fhwedel
UNIVERSITY OF APPLIED SCIENCES

Bachelor Thesis

Automated Segmentation of Bone Matter in 3D MR Images using Convolutional Neural Networks

Paul-Louis Pröve
mail@plpp.de

supervised by
Prof. Dr. Dennis Säring
dsg@fh-wedel.de

Hamburg,
August 1, 2017

Abstract

Here shall be text.

Contents

1 Introduction

Introductory Statement:

Statement of the problem: briefly describe problem

Significance of the Study

Objectives of the Study

Time and Place

Scope and Limitation

Recent advances in Artificial Intelligence have led to workflows that not only run fully automated but also often exceed human performance. State of the art neural networks can classify images into thousands of categories more accurate and magnitudes faster than humans. They translate text between hundreds of languages, navigate cars autonomously through cities and detect intruders in computer networks. In most of these cases they have been trained on tens of thousands or even millions of data samples.

Neural networks have also found great success in the medical field, where new types of problems were introduced based on datasets that are much smaller. Medical imaging data includes MRI, Ultrasound and CT of which the latter is an invasive methods because it uses electromagnetic radiation.

github link

2 Background

This chapter provides an overview of prerequisites and previous work that led to the main task of this thesis. It features a section for the necessary medical knowledge as well as a hierarchical derivation to the appropriate branch of computer science.

2.1 Medical Knowledge

I will briefly explain how the imaging technology works that was used to create the data set and what function growth plates have in the human body.

2.1.1 Magnetic Resonance Imaging

Magnetic Resonance Imaging (MRI) uses magnetic fields and radio frequencies to probe tissue structure. In contrast to x-ray and CT which require the exposure of ionizing radiation, MRI is a non-invasive imaging method. As such it has become an essential diagnostic imaging modality in the medical field [29].

96% of the human body is made up of hydrogen, oxygen, carbon, and nitrogen, all of which are referred to as MR active. These elements have an odd atomic mass number giving the nucleus a spin. Due to the laws of electromagnetic induction, the motion of unbalanced charge produces a magnetic field around itself. Hydrogen is the element used in MRI because its solitary proton in the nucleus gives it a relatively large magnetic moment [29].

The positively charged hydrogen particles in water produce a signal when exposed to a strong external magnetic field. This field is supplied by a magnet in the MRI machine aligning the magnetic field of the hydrogen atoms to its own. Gradient coils are used to cause a linear change in the strength of this field. By alternating the current of these coils on three perpendicular axes, it is possible to calculate a three-dimensional image of the tissue [29].

2.1.2 Growth Plates

Most bones in the human body grow through a chondral process, where cartilages near the end of a long bone are ossified over time. These continuously renewing cartilages are referred to as growth plates, and they are responsible for extending the longitudinal length of the shaft. The growth of a bone comes to an end when cells in the cartilages stop their proliferation, and the growth plate starts to close. This is correlated with increasing amounts

of sex hormones during puberty [3]. On average women will stop growing between age 14 to 17, while for men this happens between age 18 to 22 [1].

Figure 1: The Tibia (lower bone) in 3 different states: open at age 14 (left), centrally-closed at age 16 (middle) and closed at age 18 (right). These examples show 3 different candidates.

The closing process initializes at the center of the growth plate and extends to the edges. Literature often classifies the current state of the growth plate in open, centrally-closed and closed [2].

2.2 Computer Science

The primary task of this thesis revolves around a particular branch of computer science problems. In this section, I want to provide a path to where most of the work will take place. Figure 1 visualizes the hierarchical relationship of the following subsections.

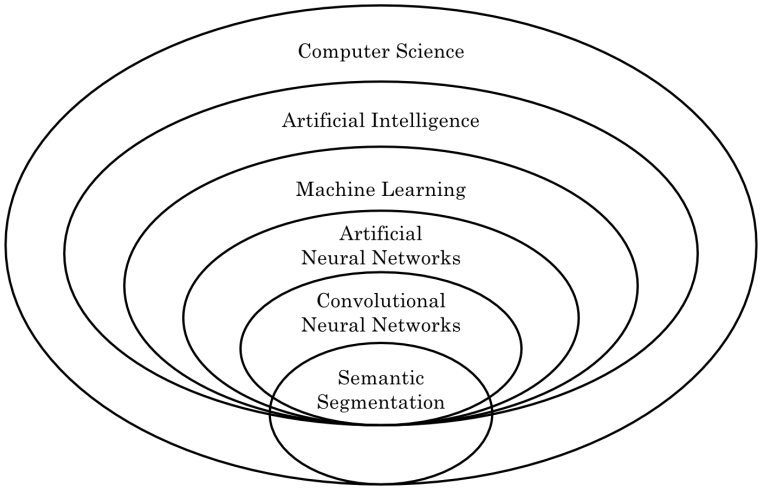

Figure 2: Hierarchical overview where convolutional neural networks and semantic segmentation are placed in the field of computer science

2.2.1 Artificial Intelligence

Artificial Intelligence (AI) is understood as the effort of automating a given task that typically needs human intelligence to solve. The history of AI started in the 1950s, where a particular type called "symbolic AI" gained popularity. It was believed that human level intelligence could be achieved through hard-coded rules that programmers specified [7].

Taking a complex problem like playing chess and continuing to break it into smaller problems, until they can be solved with known logic. While it was effective for certain tasks, fuzzy problems like image classification, speech recognition or language translation were difficult to tackle. Over the years a new approach was found, which today is referred to as machine learning (ML).

2.2.2 Machine Learning

The concept of classical programming is that an engineer defines a set of rules, called an algorithm, which uses input data to calculate some form of output data [7].

Figure 3: Classical programming pipeline

A machine learning algorithm is an algorithm that can learn from data [10]. It can be used to calculate these rules automatically, so they don't have to be specified by hand. Three components are needed for such an approach.

- Input data the algorithm is supposed to transform
- Output data the algorithm is supposed to predict
- A measurement to validate the performance of a prediction

It works by feeding input and output data into a pipeline, which will learn to transform one into the other. With the advantage that no explicit programming is needed to generate the rules, comes the disadvantage that prior input and output data is required for the initial learning process.

