# Automated 3D Bone Segmentation using Deep Learning in Scoliosis

Andreas Bennström & Filip Winzell

Master's thesis
2021:E54

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

# Automated 3D Bone Segmentation using Deep Learning in Scoliosis


LUND
UNIVERSITY

Filip Winzell
Andreas Bennström
Supervisor: Einar Heiberg
Co-supervisor: Karolina Gottschalk
Examiner: Karl Åström
Opponent: Ruben Bergengrip

June 10, 2021

# Abstract

**Background:** Scoliosis is a condition where a person's spine is curved and rotated in three dimensions. In severe cases, surgery is needed to straighten the spine. Before such complex procedures, meticulous planning is needed. By performing a CT-scan, images of the spine can be acquired. From these images, it is possible to segment the spine in three dimensions to facilitate the preparation. However, doing these segmentations manually is very time consuming.

**Aim:** The aim of this thesis was to develop a model for automatic segmentation of spines suffering from scoliosis in CT-images. Depending on the results, an expansion of the model to general bone segmentation was to be evaluated.

**Methodology:** To develop a model, deep learning with the established architecture U-net was used. For training, validation and testing of the networks, 31 datasets were available. The datasets consisted of CT-image stacks covering different bodyparts in different patients. Several models were trained and tested to evaluate the performance of different hyperparameters and segmentation algorithms. An approach for 3D segmentation based on voting between different anatomical planes was compared to a basic 2D segmentation method. Finally, the best model was extented to general bone segmentation.

**Result:** Our best model, Voting 3D (edge), scored an average Dice score of 0.927 ($\pm$0.020) and Jaccard score of 0.865 ($\pm$0.034) on the scoliosis datasets. The extended network for general bone segmentation scored an average Dice score of 0.938 ($\pm$0.052) and Jaccard score of 0.888 ($\pm$0.086).

**Conclusion:** The results show that an automatic model based on the U-net can be used to segment spines with scoliosis in CT-images. The results also suggest that by training on more types of bones, a satisfactory model for general bone segmentation can be obtained.

# Acknowledgements

First of all we would like to thank our supervisor Einar Heiberg for all the support, knowledge and engagement he has given to this project. We would also like to thank our assistant supervisor Karolina Gottschalk for support and help with the manual segmentations. In addition, we thank the entire Lund cardiac MR-group. Finally, we would also like to thank the other Master's thesis students in the group. It has been very fun to write this thesis alongside your projects. Our collaboration has given us a lot of useful feedback and meaningful discussions to improve our project and report.

# Abbreviations and Notation

## Deep Learning

Adam - Adaptive moment estimation
ANN - Artificial Neural Network
CNN - Convolutional Neural Network
FCN - Fully Convolutional Network
GPU - Graphics Processing Unit
MLP - Multi-Layer Perceptron
ReLU - Rectifier Linear Unit
SGD - Stochastic Gradient Descent
$\eta$ - Learning rate
$\omega$ - Learnable parameters (weights)

## Medical Imaging

CT - Computed Tomography
HU - Hounsfield Units
MRI - Magnetic Resonance Imaging
ROI - Region-of-Interest

## Statistics

ACC - Accuracy
BFS - Boundary $F_1$ Score
CM - Confusion Matrix
DSC - Dice Score Coefficient
FN - False Negative
FP - False Positive
IoU - Intersection-over-Union (Jaccard score)
JI - Jaccard Index (Jaccard score)
PRC - Precision
SNS - Sensitivity
SPC - Specificity
TN - True Negative
TP - True Positive

# Contents

# 1 Introduction

Recent technological advancements have made medical imaging an essential part of modern medicine, enabling accurate, non-invasive diagnostic procedures. Modern imaging techniques, such as Computed Tomography (CT), have facilitated acquisition of high resolution images, which in turn, has enabled development of more advanced biomedical image analysis methods. These methods involve object identification or classification, texture analysis etc., producing useful information for diagnosis or treatment planning. The first step of such image analysis is usually image segmentation [1]. Biomedical image segmentation refers to the process of dividing a medical image into different regions or objects of interest, for example organs, tissues or other biological structures within the body. This enables physicians to non-invasively visualize and study anatomy, track pathological processes and determine the need for surgeries, for example [2]. Another application is segmentation of the spine and ribs to visualize its structure in diagnosis and treatment of scoliosis. Scoliosis is a condition where the patients spine is crooked and rotated in a three-dimensional way, causing backache and impaired mobility. In severe cases surgery is needed to straighten the spine, which are procedures of high complexity that require meticulous planning [3]. Visualization of the spine in three dimensions by segmentation of CT images facilitates the planning significantly [4].

The company Medviso AB has together with Region Skåne developed software to compute different kinds of three dimensional biomedical segmentations, such as spine segmentation. They have developed the software *Segment 3Dprint*, where it is possible to read images from CT or MRI-scans and segment the images into objects of interest to visualize three-dimensional structures [5]. This software has many useful tools for computing the segmentation manually, such as thresholding. However, these segmentations require extensive manual corrections, in particular when the data is noisy (see Figure 1.1). The adjustments could take several hours to perform, and requires anatomical and technical knowledge. Therefore a faster, automatic method with higher precision is needed.

Automatic semantic segmentation of biomedical images using machine learning techniques has been of high interest for research during the last decade. Deep learning using convolutional neural networks (CNNs) has recently become the prelevant methodology of choice for such tasks. The more recent development of *fully convolutional networks* with advanced architectures like the U-Net by Ronneberger et al. [6] has greatly improved the results of automatic segmentation. U-Net based models have been used for many different biomedical tasks since, such as bone segmentation [7][8].

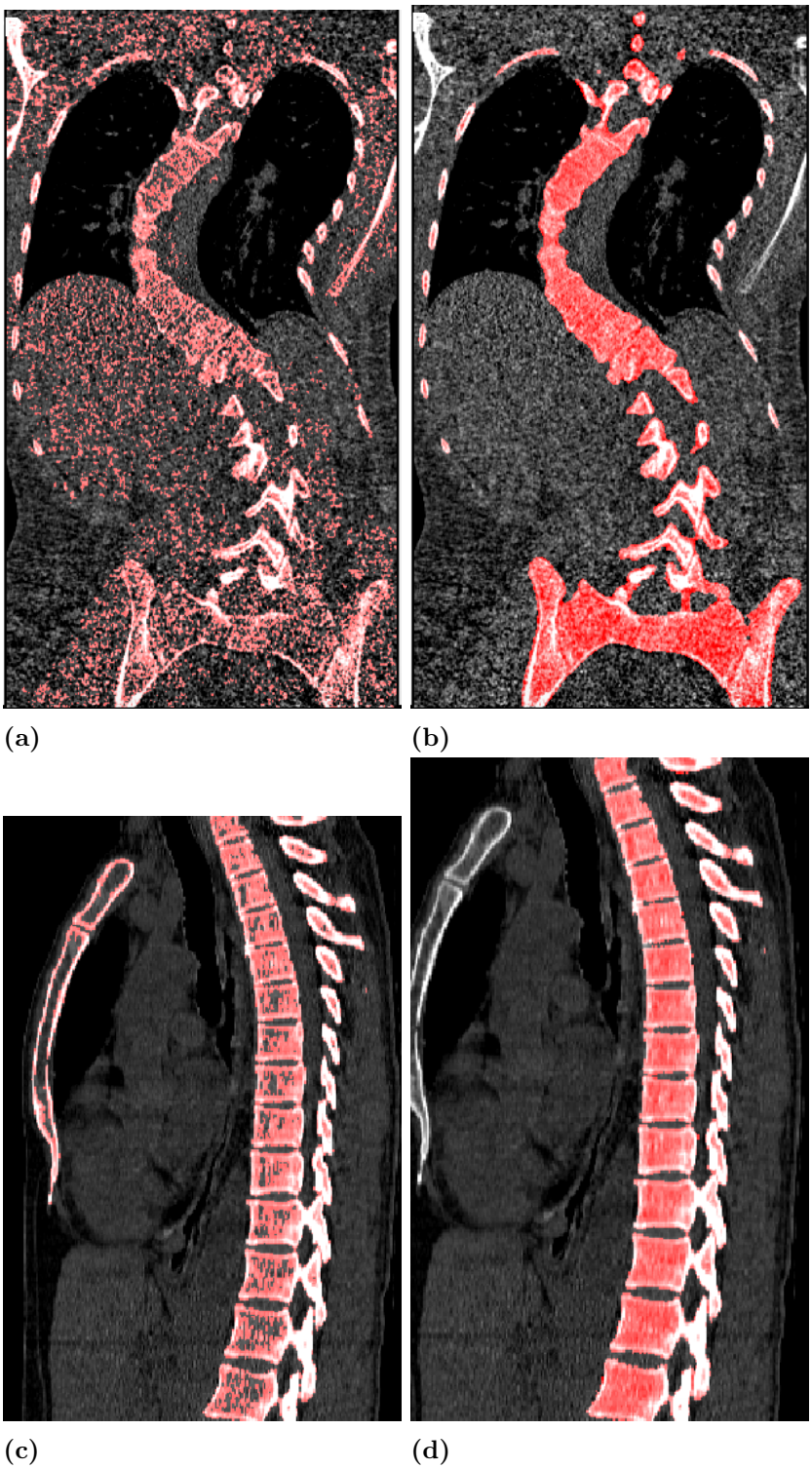



**Figure 1.1:** Comparison of spine segmentations **a),c)** using the thresholding method with **b),d)** corresponding delineated ground truth

# 2 Aim

The primary aim of this thesis is to develop a model for automatic segmentation of the spine in CT-images that is good enough to be used for clinical purposes, such as planning of scoliosis surgery. This model will be based on a fully convolutional network, and inspired by the established network architecture U-net [6]. The model will be trained on the available datasets to perform automatic segmentation of the spine as accurately as possible. Different deep learning approaches and segmentation algorithms will be evaluated to find the best method. The model should be robust towards noise in the images and variations in image size, contrast, brightness etc. The model should not require too much computational memory or power, so that it can be implemented in *Segment 3Dprint*, and run on a regular computer with a moderate size graphics card.

Depending on the result of the primary aim, the secondary aim is to investigate expansion of the model to general bone segmentation. This model will, in addition to spine and ribs, be trained on CT-images of feet, hands, and pelvises. The results will be used to evaluate if it is possible to develop a bone segmentation tool with our method, and how precise we can make the model.

# 3 Theoretical Background

## 3.1 Scoliosis

Scoliosis is a quite common condition, affecting almost 10% of all children in Sweden to some degree. Around three in one thousand have a severe scoliosis that requires treatment [3]. There are three types of scoliosis classified by etiology (origin of disease): *idiopathic*, *congenital*, and *neuromuscular*. Congenital scoliosis appears because of malformation of the spine already in embryological stages [9]. For example a cause can be uneven lengths of the legs, leading to a skewed pelvis and a crooked spine. With this type, the spine straightens out when sitting or laying down [3]. Neuromuscular scoliosis is scoliosis associated with neurological diseases such as celebral palsey or spinal cord trauma. Idopathic scoliosis is when the cause is unknown and all other types can be excluded. Adolescent idopathic scoliosis is the most common type with about 80% of all cases [9]. All types of the condition usually develop with children during the growth, predominantly during the early years of puberty. Scoliosis can cause backache and impaired mobility and is usually treated with a corset, however in more severe cases, surgery is needed [10].

### 3.1.1 Cobb's angle

The most common way of classifying the degree of scoliosis is using Cobb's angle, which is a measure of the curvature of the spine. A Cobb's angle greater than 10° is usually classified as scoliosis and an angle greater than 40° is considered a severe case that might require surgery [3]. To determine the angle, the *apex* needs to be identified, which is defined as the vertebra with the greatest deviation from the center of the vertebral column. The *end vertebrae* are defined as the vertebrae with the most tilt toward the apex, i.e. one superior (above) and one inferior (below) to the apex. Cobb's angle is defined as the angle of the intersection between two lines parallel to the superior and inferior end vertebrae, respectively (see Figure 3.1) [4].

## 3.2 Artificial neural networks and deep learning

### 3.2.1 Artificial neural networks

Deep learning is a group of machine learning methods that are based on Artificial Neural Networks (ANNs). ANNs are inspired by early models of the biological neurons of the brain, and how they are able to learn tasks. Fundamentally, ANNs are networks of connected nodes (or neurons) that send information between them. The main idea behind them is that a neuron receives input from other neurons, and then depending

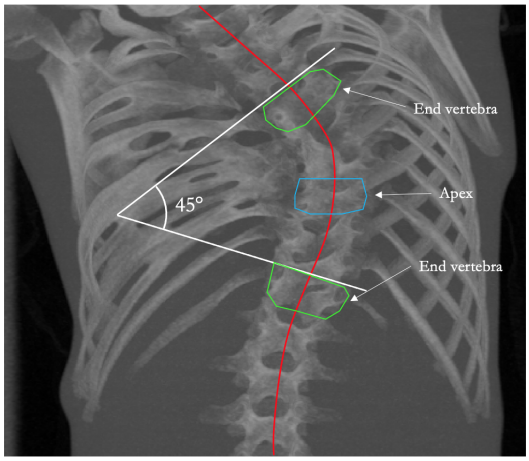



**Figure 3.1:** Example of calculation of Cobb's angle for a patient with severe scoliosis.

on whether that input sums to some threshold, that neuron gets activated or remains unactivated [11]. Each input to every node is multiplied by some weight $\omega_i$ and each node has an activation function $\varphi_0$ (e.g. a threshold function) into which the sum of all inputs are fed. Figure 3.2a shows the simplest ANN called a perceptron.

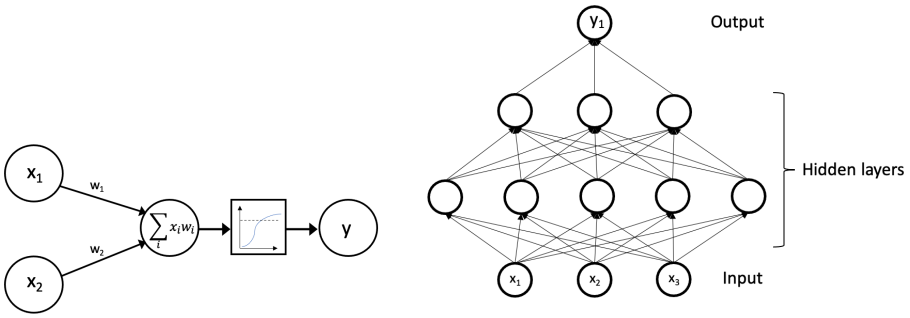
Deep learning means using ANNs with multiple layers of nodes, often refered to as multi-layer perceptrons (MLPs) or deep feedforward networks. The MLPs have an input layer of nodes that feeds information into connected "hidden" layers of nodes that ultimately generates some output [12]. The learning refers to an iterative training process where examples are presented as inputs to the network, which generates some output that is used to adjust the model. When practicing supervised learning, a desired output is known for each example presented during training. Then, by defining some loss function, a measure of the error produced by the network is obtained. The weights of the network are usually initialized to small random numbers and this error is then used to adjust them using the back-propagation learning algorithm, i.e. they are adjusted to gradually minimize the error. As we iterate through the training, the error gets smaller and the model "learns" how to perform the desired task. The trained model can then be used on new unknown examples. A common task is classification where the network is trained to classify some input into two or more categories [11].

### 3.2.2 Loss functions

An important element of deep neural networks is the loss function. There are many possible choices of loss function, however, most modern neural networks use the principle of maximum likelihood. This means that the training data can be seen as a sample from an unknown distribution $p(\mathcal{D})$, and we aim to model that distribution with our network. The model is given by the set of trainable parameters $\hat{\omega}$. Hence, we get a likelihood of getting the sample $\mathcal{D}$ given our model parameters $\hat{\omega}$,

$$p(\mathcal{D}|\hat{\omega}) = \prod_n p(d_n|\hat{x}_n, \hat{\omega})p(\hat{x}_n). \tag{3.1}$$

Here, $\hat{x}_n$ is the set of inputs from the $n$-th sample of the distribution and $d_n$ are the corresponding targets. The network can be trained by maximizing this likelihood, or




**(a)** **(b)**

**Figure 3.2:** **a)** A simple ANN called perceptron with the output function: $y(\hat{x}, \hat{\omega}) = \varphi_0(\omega_1 x_1 + \omega_2 x_2)$. Where $x_i$ denotes the inputs, $\omega_i$ the weights and $\varphi_0$ the activation function. **b)** Example of a MLP with two hidden layers. Each directed edge in the graph represents an operation where the value in the starting node is multiplied with the weight and added to the receiving node.

more commonly, by minimizing its negative logarithm. The $p(\hat{x}_n)$ can be omitted since it is independent of the weights. Hence, the loss or error becomes:

$$E(\hat{\omega}) = -\sum_n \log p(d_n|\hat{x}_n, \hat{\omega}), \tag{3.2}$$

which is the cross-entropy between samples from the distribution (training data) and the predictions of the model. For binary classification tasks this becomes the cross-entropy error and for multiple classes we get the categorical cross-entropy error. The advantage of using a maximum likelihood based loss function is that independent of the task, it can be defined by Equation 3.2 [12].

In some cases, the classes in the available data may be imbalanced. This often occurs when analyzing medical images, where one or more interesting regions or tissues may only be present in small quantities. Training on data with imbalanced classes may impede the learning process since the largest class will affect the loss function more, while the smaller classes will only make small contributions. For example, if we have two classes where one makes up 95% of the data, then the model will get a 95% accuracy simply by labeling every case with this class, and it has not been trained at all. To deal with this problem, a couple of strategies and loss functions have been developed [13].

One loss function that can be used when dealing with imbalanced classes is a further development of the cross-entropy called weighted cross-entropy (WCE). As the name implies, the different classes are here given weights. Thus, the smaller classes can be given larger weights to make them contribute more in the calculation of the loss. The WCE is calculated by,

$$E(\hat{\omega}) = -\sum_n W_n \cdot \log p(d_n|\hat{x}_n, \hat{\omega}), \tag{3.3}$$

where $W_n$ is the weight for class $n$, decided by the user [13].

Another loss function that is useful for segmentation tasks when classes are imbalanced is the Generalized Dice Loss (GDL). GDL is based on the Dice score coefficient (DSC), which is a measure of the overlap between the segmentation and the ground truth (see Equation 3.25 in Section 3.4). In the GDL each class is assigned a specific weight, typically calculated as the inverse area of each class in the ground truth:

$$w_i = \frac{1}{(T_i)^2}. \tag{3.4}$$

Here $w$ is the weight for class $i$ and $T_i$ is the total area of that class in the ground truth [13][14]. The loss functions can be combined in different ways, for example, Klein et al. [7] used a sum of GDL and WCE.

