate its class.

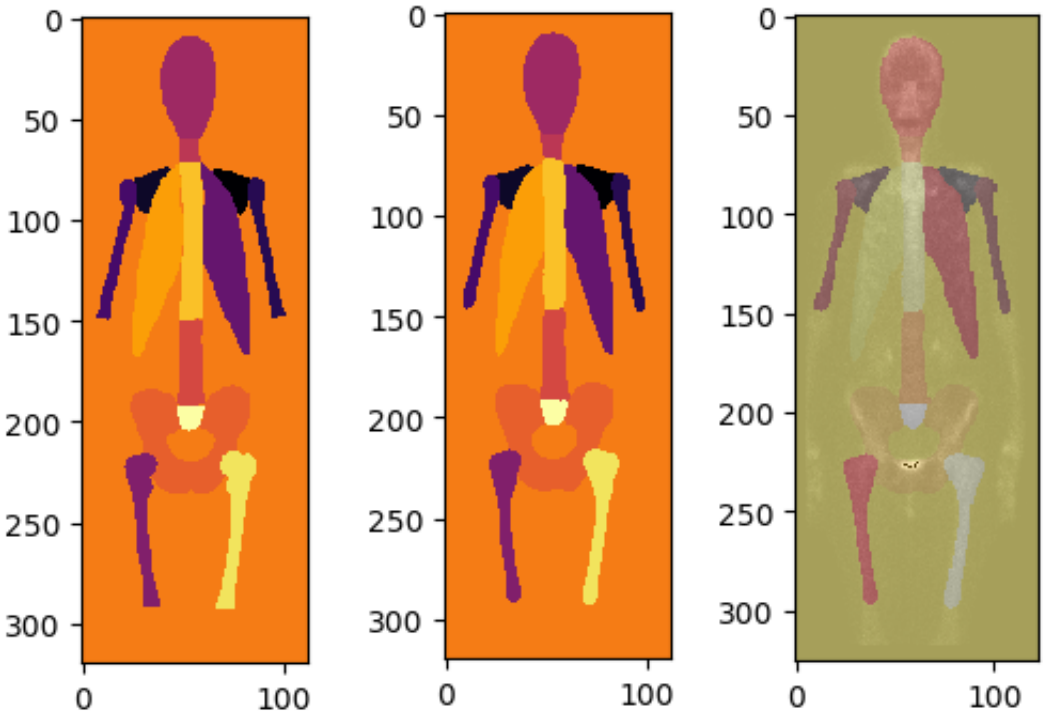


Fig. 7.2 A full body segmentation produced by the mcURI-net on a randomly selected image from the test set. The labels in the test set have different values relative the training and validation sets. This bears no significance on the model performance since each class is treated individually, but the coloring of the test set looks slightly different when plotted as compared to the other sets.