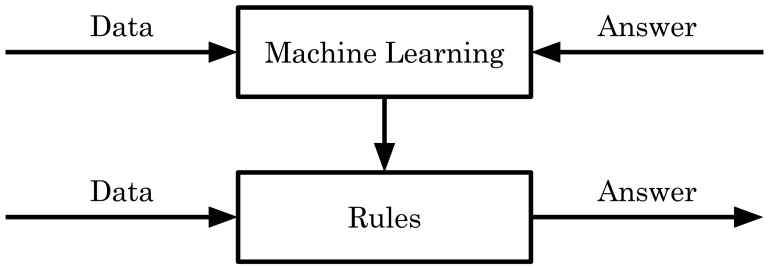

Figure 4: Machine learning pipeline that calculates a set of rules

Machine learning may be applied as an effective method if it's not feasible or possible to define an algorithm by hand and sufficient data is available for training. How much "sufficient" is depends on factors like the type of task, the complexity of the data, the uniformity of the data, the type of machine learning algorithm and others.

There are different subparts to machine learning like supervised and unsupervised learning. Supervised learning is used when it's clear what the output data looks like, whereas unsupervised learning can help to find unknown patterns in the data. Examples of supervised learning techniques include linear regression, naive Bayes, support vector machines, decision trees, random forests, gradient boosting and artificial neural networks (ANNs). Since the primary interest of this study revolves around ANNs, this will be the focus of following chapters.

2.2.3 Artificial Neural Networks

Artificial neural networks are inspired by neurobiological concepts of the human brain. However, they are not models of the human brain. There is no evidence to support that the brain implements anything like the learning mechanisms used in ANNs [7]. Artificial neural networks are mathematical frameworks for learning representations from data and one of, if not the only tool in deep learning (DL) — a subset of machine learning.

The origin of the term deep learning is twofold. On the one hand, it refers to the fact that ANNs can learn "deep" hierarchical information from data. On the other, it describes that they show multiple layers of depth within their architecture. ANNs are as old as machine learning itself, but only gained a lot of their popularity in recent years [7].

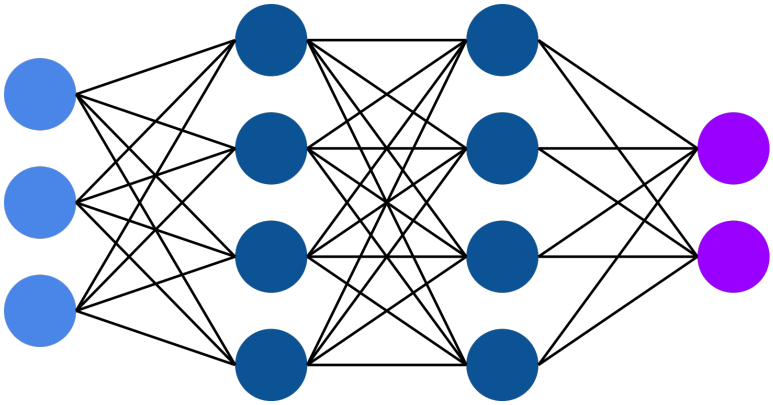

Figure 5: An ANN with 3 input nodes, 2 hidden layers with 4 nodes each and 2 output nodes

These dense layers inside artificial neural networks contain entities called nodes that are jointly linked with one another. Every node in a dense layer is connected to each node in the previous and following layer. This structure gives ANNs the capacity to approximate complex mathematical functions and learn global representations in the data. These nodes, also called parameters, can range to hundreds of millions depending on the task [26].

2.2.4 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specific type of ANN that uses an operation called convolution in at least one of their layers. The first CNN was introduced by Yann LeCunn [19] in 1990 at which time its popularity was limited. In 2012 a CNN called AlexNet [18] won the prestigious ImagetNet competition, and these types of networks have been winning ever since.

A convolution is a mathematical operation on two functions of real-valued argument. In imaging terminology, the first function refers to the input and the second function describes the kernel. The output of this operation is called a feature map [10].

Convolutions are frequently used in image processing, which is also why they were introduced to visual tasks in the field of deep learning. They allow learning local patterns in the data instead of treating the input features in a global manner like dense layers do.

Another type of layer that is frequently used in CNN architectures perform a pooling operation. By doing so, the spatial resolution is reduced, and only the most relevant features are kept. This is important to maintain a manageable network size.

2.2.5 Semantic Segmentation

CNNs can help to solve different types of supervised machine learning problems of which regression, classification, and segmentation are the most common.

A regression describes the prediction of one or multiple continuous outputs. An example for this would be the age prediction of a person based on their knee MRI. Regression is also an important part of object detections, where it's applied to determine coordinates in an image.

A classification sorts the input into one or multiple categories, like predicting if the growth plate is open, centrally-closed or closed. It can be seen as n parallel regressions where n equals the total number of classes. The continuous output for each class is the network's confidence of the input belonging to this category. For 1 out of n classifications, the most probable category is predicted. For m out of n classifications, a threshold defines at which point a category is predicted.

A segmentation creates an image of identical dimensions as the input and masks specific regions within. Every channel of the output mask belongs to a particular category that is segmented. A segmentation can be seen as a classification of each pixel/voxel. For this study, the Femur, Tibia, and Fibula needed to be masked from the rest of the MRI content. This is called a semantic segmentation because the process is based on a semantic meaning of objects in the image.

Figure 6: A landscape photograph with a possible semantic segmentation map for sky, stone and sea

3 Methods

The purpose of this chapter is twofold. It will discuss the design and processing decisions that were made for the development of this project, while also giving critical insight into how these applied technologies work.

3.1 Setup

The workstation included an Intel i5 processor, 16GB of RAM and most importantly a NVIDIA GeForce GTX1060/6GB. Neural Networks can be trained more efficiently on GPUs than CPUs. This is because the simpler but highly parallelized architecture of graphics chips plays in favor for the needed calculations in deep learning [24].

The computer ran Ubuntu 16.04 with the NVIDIA CUDA and cuDNN libraries installed to take advantage of the GPU. As the primary programming language Python 2 was chosen due to its simple syntax and popularity in the deep learning field. The code was briefly tested with Python 3 as well and seemed to work. Keras was used as the framework for training models because its top level syntax allows fast prototyping and testing. The development environment was a Jupyter Notebook, which allowed a flexible execution of code snippets instead of running the entire program for every single change.

The processing of medical image data needed a library that could handle these formats. SimpleITK is a Python binding to the ITK library written in C++. It includes many tools for image processing and is especially popular in the medical field. Other libraries were also used for smaller tasks. A complete listing can be found on the project's GitHub page, where the entire code is available.