### 3.2.3 Back-propagation algorithm

The back-propagation algorithm is the standard algorithm for computing the updates of the weights during training. It uses derivatives of the output from a node $y$ w.r.t. the weights $\omega_i$ of nodes feeding into it. This gradient gives an indication how the weights needs to be adjusted to reduce the error. Starting with the computed error by the loss function, the gradient can be calculated for the weights of the output nodes of the network. Then, using the chain rule of calculus, the gradients of the error w.r.t. each weight are computed backwards through the network. After computing the gradient, there are many different optimization algorithms for calculating the actual updates of the weights, but the general idea is that the weights descend with the gradient towards a minimum where the error is minimized [12]. The simplest strategy is called gradient descent and involves taking small steps in the negative direction of the gradient (see Figure 3.3). The size of the steps is defined by the learning rate $\eta$, which gives the update formula:

$$\Delta\omega_k = -\eta\frac{\partial E}{\partial \omega_k}. \tag{3.5}$$

There are some issues with this simple error minimization strategy, for example, it can get stuck at a local minimum, or, if there is a plateau in the loss function, it will take a long time to leave it. For large training datasets, calculating the gradient can also be very computationally heavy. A simple improvement is Stochastic Gradient Descent (SGD), where the training data is randomly split into minibatches of some size, and the weights are updated using only one minibatch at a time. Every update of the weights using a minibatch is called an *iteration* and when all training data have been used for updates once, one *epoch* have been performed. The size of the minibatches is usually adjusted according to the amount of computational memory available [15].

Furthermore, there are even more advanced optimization methods that use information from previous iterations to escape local minima and plateaus to more effectively minimize the error [16]. The information from previous updates can be added to the current update as a so called momentum term. Often, this term consists of the running average of the gradient [17].

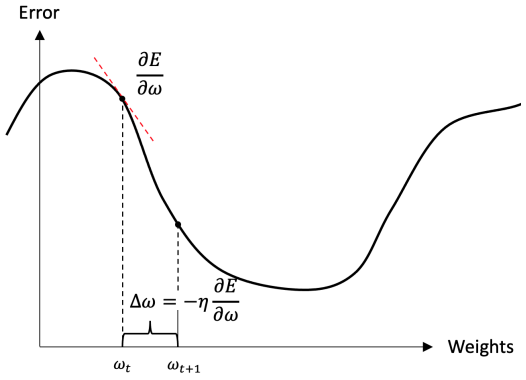



**Figure 3.3:** Illustration of the gradient descent method.

### Adam

One optimization method utilizing the idea of momentum is Adaptive moment estimation (Adam). This algorithm utilizes the concept of momentum in a more advanced way than the simplest methods by being based on an adaptive learning rate. Furthermore, Adam includes a running average of the gradient as well as a running average of the square of derivatives, making it more robust. During training, these first- and second-order moments are calculated and kept in parameters $f$ and $s$ as,

$$f_k(t+1) = \beta_1 f_k(t) + (1-\beta_1) g_k(t), \tag{3.6}$$

$$s_k(t+1) = \beta_2 s_k(t) + (1-\beta_2)[g_k(t)]^2, \tag{3.7}$$

where $\beta_1$ and $\beta_2$ are hyperparameters that have to be chosen by the user and $g(t)$ is the gradient. The moments are usually initialized as zero. To compensate for this, the method calculates estimations of both the first- and second-order moments,

$$\hat{f}_k(t+1) = \frac{f_k(t+1)}{1-\beta_1^t}, \tag{3.8}$$

$$\hat{s}_k(t+1) = \frac{s_k(t+1)}{1-\beta_2^t}. \tag{3.9}$$

Finally, the weights in the network are updated as,

$$\omega_k(t+1) = \omega_k(t) - \eta \cdot \frac{\hat{f}_k(t+1)}{\sqrt{\hat{s}_k(t+1)} + \epsilon}, \tag{3.10}$$

where $\eta$ is the learning rate chosen by the user and $\epsilon$ is a small constant to avoid division by zero [17].

Apart from these two described methods, there are several other algorithms available. Some are more popular to use, but no overall optimal method has been found. The choice of optimization method is often based on the task, as well as the user's previous experiences [17].

### 3.2.4 Generalization performance

The goal of all machine learning algorithms is to perform well on "new" data i.e. data that was not part of the training. This is called generalization performance, and means that we do not only want a small training error, but also a small test error. The two errors correspond to the two main challenges of deep learning algorithms, avoiding under- and over-fitting. Underfitting means that the model we have trainined is not complex enough to solve the task, which means that we will not be able to obtain a low training error. Overfitting means that the model is too complex and gets fitted too closely to the training data, which means that although we get a very small training error the test error will be much larger (see Figure 3.4) [15]. To be able to detect overfitting, it is important to estimate the generalization performance in an unbiased way. Therefore, it is necessary to always set aside a part of the training data for testing. This part of the data must not influence the model in any way. K-fold cross validation is a popular method to get a reliable generalization estimate, where the training data is split randomly into K different parts. Then K different networks are trained, leaving one of the parts out for testing for each one. Finally the generalization performance is given by the average performance of the K networks.

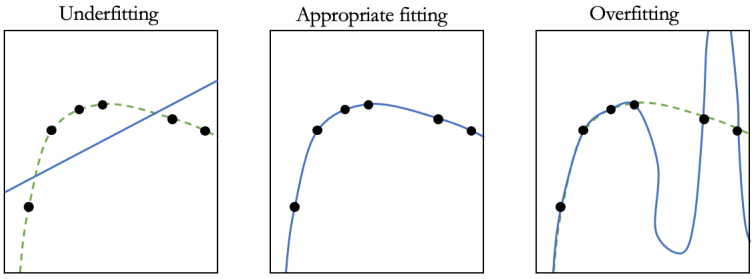



**Figure 3.4:** Illustration of underfitting and overfitting a function to some data points. The green dashed lines are ground truths and the blue lines are predictions.

#### Regularization term

A common way of preventing overfitting is the introduction of a regularization term $\Omega$ to the loss function such that $\tilde{E}(\omega) = E(\omega) + \alpha\Omega(\omega)$, where $E(\omega)$ is the loss function depending on the weights and $\alpha$ is a fixed value called the regularization strength. The regularization term penalizes large weights and makes minimizing the error a trade-off between fitting the data and being small. The suppressed weights give smaller curvature to the model, thereby decreases the risk of overfitting [15]. The most common choice of regularization term is the $L_2$ norm regularization, which is defined as follows [18]:

$$\Omega = \frac{1}{2}\sum_i \omega_i^2. \tag{3.11}$$

### 3.2.5 Convolutional neural networks

Convolutional Neural Networks (CNNs) are a form of neural networks that are useful when dealing with data in matrices or grids such as images. The networks are defined by the use of the mathematical operation convolution [19]. The convolution is essentially an operation that takes two signals, an input signal and a filter, and produces a third output signal. Often the filtering signal is referred to as a kernel. For continuous signals, the convolution is defined by the convolution integral, where $y(t)$ is the output, $x(t)$ the input and $h(t)$ the filtering kernel [20].

$$y(t) = \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)d\tau. \tag{3.12}$$

When working with data on a computer, the signals are usually discrete, therefore, the discrete convolution will be used instead [19]:

$$y(t) = \sum_{a=-\infty}^{+\infty} x(a)h(t-a). \tag{3.13}$$

This is the convolution in 1D, however, in machine learning, when the inputs are multidimensional images, a multidimensional convolution with a multidimensional kernel is needed. For example, if the input is a two dimensional image (I) and a two dimensional kernel (K) is used, the operation becomes:

$$Y(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{3.14}$$

Due to the commutative property of the convolution, the formula can be rewritten as:

$$Y(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n), \tag{3.15}$$

which is a form that is generally easier to explain and implement. The kernel $K$ is a $m$ x $n$ matrix that is multiplied with each of the $m$ x $n$ neighbourhoods of the image $I$ (see Figure 3.5a).

Similarly to a regular deep learning neural network, a CNN is built by a number of hidden layers. However, in a CNN each layer is a convolutional layer, which typically consists of three stages: 1) convolution, 2) activation function, 3) pooling. In the first stage, the input is convoluted with a kernel of weights, which are learnable parameters. A series of convolutions can mathematically be summed as just a single convolution. Therefore, a non-linear activation function is needed [19]. Hence, the linear output of the first stage is then run through the activation function, just like in a regular Multi-Layered Perceptron. A common choice of activation function is the rectifier linear unit (ReLU) defined as [21]:

$$f(x) = max(0, x). \tag{3.16}$$

The convolution with a kernel together with an activation function in CNNs is called a filter. Each filter can be seen as searching for a specific feature in the image. In order for the network to interpret the image, these features need to be combined in different ways. Therefore, in each layer, multiple filters are applied, creating multiple

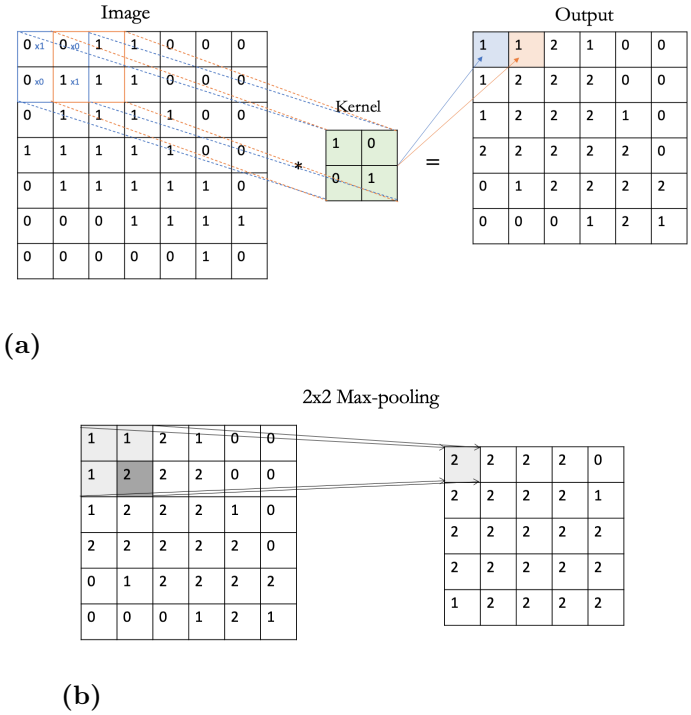



**Figure 3.5:** **a)** Example of 2D Convolution with a 2x2 kernel with stride = 1 and no zero-padding. Note that in CNNs the kernel is always considerably smaller than the input image. **b)** Example of 2x2 max pooling with stride = 1.

channels in the output. Each filter generates one channel in the output, so applying 5 filters to the input generates 5 channels in the output. Furthermore, it is important to note that a filter itself has the same number of channels as the input layer it acts upon (see Figure 3.6a) [19]. The third stage is a down-sampling operation called pooling, to make the output invariant to small translations i.e. small changes in the image would not change the output. The most common pooling function is max pooling, where the output is the maximal value within a rectangular area (see Figure 3.5b). Depending on the properties of the pooling, the function can down- or up-scale the activation map by some amount. It also provides invariance, which is also useful for preventing overfitting [19][21]. Generally, a CNN architecture consists of a couple of convolutional layers, that extract features and downsamples the data into a 1D-array. This array is then used as input to a fully-connected layer which classifies of the input (see Figure 3.6b) [21][1].

There are some general properties of CNNs that make them suitable for machine learning systems where the input is multidimensional arrays. Firstly, they are not dependent on the size of the input image. A regular fully-connected neural network needs exactly one input to every input node. However, in a CNN, as long as the image size is bigger than the kernel, it can take any input size with the only difference being the number of times that the kernel is applied. Hence, after training a CNN, it can perform the desired task on any image, regardless of size [19]. Secondly, they have a *sparse connectivity* of the layers and use *shared weights* between them. If we consider a fully-connected layer, every node of the hidden layer is connected with every input node, with a separate weight for each connection. In a CNN, each node in the hidden layer is only connected with as many nodes from the input as the kernel size. The

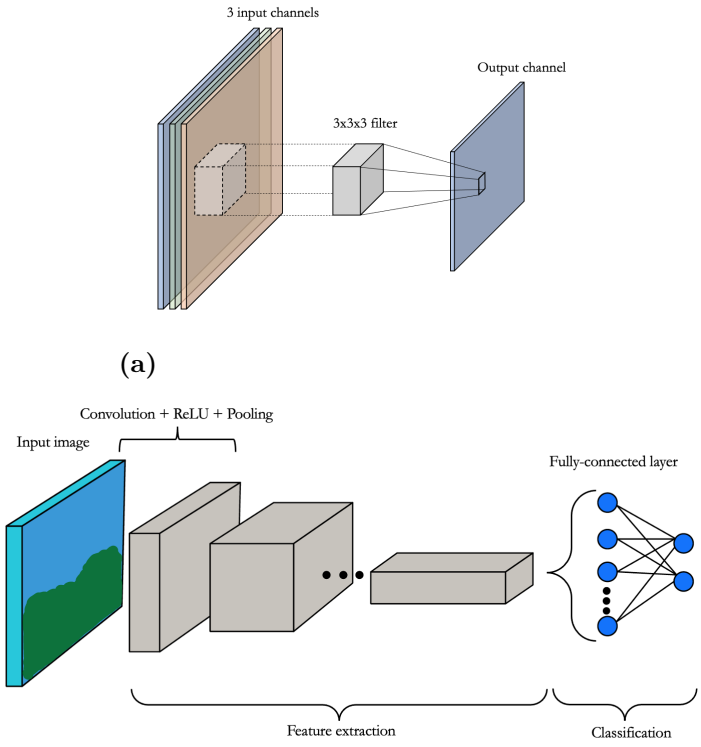



**Figure 3.6:** **a)** An input with 3 channels filtered with a 3x3x3 kernel generating one output channel. **b)** Illustration of a CNN.

weights of the kernel will also be shared for all operations in that layer. For example, an input image can contain thousands of pixels, yet with a smaller kernel (e.g. 10 x 10) we could detect edges in the image using only 100 weights. This means that we can perform many operations like this through the CNN without having to store millions of parameters, which enables detection of more features for a given amount of memory. It also means that we need fewer operations to compute the output of each layer, drastically increasing efficiency and reducing the computational power needed [19].

Besides the kernel size, there are two more hyper-parameters that are related to connectivity and can be used for reducing the complexity of the model: stride and zero-padding. Stride refers to the size of the steps that the kernel takes between each operation. This can be increased from one to bigger steps, further decreasing the connections. Stride is also used in the pooling stage to control the down-sampling. With larger steps, there is a risk of missing features close to the edges of the image. By padding the edges with a border of zeros, this can be avoided [21].

## 3.3 Biomedical image segmentation using deep learning

Biomedical image segmentation used in most diagnostic procedures is computed with manual or semi-automatic methods, where a physician or medical expert uses different computational tools to outline an organ of interest. Completely manual methods require the expert to manually outline the boundaries, image-by-image, which can be highly time consuming and not possible for larger datasets [1]. Semi-automatic methods involve some user-interaction and different algorithms to compute the segmentation such as level-set methods and thresholding. These methods can be more effective than manual segmentation but are often not robust towards imaging artifacts. They also require some knowledge of both the anatomy of the patients and the computational tools [1]. However, high resolution image acquisition, and the increasingly faster CPUs and GPUs of modern computers, has enabled the development of automatic segmentation methods using deep learning.

### 3.3.1 U-Net

In 2015, Ronneberger et al. [6] developed a deep learning architecture named *U-net*. This architecture builds upon the *Fully Convolutional Network* (FCN) developed by Long et al. [22]. Most of the recent studies using deep learning approaches to medical segmentation tasks use either regular CNN architectures or modified FCNs. The main difference between regular CNNs and the FCNs is that the fully connected layers at the end of the CNN (see Figure 3.6b) are replaced with convolutions with kernels that cover the entire input region. This does essentially the same operation, except it produces a heatmap as output which preservers spatial information. This part of the network, from input image to heatmap, is often referred to as the encoding stage. The heatmap is then upsampled using the inverse of a convolution called deconvolution, back to the same dimensions as the input. The deconvolution filters are not fixed, but are also part of the learning [22]. The upsampling path (or decoding stage) eventually generates an output segmentation map that automatically classifies each pixel (see Figure 3.7a). The motivation behind using FCNs over regular CNNs is that they preserve spatial information, that is normally lost in the fully connected layers. When dividing the input image into patches, they can also reuse information from overlapping patches [1].

The U-Net architecture has been modified and extended to return accurate segmentations even with a small number of training images. The architecture proposed by Ronneberger et al. is shown in Figure 3.8. The network has one contracting path (encoding stage) and one expansive path (decoding stage). Since these two paths are almost symmetrical, the network takes the shape of the letter *U*. The left side of the network downsamples the images by following the common architecture of a CNN. It repeatedly utilizes the operation of two 3x3 convolutions, each followed by a ReLU and 2x2 max pooling with stride 2. At every step in the downsampling path, the number of feature channels is doubled. The total number of max-pooling operations gives the "encoder depth" of the architecture. The proposed U-net architecture uses an

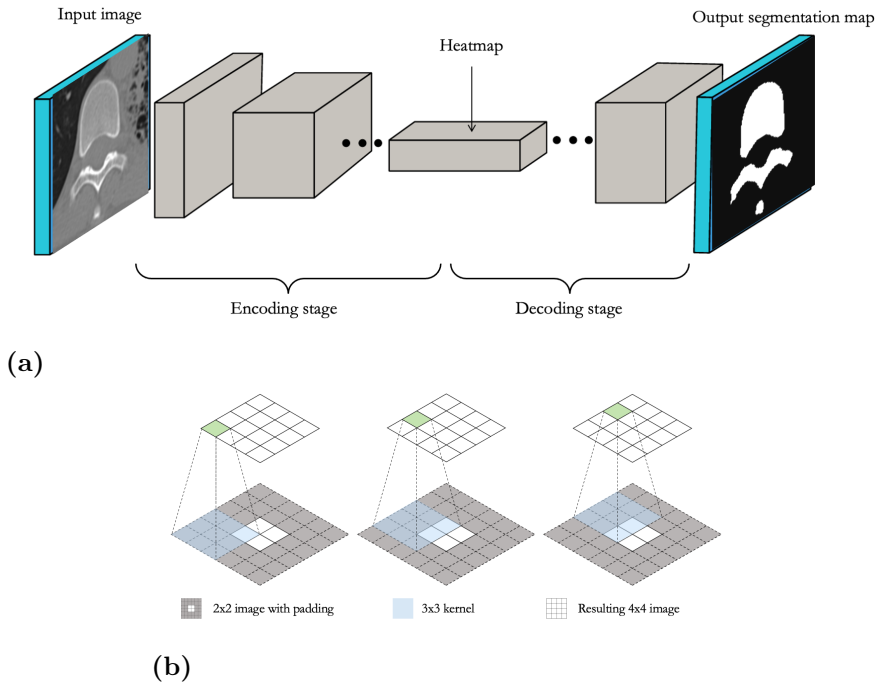



**Figure 3.7:** **a)** Illustration of a FCN **b)** Example of the deconvolution operation, here upsampling a 2x2 image to 4x4 by padding the original before convoluting with a general 3x3 kernel. Illustration by Ruben Bergengrip, used with permission.

encoder depth of four. In the expansive path, the feature map is upsampled followed by a 2x2 convolution. This operation halves the number of feature channels in the images. After the upsampling, a concatenation is performed with the corresponding cropped feature map from the same step in the contracting path. These connections between the contracting and expansive paths are called skip-connections and let the network predict finer details, since the upsampled semantic information is combined with the high-resolution feature maps. Every step is finished with two 3x3 convolutions, each followed by a ReLU. The last layer also performs a final 1x1 convolution to map every feature vector to the number of classes in the segmentation task [6]. The choice of activation function, size and stride of convolutions, encoder depth etc. are all hyperparameters that can be tuned for specific segmentation tasks.