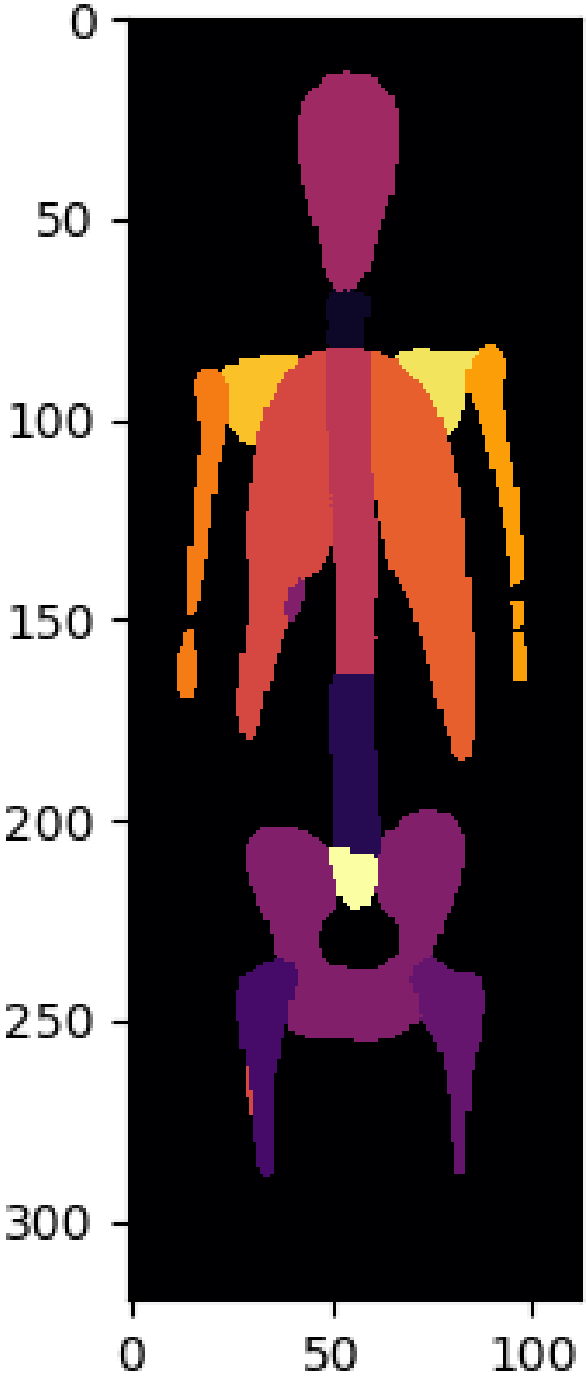


Fig. 7.3 Each row shows a segmentation-label pair made with the best performing model on the test, i.e. the FDN-13 model, where the left column contains the segmentations generated by the model and the right column the corresponding label seen as the ground truth. Each example image has been drawn randomly from the test set. The mean dice score on the entire test set is 0.9341, as can be seen in Table 7.1.

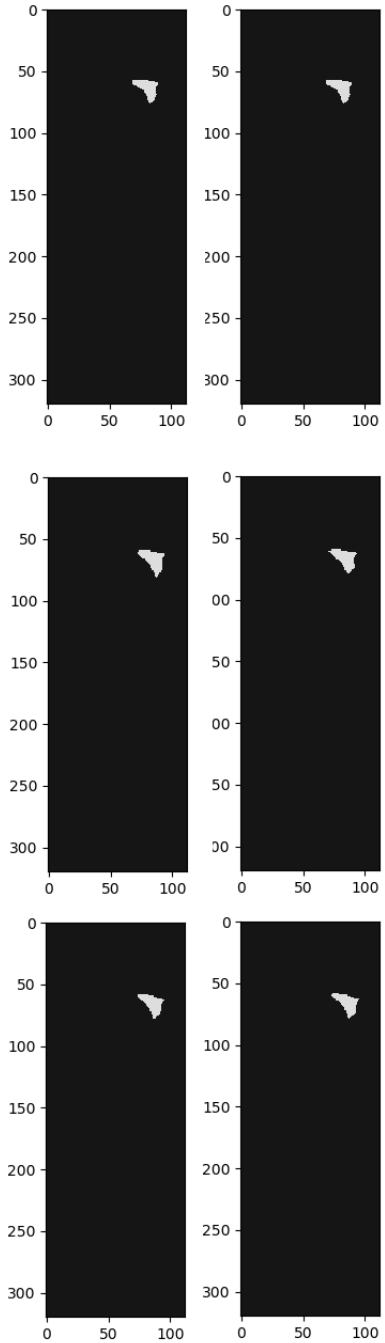
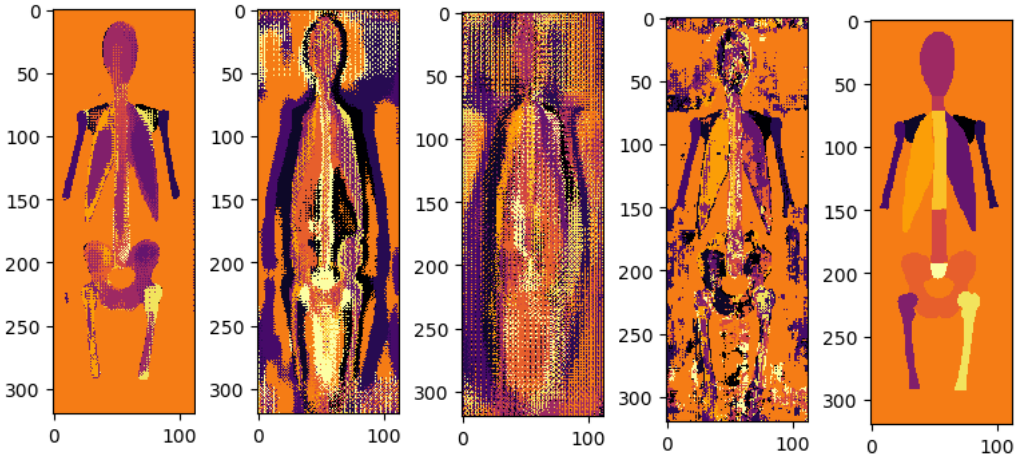


