Master of Science Thesis in Computer Science
Department of Electrical Engineering, Linköping University, 2019

Bone Fragment Segmentation Using Deep Interactive Object Selection

Martin Estgren

LINKÖPING
UNIVERSITY

Master of Science Thesis in Computer Science

Bone Fragment Segmentation Using Deep Interactive Object Selection

Martin Estgren

LiTH-ISY-EX–19/5197–SE

Supervisor: **Karl Holmquist**
ISY, Linköpings Universitet
Jonas Hellsten
Sectra AB

Examiner: **Maria Magnusson**
ISY, Linköpings Universitet

Computer Vision Laboratory
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

In recent years semantic segmentation models utilizing Convolutional Neural Networks (CNN) have seen significant success for multiple different segmentation problems. Models such as U-Net have produced promising results within the medical field for both regular 2D and volumetric imaging, rivalling some of the best classical segmentation methods.

In this thesis we examined the possibility of using a convolutional neural network-based model to perform segmentation of discrete bone fragments in CT-volumes with segmentation-hints provided by a user. We additionally examined different classical segmentation methods used in a post-processing refinement stage and their effect on the segmentation quality. We compared the performance of our model to similar approaches and provided insight into how the interactive aspect of the model affected the quality of the result.

We found that the combined approach of interactive segmentation and deep learning produced results on par with some of the best methods presented, provided there were adequate amount of annotated training data. We additionally found that the number of segmentation hints provided to the model by the user significantly affected the quality of the result, with convergence of the result around 8 provided hints.

Acknowledgments

I would like to thank my supervisor Karl Holmquist and my examiner Maria Magnusson for their help and tireless support during this thesis. I would like to thank Mattias Bergbom, Jonas Hellsten, and the rest of the team at Sectra Orthopaedic Solutions for this thesis opportunity, and for their assistance during the course of the thesis. I would also like to acknowledge Dr. Jörg Schilcher for his advice during the initial thesis formulation, as well as for his and Region Östergötland's help in procuring relevant medical data.

Linköping, May 2019
Martin Estgren

Contents

1 Introduction

Many problems within the field of medical imaging involves segmenting specific parts of an anatomy from each other. Typical examples involves problems such as segmenting cancerous tissue from surrounding healthy tissue, and separation of distinct organs for diagnosis and treatment planning. Many of these problems are either done manually by professional radiologists, or with classical image processing algorithms such as: *level-set*, *watershed*, or *clustering*.

Accurate segmentation of bones in CT-volumes are of significant importance for orthopaedic medical professionals, to help with diagnosis and surgical planning. During the last decades, multiple potential solutions have been presented. Many of them suffers from performance and reliability problems, related to advanced pathologies and the image-quality of the CT-scans. As it is desired to limit the dose of ionizing radiation used during clinical CT-scans, the image-quality is reduced, resulting in scans with lower signal-to-noise ratio.

1.1 Background

Sectra Orthopedic Solutions develops and markets a software suite with a segmentation tool based on *C. Wang and O. Smedby* [26] which provides guides and visual aids for orthopaedic professionals. This software is appreciated within the clinical orthopaedic community but the current segmentation algorithm suffers from the typical problem described above. As a result, there is an interest in examining alternative solutions.

Some interactive segmentation methods such as the *probabilistic watershed transform* [25] and *fuzzy connectedness* [26] have both seen some success in clinical settings but both suffers to some extent from the above mentioned problems since they operate in the *gray-level* image space without anatomical or shape context, often resulting in segmentations with varying quality depending on the bone-tissue density and signal quality.

Meanwhile exploration into deep learning based methods, specifically involving *Convolutional Neural Networks* (CNN), have seen novel approaches reaching cutting-edge results in medical imagining competitions and benchmarks [2], [20]. As a result, both the medical academic and industrial communities have shown significant interest in the topic of deep learning for medical imagining during the last couple of years [5].

Additionally, traditional interactive graph-cut based methods such as *Y. Y. Boykov and M-P Jolly 2001* [4] have seen promising results for segmentation problems when combined with deep learning-based methods. One example is *N. Xu et al. 2016* [27] who utilize a CNN model to produce a rough segmentation of a given object, followed by a set of user-provided hints and a graph-cut based algortihm for edge-refinement. The refinement is done since typical semantic segmentation models often produces uncertainty around the edges of the objects [15].

Evaluation of segmentation tasks are often done by comparing evaluation metrics oriented around the confusion matrix of a binary classifier, such as *Intersection over Union* and *Dice score*. These metrics serves as the primary metrics for which we examine the performance of our solution and are explained in greater detail in Section 3.8.