### 3.3.2 Data augmentation

In order to train a network that performs well, a large amount of data is usually needed. However, when it comes to medical images, the available data is often limited. Therefore, several different methods of augmentations can be applied to the original training data. By using data augmentation, more data can be generated and the variability in the data is increased [1]. In biomedical images, there is often a variation in tissues and organs regarding shape and size between different patients. Hence, the use of augmentation can help to obtain a more robust network with a higher generalized performance. Especially when it comes to medical images, it is simple to simulate realistic deformations and thereby increase the variability in the images.

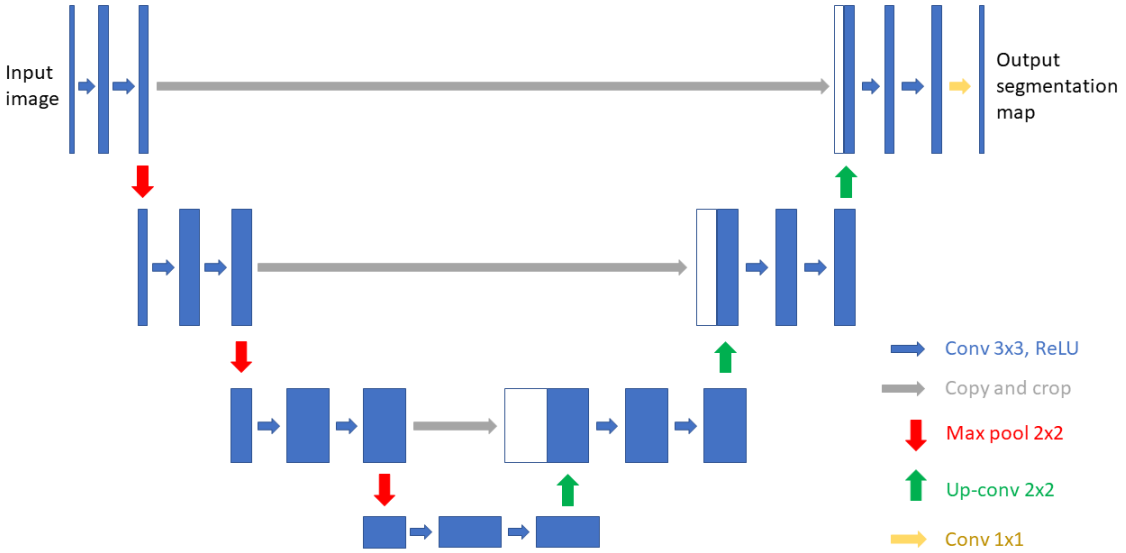



**Figure 3.8:** An overview of the architecture of the U-net proposed by Ronneberger et al. [6]. The blue boxes are multi-channel feature maps. The copied feature maps are shown by the white boxes and the different steps are visualized by the arrows.

When using augmentation, the risk of overfitting is also decreased, since there is a larger variation in the training data [6][23].

As mentioned before, there is a large number of different techniques for augmentation. Listed below are a few examples of commonly used methods [1] [24] [8].

- *Mirroring*: The image can be mirrored both horizontally and vertically by reversing the pixels in the rows and columns.
- *Rotation*: The image can be rotated either left or right in the range 0-360°.
- *Scaling*: The image is either zoomed in or out by multiplying a scaling factor to the coordinates.
- *Brightness*: The brightness can be altered to make the image brighter or darker.
- *Contrast*: The contrast in the image can be increased or decreased by adjusting the intensity.
- *Noise*: The noise level in the image is increased by adding random values. The values are taken from a Gaussian distribution.
- *Blurring*: By applying a Gaussian kernel filter, a blurrier image will be obtained.

Choosing which techniques to use is depending on for example the data and the application at hand [24].

## 3.4 Performance measures

In order to develop a segmentation algorithm it is essential to evaluate the performance of the algorithms on ground truth data. To quantify the performance, different measures can be calculated. The performance measures are important in the process of tuning the classifier parameters to find an optimal model [25] [26].

In semantic segmentation, each pixel of an image is assigned a label from a fixed set. For example, in a two-class problem there are two different labels in the fixed set, one for each class. The measures calculated for evaluation should then depict how well this labelling has been performed, compared to the ground truth [25]. One of the most common methods used to present the results from a classifier is a confusion matrix. For a classification task with C classes, the confusion matrix is defined as,

$$CM = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1C} \\ m_{21} & m_{22} & \dots & m_{2C} \\ \vdots & \vdots & \ddots & \vdots \\ m_{C1} & m_{C2} & \dots & m_{CC} \end{bmatrix}. \tag{3.17}$$

In the confusion matrix, element $m_{ij}$ represents the number of members actually belonging to class $i$, but that have been classified as members of class $j$. In the binary case there are only two classes present. If one class is called "Positive" and the other class is called "Negative", the confusion matrix can be constructed as,

$$CM = \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix}. \tag{3.18}$$

The elements in this matrix are short for the following: TP = True Positives, FN = False Negatives, FP = False Positives, TN = True Negatives. Hence, TP and TN are the number of members that have been correctly classified while FN and FP are the number of members that have been wrongly classified. See also Figure 3.9. From the confusion matrix, a number of different performance metrics can be calculated [26].

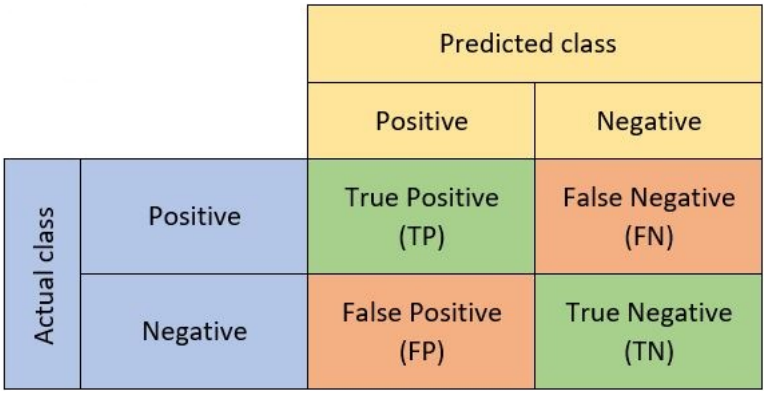



**Figure 3.9:** Confusion matrix for binary classification.

Listed below are some commonly used performance measures. The equations to calculate the different metrics are based on binary classification. However, the same types of metrics can be calculated in classification problems with more than two classes as well.

- Sensitivity (SNS) measures the fraction of how many of the truly positive elements that have been correctly classified, [26]

$$SNS = \frac{TP}{TP + FN}. \tag{3.19}$$

- Specificity (SPC) measures the fraction of how many of the truly negative elements that have been correctly classified, [26]

$$SPC = \frac{TN}{TN + FP}. \tag{3.20}$$

- Precision (PRC) measures the fraction of how many of the elements that have been classified as positive that truly are positive, [26]

$$PRC = \frac{TP}{TP + FP}. \tag{3.21}$$

- Accuracy (ACC) measures the overall fraction of correctly labelled elements, [26]

$$ACC = \frac{TP + TN}{TP + FN + FP + TN}. \tag{3.22}$$

  The accuracy is a commonly used measure and it is easy to calculate. However, this measure has the drawback of introducing bias when the classes are imbalanced [25].

- Per-Class accuracy or mean accuracy sums up the fractions of correctly classified elements for each class and then divides by the number of classes. For the two-class problem, this is the sum of sensitivity and specificity divided by two, [25]

$$Mean = \frac{SNS + SPC}{2}. \tag{3.23}$$

- Jaccard Index (JI) measures the intersection over the union of the segmented elements. This measure can then be reported as a measure for each class or as an average over all classes. The intersection over union for the positive class is calculated as,

$$JI = \frac{TP}{TP + FN + FP}. \tag{3.24}$$

  Since this measure takes both the false positives and the missed values into account, it is often considered a more robust evaluation tool. Yet, it has the constraint to only assess the number of correct elements, and not necessarily how accurate the boundaries have been segmented. An illustration of JI is shown in Figure 3.10 [25].

- Dice score coefficient (DSC) is generally a measure of similarity between two samples. Originally it was used by Dice [27] to compare ecologic association between species, but it has since been adapted as a standard statistical metric. It can be applied to classification problems as a measure of similarity with the ground truth by defining it as [26]:

$$DSC = 2 \times \frac{PRC * SNS}{PRC + SNS} = \frac{2TP}{2TP + FP + FN}. \tag{3.25}$$

- Boundary $F_1$ (BF) score is a measure of how well the boundaries in the segmentation match the contours in the ground truth. $F_1$ score is simply another name for the DSC. The measure introduces a distance error threshold ($\theta$) to decide whether a point on the contour has a match or not, i.e. the precision and sensitivity is calculated based on whether the predicted boundary is within this error threshold of the ground truth boundary. A common choice of threshold for image segmentation is 0.75% of the image diagonal [25] [26],

$$BF = 2 \times \frac{PRC_\theta * SNS_\theta}{PRC_\theta + SNS_\theta}. \tag{3.26}$$

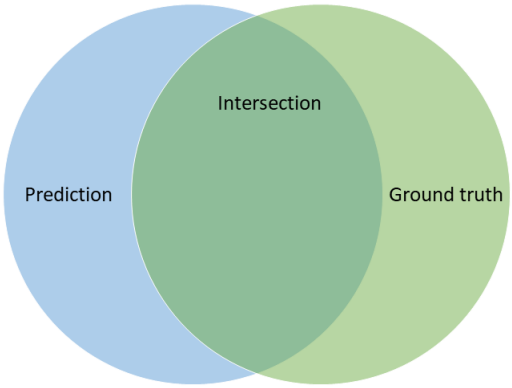



**Figure 3.10:** Graphical illustration of Jaccard Index. JI is calculated by dividing the intersection (the overlap) by the union (the entire area). Illustration by Ruben Bergengrip, used with permission.

## 3.5 Medical imaging

### 3.5.1 Computed tomography

Computed Tomography, often shortened as CT, is a medical imaging technique. It uses x-ray technology to depict detailed images of organs within the body. In a CT-scan, the radiation is sent through a slice of the body from many different directions. This makes it possible to obtain images showing the shape and structure of tissues and organs [28].

The radiation called x-rays was first discovered in 1895 by physicist Wilhelm Conrad Röntgen [29]. X-rays are electromagnetic radiation, just like radio waves and visible light, but with a higher frequency and shorter wavelength [30]. X-rays arise when particles or photons with a high energy are decelerated in a material. The most common source of x-rays is the x-ray tube, where electrons are accelerated from a cathode by a high voltage. The anode consists of a piece of metal, for example tungsten.

When electrons hit the anode with a high speed, they interact with the atoms in the metal, creating x-rays [31][32]. To collect medical images, the produced radiation is sent through the patient and detected by devices sensitive to radiation on the other side. Different tissues in the body attenuate different amounts of x-rays, making it possible to produce an image. In conventional radiography, the x-rays are sent and detected from only one direction. This technique has the benefit of producing images with high resolution and high contrast with a relatively low radiation dose to the patient. However, some disadvantages are geometric distortion and an inability to distinguish information about the depth in the images. This is because a three-dimensional space is reconstructed as a two-dimensional image [30].

Computed Tomography takes on a different approach than conventional radiography. The beginning of how to collect and reconstruct an image is quite similar, with a beam of x-rays being sent through the body to a detector on the other side. However, by connecting the scanning and detector system with a computer and a display medium, the CT-scan is able to display any internal organ in two-dimensional axial slices or reconstruct the three dimensional volume along the Z-axis. As an overview, CT scanners consist of three major elements: a scanning gantry, a data-handling unit and a storage facility [30].

The scanning gantry sends and detects the x-rays to collect enough information about the interesting cross-section to build an image. It consists of one or two x-ray tubes creating and sending x-rays. The x-rays are continuously sent from the tubes as a fan beam, covering the entire width of the patient, as the tubes rotates 360 degrees around the patient. The transmitted x-rays are detected by a stationary ring of detectors. For an illustration, see Figure 3.11a. At the same time as the x-ray tubes are rotating, the patient couch can be moved in or out of the scanning plane. This makes it possible to collect information from several interesting slices in the same scan [30].

In the data handling system, the information collected by the gantry is processed to construct images. From the x-ray absorption profiles from the different directions, the system should reconstruct the x-ray absorption properties of the different tissues in the body, and display it as an image. There are several different mathematical algorithms available for this task and they can be either iterative or analytic. Iterative algorithms start with an initial guess of the two-dimensional absorption pattern. This guess is then compared with the measured data and the difference is used in an iterative manner to update the guess. When the difference between the measured data and the calculated guess is small enough (often below a specified limit), the calculation stops. There are several different analytic techniques, but one example is the filtered back-projection. In this technique, the collected attenuation data for a given scan angle are convoluted with a spatial filter function. The filtered data are then back-projected along the line it was collected from by using the measured value for each pixel along the line. The final reconstructed image is the summation of the back-projections from all scan angles [30]. An example of a reconstructed image is shown in Figure 3.11b.

**(a)** **(b)**

**Figure 3.11:** **a)** Schematic illustration of a CT scan. The x-ray tube (blue) sends radiation as a fan beam through the patient (orange) and the transmitted x-rays are detected by a ring of detectors (yellow). The x-ray tube rotates 360 degrees around the patient. **b)** CT image of a patient's thorax.

### Hounsfield scale

The parameter being measured in a CT scan is the absorption coefficient in different parts of the body. Different tissues absorb different amounts of the incoming x-rays. By knowing how much radiation is sent out and then measure the amount of transmitted radiation through the body, the absorption coefficient can be calculated. When the absorption coefficients are put together, they can be displayed as gray or color scale images. Every pixel in the images represent the relative absorption coefficient of a volume in the cross section being scanned [30]. To get a relative, quantitative measure of the absorption coefficient, Sir Godfrey Hounsfield came up with the Hounsfield scale. The units (Hounsfield Units, HU) in the scale are calculated based on a linear transformation of the standard attenuation coefficient of the x-rays, [33]

$$HU = 1000 \times \frac{\mu - \mu_{water}}{\mu_{water} - \mu_{air}}, \tag{3.27}$$

where $\mu$ = original linear attenuation coefficient of the body, $\mu_{water}$ = linear attenuation coefficient of water and $\mu_{air}$ = linear attenuation coefficient of air. On the scale, water at atmospheric pressure and standard temperature is defined as zero HU and air is defined as -1000 HU. Tissues with lower values appear darker in CT images, while tissues with higher values appear brighter. Soft tissues in the body generally have values between -100 and 100 HU, while cortical bone take on values from 300 to more than 1000 HU [34].

## 3.5.2 Anatomical planes

When it comes to the human body, there is a well established nomenclature describing the anatomy. This includes anatomical planes and terms describing direction and motion. For the terminology to apply, the body is expected to be in the so called standard anatomical position. This means standing up, looking forward with the feet together pointing forward. The arms shall hang along the sides with the palms of the

hands forward and thumbs pointing away from the body. When in this position, the body can be divided by the three anatomical planes. The sagittal plane splits the body in right and left halves. The coronal plane creates anterior and posterior (front and back) parts and the transversal plane divides the body in an upper and lower part (see Figure 3.12) [35].

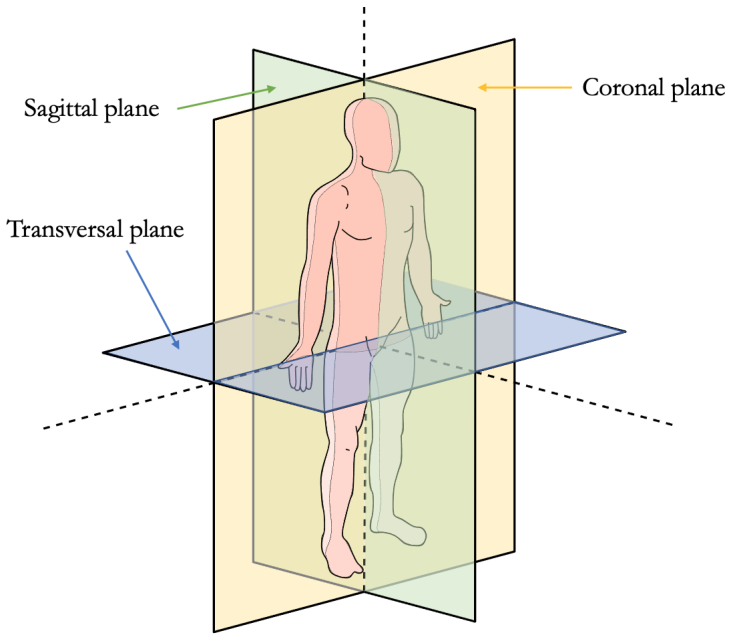



**Figure 3.12:** The anatomical planes.

## 3.6 Related work

Klein et al. [7] developed in 2019 an architecture based on the U-Net for automatic bone segmentation of whole-body CT-images. They compared three different training procedures: training on 2D transversal slices, a "pseudo 3D approach" that uses transversal, sagittal and coronal slices, and an approach with unsupervised pre-training. They performed six-fold cross validation loops on 18 datasets (approximately 7200 transversal slices), with training on 12, validating on 3, and testing on 3 of the datasets. In their "pseudo 3D" approach, the segmentation is run in all direction, and then merged using the mean. To solve the issue with imbalanced classes in bone segmentation, they implemented a combination of cross-entropy error and dice loss as loss function.

In 2020, Isensee et al. [8] developed a deep learning framework for biomedical segmentation based on U-net, that automates the manual tuning of the networks. Their method was trained and evaluated on segmentations of different organs in images captured with different modalities. When running on CT-images, they employed a scheme to preserve the HU-scaling when normalizing images during training, to retain the anatomical information that the HU values reflect. They used the 0.5 and 99.5 percentiles of foreground voxels for clipping, and the foreground mean for normalization.

# 4 Methodology

## 4.1 Data and resources

All computer code was written in Matlab, version 2019a [36]. The reason behind this was to make it easy to integrate the final code and network in *Segment 3DPrint*. In Matlab there are also a number of frameworks and functions available to use for deep-learning. For training and segmentation, two NVIDIA GeForce RTX 3090 and two NVIDIA Titan RTX GPUs were available from the Cardiac MR group.