Fig. 7.4 Experiments performed with multi-class segmentation using only a single convolutional neural network on a randomly selected image from the validation set. The leftmost image shows an attempt with the standard U-net model to segment all classes at once. The three images in the center illustrates different attempts to perform a semantic segmentation of all 15 classes with different configurations of the URI-net. The rightmost image shows the ground truth labels of the image for reference.



Chapter 8

Discussion

This chapter will first hold separate discussions of the different branches of research conducted in this thesis with regards to the results presented in Chapter 7. Thereafter a general discussion will be made where final conclusions will be drawn as well as thoughts on possible extensions and future work will be given.

8.1 U-net

As can be seen in Table 7.1, the URI-net model performed much better than the standard U-net model on both the training set and validation set. The mean dice scores increased by about four percentage points in both the training set and validation set, where the training and validation scores were 0.99 and 0.95 respectively as compared to the dice scores of 0.95 and 0.91 achieved with the standard U-net model. In addition to an increased performance, URI-net also reduced the amount of parameters by 43% from the standard U-net model, from about 34 million to roughly 20 million parameters, which reduces the model complexity as well as training time and in theory should make it less susceptible to overfitting. These facts did indicate that the URI-net would perform better than the U-net on the test set as well, which proved a faulty assumption since the URI-model only managed to achieve a mean dice score of about 0.83 on the test set while the standard U-net model achieved a mean dice score of approximately 0.86. Both scores are significantly worse than the scores achieved in both the training set and the validation set and destroyed the intuition of which of the two models would perform the best out of sample. In lieu it appears that the URI-net model has managed to overfit the training data to a larger extent than the U-net model, in spite of the large number of "tricks" employed in the model to reduce such an occurrence as well as the considerable amount of dropout used throughout the model. This fact is also evident from the example segmentations presented in Chapter 7. Figure 7.1 illustrates an

almost flawless segmentation on an example from the validation set while Figure 7.2 shows a less than ideal segmentation of a randomly selected image from the test set segmented with the same model. This indicates that the mcURI-net model has overfitted the training samples. It is hard to tell which parts of the URI-net caused the decreased performance since the individual alterations have not been examined individually. While it would be interesting to identify the effect of each individual alteration incorporated in the URI-net as compared to the original U-net structure, this would be too costly to conduct in the context of this work. Instead, we have settled for the conclusion that the URI-net model can be a powerful model in some segmentation tasks [44] but can easily fit the training samples too well, which results in poor generalization capabilities. The same argument can be made for the standard U-net model, for while Ronneberger et al. [31] achieved impressive results on various data sets, but the model failed to generalize well in the herein described problem domain despite the addition of dropout in numerous locations in the network.