3.2 Data Analysis

The data set was a collection of three dimensional MR images showing the human right knee. The number of available samples grew during the project. For most of the development time, it included 150 samples that came from 3 different sources. All images were provided in the MHD file format, which is very common in the medical field. It features lossless compression, as well as certain meta information about each image. The resolution of these samples differed from one source to the other.

The Maas data featured a sagittal perspective, while all other samples were taken coronal. This mattered because the resolution of all images was roughly

Source	Prospective	Perspective	Samples	Maps	Resolution
Epi	Yes	Coronal	80	40	800x800x41
Jopp	No	Coronal	65	36	512x512x24
Maas	No	Sagittal	5	0	multiple

Table 1: Details of available images sorted by their source

20 times lower on the z-axis. The 65 Jopp images were another source of retrospective data, of which 36 had segmentation maps. The origin of this data was a previous study that investigated the condition of the growth plates of Tibia and Femur. A total of 80 MR images were taken specifically for this project. They showed the highest resolution in all dimensions and half of them had segmentation maps.

Figure 7: Six equally distributed slices of a single 3D MRI sample. Femur and Tibia are visible starting at the second image, whereas the Fibula becomes visible in the lower left corner of the 4th image.

The total of 76 segmentation maps were all manually labeled by two people. Each mask featured three channels that showed Femur, Tibia, and Fibula separately. They did not segment the entirety of each 3D image but masked a window around the knee cap where the three bones meet.

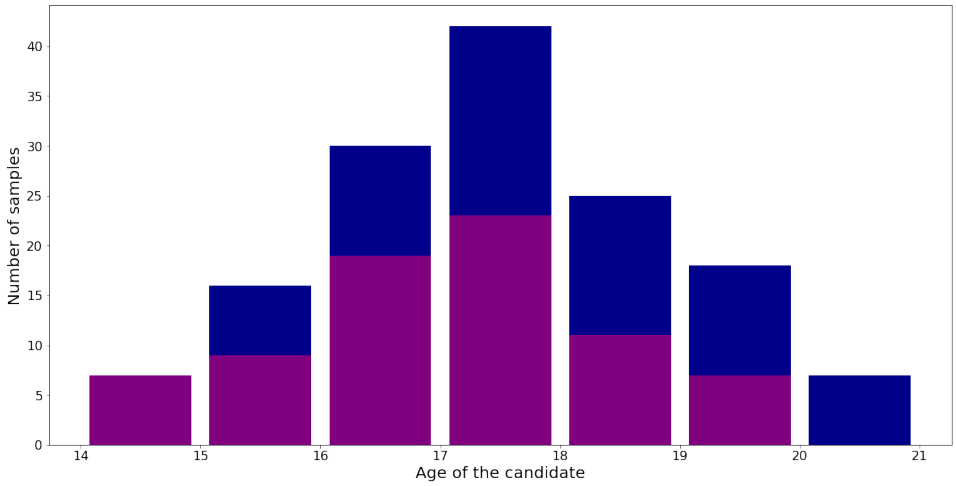

Figure 8: Age distribution in the data set. Purple shows the samples for which segmentation maps were available. Blue includes the rest.

The candidates chosen for the MRI recordings were all german males between the age of 14 and 20. Their age was normally distributed, and a majority of the candidates were recorded twice roughly a year apart.

3.3 Preprocessing

The parameters in a Neural Network commonly range from tens of thousands to hundreds of millions. This complexity allows the model to learn on its own what features of an image are relevant for any given task. It works in conjunction with the fact that high volumes of data are available for the training.

Because of the small data set that was available for this study, several types of preprocessing were applied to the images. For the most part, these techniques remove irrelevant information and reduce variance between multiple samples.

Other preparation methods experimented with the difference between 2D and 3D data as well as the influence of separate segmentation channels on the output.

3.3.1 Training, Validation, and Testing Sets

The data was split into three subsets of which each was used for a different purpose. The training set is commonly the largest of the three and contains

the data that is applied to the actual learning process. It's the only portion of the data the network will draw direct conclusions from.

The validation data is used to regularly measure the performance of the model and check whether overfitting occurs or not. If the accuracy of the validation set drops below the results of the training data, the network is starting to memorize the data it knows rather than generalizing on the concept.

The third subset is referred to as the testing data. In contrast to the validation set, it's only used once in the very end, to give a final score. The idea is that by building a model based on the validation results a certain amount of information bleed occurs, where the network will implicitly learn from the validation data. In order to prevent biased results, the testing data is used as a last performance reference.

- Traning Set: 74% of the data
- Validation Set: 13% of the data
- Test Set: 13% of the data

3.3.2 Cropping, Resizing, and Resampling

The framing included vast areas of the upper and lower leg to be visible in the picture. Since these were not seen as relevant, they were cropped out. An existing algorithm was used to detect the center where Tibia and Femur meet and only use a square window around this point.

Although there is no theoretical size limitation to using convolutional neural networks, it is desirable to reduce the spatial resolution and therefore decrease the number of calculations. 224x224 pixels for each slice still gave enough detail to identify the shape of Femur, Tibia, and Fibula, while also being a common resolution for CNNs.

Resizing the z-axis was problematic because the resolution was roughly 20 times lower. When scaling along this dimension, the segmentation maps of different slices started to blend and create merged maps of multiple layers. In order to get the images from two different main sources on the same scale, two different approaches were tried.

The 41 slices per image of the Epi data were padded with empty pixels to 48 slices. Afterward, every second 2D image was resampled to the same 24 slices the data from Jopp et al. showed. It was not possible to do it the other way around and upscale 24 slices to 48, because the interpolated segmentation maps may have falsified the data.

Throwing away perfectly good data is very uncommon in the machine learning field, especially if data sets are small. However, one slice shares a lot of similar information with its neighboring slices and can be seen as a sort of image augmentation between the two. By using the same spatial resolution for both sources, the balance between the amount of Epi and Jopp data was kept. A total of 76 24x224x224 images were now available for training.

The second approach converted the volumetric data to 2D while using all 41 slices from the Epi data and all 24 slices from Jopp. No samples were thrown away, but the data was no longer volumetric.

3.3.3 Normalization

The normalization of images refers to the process of transforming all samples on the same scale of values. Two techniques for this are popular in the field of deep learning. The first one is called feature scaling, where every sample is normalized between 0 and 1.

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

The second technique calculates the standard score, where the mean is subtracted from the samples and then divided by the standard deviation.

$$x' = \frac{x - \mu}{\sigma}$$

In this case, the mean and standard deviation are not calculated for every image, but for all training samples. This centers the intensities around the average brightness of the training data. When normalizing validation and test data, one would also use the mean and standard deviation from the training set because those are the measures the architecture is expecting.