### 4.1.1 Data

Every dataset was pseudonomized before usage to prevent the spread of patient information. The data consisted of image-stacks collected with CT. 14 datasets came from *The Cancer Imaging Archive* [37]. The other datasets were clinical cases from 3D Centre at Skåne University Hospital. In total there were 31 datasets available for the thesis. The datasets consisted of transversal image slices of certain parts of the body, for example the thorax or pelvis. All datasets came from different patients. 30 datasets had corresponding ground truths from manual segmentations performed by either Karolina Gottschalk or Einar Heiberg. The data were given as .mat files for easy importation to *Segment 3Dprint* for visualization. The images were acquired over a very long period of time. The earliest scans were performed in 1990 and the latest in 2020. Among the patients there were about the same number of males and females. The ages among the patients were in the range 2-80 years. No images were acquired for this project specifically. The type of datasets and type of bone manually segmented in their ground truths is summarized in Table 4.1. The dataset missing a ground truth covered the lower part of a spine suffering from both scoliosis and osteoporosis.

## 4.2 Segmentation algorithms

Two different segmentation algorithms were used, a standard 2D algorithm and a pseudo 3D algorithm based on voting. In both algorithms, a three-dimensional region-of-interest (ROI) is selected containing the spine. This ROI will be automatically divided into 128x128 patches that will be sent to the network. The patches will overlap to ensure that the whole ROI of any dimensions is covered completely (see Figure 4.1b). To avoid splitting the spine between patches in the transversal direction, a patch that automatically tracks the biggest object in each slice was added. Each patch was sent through the network generating a segmentation map, where each pixel is labeled. The confidence scores for the labels were used when deciding the label of the overlapping patches where the label with the highest score was picked.

**Table 4.1:** Summation of all datasets available and used in the thesis. The first column (Type of data) describes the scanned part of a patient. The second column (Ground truth) describes the bones labeled in the corresponding ground truths. The third column (Number of patients) shows the number of patients (number of datasets) in each type.

| Type of data | Ground truth | Number of patients |
|---|---|---|
| Thorax with healthy spines | Spine, ribs | 14 |
| Lumbar spine and pelvis | Spine, pelvis, upper femur | 1 |
| Right side of upper thorax | Spine, ribs, right shoulder | 1 |
| Middle thorax | Spine, ribs | 1 |
| Thorax with scoliosis in spine | Spine, ribs | 2 |
| Arms and hands | Arms, hands | 3 |
| One foot and lower leg | Foot, lower leg | 4 |
| Pelvis | Pelvis | 4 |
| Lower spine with scoliosis | No ground truth | 1 |

In the standard 2D version, the network is only trained on transversal images and the segmentation is run on 2D transversal slices, iterating through the slices of the ROI. A possible issue with this approach is that when the spine is crooked the algorithm might fail to segment the spine correctly since the slices will "cut" the spine at different angles. The voting 3D approach aims to solve this by running the segmentation in the coronal and sagittal directions as well (see Figure 3.12). For this approach the network will be trained on patches from all three directions. Then the ROI will be segmented once in every direction generating three 3D segmentation maps. The label of each voxel in the final segmentation will be picked by voting i.e. if two or more of the segmentation maps classify a specific voxel as spine, it will be labeled as spine in the final map.

## 4.3 Our network

### 4.3.1 Architecture

The artificial neural network used for deep-learning and semantic segmentation in this thesis was inspired by the U-net. As described in Section 3.8, this network consists of an encoder subnetwork connected to a decoder subnetwork. The bias term in every convolutional layer was initialized to zero. In order to make the output segmentation map the same size as the input image, zero padding was applied to the inputs to convolutional layers. Adam was used as optimizer and to prevent overfitting, a L2-regularization term was added. During the project, the depth of the subnetworks was changed between trainings. Other hyperparameters such as learning rate and loss function were also changed. For more information about the parameters used, see Section 4.3.3.

### 4.3.2 Pre-processing

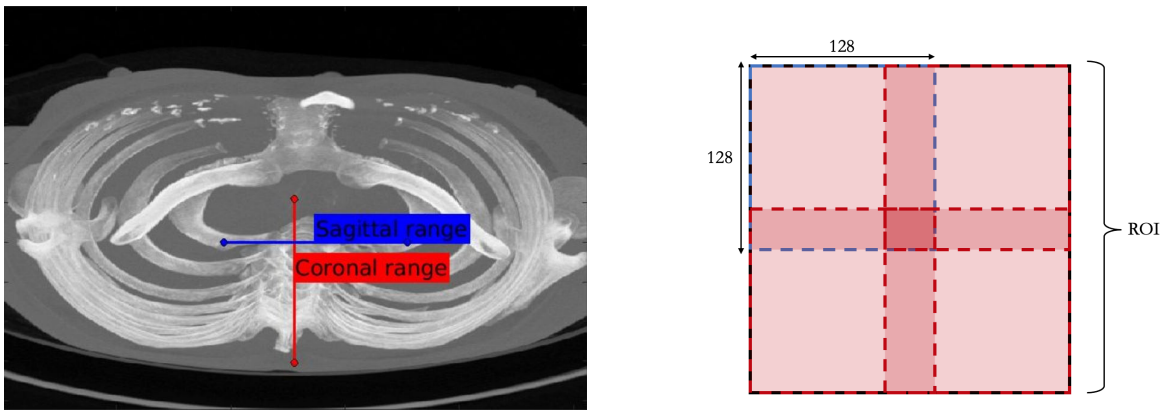
#### Patches

To limit the number of parameters and operations in the network, the input image size was set to 128x128. The size of the output segmentation map was also 128x128 to assign a label to every pixel in the input image. Since the input images are of bigger sizes, they needed to be divided into patches, which the network was trained on (the ground truth maps were also divided in the same way). One advantage of using patches in this manner is that the network can be used for segmentation of an image of any size, as long as it is divided into 128x128 patches.

To create patches that could be used for training and validation, patches were extracted from the original training datasets. The procedure of extracting 128x128 patches from the original images was developed during the course of the project. The final method consisted of one strategy for extracting patches in datasets containing the spine and another strategy for the other types of datasets.

In the spine datasets, a rectangular area containing the spine was selected from the transversal maximum intensity projection of an entire dataset. See Figure 4.1a. Following this, patches were extracted by simply covering the selected area in each slice. The patches were allowed to overlap to avoid missing parts and ensure they were kept within the selected area (see Figure 4.1b). All coronal and sagittal slices within the area were used.

Furthermore, a type of identification of the spine was performed to make sure patches that did not split the spine in half were extracted. In each transversal image slice, the largest object, i.e. the spine, was located in the corresponding ground truth. To determine which patch to be collected, the center pixel of this object was found. Then a 128x128 patch with its center in this pixel was extracted. See Figure 4.2a. In coronal and sagittal image slices, this was done in a slightly different way. To collect patches,




**(a)** **(b)**

**Figure 4.1:** **a)** Example of selection of the interesting area from which patches were extracted. The lines specifying the ranges enabled interaction to choose the ROI depending on the shape of the spine. **b)** Schematic showing overlapping patches covering a ROI.

there was an iteration over the masks row by row, to find the pixel with the highest value. This pixel was presumed to be located in the spine, and thereby a patch was extracted from the image with this pixel in the center. If a patch was extracted, the next 64 rows were skipped to have a maximum overlap of 64 pixels between patches. See Figure 4.2b.

**(a)** **(b)**

**Figure 4.2:** **a)** Example of patch to extract from a transversal image. This patch is shown on the ground truth. **b)** Example of patches to extract from a sagittal image. The patches are shown on the ground truth. The procedure with overlapping patches was performed in the same way with coronal images.

In the datasets covering other parts of the body, the extraction of patches was similar to the strategy used on the spine. However, since the interesting bones in this data were more irregular regarding their shape and generally covered larger parts of the images, no tracking was used. Instead, a rectangular area was once again selected from a maximum intensity projection, containing the interesting bones. The area in each image slice was covered with patches and extracted. To avoid getting too much background in the patches, patches that did not contain any bone segments were not included.

### Bits and hounsfield mapping

After extracting the patches their datatype was converted to 8-bit unsigned integer (`uint8`). This data type can take integer values between 0-255. The reason behind this conversion was that the u-net in Matlab is compatible with `uint8`. It was also to make sure the images and their ground truths had the same data type. The original images have pixel values from the hounsfield scale. Since these values are anatomically dependent, it was desired to keep their relationship after converting to `uint8`. To do that, the images were first converted to grayscale (floating values between 0-1). All pixel values between an upper and lower boundary were distributed over the interval. In order to keep as much relevant information as possible the pixels in the spines in the training data were analyzed. The 2.5 and 97.5 percentiles of all pixel values labeled as spine in the ground truth were set as the lower and upper boundary respectively. The percentiles were chosen to keep as much information as possible of the spine

but avoiding any outliers. These values were rounded to -100 and 1100 and kept for the extraction of all patches. After mapping the images to grayscale, all values were multiplied with 255 and converted to `uint8`.

In the standard method, as expected, two classes were used; "spine" and "background", i.e. each pixel was labeled with one of the classes. Hence, these two classes were the options in the ground truth masks. The values in the masks were in the range 0-255. All values larger than 127 were considered "spine" and smaller values "background".

### Edge class

In an attempt to improve performance of the networks, an artificial class was added to the ground truths. The idea was that all border-pixels of the objects in the masks would belong to this class. A number of different approaches were made to create the "edge". The final method, which is the method that was used in practice, made use of morphological erosion. The erosion was applied on an entire, binary ground truth dataset at once. This three-dimensional erosion was performed with a spherical structuring element with a radius of two pixels. The reason behind selecting such a small structuring element was to make sure only the outer boundary of each object was labeled "edge". Thus, small objects were less likely to be labeled as only edge. Thereafter, the remaining objects were still labeled as "bone" with a pixel value of 255. The pixels that were labeled as "bone" in the original masks but were not present in the eroded objects obtained the label "edge", with a pixel value of 128. The remaining pixels were labeled "background" with a pixel value of 0. An example of a ground truth with three classes is shown in Figure 4.3. In the final segmentation algorithms used in the GUI, all pixels the network classified as "edge" was included as belonging to the spine.

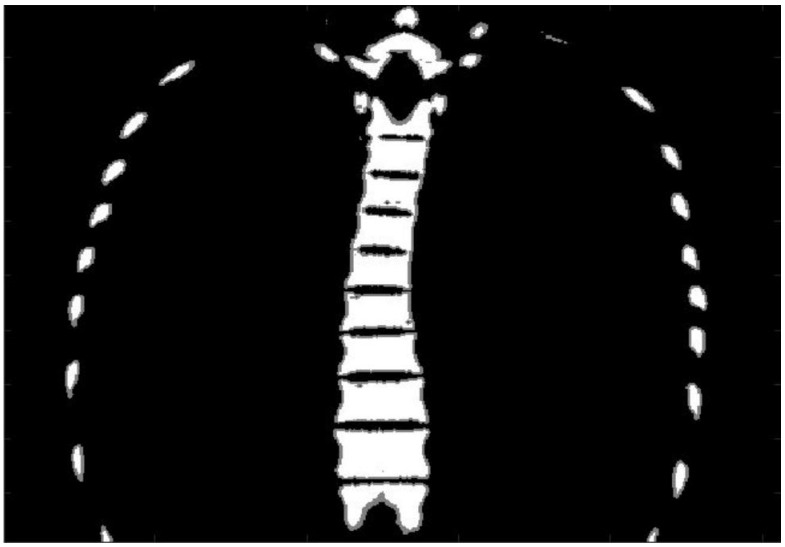


**Figure 4.3:** Example of a coronal ground truth with three classes. Black (value 0) is background, gray (value 128) is the artificial edge class and white (value 255) belongs to the spine and ribs class.

### 4.3.3 Training

#### Hyperparameters

The early stages of the project involved testing different hyperparameters to see how they influenced the model. Adam was early chosen as our optimizer, and kept for all networks. A piecewise learning rate drop schedule was used where the learning rate was multiplied by a factor every epoch, allowing the model to get closer to an optimum. The mini-batch size for each iteration was set to 32 and the data was shuffled every epoch to ensure that no mini-batches consisted of the same data. An L2 regularization term was added to prevent overfitting. Most models were trained for 25 epochs, since the models converged before this, and no improvement of early models trained for longer could be observed. An example of a training curve is presented in Section 5.1. The encoder depth was set to either 4 or 5 for most models. The values for intial learning rate, learning rate drop factor, regularization strength and the Adam hyperparameters were varied in inital tests, but the observed effect of these changes were low. Hence, these parameters were set to the standard values summarized in Table 4.2. Different combinations of loss functions were also tested. The functions evaluated were the dice loss, cross entropy error and weighted cross entropy error. The weights in the weighted cross entropy were calculated by counting the relative frequency of each class in the training data and then subtract it from 1. So if 80% of the pixels in the ground truths were labeled as "background", this class was given the weight 0.2. Combinations of these loss functions were also evaluated, with "dice + regular cross entropy" and "dice + weighted cross entropy".

**Table 4.2:** Standard hyperparameters used. Explanation of the Adam parameters can be found in 3.2.3.

| Hyperparameter | Value |
|---|---|
| Initial learning rate | 0.001 |
| Learning rate drop factor | 0.9 |
| Max epochs | 25 |
| Mini-batch size | 32 |
| L2 regularization strength | 0.00005 |
| Gradient decay factor (Adam) | 0.9 |
| Squared gradient decay factor (Adam) | 0.99 |
| Epsilon (Adam) | 1e-8 |

#### Normalization

Before every training, an additional normalization was applied to all images. The normalization was performed on patches that had already been converted to `uint8`. This was done by subtracting all images with a specific coefficient. The coefficient was calculated by taking the mean of all pixels in all converted training images labeled as "bone". By calculating the mean of all foreground pixels and normalize all images with the same number, their relationship to the Hounsfield values was kept after the

normalization. The numbers used for normalization in the cross-validations and final trainings can be found in Table 4.3.

**Table 4.3:** Values used when normalizing input images to the networks.

| Type of training (training data) | Normalization value |
|---|---|
| Cross-validation (15 spine datasets) | 85.3 |
| Final training spine (15 spine datasets) | 108.6 |
| Final training bone (15 spine + 8 bone datasets) | 117.2 |

### Augmentations

During the process of training networks, augmentation was performed to obtain more robust models. For the first networks only a few augmentation methods were used, but from about halfway through the project, a lot of data augmentation was performed for each training. Most methods that were used are often applied when dealing with medical images. Inspiration to which methods and parameters to use was taken from Isensee et al. [8]. However, only the methods considered relevant for the data and purpose of this project were included. Since it was desired to obtain a model that performed well on scoliotic spines, the methods of rotation, scaling and mirroring were regarded as extra important, especially since there were no datasets with scoliosis included in the training data. Thus, a relatively large amount of rotation and scaling was applied to the images.

One special type of augmentation that was used, was a simulation of osteoporosis in the spine. Osteoporosis is a condition where the balance between degradation and rebuilding of the bones in the body is disturbed such that the bones are gradually weakened from the inside out [38]. This simulation was performed on either one or two entire datasets prior to training. By using the ground truths, a 3 pixels wide edge of the spine could be found in each patch. The edge was then kept unaltered, but all pixels inside the edge was divided by five to make their intensities lower. This augmentation was used for a couple of networks, but the method was omitted when the edge class was introduced. The reason behind the omission was that the edge class was expected to bring a natural boundary to the segmentations. By this, the networks were presumed to "fill" the bones inside the boundaries, even when they suffered from osteoporosis.

Apart from the simulation of osteoporosis, seven other augmentation methods were used in every training. Three methods were performed on-the-fly and the other four were performed before each training. When augmentation is applied on-the-fly, it means that a new augmentation is performed before every new epoch. This means the network never "sees" the exact same patch twice, reducing the risk of overfitting. Therefore, all augmentations should ideally have been made on-the-fly. However, Matlab only have built-in functions to perform a couple of augmentations on-the-fly. There are also difficulties in implementing own methods for on-the-fly augmentation, and therefore some augmentations had to be made before each training.

For all augmentations below, the notation $x \sim U(a, b)$ means that $x$ was picked from a

continuous uniform distribution between $a$ and $b$. The following augmentations were applied for all patches on-the-fly:

- **Rotation.** The images were randomly rotated either left or right. The angle in degrees of which an image was rotated, was drawn from $U(-60°, 60°)$.

- **Scaling.** The images were zoomed in or out depending on the scale factor, which was drawn from $U(0.6, 1.4)$. A scale factor smaller than one meant a zoom out and bigger than one a zoom in.

- **Mirroring.** Every image had a 50% chance of being reflected horizontally. They also had a 50% chance of being reflected vertically.

The four other augmentations that were used were applied to alter the quality of the images. Since images may be acquired under different circumstances and at different locations, the quality can vary. Therefore, noise and blurring could be added and the brightness altered. One recurrent problem for the networks was when the contrast in a dataset was large. Especially when the aorta and other blood vessels had been infused with contrast agents, the networks classified the vessels as bone as well. Because of this, altering the contrast was also added as an augmentation method. Hence, the following augmentations were applied on the patches before training. Every augmentation was applied to a certain image with a probability of 25%.

- **Noise.** Gaussian white noise was added to all pixels in the image. The noise was zero-mean and the variance was drawn from $U(0, 0.1)$. Before noise was added to a certain image, it was turned into a grayscale image with intensities between 0-1. After the noise had been added, the image was turned back to values between 0-255.

- **Blurring.** With this augmentation, the image was blurred by filtering it with a Gaussian smoothing kernel. The standard deviation of the kernel was drawn from $U(0.5, 1.5)$. The size of the filter was calculated from the standard deviation $\sigma$ as $2 \times \texttt{ceil}(2 \times \sigma) + 1$, where `ceil` means rounding the number to the nearest integer larger or equal to itself. Thereby, the size was in the range 3-7.

- **Brightness.** The image was made brighter or darker by multiplying the pixel values with a value $x$ drawn from $U(0.7, 1.4)$. After the multiplication, the values were still kept in the range 0-255.

- **Contrast.** When this augmentation was applied, the contrast in the image was altered. The values in the image was mapped to new intensities (still between 0-255) with a scalar describing the relationship between the original image and the augmented one. The scalar was drawn from $U(0.65, 1.5)$, where a number less than one meant higher output values.

### Cross-validation

To estimate generalization performance and contribute to the choice of hyperparameters, a 5-fold cross-validation scheme was used when training (see Figure 4.4). The datasets used in training were randomly partitioned into 5 folds. Note that each dataset came from a separate patient and it was never split between folds. After finishing the cross-validation loop, a final network was trained and its performance tested on the test data.