Since the URI-net was believed to be the best constructed model – based on the achieved results in the training set and validation set – it was used to perform fully-body segmentations in the mcURI-net ensemble model. The result can be seen in Table 7.2. The results corresponds well to the quality of those obtained from the experiments performed on the left scapula. The mean dice score over all classes in the training set was 0.9798 and the corresponding standard deviation was 0.0075, which is an extremely good score. Since the labels used as ground truth during the training phase are automatically generated by image registration using morphons, an overlap measure of approximately 0.98 might in fact be too good a score, where the CNN is learning to act like a morphon rather than to learn the actual actual structure of and patterns in the data. Such a high training score might also be an indication of overfitting, which we see clear trends of in the validation set, and especially in the test set. The dice score is reduced by about three percentage points from the training set to the validation set and by another 15 percentage points from the validation set to the test set. We also see that the results become less uniform in the test set as compared to the other sets since the standard deviation is increased by a factor 10 between the training set and the test set.

While the main focus of this thesis has been a one class segmentation where a single class has been distinguished from the background, experiments have also been made with multi-class semantic segmentation where some example results can be seen in Figure 7.4. As the figure clearly illustrates, the task has been all but successful. Interestingly, the standard U-net model managed to achieve a higher performance than the URI-net model, even though the latter achieved a higher dice score on the same set (the validation set) on the one versus all problem. It is not known why the models failed to perform multi-class segmentations

since this is not the line of research which has been the main focus of this project. Possibly, the poor results might be due to the relatively small size of the multi-class models. The mcURI-net for example is of a magnitude which is 15 times larger than the models which attempt multi-class segmentation. Perhaps deeper architectures would fare better than the ones examined for this project. Another significant factor is believed to be the loss function. The performance gained from a transition from the standard cross-entropy function to our dice score was very significant, so it is possible that a reformulation of the loss function would make the multi-class models work as well. Another simpler explanation is that the hyperparameters are simply tuned incorrectly for multi-class segmentation. For example, the learning rate might be set to a value which has shown to be suitable for segmentations of a single class, but ill fitted for work on many classes at once.

8.2 Atrous Convolutions

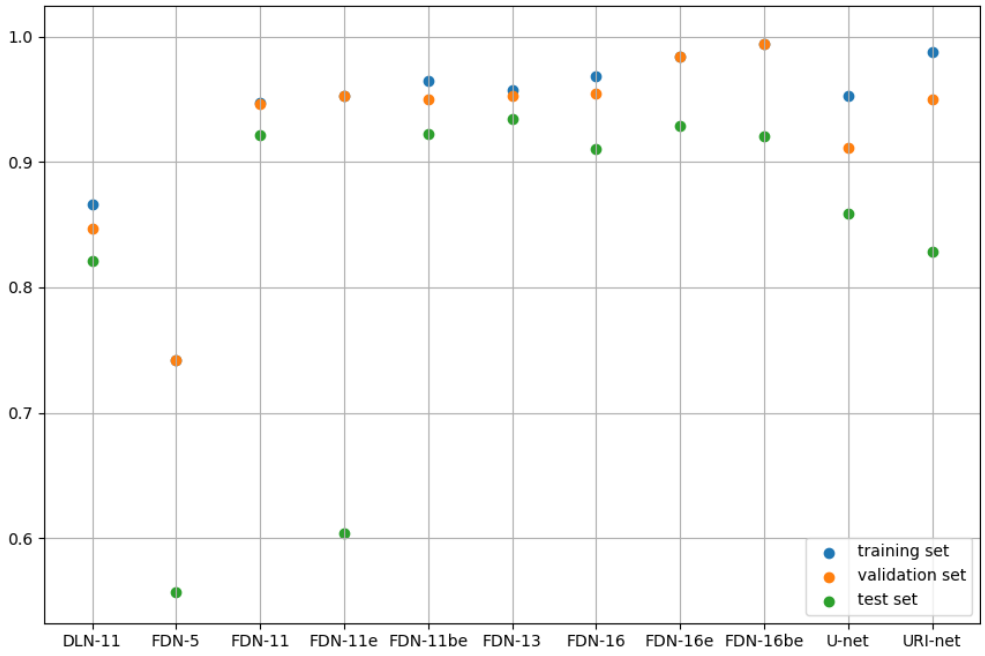
Contrary to the experiments with the U-net based models, those made with the dilated convolutional neural networks have been very successful. The addition of the scheme of dilation rates alone in the FDN-11 network decreased the out of sample error by 56.32%, see a comparison between the mean dice score on the test set for DLN-11 and FDN-11 in Table 7.1 where the only difference between the two networks is that DLN-11 lacks the dilation scheme applied in the FDN networks. Evidently, the increased receptive field of the deeper convolutional kernels that comes with an increased dilation rate is essential for the model to understand the data and perform accurate segmentations. The main disadvantage of the dilated networks is the heavy memory requirements. Since the spatial resolution is maintained as the image is propagated through the network – which was the hole point with the dilated kernel approach – it significantly increases the memory requirements. This is a potential problem for data sets with images of a larger resolution or data of higher dimensionality.