The standard score presents a more desirable approach because its mean centering helps to prevent that parameters get abnormally large or get set to zero during training. It is also less sensitive to outliers e.g. extremely bright spots in the image, which frequently appeared in the form of high intensities.

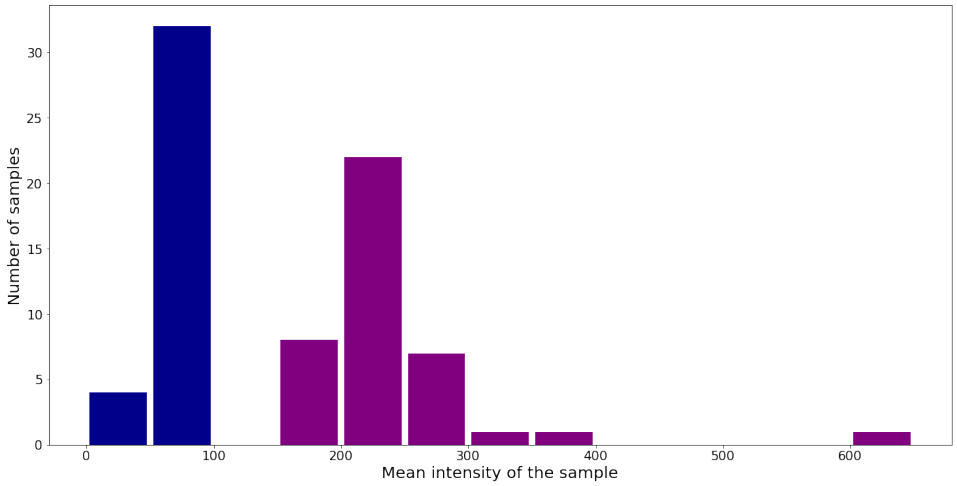

Figure 9: Intensity distribution in the data set. Blue shows the Epi samples and purple shows Jopp data.

In contrast to the regular use of the standard score, every image was normalized on its own mean and standard deviation instead of calculating it on the entire training data. The Jopp images featured roughly five times higher intensities than the Epi data because they were recorded on different MRI machines. Figure 9 shows two spikes in mean intensities that had to be dealt with. By normalizing each 3D image on its own values, these two distributions were unified. Furthermore, this procedure will normalize future samples on the same scale the network expects no matter what the bit depth of an image is.

This approach also has a downside. The network can no longer use the mean intensity and the standard deviation as a feature to learn from because every image will be identical according to these measures. However, these two features showed no correlation to the accuracy of a prediction, which is why it was decided to use this approach.

3.3.4 N4 Bias Field Correction

A bias field is a low-frequency non-uniform intensity that is an unwanted byproduct Magnetic Resonance imaging. Several methods have been developed in the past of which N4ITK is the de facto standard in the field [28]. This algorithm approximates the nonuniform field and balances intensities using B-Splines.

Figure 10: Raw image (left), corrected image (middle) and visualization of intensity differences between the two (right)

As the name suggests, it is included in the Insight Toolkit (ITK) for which SimpleITK delivered an available Python binding. Using the N4 Bias Field Correction with its default settings on a single 24x224x224 image took between 60 and 120 seconds.

3.3.5 Augmentation

Image augmentation is a popular approach to virtually increase the size of the dataset. The general idea is that a neural network will overfit more when learning one sample n times, in comparison to learning n alternations of this sample just once. It helps to generalize on new images instead of memorizing patterns in the training data.

Lossless augmentation techniques are those that don't change the values and relative localities in a sample. In 2D these include vertical and horizontal flips, as well diagonal flips if width and height are equal. Note that any multiples of 90-degree rotations can also be created using a combination of these flips. Although the samples are changed as a whole, they do not vary concerning their contained values.

The term "lossless" only refers to the technical change and not necessarily to the semantic change. Flipping a slice of this dataset vertically, switches the absolute positions of Femur and Tibia. This might make it harder for the network to distinguish between the two.

Lossy augmentation methods include a variety of image transformations. Common choices include:

- Horizontal shifts
- Vertical shifts
- Rotations

- Shear mapping
- Brightness adjustments
- Contrast adjustments
- Gamma adjustments

For this study horizontal flips were implemented to give the impression that images from both knees were available. In addition to this, these images were shifted 24 pixels on the horizontal axis to add another type of augmentation.

Interestingly, these methods did not improve the accuracy of the model. The horizontal flips even hurt the performance and were therefore removed. The horizontal shifts were kept in the training set, so the network would perform better on unseen data that was not perfectly aligned.

3.3.6 2D and 3D data

Although convolutional neural networks are most commonly connected with 2D data such as photos or drawings, they can be applied in any dimensional space. In Natural Language Processing (NLP) 1D convolutions are often used on sentences and in finance they can be applied to time series forecasting. In the medical field where a variety of volumetric data exists, 3D convolutions have become a popular choice for building deep learning solutions.

In the context of neural networks, one has to differentiate between volumetric data with one channel and 2D data that consists of multiple color channels. Both are an example of 3D data, but the volumetric images feature three spatial dimensions, whereas photos feature only two. Convolutions commonly traverse the spatial dimensions, which means photos with multiple channels are usually not subject to 3D convolutions. Instead, various input channels are fed into a 2D CNN.

Although the data set consisted of volumetric MRI data, the z-axis showed 20 times fewer voxels than the other two dimensions. Not knowing the influence of this situation both 3D and 2D architectures were investigated for this project. The 3D model didn't use any MaxPooling on the already small z-axis. A kernel size of 3x3x3 was used similarly to the 3D version of U-Net. The 2D data was created by using each slice as a single input image.

As described in 3.3.1 the 3D convolutional network used 24x224x224 voxel images from both the Epi and Jopp data. In contrast, the 2D network used all of the available slices from each source as separate images.

While in theory, the spatial information of the z-axis gave the 3D network more contextual information of every slice, the 2D model performed better

at the end. The latter was also less computationally expensive and allowed working with data where only single slices were available per sample.

3.3.7 Separate Bone Maps

The initial segmentation maps came with three separate channels for the Femur, Tibia, and Fibula. With this information, it was possible to train a network that would segment the three bones while still differentiating between them.