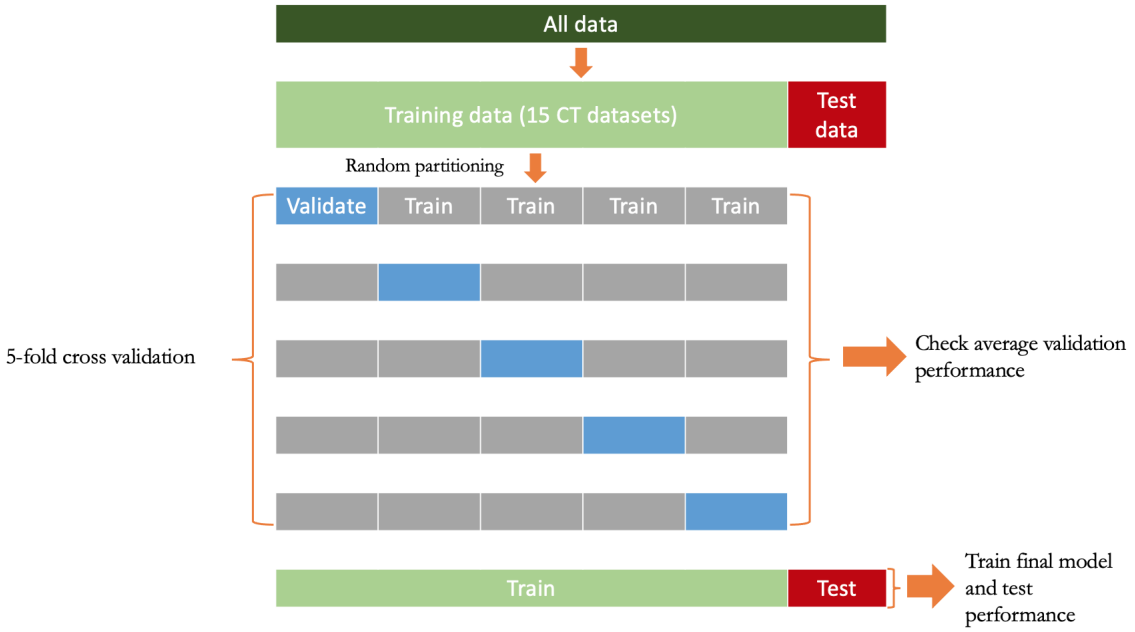



**Figure 4.4:** Scheme of the cross validation used in the training of the networks. The training data consisted of 15 CT-image datasets from healthy (non-scoliosis) patients. CT images of patients with scoliosis are included in the test data. Illustration by Ruben Bergengrip, used with permission.

The 30 available datasets with ground truths were split into two parts, one for training/validation and one for testing. 23 patients were put in the training/validation part and the seven remaining patients were used for testing. The partitioning of datasets were both made at random and by choice. There were only two datasets available from patients with scoliosis. Since it was desired to evaluate how the automatic segmentation performed on spines with scoliosis, both of them were used for testing. The 14 datasets over thoraxes with healthy spines and the dataset over the pelvis and lower spine were put in the test/validation. The other datasets were chosen at random to testing. An overview of the types of datasets used for training/validation and testing can be found in Table 4.4. The 15 datasets used in the cross-validation were the 14 thoraxes with healthy spines and the one over the lumbar spine and pelvis.

### Post-processing

After the networks were used on validation or test images, mainly two methods were used to post-process the segmentations. One method was to extract and use only the

**Table 4.4:** Summation over the number of datasets of each type used for training/validation and testing. The numbers in parentheses are the total number of transversal images.

| Type of data | # Traing/Validation | # Testing |
|---|---|---|
| Thorax with healthy spines | 14 (6121) | 0 (0) |
| Lumbar spine and pelvis | 1 (563) | 0 (0) |
| Right side of upper thorax | 0 (0) | 1 (190) |
| Middle thorax | 0 (0) | 1 (306) |
| Thorax with scoliosis in spine | 0 (0) | 2 (1440) |
| Arms and hands | 2 (1593) | 1 (899) |
| One foot and lower leg | 3 (984) | 1 (513) |
| Pelvis | 3 (1363) | 1 (423) |
| **Total** | **23 (10624)** | **7 (3771)** |

largest connected object in the three-dimensional segmentation. This was performed by a built-in function in *Segment 3D Print* and thereby got rid of small, misclassified parts in the images. The largest object was always used when evaluating the network performance on spines. However, the largest object was not extracted in the shoulder, lower arms, pelvis or foot datasets. This was to not lose "real" objects in the segmentations. For example, since the two arms are not connected, the method would have kept only one of the arms. The other method used for post-processing was a default smoothing. The smoothing was also performed by using built-in functions in *Segment 3D Print*. The smoothing was only applied to create a segmentation more suitable for visualization in three dimensions. Evaluations of the segmentations were made before smoothing.

### Performance measures

Evaluation of the networks was done in two ways. After training with cross-validation, the networks performed segmentation of extracted patches (as described in Section 4.3.2) from the validation data. Considered metrics of these segmentations were global accuracy, mean Jaccard score, mean boundary $F_1$ score as well as Jaccard score and boundary $F_1$ score for both the "bone" and "edge" class. The networks from the cross-validation later also performed segmentation of entire datasets, as described in Section 4.2. For evaluation, the Dice score, the Jaccard index, the boundary $F_1$ score and the accuracy of the "spine/bone" class were considered. These metrics were also used for evaluation of the final networks.

### Training procedure

In the beginning of the project, a number of models with different hyperparameters were tested. Here, the 14 datasets of thoraxes with healthy spines were used for training and validation. For the trainings, two random patients were selected as validation data. Then, when the standard hyperparameters listed in Table 4.2 were established,

training through 5-fold cross-validation was performed. This enabled estimation of the generalization performance of different networks and comparison of performance when using different parameters. In the cross-validation, the 14 datasets of thoraxes with healthy spines and the dataset of the lumbar spine and pelvis were used. Since the cross-validation was performed on 5 folds, every fold consisted of three datasets. The folds were randomized before the start of every cross-validation to avoid introducing bias. The hyperparameters evaluated through cross-validation were encoder depth, choice of loss function and finally the segmentation algorithms (see Table 4.5).

**Table 4.5:** The hyperparameters tested when performing cross-validation. The second column describes which values and types of parameters that were used.

| **Parameters evaluated** | **Values/type** |
|---|---|
| Encoder depth | 4 \| 5 |
| Loss function | Dice \| W CEE \| Dice + CEE \| Dice + W CEE |
| Segmentation algorithm | Transversal 2D \| Transversal 2D (edge)<br>Voting 3D \| Voting 3D (edge) |

After evaluating the encoder depth and loss functions, the choices with the best average performance over the cross-validation loop were selected for the evaluation of the segmentation algorithms. In total, four different segmentation algorithms were tested, each of the algorithms mentioned in Section 4.2, with and without the edge class mentioned in Section 4.3.2. These will be referred to as: Voting 3D (edge), Voting 3D, Transversal 2D (edge) and Transversal 2D in the rest of the report.

In addition, one more network was trained with the spine datasets, as well as three foot, three pelvis and two arm datasets. This was to evaluate how well a network could perform general bone segmentation. The "bone-network" was trained on images from all directions with three classes.

#### Additional testing

After the final networks had been tested on the test data, one more test was performed. In this test, the bone-network was used to segment an additional dataset, which had not been used before. This dataset consisted of images of the lower parts of a spine. The spine in question suffered from both scoliosis and osteoporosis. Because of these complications, the dataset had no corresponding ground truth from manual segmentation. Hence, no performance measurements could be calculated on the segmentations. Instead, only visual evaluation was done to analyse how the network performed on such a difficult dataset.

## 4.4 Statistical analysis

To be able to draw any conclusions from the results, some statistical analysis is needed. After running the cross-validation loop, 15 different scores (one for each dataset) will be obtained for each performance measure. These scores can be further analyzed to

find if there is a signficant difference between two methods, e.g. using the edge class or not. A suitable test for evaluating if two samples from unknown distributions are different is the Student's t-test. The t-test gives a probability (p-value) that the null hypothesis ($H_0$), that there is not any significant difference, is true. The t-score is calculated according to Equation 5.2 below, where $\bar{x}$ is the mean of the sample, $s$ is the standard deviation and $n$ is the number of datapoints. The p-values is taken from the Student's t-distribution using this score.

$$t = \frac{|\bar{x_1} - \bar{x_2}|}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{4.1}$$

In our case, since the performance scores are in paires (e.g. Dice score for Network A and Network B on Dataset D), we can not use the general Student's t-test. However, a solution is calculating the difference between the scores of Network A and Network B for each dataset. If the null-hypothesis is true, the mean of this distribution will be zero, hence we can use the one-sample t-test. This means testing if the population mean is equal to some value $\mu_0$ (in this case, $\mu_0 = 0$). The t-score is calculated as:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \tag{4.2}$$

where $\bar{x}$ is the sample mean, $s$ is the sample standard deviation and $n$ is the number of data points [39]. If the difference in scores between Network A and B has a mean that is significantly higher than zero (i.e. p-value below a pre-determined threshold e.g. 0.05), then A can be considered a better network.

## 4.5 Graphical user interface

As part of the project, a graphical user interface was created for the spine segmentation tool in *Segment 3Dprint*. This GUI will be used in the software for future segmentation tasks. Details can be found in Appendix A.

# 5 Results

In the following sections, the results from the cross-validations and final trainings are summarized. The first cross-validation results are presented with regular plots. In each plot, all validation results for each parameter is plotted with red "+". The results from cross-validations with different segmentation algorithms are presented as box-plots, because of more data points. In every box-plot, the central red mark is located at the median score from the five networks. The bottom and top of each box marks the 25th and 75th percentiles, respectively. The whiskers are extended to the most extreme values, not considering outliers, which are plotted with "+". An example of a box-plot is found in Figure 5.1. The performance of the final networks are presented in tables. In every table, the largest value for each metric is written in bold type for easy localization. In addition, a few images are presented to visualize final, two- and three-dimensional segmentations in the test datasets.

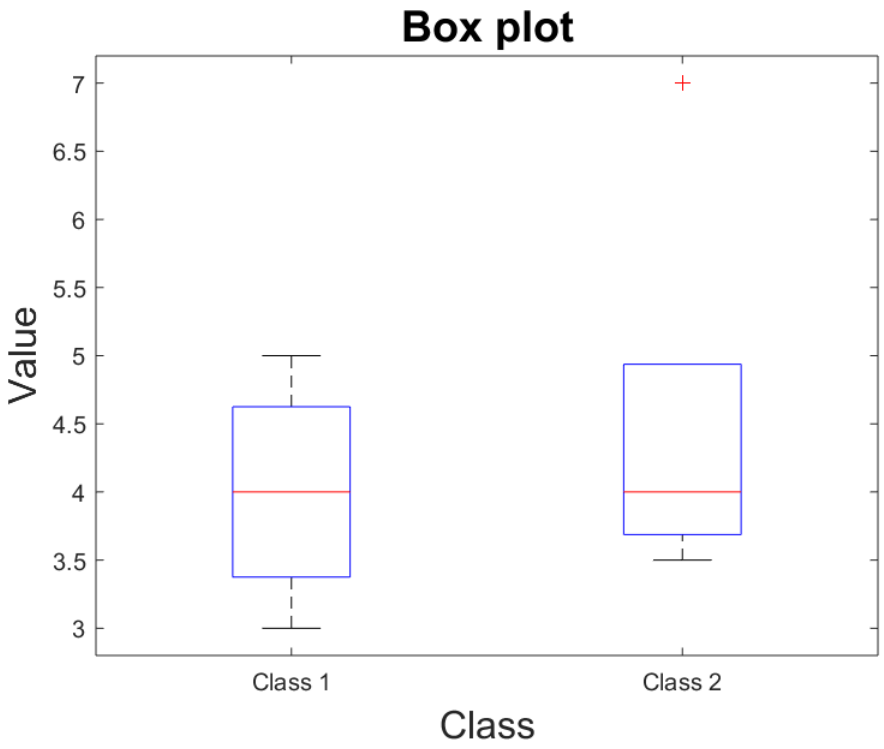



**Figure 5.1:** Example of a box-plot with two classes.

## 5.1 Training curve

An example of a training curve is presented in Figure 5.2. The network was trained on images from all directions with three classes. It used the standard hyperparameters with an encoder depth of 4 and loss function "Dice + Weighted Cross Entropy".

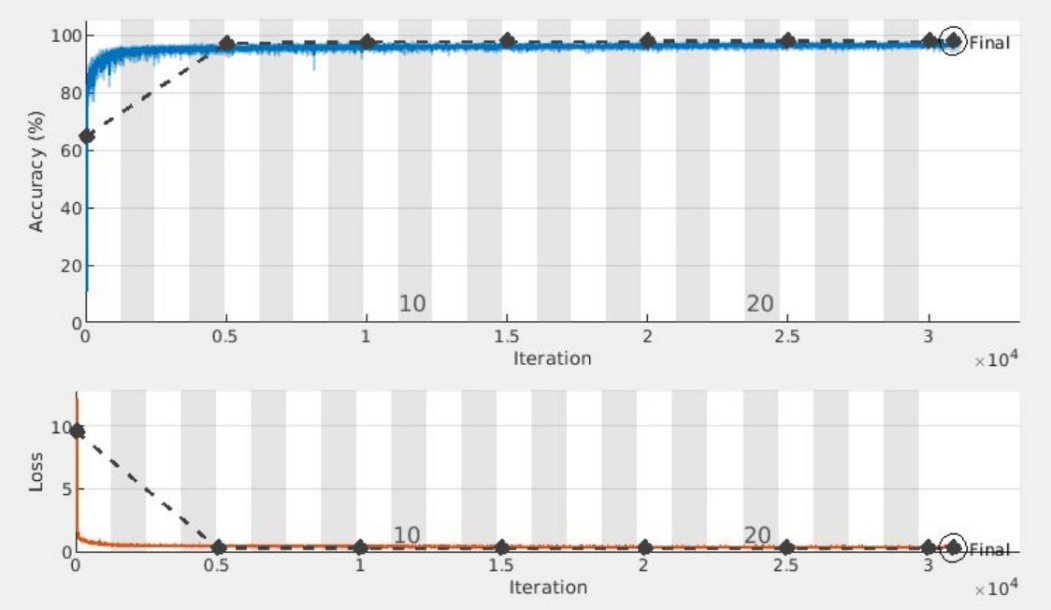



**Figure 5.2:** Example of a training curve. The upper graph depicts the accuracy, where the blue line is the training accuracy and the black, dashed line is the validation accuracy. The lower graph depicts the loss, where the orange line is the training loss and the black, dashed line is the validation loss.

## 5.2 Encoder depth

The results from cross-validation with encoder depth 4 and 5 are presented in Figure 5.3. The only parameter that was changed between these two cross-validations was the encoder depth. The standard hyperparameters listed in Table 4.2 were used, together with "Dice + Weighted Cross Entropy" as loss function. The networks were trained on images from all three directions with three classes. The metrics that were chosen for evaluation were the global accuracy, i.e. the total accuracy of all pixels, the mean Jaccard score and the mean BF score. The networks performed segmentations on extracted patches from the validation folds. From these segmentations the metrics were calculated.

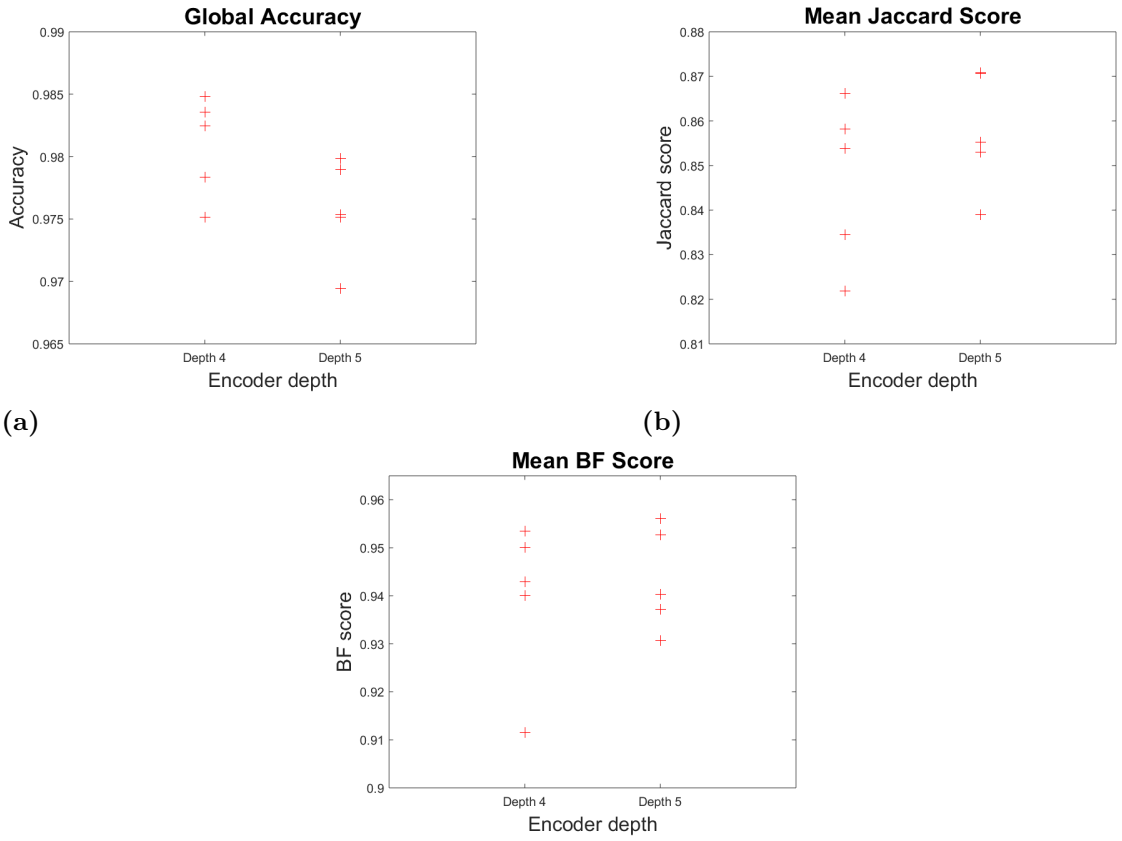



**Figure 5.3:** Results from 5-fold cross validation of different encoder depths of the networks. Evaluated by the performance metrics: **a)** Global Accuracy, **b)** Mean Jaccard score, **c)** Mean BF Score.

## 5.3 Loss functions

The results from cross-validation with different loss functions are presented in Figure 5.4. The only parameter that was changed between these four cross-validations was the loss function. The standard hyperparameters listed in Table 4.2 were used, together with an encoder depth of 4. The networks were trained on images from all three directions with three classes. The metrics that were chosen for evaluation here were also the global accuracy, the mean Jaccard score and the mean BF score. The networks performed segmentations on extracted patches from the validation folds. From these segmentations, the metrics were calculated.