An observation of the training and validation scores indicate that the deeper the network, the better the performance of the model. The overall best training score was with the FDN-16be model which achieved an in sample error of 0.0062. Clevert et al. [7] found no improvement with the introduction of batch normalization in their models when ELU was applied, which also corresponds well to the results obtained in this work, see Table 7.1. While FDN-16be achieved the highest global mean dice score in the training set, it was its sister model, FDN-16e, which achieved the highest validation score and also outperformed the FDN-16be network in the test set. Batch normalization is as already discussed a normalization technique to prevent internal covariate drift within the models, and is an effective technique

for decreasing training time while increasing the performance of the models. The exponential linear unit however does already perform a sort of normalization, since it forces the mean activation closer to zero. In other words, as is in line with the theoretical argumentation, batch normalization has been found to be redundant or even detrimental to the out of sample performance when the activation function used in the network is the exponential linear unit. Furthermore, Clevert et al. [7] also argue that the ELU function can serve as a form of regularizer and can in some cases remove the need entirely for dropout in convolutional neural networks. These arguments also correspond well to our experiments, where FDN-16e shows a higher validation score as well as a higher test score compared to the basic FDN-16 model which employs the standard ReLU activation function. However, an internal comparison within the different data sets for FDN-16e shows that FDN-16e still suffers from a significant amount of overfitting and thus might benefit from the addition of dropout. Do notice that no regularization techniques apart form the ELU function have been incorporated in any of the dilated models.

Interestingly, it is FDN-13 – the second largest model which is also the model which has been experimented with the least – that achieved the highest performance in the test set of all the models constructed for this thesis, and is thus crowned as the best model of all the trained networks. As already mentioned, we see a clear correlation between the depth and size of the models and their performance, where deeper and larger models generally perform better than their shallower counterparts. Compare the performances of the FDN-5, FDN-11, FDN-13, and the FDN-16 networks on the different data sets in Table 7.1, where FDN-11e is the only outlier with an exceptionally bad dice score in the test set. It is unknown what lead to this poor generalizing capability, but some hypothetical reasons could be extremely bad random initialization or that such a shallow network becomes too sensitive to variations in the data when trained with the ELU activation function. While the performance increases with the depth of the networks, it is important to remember that with an increased complexity there is an increased risk of overfitting. This is also represented in the results in the family of FDN-16 networks where FDN-16 and FDN-16e both achieved a higher dice score in the training set and validation set than FDN-13, but appears to have been overfitting the training data to the point where they performed worse than FDN-13 on the test data. See Figure 8.1 for an additional, more graphical, illustration of the performance of the different models on the different data sets. It is believed that the addition of dropout in the heavier networks might increase their out of sample performance, but by how much is not clear since it is difficult to have a good intuition when it comes to the generalizing abilities of deep neural networks [50].

Fig. 8.1 This figure gives a graphical overview of the performances of the different models on the three different data sets: training set, validation set and test set. The X-axis shows the different produced models where the Y-axis shows the corresponding dice score. The training set is illustrated as blue dots, the validation set as orange dots and the test set as green dots. We see a clear trend in the the training and validation sets for the dilated networks where the dice score increases with the depth. The test score however peaks at FDN-13 and then decreases with the FDN-16 networks due to overfitting. For some models the training set and validation set have almost identical values, see e.g. FDN-5, FDN-11e, FDN-16e and FDN-16be.