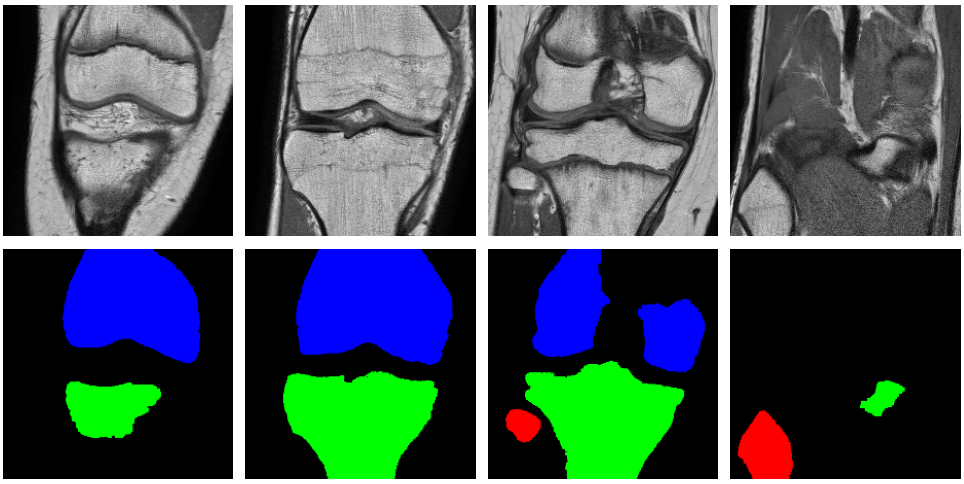

Figure 11: Four sample slices with their ground truth segmentations showing the three separate bones

However, this led to problems with the Fibula, which was not recognized by any of the applied architectures. The channel was always predicted empty. It's appearance in the images accounted for 3.4% of the segmented area, whereas the Tibia made up 41% and the Femur accounted for the remaining 55.6%. This led the network to learn that an empty prediction of this channel is valid in 96.6% of the cases, which is very accurate on its own. Instead of spending capacity on improving a channel that only makes up a tiny fraction of the result, the network improved the performance of Femur and Tibia.

The only successful method was training a separate model for each of the three bones. The network started to predict empty segmentation maps in the beginning because the actual masks only made up a fraction of the frame. The only way to improve from here was to find the Fibula in the non-empty slices and segment its region. This was different from the tests before, where the unsatisfactory performance of the Fibula was balanced by making more accurate segmentations of the other two bones.

For another experiment, the three channels were merged to create a network that would segment any bone in the image. Not only did this solve the problem with the Fibula as well, but the predictions of Femur and Tibia were also more accurate. The network didn't need to learn any more that the global position in the image accounts for whether or not an object should be segmented. Instead, it could segment any bone in any position in the frame. It was decided to continue with this approach for further tests, but the same architecture can be used on single bone channels as well.

3.4 Architecture

The majority of classification CNNs use a recurrent combination of convolutional and pooling layers [18][12][15][25]. By combining the local learning patterns of convolutions with the spatial reductions of pooling, the input is compressed to a dense representation of its most important features.

As mentioned in 3.2.5 a segmentation can be seen as a classification for every pixel. As such, early CNN segmentation models used small patches of the input image only to predict a single pixel through a classification pipeline. Afterward, a full segmentation map was assembled using each of these pixels. This process was very slow, and it also prevented the network to have a field of view larger than the inserted patch.

An improvement for segmentations came through architectures referred to as encoder-decoder models. The encoding process describes the same spatial compression used in classification networks. Afterward, in the decoding step, the spatial resolution is brought back to its original shape and further processed by additional convolutions. This yields huge speed improvements, while also increasing the accuracy of the prediction. Encoder-decoder networks are the de facto standard in the current field of image segmentation [6][30][20][5][23][21][8][25][4], and they were the initial starting point for building architecture.

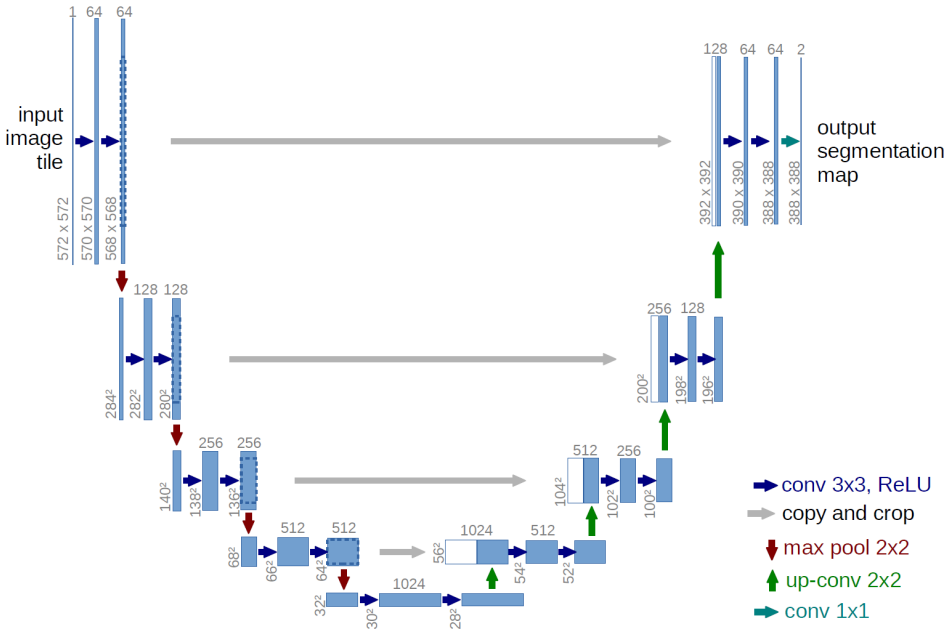

Figure 12: Visualization of the U-Net architecture

Figure 12 shows a particular implementation of an encoder-decoder model known as U-Net [25]. It has been successfully used on a variety of projects in and outside the medical field. The contracting side on the left shows a similar architecture to classification CNNs, after which the process is mirrored on the expanding side.

The following subchapters will discuss key elements that go into the design process of such an architecture. It was also analyzed how technologies that were introduced after the U-Net paper could improve the performance on this data set.

3.4.1 Channels, Growth Rate and Depth

The number of parameters in a neural network has a high correlation with its learning capacity. By adding more nodes that can be adjusted during training, the model can approximate a more complex function that transforms the input into the output. The downside is that a larger parameter count will also increase the possibility of overfitting the data. A convention in the field of CNNs is to gradually increase the number of channels, while the spatial resolution is reduced due to the use of MaxPooling [18][12][15][25]. U-Net also shows this behavior on the left side of its architecture.