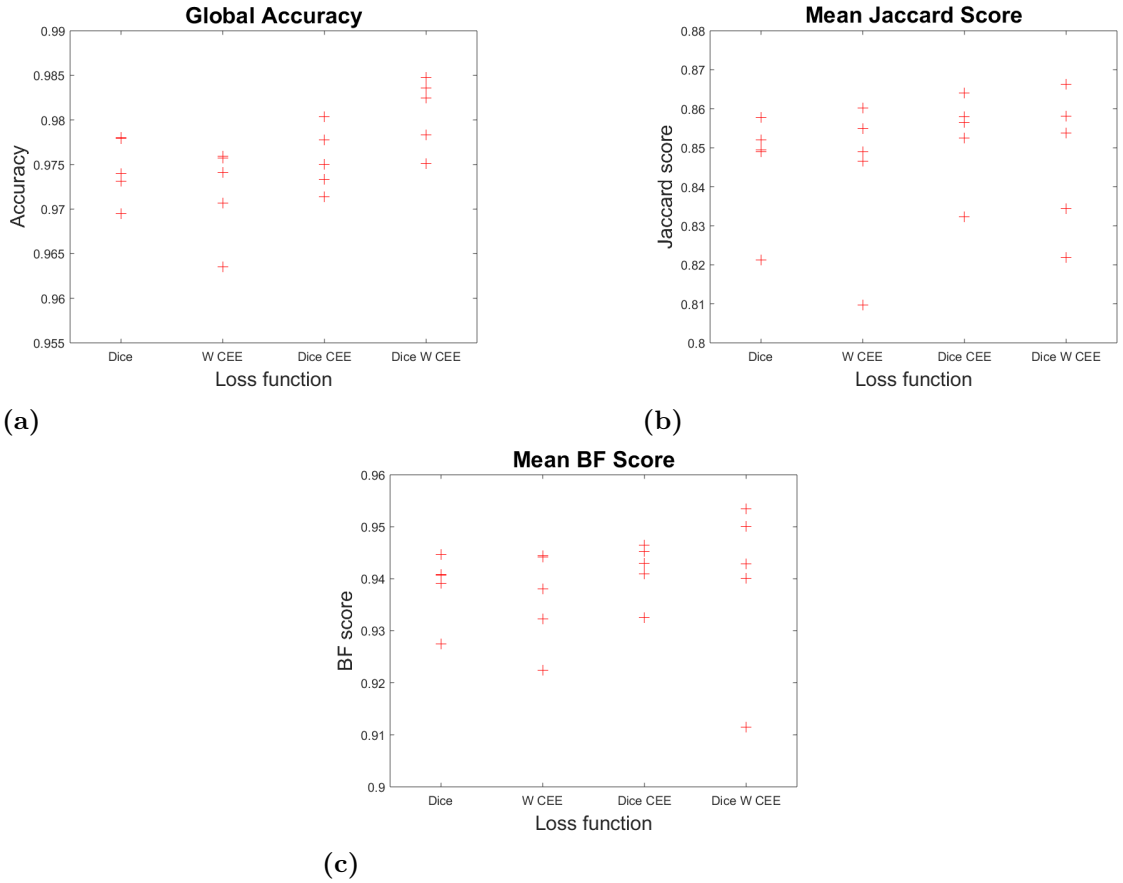



**Figure 5.4:** Results from 5-fold cross validation of the loss functions. Dice = Dice loss, W CEE = Weighted Cross Entropy, Dice CEE = Dice loss + Cross Entropy, Dice W CEE = Dice loss + Weighted Cross Entropy. Evaluated by the performance metrics: **a)** Global Accuracy, **b)** Mean Jaccard score, **c)** Mean BF Score.

## 5.4 Segmentation algorithms

### 5.4.1 Cross-validation networks

The results from cross-validation with the different segmentation algorithms are presented in Figure 5.5 and Table 5.1. For the networks, the standard hyperparameters were used together with an encoder depth of 4 and "Dice + Weighted Cross Entropy" as loss function. The networks used for Transversal 2D segmentation were trained using only transversal image patches. The number of training patches were doubled during augmentation to obtain more training data. For the networks used in Voting 3D segmentation, images from all three directions were used as training data.

The metrics that were chosen for evaluation were the Dice score, the Jaccard score, the BF score and the spine accuracy. The metrics were calculated on segmentations the networks performed on datasets in the validation folds. When doing the segmentations, the attempt was to only include the spine and parts of the ribs. For example, the collar bones were excluded since they were not present in the ground truths. With all the segmentations with different algorithms, the aim was to always segment the same volume in the datasets. All metrics were calculated on the largest segmented object.

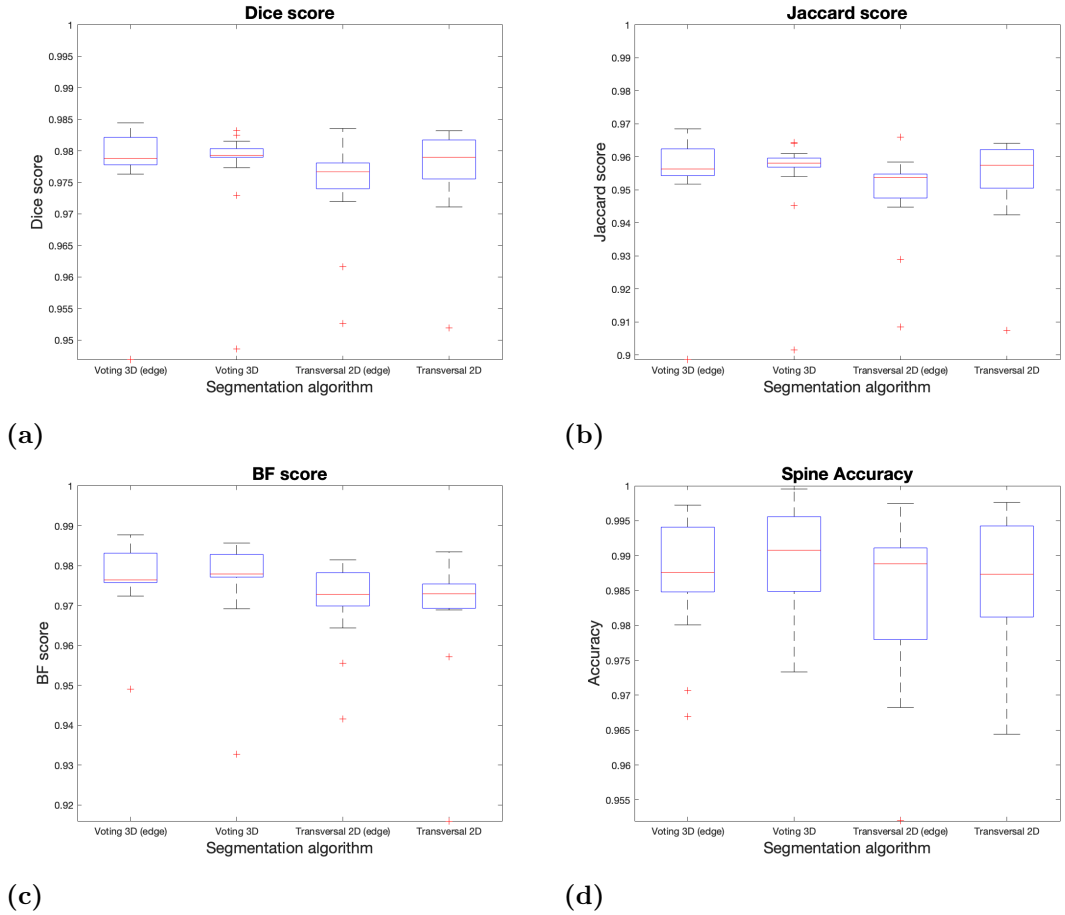



**Figure 5.5:** Results from 5-fold cross validation of the segmentation algorithms. Evaluated by the performance metrics: **a)** Dice score, **b)** Jaccard score, **c)** BF Score, **d)** Spine class accuracy.

**Table 5.1:** Median values (red lines) from Figure 5.5. Voting 3D has the highest median on all scores

| Method | Dice | Jaccard | BF | Accuracy |
|---|---|---|---|---|
| Voting 3D (edge) | 0.9788 | 0.9563 | 0.9764 | 0.9876 |
| Voting 3D | **0.9793** | **0.9581** | **0.9779** | **0.9908** |
| Transv. 2D (edge) | 0.9767 | 0.9537 | 0.9728 | 0.9888 |
| Transv. 2D | 0.9790 | 0.9574 | 0.9729 | 0.9873 |

Using the scores of the different networks, a one-sample t-test was performed highlight any significant differences between the metrics (see Table 5.2).

**Table 5.2:** P-values from the one-sample Student's t-test on the difference of scores between method A and B. P-values below 0.05 are in bold. V3DE - Voting 3D edge, V3D - Voting 3D, T2DE - Transversal 2D edge, T2D - Transversal 2D. *The mean was below zero so no test was performed.

| Method A | Method B | Dice | Jaccard | BF | Accuracy |
|---|---|---|---|---|---|
| V3DE | T2DE | **0.011** | **0.009** | **0.001** | **0.041** |
| V3D | T2D | 0.290 | 0.285 | **0.001** | **0.029** |
| V3DE | V3D | 0.386 | 0.336 | 0.229 | -* |

### 5.4.2 Final networks

The final networks were trained on all 15 datasets used in the cross-validation. One network for each segmentation algorithm was trained. The standard hyperparameters were used together with an encoder depth of 4 and "Dice + Weighted Cross Entropy". The networks were used on the four test datasets with spines (two scoliosis, one middle thorax and one right side of upper thorax).

The test results on the first scoliosis dataset can be found in Table 5.3 and the second scoliosis dataset in Table 5.4. All metrics were calculated on the largest segmented object. Images of the final scoliosis segmentations performed by the Voting 3D (edge) algorithm are shown in Figure 5.6. As these images depict, the volumes including the entire spines and some parts of the ribs were included in the segmentations.

The test results on the middle thorax dataset can be found in Table 5.5 and the right shoulder and spine dataset in Table 5.6. All metrics were calculated on the largest segmented object. For both these test sets, the volumes including the spines and small parts of the ribs were included in the segmentations. Figure 5.6 shows images of the final segmentations performed by the Voting 3D (edge) algorithm.

**Table 5.3:** Final test results of segmentation algorithms for the first scoliosis set.

| **Segmentation algorithm** | Dice Score | Jaccard Score | BF Score | Spine Accuracy |
|---|---|---|---|---|
| Voting 3D (edge) | **0.941** | **0.889** | 0.903 | **0.932** |
| Voting 3D | 0.926 | 0.866 | 0.917 | 0.876 |
| Transversal 2D (edge) | 0.935 | 0.880 | **0.927** | 0.921 |
| Transversal 2D | 0.915 | 0.848 | 0.903 | 0.881 |

**Table 5.4:** Final test results of segmentation algorithms for the second scoliosis set.

| **Segmentation algorithm** | Dice Score | Jaccard Score | BF Score | Spine Accuracy |
|---|---|---|---|---|
| Voting 3D (edge) | **0.912** | **0.841** | **0.848** | **0.889** |
| Voting 3D | 0.906 | 0.831 | 0.844 | 0.876 |
| Transversal 2D (edge) | 0.880 | 0.792 | 0.833 | 0.864 |
| Transversal 2D | 0.883 | 0.799 | 0.829 | 0.839 |

**Table 5.5:** Final test results of segmentation algorithms for the thorax set.

| **Segmentation algorithm** | Dice Score | Jaccard Score | BF Score | Spine Accuracy |
|---|---|---|---|---|
| Voting 3D (edge) | **0.968** | **0.936** | **0.930** | **0.950** |
| Voting 3D | 0.959 | 0.920 | 0.919 | 0.928 |
| Transversal 2D (edge) | 0.960 | 0.923 | 0.925 | 0.931 |
| Transversal 2D | 0.951 | 0.905 | 0.912 | 0.912 |

**Table 5.6:** Final test results of segmentation algorithms for the spine (shoulder) set.

| **Segmentation algorithm** | Dice Score | Jaccard Score | BF Score | Spine Accuracy |
|---|---|---|---|---|
| Voting 3D (edge) | 0.933 | 0.872 | 0.902 | **0.998** |
| Voting 3D | 0.944 | 0.891 | 0.913 | 0.985 |
| Transversal 2D (edge) | 0.952 | 0.906 | **0.927** | 0.967 |
| Transversal 2D | **0.953** | **0.908** | 0.923 | 0.987 |

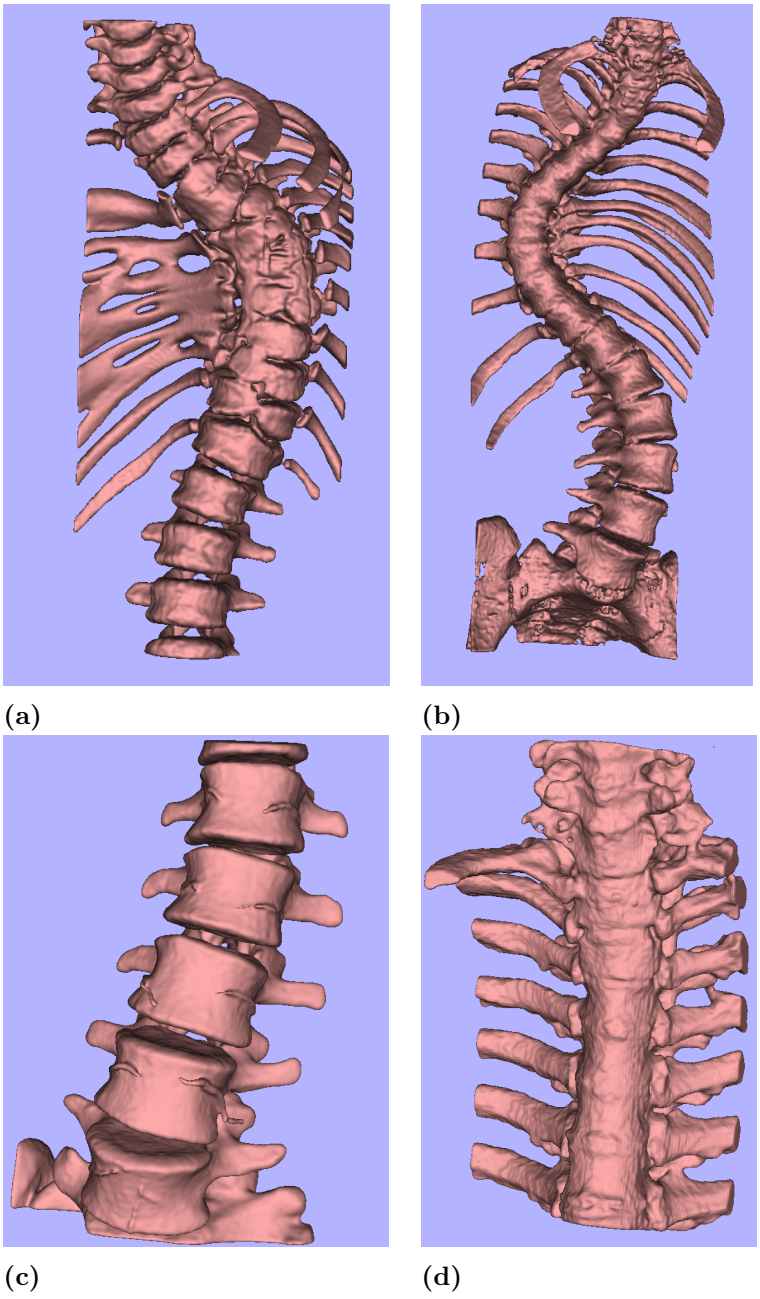


**Figure 5.6:** Results of the final 3D Voting edge network. 3D views of automatic segmentations of datasets: **a)** Scoliosis 1 **b)** Scoliosis 2 **c)** Thorax **d)** Spine (shoulder).

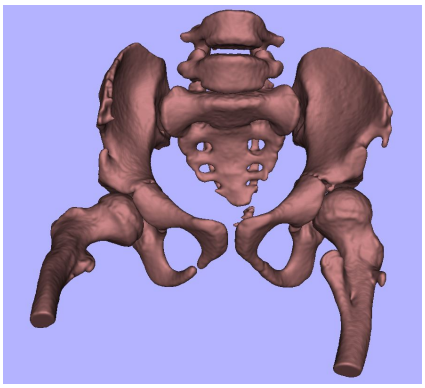
## 5.5 Bone segmentation

The last model was trained for general bone segmentation. In addition to the 15 spine datasets, this network was also trained on three pelvises, three feet and two lower arms. It was trained on images from all three directions with three classes in the ground truths. The standard hyperparameters were used with an encoder depth of 4 and the "Dice + Weighted Cross Entropy" as loss function. The network was tested on all test datasets and Voting 3D (edge) was used as segmentation algorithm. Thereafter, the network trained on only spines for the Voting 3D (edge) algorithm was also tested on all test datasets. This was to be able to compare the performance of the general bone segmentation to a network trained on only spines.

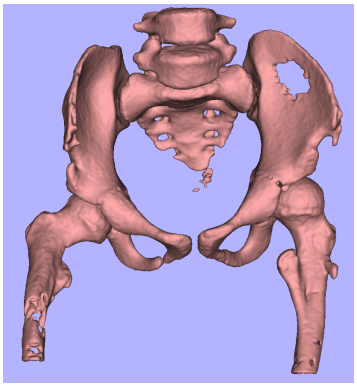
The test results on the pelvis dataset are listed in Table 5.7. The entire volume was included in the segmentations. Figure 5.7 shows the final segmentation performed by the "bone-network" compared with the segmentation by the "spine-network". It can be seen that the spine-network leaves a hole in the pelvis and misses a part on the right femur.

**Table 5.7:** Final test results on the pelvis set from the networks trained on only spine and general bones. Both with the Voting 3D (edge) algorithm.

| **Type of network** | Dice Score | Jaccard Score | BF Score | Bone Accuracy |
|---|---|---|---|---|
| Only spine | 0.900 | 0.840 | 0.931 | 0.890 |
| Bone | **0.985** | **0.969** | **0.990** | **0.994** |



**(a)**



**(b)**

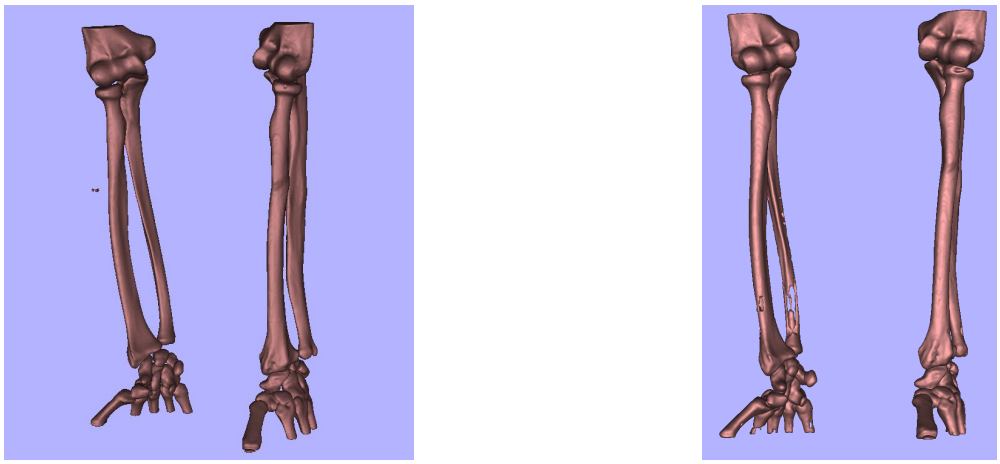
**Figure 5.7:** Anterior views of segmentations of the pelvis test set by **a)** the bone-network and **b)** the spine-network.