8.3 Conclusion

When the dilated models are put against the U-net based architecture we have observed that the dilated networks outperform the U-net models in almost every way. First of all, they achieved a significantly better dice score on the out of sample data, where FDN-13 reduced the test error by 50.67% as compared to the standard U-net and 59.57% relative the URI-net. Secondly, they are significantly easier to implement. This is mostly due to the authors of [3, 4] who added their dilated implementation in the open source library TensorFlow, which has been used as the back end library for the duration of this thesis, but also due to the fact

that the architectures are conceptually simpler. Thirdly, the dilated networks do generally require less training time than the U-net models. The exception is the FDN-16 family of networks which required a training time which was slightly longer than the U-net models.

Future work with the dilated networks include the addition of regularization, especially in the more complex models such as FDN-16. While we have established that the depth of the network is relevant for its performance, there is still a large number of established architectures which remain untested in this problem domain. Chen et al. [3, 4] experiment with different sampling techniques and introduce the Atrous Spatial Pyramid Pooling (ASPP) where the image is processed through different channels with different dilation rates and found that it had a positive impact on the model performance. Such an architecture would indeed be interesting to study, as well as entirely novel architectures using a dilation scheme. Perhaps it is possible to incorporate structures such as the inception module and the residual building block in dilated networks as well. Perhaps an ensemble of networks with configurations of the aforementioned components can be used to take the segmentation performance to new heights. In addition to these experiments as well as tweaking other hyperparameters, the next big step is to work on multi class segmentation. If one network would be able to properly segment all 15 classes at once it would significantly decrease the time required for both training and inference. Another line of research which has not been discussed in this thesis is the addition of fully connected Conditional Random Fields (CRF) as post-processing tools used on the output of the convolutional neural networks [3, 4]. This might a useful addition to multi-class segmentations, where the CRF can help make the segmentation more exact and remove noise.

All in all we can conclude that deep convolutional neural networks is good enough a technique for use in medical image analysis, at least as a powerful segmentation tool. Some of the networks achieved a negligible in-sample error and has thus been successfully trained to act like a Morhpon segmentation-by-registration algorithm. We also see that some networks perform really well in the out of sample data, such as FDN-13, which shows a remarkable generalizing capability. Such a network might in fact be superior to the traditional image registration technique used as ground truth, since the latter is incapable of performing exact segmentation but instead provide a rougher estimation. Even if the DNNs is found to perform worse than the morphon after qualitative evaluation of experts, it is likely that they would outperform their predecessor given accurate labels constructed by human experts. The work done in this thesis gives a strong foundation for further research for image segmentation with convolutional neural networks on nuclear medical data .

References

[1] Abu-Mostafa, Y. (2012). Machine learning course - cs 156. https://www.youtube.com/watch?v=EQWr3GGCdzw&list=PLD63A284B7615313A&index=11. (Accessed on 03/27/2017).

[2] Anand, A., Morris, M. J., Kaboteh, R., Båth, L., Sadik, M., Gjertsson, P., Lomsky, M., Edenbrandt, L., Minarik, D., and Bjartell, A. (2016). Analytic validation of the automated bone scan index as an imaging biomarker to standardize quantitative changes in bone scans of patients with metastatic prostate cancer. *Journal of Nuclear Medicine*, 57(1):41–45.

[3] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*.

[4] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2016). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*.

[5] Chollet, F. (2017). Keras documentation. https://keras.io/. (Accessed on 04/28/2017).

[6] Ciresan, D. C., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745.

[7] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

[8] Cootes, T. F., Taylor, C. J., Cooper, D. H., and Graham, J. (1995). Active shape models-their training and application. *Computer vision and image understanding*, 61(1):38–59.