A test was set up that compared the original U-Net against five smaller variants. They varied in the total number of parameters and how the number of channels changed from one layer to the next.

Name	C1	C2	C3	C4	C5	C6	C7	C8	C9	Param.
Model A	32	32	32	32	32	32	32	32	32	211k
Model B	16	24	36	54	81	54	36	24	16	340k
Model C	48	48	48	48	48	48	48	48	48	474k
Model D	8	16	32	64	128	64	32	16	8	485k
Model E	32	48	72	108	159	108	72	48	32	1,358k
U-Net	64	128	256	512	1024	512	256	128	64	31,000k

Table 2: Convolution blocks 1 to 9 of the candidates with their respective output channels and total number of parameters

Training U-Net was very slow in two regards. Each training step lasted six times longer compared to the smallest model, while it also took 25 times more training steps to reach the same results. Since it was also overfitting to a significant amount, it was not trained until the end. The second largest model E also overfitted on the training data and never achieved comparable results to the other four models.

D started with eight channels which were then increased by a factor of 2, whereas B started out larger but only increased the channels by a rate of 1.5. Although being the smaller model, B showed a higher accuracy in the end. It was concluded that the initial number of output channels has a larger effect on the result than the growth rate.

This theory was supported by the results from A and C, both of which kept their initial number of channels throughout the network. These two showed the highest scores in comparison to the other models while maintaining their relatively small size. Since their results were identical, model A was kept because of its faster training speed. It was interesting to see that the smallest model performed best across the candidates.

The original U-Net featured a depth of 5 convolutional blocks until reaching the bottom of the "U"-shape. It was also investigated that a depth of 6 would not improve performance, while a depth of 4 decreased the accuracy. Therefore, Model A was left unchanged.

3.4.2 Skip Connections

U-Net uses skip connections to copy values from the left side to the corresponding layer on the right side. This improves performance because oth-

erwise spatial information would get lost while contracting the input. An experiment showed that removing these paths indeed hurt the performance badly.

ResNet [12] is another popular classification architecture that was the first to use residual connections. These are another type of skip connection that bridge the beginning and end of a convolutional block. According to the original paper, they are also one of the main reasons CNNs can be expanded to depths of thousands of layers. Since their use in the industry has found great success in many models, they were also applied in this network. However, they did not have an impact on the performance or the training speed, which is why they were not used in the end.

3.4.3 Dropout

Dropout is a popular regularization technique that randomly zeros out a fraction of the nodes during training [27]. It is understood that this helps the model to generalize better and reduce overfitting on a given data set. Well known image classification architectures like VGG16 [26], SqueezeNet [15] or AlexNet [18] use Dropout near the end of the network. Similarly, U-Net uses Dropout at the end of the contracting path to prevent overfitting.

Since a single unit of dropout with a rate of 0.5 is common in other architectures, this was also chosen as a first option. Other tests included adding multiple dropout units between the convolutions on the contracting side, which led to slower training and lower scores. Adding dropout on the expanding side is rather unusual and also didn't perform well in the tests. In the end, the first candidate was chosen for future training.

3.4.4 Activation Functions

Activation functions sit between layers in a network to introduce a nonlinearity. Otherwise, the dense and convolutional operations could be regenerated to a simple linear transformation and remove the benefits of building a deep model.

$$relu(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

The rectified linear unit or ReLU [22] is the default recommendation for an activation function in modern neural networks. It's defined as the maximum of 0 and the input value and can be described as a nonlinear function made

up of two linear pieces. Because of this, it preserves many of the properties that make models easy to optimize and generalize well on new data [10].

$$elu(x) = \begin{cases} e^x - 1 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

Since the introduction of ReLU other variants have built on its success like PReLU [13], LReLU and ELU [9]. The latter titled exponential linear unit shows the same linear behavior for positive values but adds an exponential function to negative inputs instead of erasing its value. This has shown further improvements by keeping the mean of values centered at 0 while only being slightly more computationally expensive.

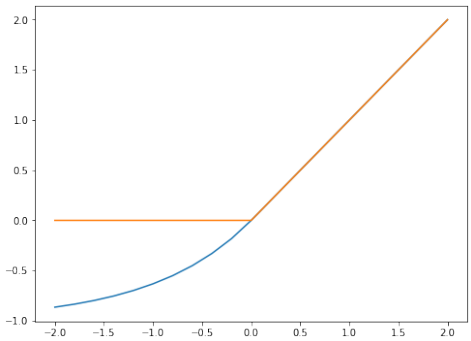

Figure 13: Visualization of ReLU in orange and ELU in blue

ELU showed superior results in comparison to ReLU and was chosen for all future tests.

3.5 Training

The training is where the actual learning takes place, and its process is inspired by how humans improve their performance on tasks. When opposed to a difficult question the first step would be a mere guess of what the answer might be. In a second phase, this information is compared to the right answer and processed by the brain accordingly.

Similarly, the training of a neural network consists of two parts. In the first step, the input is fed forward through the network, and an answer prediction is made. In the very beginning, this is just a random guess because ANNs are initialized with random values.

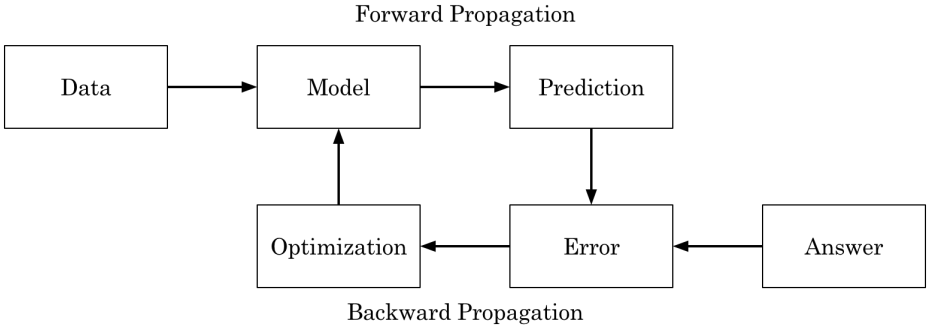

Figure 14: The iterative process of forward and backward propagation

The prediction is then compared to the right answer, and the error is calculated. This metric can be fed back through the network to optimize the parameters in the model. These two steps are called forward and backward propagation.