The test results on the lower arms dataset are listed in Table 5.8. The entire volume was included in the segmentations. The final segmentations performed by the networks are shown in Figure 5.8. The spine-network has some troubles in segmenting the right arm.

The test results on the foot dataset can be found in Table 5.9. The entire volume was included in the segmentations. Images of the final segmentations performed by the networks are shown in Figure 5.9. These images depict that the spine-network did not segment all the small bones in the foot.

**Table 5.8:** Final test results on the lower arms set from the networks trained on only spine and general bones. Both with the Voting 3D (edge) algorithm.

| **Type of network** | Dice Score | Jaccard Score | BF Score | Bone Accuracy |
|---|---|---|---|---|
| Only spine | 0.884 | 0.794 | 0.824 | 0.895 |
| Bone | **0.974** | **0.948** | **0.997** | **0.998** |

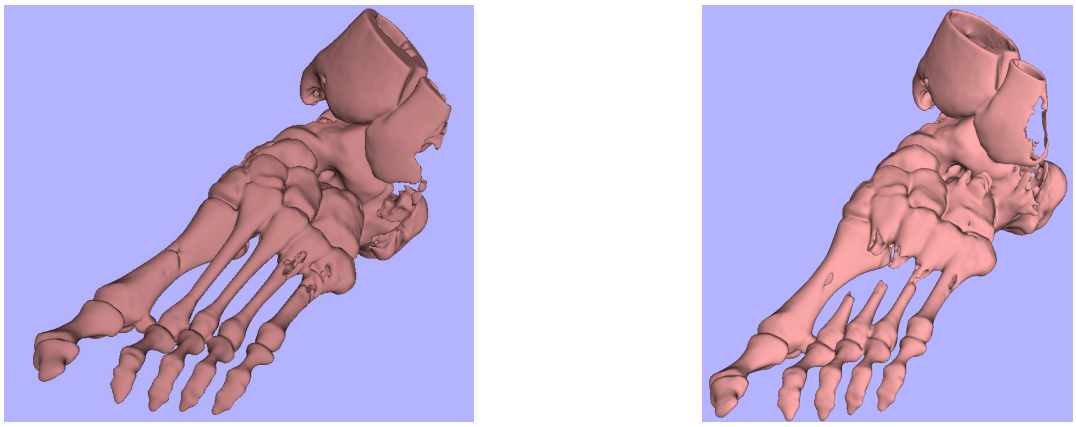


**(a)** **(b)**

**Figure 5.8:** Segmentations of the arms test set by **a)** the bone-network and **b)** the spine-network.

**Table 5.9:** Final test results on the foot set from the networks trained on only spine and general bones. Both with the Voting 3D (edge) algorithm.

| **Type of network** | Dice Score | Jaccard Score | BF Score | Bone Accuracy |
|---|---|---|---|---|
| Only spine | 0.782 | 0.662 | **0.741** | 0.686 |
| Bone | **0.873** | **0.783** | 0.728 | **0.788** |



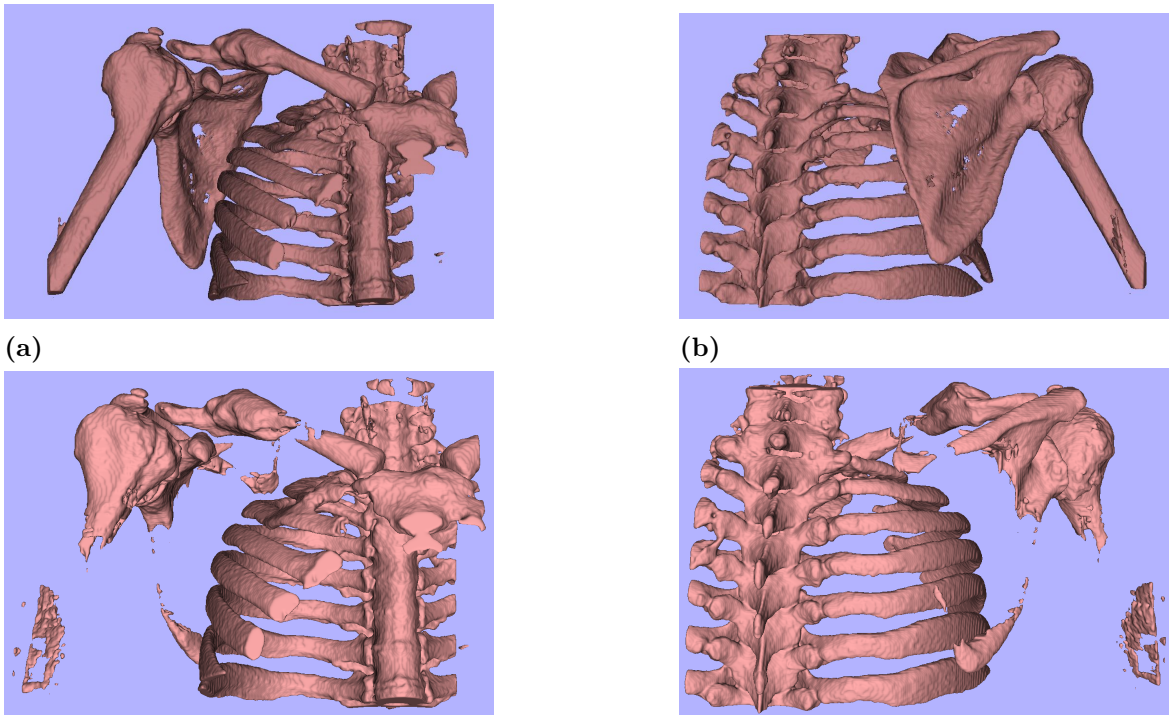
**(a)** **(b)**

**Figure 5.9:** Segmentations of the foot test set by **a)** the bone-network and **b)** the spine-network.

The test results on the dataset with the shoulder and upper thorax are summarized in Table 5.10. Images of the final segmentations performed by the networks are shown in Figure 5.10. As the images depict, the shoulder, the spine and a large part of the ribs on the right side were included in the segmentations. When observing the images, there are large differences regarding the segmentations of the shoulder and upper arm.

**Table 5.10:** Final test results on the shoulder set from the networks trained on only spine and general bones. Both with the Voting 3D (edge) algorithm.

| **Type of network** | Dice Score | Jaccard Score | BF Score | Bone Accuracy |
|---|---|---|---|---|
| Only spine | 0.835 | 0.722 | 0.836 | 0.852 |
| Bone | **0.921** | **0.853** | **0.930** | **0.985** |



**(a)** **(b)** **(c)** **(d)**

**Figure 5.10:** Segmentations of the shoulder test dataset by the bone-network in **a)** anterior view and **b)** posterior view. Segmentations by the spine-network in **c)** anterior view and **d)** posterior view

The test results on the first scoliosis set are summarized in Table 5.11 and the results on the second scoliosis set can be found in Table 5.12. Lastly, the test results from segmenting the middle thorax set are listed in Table 5.13. When observing these segmentations visually, the differences were small. Hence, no images of the segmentations are presented.

**Table 5.11:** Final test results on the first scoliosis set from the networks trained on only spine and general bones. Both with the Voting 3D (edge) algorithm.

| **Type of network** | Dice Score | Jaccard Score | BF Score | Bone Accuracy |
|---|---|---|---|---|
| Only spine | **0.941** | **0.889** | 0.903 | **0.932** |
| Bone | 0.924 | 0.863 | **0.932** | 0.884 |

**Table 5.12:** Final test results on the second scoliosis set from the networks trained on only spine and general bones. Both with the Voting 3D (edge) algorithm.

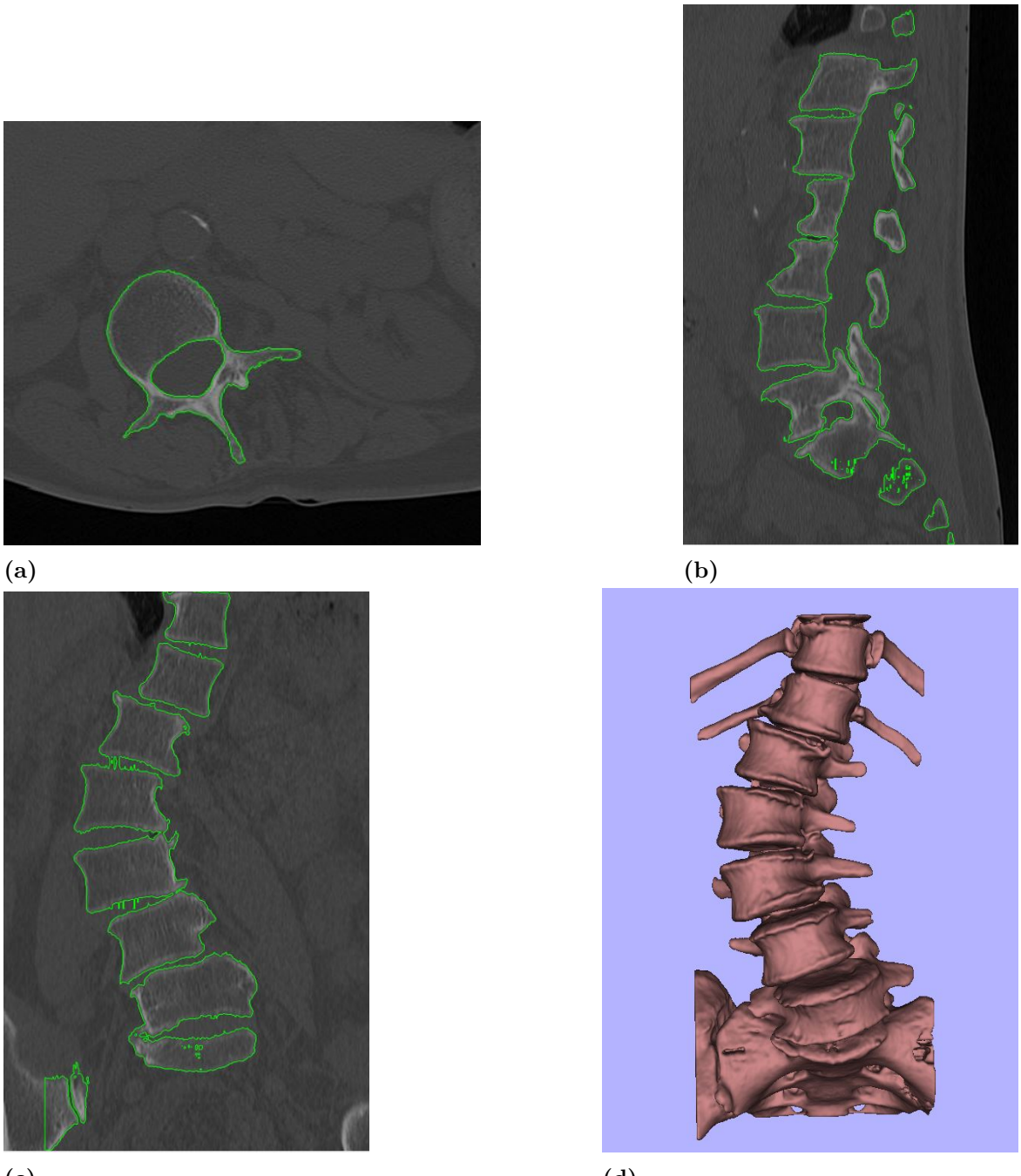
| **Type of network** | Dice Score | Jaccard Score | BF Score | Bone Accuracy |
|---|---|---|---|---|
| Only spine | **0.912** | **0.841** | 0.848 | **0.889** |
| Bone | 0.892 | 0.807 | **0.867** | 0.836 |

**Table 5.13:** Final test results on the middle thorax set from the networks trained on only spine and general bones. Both with the Voting 3D (edge) algorithm.

| **Type of network** | Dice Score | Jaccard Score | BF Score | Bone Accuracy |
|---|---|---|---|---|
| Only spine | **0.968** | **0.936** | **0.930** | **0.950** |
| Bone | 0.950 | 0.904 | 0.891 | 0.916 |

### 5.5.1 Osteoporotic spine

Since the osteoporotic, scoliotic spine did not have any ground truth from manual segmentation, no performance metrics could be calculated for this dataset. Instead, only visual evaluation could be performed. Examples of the segmented images by the bone-network are shown in Figure 5.11. Apart from small green areas in the lower vertebrae in Figure 5.11b and Figure 5.11c, the spine is filled within the outer boundary (green line).



**(a)** **(b)**

**(c)** **(d)**

**Figure 5.11:** Example of segmented image slices in the osteoporotic, scoliotic spine dataset. **a)** Transversal image **b)** Sagittal image **c)** Coronal image **d)** Frontal 3D view of segmentation (largest object).

# 6 Discussion and Conclusion

## 6.1 Discussion

In general, the type of data available for this thesis was considered to be good. The data was acquired over a very long period of time. Because of the technical development, the images acquired in 1990 were collected with a different type of CT-scanner than the ones in 2020. Thus, the images and their quality may differ, which possibly made the networks more robust. Since almost half of the datasets were taken from *The Cancer Imaging Archive*, it means that the images were acquired at different locations. Among the patients there were about the same number of males and females and their ages had a wide range. This is favorable, since it means the networks were not exclusive for one gender or one specific age. However, one negative aspect with the available data was the amount of test data. For example, it could have been valuable to have a few more datasets covering spines with scoliosis. Then, an even stronger conclusion could have been drawn on how well a certain model performs on this type of data. In addition, it also would have been interesting to evaluate the performance on more bones that were not present in the training data, like the shoulder dataset.

The results suggest that the augmentation methods worked well in increasing robustness of the models to difficult datasets, such as the scoliosis datasets. The special augmentation method with simulation of osteoporosis was, as previously mentioned, used for a couple of models in the beginning. This augmentation method was however omitted when the "edge" class was introduced. This was due to the belief that the artificial class marking the boundaries of the bones would serve the same purpose as artificially weakening the bones in a dataset. The "edge" class gave the networks the opportunity to learn the appearance of boundaries and from that hopefully learn to classify the insides as bone. This hypothesis is further validated from the results in Figure 5.11. Unfortunately, the dataset did not have a ground truth, but from visual evaluation, the network fills the vertebrae and does not leave any large holes. Even though this dataset suffered from both scoliosis and osteoporosis, the bone-network performed very well.

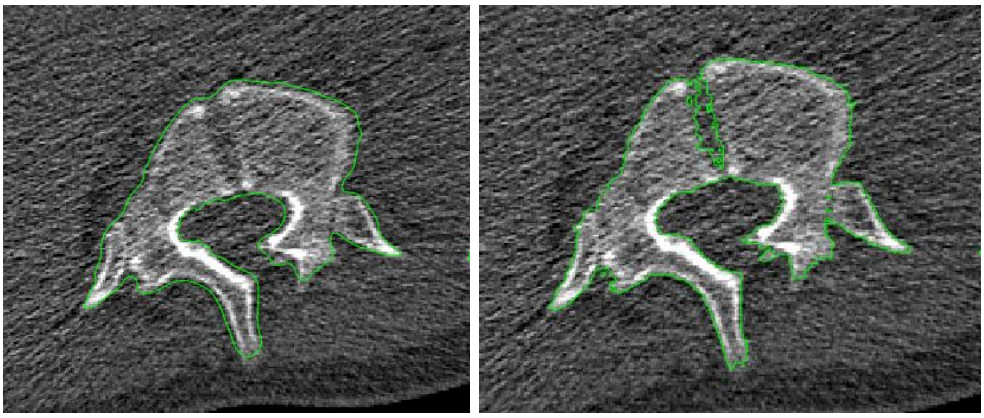
Overall, the calculated metrics from the cross-validations were high. When evaluating different encoder depths in Figure 5.3, it is difficult to say that one depth is better than the other. In all three box-plots the results are vastly overlapping. Hence, these results indicate that networks with encoder depth four do not perform worse than networks of depth five. A network with depth five has a size of almost 400 MB while a network with depth four has a size of less than 100 MB. Therefore, since the networks with depth four performs well, they were preferred for this application and implementation in the software.

The results from different loss functions shown in Figure 5.4 are difficult to draw conclusions from. Most results overlap, and no loss function shows a significant better

or worse performance than the others. Although it seems as if the "dice + weighted cross entropy" has a higher global accuracy. At the same time, this loss function has the widest range for the BF score, with both the largest and smallest value. Since the results do not show that one loss function is significantly better, the "dice + weighted cross entropy" was chosen for the final networks. The main reason behind this was that both parts in this sum of loss functions were depending on the size of the classes. This loss function was also similar to the one used by Klein et al. [7], which worked well for a similar application as ours.

When looking at the box-plots in Figure 5.5, it is clear that there is little significant differences between the algorithms. Voting 3D seems to have the least spread, and looking at the BF score plot in Figure 5.5c, it seems that Voting 3D edge and Voting 3D gets slightly higher scores than the other methods, but other than that, it is hard to draw any conclusions from this result. Unfortunately, by randomizing the folds each iteration and by segmenting an arbitrary area around the spine etc. we introduced random nuisance variables. However, even though our cross-validation procedure is not completely valid from a statistical point-of-view, Table 5.2 does show that using either of the voting 3D algorithms seems to be advantageous over the transversal 2D methods. For example, the Voting 3D (edge) algorithm gets significantly better scores than Transversal 2D (edge) for all metrics. Furthermore, when comparing the voting 3D algorithms, the mean of the difference (V3DE-V3D) is above zero in all metrics except for spine accuracy, even though there was no significance. This coincides with our visual assessment of the segmentations, that Voting 3D edge generally produces the best results.

For the final networks tested on the scoliosis datasets in Tables 5.3-5.4, Voting 3D edge scores the highest on 7 out of 8 performance scores. Furthermore, it also scores the highest for 5 out of 8 scores on the thorax and shoulder datasets in Tables 5.5-5.6. A point of discussion is how sensitive these performance measures are to changes in segmentation result. Since we are segmenting 3D volumes, an algorithm that fills the vertebrae and the ribs will get generally good scores, especially for jaccard score and spine accurcacy. For example, displacing the boundary, or wrongly classifying voxels outside of the spine does will not have a big effect on these scores, since the amount of correctly classified voxels inside the spine will greatly outnumber the wrongly classified voxels. However, misclassifications like this will be evident when evaluating the results visually, hence our visual assessment should be considered a valid measure of the performance. It should also be noted that these metrics depend greatly on the ground truth segmentations, which were computed manually. These manual segmentation can be difficult to get completely right, especially for the scoliosis cases. When studying some slices more carefully, the automatic segmentation seemed to better fit the edge of the spine than the ground truth in some cases (see Figure 6.1). For cases like this, the algorithm will get lower scores, even though it generates a better segmentation than the manual method.