[9] DeepLearning4J (2017). A beginner's guide to recurrent networks and lstms - deeplearning4j: Open-source, distributed deep learning for the jvm. https://deeplearning4j.org/lstm.html. (Accessed on 04/26/2017).

[10] Diagnostics, E. (2017). Exini diagnostics. http://exini.com/. (Accessed on 05/10/2017).

[11] Dwyer, H. (2015). Lenet - convolutional neural network in python - pyimagesearch. http://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/. (Accessed on 04/25/2017).

[12] EMC (2014). The digital universe of opportunities: Rich data and the increasing value of the internet of things. https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm. March 24, 2017.

[13] Fulkerson, B., Vedaldi, A., and Soatto, S. (2009). Class segmentation and object localization with superpixel neighborhoods. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 670–677. IEEE.

[14] Gera, D. L. (2003). *Ancient Greek ideas on speech, language, and civilization*. Oxford University Press, USA.

[15] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[16] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

[17] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[18] Johnson, L. (2015). Classifying mnist digits — theanets 0.7.3 documentation. http://theanets.readthedocs.io/en/stable/examples/mnist-classifier.html. (Accessed on 03/24/2017).

[19] Karpathy, A. (2017). Cs231n convolutional neural networks for visual recognition. http://cs231n.github.io/neural-networks-1/. (Accessed on 04/26/2017).

[20] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[21] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[22] Kurenkov, A. (2016). A 'brief' history of game ai up to alphago, part 2 – andrey kurenkov's web world. http://www.andreykurenkov.com/writing/a-brief-history-of-game-ai-part-2/. (Accessed on 05/05/2017).

[23] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137.

[24] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440.

[25] Marko, J. (2016). Github - jocicmarko/ultrasound-nerve-segmentation: Deep learning tutorial for kaggle ultrasound nerve segmentation competition, using keras. https://github.com/jocicmarko/ultrasound-nerve-segmentation/. (Accessed on 05/02/2017).

[26] Mitchell, T. M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877.

[27] Needham, J. (1960). Science and civilization in china: Volume 2, history of scientific thought.

[28] Ng, A. (2008). Gradient descent landscape visualization (703×366). https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/03/06100746/grad.png. (Accessed on 04/03/2017).

[29] Overgaard, N. C. (2016). Medical image analysis, clustering and segmentation - fman30. http://www.ctr.maths.lu.se/media/FMAN30/2016/Lecture-10-Clustering-and-segmentation.pdf. (Accessed on 03/27/2017).

[30] Richter, J. (2006). Automated interpretation of cardiac scintigrams. Master's thesis, Centre for Mathematical Sciences LTH, Lund University, Sweden.

[31] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer.

[32] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[33] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229.

[34] Shotton, J., Johnson, M., and Cipolla, R. (2008). Semantic texton forests for image categorization and segmentation. In *Computer vision and pattern recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE.

[35] Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2009). Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 81(1):2–23.

[36] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[37] Sminchisescu, C. (2016). Machine learning - fman45. http://www.ctr.maths.lu.se/course/machinlearn/2016/. (Accessed on 03/27/2017).

[38] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[39] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

[40] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826.

[41] Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine learning*, 8(3-4):257–277.

[42] Tu, Z. and Bai, X. (2010). Auto-context and its application to high-level vision tasks and 3d brain image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(10):1744–1757.

[43] Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

[44] Tyantov, E. (2016). Github - edwardtyantov/ultrasound-nerve-segmentation: Kaggle ultrasound nerve segmentation competition [keras]. https://github.com/EdwardTyantov/ultrasound-nerve-segmentation. (Accessed on 05/02/2017).

[45] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

[46] Winston, P. (2015). 12a: Neural nets - youtube. https://www.youtube.com/watch?v=uXt8qF2Zzfo&t=2378s. (Accessed on 05/09/2017).

[47] Wrangsjö, A., Pettersson, J., and Knutsson, H. (2005). Non-rigid registration using morphons. *Image Analysis*, pages 501–510.

[48] Yann LeCun, C. C. (2017). The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/. (Accessed on 03/24/2017).

[49] Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*.

[50] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*.