3.5.1 Metrics and Loss Functions

Metrics are used in deep learning to measure the performance of a model. For example, the accuracy is often chosen to describe how well a neural network is doing on a classification task. An accuracy of 0.9 indicates that 9 out of 10 samples are classified correctly.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

In the formula above T and F indicate whether a prediction was true or false. P and N stand for a positive or negative outcome.

The result of a loss function is a metric that will be minimized during the backward propagation process. It needs to be a differentiable function, which is why the accuracy cannot be used as a loss function. It is a binary metric that works with true or false values and not with probabilities.

In situations like these, a surrogate function is used that has a high correlation with the target metric. For classification problems, this is often the cross entropy. Because a segmentation can be seen as a classification for every output pixel, it was also chosen as a candidate for this study.

$$H(p, q) = -\sum p(x)\ log\ q(x) \tag{2}$$

Another option was the $F_1 score$, which is a particular implementation of the F-Measure when β is 1. It is also known as the DICE coefficient (DSC).

$$F_\beta \ score = \frac{(1+\beta^2)TP}{(1+\beta^2)TP + \beta^2 FN + FP} \tag{3}$$

Although the F1 score is commonly applied as a binary measure and therefore not differentiable, a "soft" version can be used that accounts for continuous probabilities. To use it as a loss function where 0 describes the best possible outcome the $F_1 loss$ was defined as $1 - F_1 \ score$.

$$F_\beta \ loss = 1 - F_\beta \ score \tag{4}$$

The value of β can be adjusted to change the emphasize between precision and recall. Precision describes how much of the predicted area was true, whereas recall describes how much of the ground truth was recognized by the model. This can be useful for data sets with large imbalances between classes, like this study showed between Fibula, Femur, and Tibia.

The $F_2 score$ was tried for the three channel segmentation, where the Fibula could not be recognized. Although this model showed better scores regarding the recall measure, the Fibula was still not segmented.

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$Error = \frac{FP + FN}{TP + TN + FP + FN} \tag{7}$$

Both loss functions were used in separate runs to compare the results of the $F_1 \ score$ and the cross entropy. The F_1 trained model showed better results regarding the $F_1 \ score$ and the cross entropy model showed better results on the cross entropy loss. In regards to Precision, Recall, Accuracy, and Intersection-over-Union (IoU) the performance of the F_1 trained network showed higher scores.

$$IoU = \frac{TP}{TP + FP + FN} \tag{8}$$

Another test run used both loss function at the same time. This was done by feeding the output of the next-to-last layer into two separate segmentation outputs. Each of these last layers was trained with the F_1 *loss* and cross entropy loss respectively. All previous layers were therefore trained by the sum of both loss functions. This architecture matched the best results of the previous runs in a single model but also increased the training time by 30%.

Evaluating these two outputs on the other metrics played in favor for the $F_1 loss$ output. Combining both maps didn't improve the scores any further. Since the cross entropy was chosen as a surrogate function and not as a performance metric, all future tests used only the $F_1 loss$. This delivered faster training than the dual output method.

3.5.2 Optimizer

The previous chapter provided an overview of the loss function, which measures the performance of a prediction during the training process. This chapter discusses how the result can be used to execute the actual optimization step.

The derivative of a single variable function defines the slope of this function at any given point. Knowing this, one can tell in which direction the original function declines. Adjusting the independent variable along the descent of a loss function will minimize the error in respect to its dependent variable.

The gradient is the derivative for functions of multi-dimensional inputs, such as the loss functions used in deep learning. A process that minimizes its result is called gradient descent. While it is possible to determine its minimum analytically, it is intractable for artificial neural networks due to the high number of parameters [7].

Instead, the stochastic gradient descent (SGD) is used which will take a random batch of the training data and iteratively adjust the parameters in small steps. SGD is the basis for all common ANNs, but over the years different variants were introduced to the field.

The Adam optimizer [17] is such a variant, which enhances SGD amongst other things by using what's called an "adaptive momentum estimation." This is also where its name derives from. It adaptively adjusts the learning rate which defines how much the parameters will be changed in one training step. By incorporating the previous and current shape of the slope, Adam can tremendously speed up the training.

3.5.3 Batch Size

The number of random samples per training step is referred to as the batch size. In the past, it was believed that larger batches led to something called the generalization gap [16], where the accuracy of a model would drop if it was trained on unusually large batches. Recent work [14] suggests other reasons for this decrease in accuracy. While common batch sizes range from 32 to 256, Goyal et al. showed accurate results using 8192 images per batch when training a model on ImageNet [11].

Using batches smaller than 32 samples can introduce a different kind of problem. Having too few data points that don't represent the mean of the data well, may lead to slow and unstable training.

Based on hardware limitations the largest possible batch sizes ranged from 24 to 48 samples depending on the current architecture. Whatever could be fit into memory was used for these experiments. Exceptions occurred when working with 3D convolutions in which case the batch size had to be limited to just 4 samples.

3.5.4 Learning Rate Policy

One full iteration over the training samples is referred to as an epoch, and the learning rate policy describes how the learning rate is changed from one epoch to another. With the introduction of adaptive optimizers like Adam or RMSProp, there has been a lower emphasize on this topic. Learning rates that were set too high or too low will be adjusted by the optimizer after a few iterations. Even though this reduces the number of possible defects, a lot of training time can be saved with the right policy.

Ten epochs were run at different learning rates to compare initial results and to examine the point at which the model wouldn't converge at all. 0.002 was the highest rate at which the model started training, but 0.001 resulted in the best score.

$$lr(x) = \frac{1}{1 + decay * x}$$

Figure 15: Learning rate at epoch x based on the decay value

After the model had stopped to improve at epoch 65, the learning rate was changed to 0.0001. This gave a small boost of accuracy. To have a smooth transition between learning rates, the decay was set to 0.001. This meant that the initial learning rate of 0.001 would reach 0.0001 after 68 epochs and then continue to decrease even further.

3.5.5 Early Stopping

Neural networks will continuously minimize the loss on the training set. This result needs to be validated on data the network hasn't seen before. At a certain point during training, the performance on the validation set will start to decrease because the model is overfitting on the training data. The number of iterations to reach this point is dependent on many hyperparameters, as well as the random values the network has been initialized with. As such it's difficult to calculate how many epochs the training will need to reach its peak.

Early stopping is a simple technique that will end the training process as soon as the model stops improving on the validation data. To do this, a patience value is defined how long the network should continue training after the score has stopped increasing. This is important because not every epoch will lead to a new best score on the validation data.