**(a)** **(b)**

**Figure 6.1:** Comparison between the manual and automatic segmentation on a transversal image slice in the second scoliosis dataset. **a)** Manual segmentation **b)** Automatic segmentation performed by the bone-network.

Since it performed the best, the Voting 3D (edge) algorithm was used for the general bone segmentation network. When comparing the performance of the bone-network with the final Voting 3D (edge)-spine-network on the general bone datasets (Tables 5.7-5.10), the bone-network outperforms the spine-network on 15 out of 16 performance measures. The one out of the 16 was the BF score for the foot set (see Table 5.9, Figure 5.9). Both the algorithms struggled with segmenting this dataset, which is likely due to the presence of metal bone screws and severe osteoporosis. It is clear from the visual results that the bone-network performs better, especially for the shoulder dataset in Figure 5.10. One interesting observation from these results is that the bone-network performs well on the shoulder dataset. The network did not have any similar data to train on. Nonetheless, it shows relatively high metrics and visually the segmentation only leaves one small hole in the shoulder blade (see Table 5.10, Figures 5.10a-5.10b). When testing the bone-network on the scoliosis and thorax datasets, the spine-network outperformed the bone-network in 10 out of 12 performance measures (see Tables 5.11-5.13). However, these differences were smaller than for the general bone datasets.

When choosing which type of network and algorithm to implement in *Segment 3DPrint*, we suggest the Voting 3D (edge) algorithm. This segmentation algorithm would be used together with a network trained on images from all directions with three classes. The type of network to use may depend on the desired application in the software. The bone-network performs much better on general bones, while the spine-network seems to be slightly better on spines with and without scoliosis. One suggestion is therefore to have both types of networks available in the software, and depending on the dataset, choose the appropriate one for segmentation. The implemented networks will be trained on all available data (both the training and testing data) to obtain a network trained on as much data as possible. In theory, those networks should be as good as possible, since more training data should mean a better model. However, the drawback with such an approach is that there is no data available to test how well the networks actually perform.

## 6.2 Future Work

### 6.2.1 Limitations and possible improvements

In addition to the first normalization of the images where they were mapped to `uint8`, one more normalization was performed before training. Here, all pixels were subtracted with the mean value of all bone pixels. All images were subtracted with the same coefficient to keep their relation to the Hounsfield scale. Ideally, the images also would have been divided with the standard deviation of the bone pixels. This would have transformed them closer to zero mean and unit variance. However, Matlab 2019a does not support division in the normalization, hence, only subtraction of the mean was used for normalization.

The parameters evaluated through cross-validation were encoder depth, loss function and segmentation algorithm. Preferably, even more differences in hyperparameters would have been used in the cross-validation. However, since the cross-validation algorithm was very time-consuming (one cross-validation took approximately three days), there was not enough time to evaluate more hyperparameters.

When it comes to the cross-validations, the folds were randomized for every new type of parameter. The reasoning behind this was to avoid introducing the bias correlated to always using the same folds. However, in hindsight it actually could have been an advantage to always use the same folds. Then, when validating a certain dataset, the networks would always have been trained on the same data, eliminating that variability. Therefore, a more conclusive statistical analysis of the validation results could have been used to see if there were any significant differences between parameters. Now, since different folds were used, there was a natural variance between the validation results depending on the training data. When this variance is larger than the difference between parameters, the optimal parameter values are difficult to find. Furthermore, to better evaluate the performance of the final networks, having more datasets of scoliosis patients with ground truth available would have been an improvement. Finally, to further examine the statistical significance of our results, it would have been interesting to test the intra- and inter-observer variability for the ground truth segmentations. This would require multiple physicians to outline the same segmentations, multiple times, and finding the variance between them. If the performance of our model would fall within the same variance, then we could conclude that it had reached the optimal performance.

### 6.2.2 Cobb's Angle

Additionally, a simple algorithm for calculating Cobb's angle on a segmented spine was developed. The algorithm used the centers of segmented objects, to identify keypoints along the vertebral column. The keypoints were identified in 3D, however the Cobb's angle was only analyzed in the coronal plane. A polynomial of degree five was fitted to the keypoints (using a least-squares method), to obtain a curve following the spine. Instead of identifying specific vertebrae, the local max and/or min of this curve was considered the apex and the corresponding inflection points the end vertebrae. Straight

lines were drawn perpendicular to the spine curve at the inflection points. The Cobb's angle was calculated at the intersection of these lines (see Figure 6.2). We were not able to evaluate the algorithms performance since we did not have any verified ground truth data of Cobb's angles. Hence, this is just an example of how our segmentation algorithm can be used for future work.

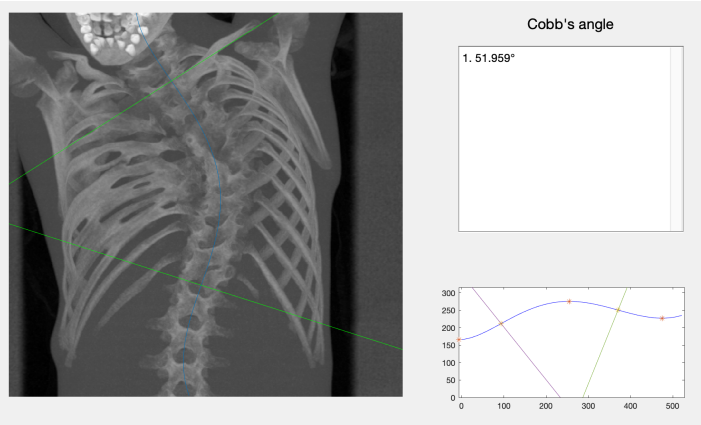



**Figure 6.2:** Example of running our Cobb's angle algorithm on a segmented scoliosis dataset.

## 6.3 Conclusion

The primary aim of this master thesis was to develop a U-net based model for automatic segmentation of the spine in 3D CT-images, that is good enough for clinical use in scoliosis. With an average Dice score of 0.927 (±0.020) and Jaccard score of 0.865 (±0.034) on the scoliosis datasets, our Voting 3D edge model performed the best. We can with the high scores, together with our visual assessments, conclude that we successfully achieved the primary aim. The final model will be implemented in the next version of *Segment 3D Print* [5], where it will be used clinically .

Furthermore, we managed to extend our model to general bone segmentation where our bone-network scored an average Dice score of 0.938 (±0.052) and Jaccard score of 0.888 (±0.086) on the four different bone datasets (pelvis, arms, foot, shoulder). It also becomes clear that training on only spine data is not sufficient for segmenting general bone datasets. This is obvious when studying the segmentation of the shoulder dataset (Figures 5.10c-5.10d), where the spine-network makes a good segmentation of the spine and ribs, but misses the entire shoulder. It should also be noted that the bone-network performed slightly worse than the spine-network on the spine datasets. So, in conclusion, to generate a good general bone segmentation, it is necessary to train on multiple different bone structures of the body. However, to generate an as-good-as-possible model for the spine segmentation task, it would be better to train on only spine images.

# Bibliography


[1] Intisar Rizwan I Haque and Jeremiah Neubert. "Deep learning approaches to biomedical image segmentation". In: *Informatics in Medicine Unlocked* 18 (2020), p. 100297. ISSN: 2352-9148. DOI: https://doi.org/10.1016/j.imu.2020.100297. URL: https://www.sciencedirect.com/science/article/pii/S235291481930214X.

[2] Ahmed Elnakib, Georgy Gimel'farb, Jasjit Suri, and Ayman El-Baz. "Medical Image Segmentation: A Brief Survey". In: *Multi Modality State-of-the-Art Medical Image Segmentation and Registration Methodologies* (Apr. 2011), pp. 1–39. DOI: 10.1007/978-1-4419-8204-9_1.

[3] Ernesto Martinez and Björn Strömqvist. *Skolios*. URL: https://www.1177.se/sjukdomar--besvar/skelett-leder-och-muskler/rygg-och-nacke/skolios/. accessed: 2021-03-11.

[4] Hana Kim, Hak Sun Kim, Eun Su Moon, Choon-Sik Yoon, Tae-Sub Chung, Ho-Taek Song, Jin-Suck Suh, Young Han Lee, and Sungjun Kim. "Scoliosis Imaging: What Radiologists Should Know". In: *RadioGraphics* 30.7 (2010). PMID: 21057122, pp. 1823–1842. DOI: 10.1148/rg.307105061. URL: https://doi.org/10.1148/rg.307105061.

[5] *Segment 3DPrint - Software solution for medical 3D printing*. Medviso AB. URL: http://medviso.com/segment-3dprint/. accessed: 2021-03-31.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597.

[7] André Klein, Jan Warszawski, Jens Hillengaß, and Klaus H. Maier-Hein. "Automatic bone segmentation in whole-body CT images". In: *International Journal of Computer Assisted Radiology and Surgery* 14 (2019), pp. 21–29. DOI: 10.1007/s11548-018-1883-7.

[8] Fabian Isensee, Paul F. Jaeger, Simon A. A. Kohl, Jens Petersen, and Klaus H. Maier-Hein. "nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation". In: *Nature Methods* 18.2 (2020), 203–211. ISSN: 1548-7105. DOI: 10.1038/s41592-020-01008-z. URL: http://dx.doi.org/10.1038/s41592-020-01008-z.

[9] *Scoliosis*. American Association of Neurological Surgeons. URL: https://www.aans.org/Patients/Neurosurgical-Conditions-and-Treatments/Scoliosis. accessed: 2021-04-27.

[10] *Scoliosis in Children and Teens*. National Institute of Arthritis, Musculoskeletal, and Skin Diseases. URL: https://www.niams.nih.gov/health-topics/scoliosis. accessed: 2021-03-31.

[11] Anders Krogh. "What are artificial neural networks?" In: *Nat Biotechnol* 26 (2008), pp. 195–197. DOI: https://doi.org/10.1038/nbt1386.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 6 - Deep Feedforward Networks.

[13] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations". In: *Deep learning in medical image analysis and multimodal learning for clinical decision support*. Springer, 2017, pp. 240–248.

[14] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation". In: *2016 Fourth International Conference on 3D Vision (3DV)*. 2016, pp. 565–571. DOI: 10.1109/3DV.2016.79.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 5 - Machine Learning Basics.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 4 - Numerical Computation.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 8 - Optimization for Training Deep Models.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 7 - Regularization for Deep Learning.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 9 - Convolutional Neural Networks.

[20] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997. Chap. 13 - Continuous Signal Processing / Convolution. URL: http://www.dspguide.com/ch13/2.htm.

[21] Keiron O'Shea and Ryan Nash. "An Introduction to Convolutional Neural Networks". In: *CoRR* abs/1511.08458 (2015). arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458.

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1411.4038 (2014). arXiv: 1411.4038. URL: http://arxiv.org/abs/1411.4038.

[23] Shunjiro Noguchi, Mizuho Nishio, Masahiro Yakami, Keita Nakagomi, and Kaori Togashi. "Bone segmentation on whole-body CT using convolutional neural network with novel data augmentation techniques". In: *Computers in biology and medicine* 121 (2020), p. 103767.

[24] Connor Shorten and Taghi M Khoshgoftaar. "A survey on image data augmentation for deep learning". In: *Journal of Big Data* 6.1 (2019), pp. 1–48.

[25] Gabriela Csurka (Xerox Research Centre Europe), Diane Larlus, and Florent Perronnin (Xerox (XRCE) Grenoble). "What is a good evaluation measure for semantic segmentation?" In: *Proceedings of the British Machine Vision Conference*. BMVA Press, 2013.

[26] Amalia Luque, Alejandro Carrasco, Alejandro Martín, and Ana de las Heras. "The impact of class imbalance in classification performance metrics based on the binary confusion matrix". In: *Pattern Recognition* 91 (2019), pp. 216–231.

[27] Lee R. Dice. "Measures of the Amount of Ecologic Association Between Species". In: *Ecology* 26.3 (1945), pp. 297–302. ISSN: 00129658, 19399170. URL: http://www.jstor.org/stable/1932409.

[28] Peter Tuominen and Björn Relefors. *Datortomografi*. URL: https://www.1177.se/behandling--hjalpmedel/undersokningar-och-provtagning/bildundersokningar-och-rontgen/datortomografi/. accessed: 2021-04-12.

[29] Wilhelm Conrad Röntgen. "On a new kind of rays". In: *Science* 3.59 (1896), pp. 227–231.

[30] John D. Enderle and Joseph D. Bronzino. *Introduction to Biomedical Engineering, Third Edition*. Academic Press, 2012. Chap. 15 - Radiation Imaging. ISBN: 978-0-12-374979-6.

[31] Eva Selin Lindgren. *Nationalencyklopedin, Röntgenstrålning*. URL: https://www.ne.se/uppslagsverk/encyklopedi/lng/rntgenstrlning. accessed: 2021-04-12.

[32] Mats Wager. *Nationalencyklopedin, Röntgenrör*. URL: https://www.ne.se/uppslagsverk/encyklopedi/lng/rntgenrr. accessed: 2021-04-12.

[33] Tami D. DenOtter and Johanna Schubert. "Hounsfield Unit". In: *StatPearls [internet]* (2021). URL: https://www.ncbi.nlm.nih.gov/books/NBK547721/. accessed: 2021-04-15.

[34] Anil Kalra. *Basic Finite Element Method as Applied to Injury Biomechanics*. Academic Press, 2018. Chap. 9 - Developing FE Human Models From Medical Images, pp. 390–391. ISBN: 978-0-12-809831-8.

[35] Tim D. White, Michael T. Black, and Pieter A. Folkens. "Chapter 2 - Anatomical Terminology". In: *Human Osteology (Third Edition)*. Ed. by Tim D. White, Michael T. Black, and Pieter A. Folkens. Third Edition. San Diego: Academic Press, 2012, pp. 11–24. ISBN: 978-0-12-374134-9. DOI: https://doi.org/10.1016/B978-0-12-374134-9.50002-7. URL: https://www.sciencedirect.com/science/article/pii/B9780123741349500027.

[36] *MATLAB version 9.6.0.1472908 (R2019a)*. The Mathworks, Inc. Natick, Massachusetts, 2019.

[37] *The Cancer Imaging Archive (TCIA)*. 2021. URL: https://www.cancerimagingarchive.net/. Funded in part by: Frederick Nat. Lab for Cancer Research.

[38] Ernesto Martinez and Ewa Waern. *Benskörhet - osteoporos*. URL: https://www.1177.se/Skane/sjukdomar--besvar/skelett-leder-och-muskler/benskorhet/benskorhet---osteoporos/. accessed: 2021-05-06.

[39] Gunnar Blom. *Statistik med tillämpningar*. Vol. 2. Lund: Studentlitteratur, 1984. Chap. 20 - Intervallskattning. ISBN: 91-44-05592-7.

# Appendix A

# GUI

The layout of the implemented Graphical User Interface is shown in figure A.1. In the GUI, images from all three directions are shown. The users can decide if they want to see one image slice at a time or if they rather want the maximum intensity projection of all slices. When opening the GUI, a tracking is performed to find the largest object in the images. The rectangular area for segmentation is initially placed as a bounding-box around this object. However, the user can move the area as well as change its size, to fit the bones wanted for segmentation. The user can also choose which segmentation algorithm from the drop-down menu to use for the segmentation. To start the segmentation, the "OK" button is pressed, and a waitbar pops up, that displays the progress. When pressing the "Reset" button the segmentation area is reset to the initial location and size. Finally, the "Close" button shuts down the GUI. When a segmentation is finished, the result is transferred to *Segment 3DPrint*.

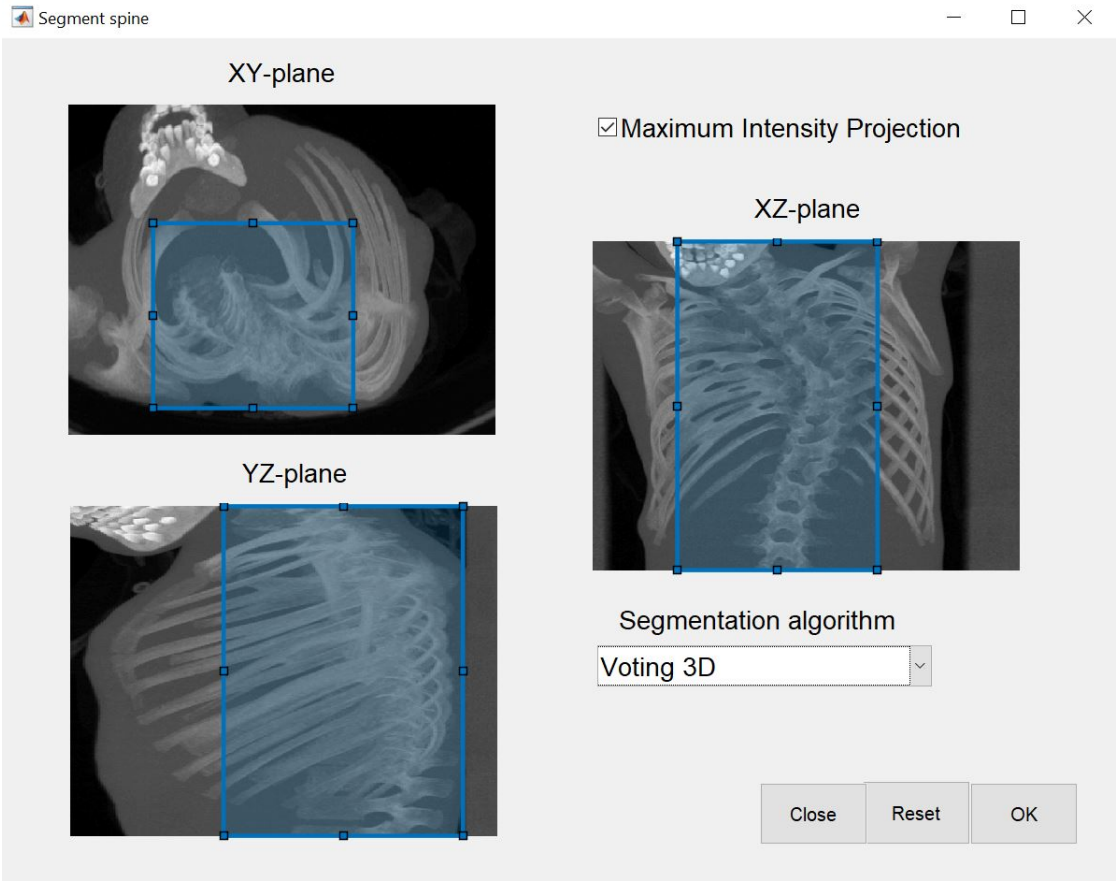



**Figure A.1:** Image showing the layout of the Graphical User Interface.

Master's Theses in Mathematical Sciences 2021:E54
ISSN 1404-6342

LUTFMA-3456-2021

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/