For test runs in this study, a patience of 9 was selected, which meant the training would stop after 10 epochs without improvement. Depending on the architecture and other hyperparameters this point was reached after 50 to 90 epochs. For the last training with the final set of hyperparameters, the patience was increased to 19, which didn't improve the accuracy. This was also a verification that the initial value of 9 was a good fit for this problem.

3.6 Postprocessing

After the training finished, it was investigated how the results on the validation data could be improved any further. Since the predicted segmentations showed a small number of values between 0 and 1, a threshold of 0.3 showed the best results to create a fully binary map. Furthermore, any 2D slice that showed a prediction smaller than 2% of the total frame was predicted as empty instead. This helped against the false prediction of tiny fractions in upper and lower layers.

3.7 Summary

In summary, the architecture features 4 "Down Blocks" on the contracting side that are mirrored through 4 "Up Blocks" on the expanding side. Each level is connected to the other side to share information at the current spatial resolution.

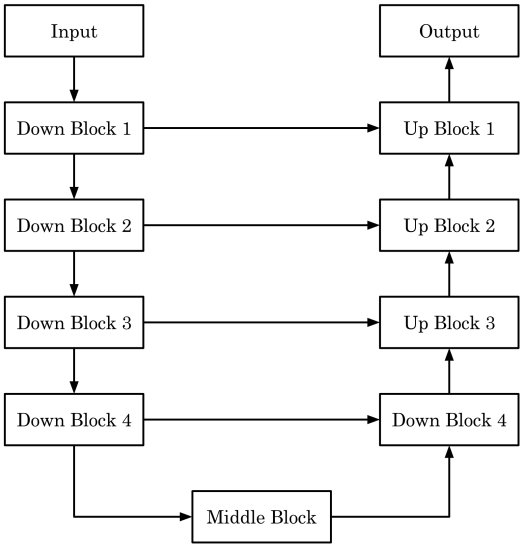

Figure 16: The proposed architecture for the segmentation

A Down Block consists of 2 convolutional layers each followed by an ELU activation function, after which A MaxPooling layer halves the width and height of the image. The Up Block looks similar but features a transposed convolutional layer in the beginning to upsample the resolution and no MaxPooling in the end. The Middle Block features a Dropout layer with a value of 0.5, which is surrounded by convolutional layers as well. All convolutional layers have 32 channels. The output layer includes another convolution that reduces the channels to 1, which represents the segmentation map.

Other options that were tried but didn't improve performance included using convolutional blocks with a stride of 2 instead of MaxPooling to downsample the images. Batch normalization was also applied but resulted in a high amount of overfitting the network never recovered from. The fact that it did not improve the performance came as a surprise because it usually improves any model. A possible reason for this could be the small batch size in combination with a high variance of intensities in the data. If the mean of each batch varies significantly from the mean of the entire training data, it may hurt the performance of batch normalization.

4 Results

The results in this section are entirely based on the test set, which had no contact with the network up to this point. 5 images were chosen randomly from each of the two sources in the very beginning. Since the Epi data featured 41 slices and the Jopp data featured 24 slices, a total of 325 2D samples were available for the verification.

4.1 Numeric Evaluation

The

The following results were evaluated on the test set, which consisted of 325 slices.

The numeric results show different modes applied to the 224x224 pixel test set. Bone carried the focus of hyperparameter tuning and describes all bone channels merged into one. Femur, Tibia and Fibula show the results of models that were trained on each bone alone. Combined assembles the three separate channels into one prediction.

Mode	DSC	IoU	Error
Bone (1 Ch.)	0.980 (±0.174)	0.980 (±0.174)	0.980 (±0.174)
Femur	0.980 (±0.174)	0.980 (±0.174)	0.980 (±0.174)
Tibia	0.980 (±0.174)	0.980 (±0.174)	0.980 (±0.174)
Fibula	0.980 (±0.174)	0.980 (±0.174)	0.980 (±0.174)
Bone (3 Ch.)	0.980 (±0.174)	0.980 (±0.174)	0.980 (±0.174)

Table 3: Evaluation of the test set

4.2 Visual Evaluation

5 Discussion

age prediction, size of data, augmentation

References

[1] David E. Attarian. Your Bones: What are growth plates?, 2013.

[2] Markus Auf der Mauer. *Automated Quantification of the Growth Plate of the Proximal Tibia for the Age Assessment in 3D MR Images Using a Fuzzy-Logic Classification Approach*. PhD thesis, 2015.

[3] Gerhard Aumüller, Gabriela Aust, Andreas Doll, Jürgen Engele, Joachim Kirsch, Siegfried Mense, and Laurenz Wurzinger. *Anatomie*. Number 2. 2010.

[4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. nov 2015.

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. jun 2016.

[6] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. jun 2017.

[7] Francois Chollet. *Deep Learning With Python*, volume 1. 2017.

[8] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3D U-net: Learning dense volumetric segmentation from sparse annotation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9901 LNCS, pages 424–432, jun 2016.

[9] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). nov 2015.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[11] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. jun 2017.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. dec 2015.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. feb 2015.

[14] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. may 2017.

[15] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and. feb 2016.

[16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. sep 2016.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. dec 2014.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks.

[19] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, and W. Hubbard. Handwritten Digit Recognition with a Back-Propagation Network. *Advances in Neural Information Processing Systems*, pages 396–404, 1990.

[20] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation. nov 2016.

[21] Fausto Milletari, Nassir Navab, and Seyed Ahmad Ahmadi. V-Net: Fully convolutional neural networks for volumetric medical image segmentation. In *Proceedings - 2016 4th International Conference on 3D Vision, 3DV 2016*, pages 565–571, jun 2016.

[22] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines.

[23] Vladimir Nekrasov, Janghoon Ju, and Jaesik Choi. Global Deconvolutional Networks for Semantic Segmentation. feb 2016.

[24] NVIDIA. GPU vs CPU? What is GPU Computing?

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Miccai*, pages 234–241, may 2015.

[26] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. sep 2014.

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[28] Nicholas J. Tustison, Brian B. Avants, Philip A. Cook, Yuanjie Zheng, Alexander Egan, Paul A. Yushkevich, and James C. Gee. N4ITK: Improved N3 bias correction. *IEEE Transactions on Medical Imaging*, 29(6):1310–1320, 2010.

[29] Catherine Westbrook. *MRI at A Glance*. Wiley Blackwell, 3 edition, 2016.

[30] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid Scene Parsing Network. dec 2016.