1.2 Aim

The aim of this thesis is to examine the potential of combining user-interactions with deep learning based segmentation methods, for the purpose of segmenting bone fragments from surrounding soft- and bone-tissue. Ideally, the resulting method should be able to serve as a robust and reliable segmentation model, which can be used for clinical purposes, with minimal impact on the current user-interface.

1.3 Research Questions

The following questions will serve as guides to structure the different parts of this thesis:

1. How well does the model perform segmentation of bone-tissue in regards to *Intersection over Union*?
2. How well does the model perform segmentation of bone-tissue in regards to *Dice Score*?
3. Does the number of user-provided hints affect the segmentation performance and in what way?

1.4 Delimitations

To keep the project within a reasonable scope some delimitations have been set:

- Only user interactive deep learning based methods will be examined.
- Training and evaluation will only be done using CT-scans.
- Limitation on the size of the dataset to what can be reasonable be procured during the project.
- We assume that the user-provided hints always are correct.
- Models, training, and evaluation is limited to the hardware available, for this project a NVIDIA GTX 1080Ti.

2 Related Work

This chapter provides technical background to this thesis and describes the different components that will constitute the method, as well as some alternative approaches. The first section describes classical interactive segmentation using a *graph-cut* method (section 2.1), followed by an explanation of how typical CNN-models are built (section 2.2), with some example implementations relating to semantic segmentation (section 2.3). The chapter concludes with an explanation of a combined model utilizing both *graph-cut* and a CNN which serves as ground for the method (section 2.4).

2.1 Graph-cut segmentation

Y. Y. Boykov and M-P Jolly 2001 [4] present a method for interactive region segmentation where the user manually marks a set of pixels as belonging to the correct segment, which acts as as hard-constraints for the remaining segmentation. The remaining pixels are segmented by optimizing the function

$$E(A) = \lambda \cdot R(A) + B(A), \tag{2.1}$$

where A denotes a potential class-assignment of all pixels, λ is the relative importance between boundary $B(A)$ and region term $R(A)$. The latter two are defined as

$$R(A) = \sum_{p \in \mathcal{P}} R_p(A_p), \tag{2.2a}$$

$$B(A) = \sum_{\{p,q\} \in \mathcal{N}} B_{\{p,q\}} \cdot \delta(A_p, A_q), \tag{2.2b}$$

with $\mathcal{P}$ denoting the set of all pixels, $\mathcal{N}$ the set of all edges, R_p the region term for pixel p, $B_{\{p,q\}}$ the boundary term for a pixel p and a neighbouring pixel q. The function

$$\delta(A_p, A_q) = \begin{cases} 1 & \text{if } A_p \neq A_q \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

defines a discontinuity measure where neighbouring pixels, which have the same assignment, do not require the boundary term to be computed.

The region term $R_p(A_p)$ provides a measure of how well the given pixel p belongs to the foreground and background segments. This can be done in a multitude of ways, *Y. Y. Boykov and M-P Jolly 2001* [4] give the example of a measure of how well a given pixel intensity fits into a known intensity model. One example is

$$R_p(A_p) = \begin{cases} -\ln P(I_p|\mathcal{O}), & \text{if } A_p = \text{"obj"} \\ -\ln P(I_p|\mathcal{B}), & \text{if } A_p = \text{"bg"} \end{cases}. \tag{2.4}$$

That is, the *negative log-likelihood* for p belonging to the *"obj"* set or *"bg"*, where I_p indicate pixel intensity, $\mathcal{O}$ representing all pixels manually assigned as *"obj"*, and $\mathcal{B}$ all manually assigned as *"bg"*.

The boundary term $B_{\{p,q\}}$ describes the relative difference between the neighbouring pixels p and q to create a boundary between the segments, where there is a large discontinuity in the pixel intensity. *E. N. Mortensen and W. A. Barret 1998* [21] provide a number of examples of such functions, for example

$$B_{\{p,q\}} \propto \exp\left(-\frac{(I_p - I_q)^2}{2\sigma^2}\right) \cdot \frac{1}{\text{dist}(p,q)}, \tag{2.5}$$

where σ denotes some form of sensor noise and $\text{dist}(p,q)$ corresponding to a distance metrics, such as euclidean distance.

The *E. N. Mortensen and W. A. Barret 1998* [21] names the *Ford-Fulkerson* graph-cut algorithm presented by *L. Ford and D. Fulkerson 1962* [9] as the most straightforward way to find the optimal segmentation A. The segmentation problem is concretized through a graph-representation of the image, where each pixel denotes a node and where each node has a directed link to all neighbouring nodes. Additionally, each node has an incoming edge from a node denoted as *source* (S) and an outgoing edge connected to a node denoted as *sink* (T). Note that these two nodes are not part of the image but represents the two sets each pixel may be assigned. Figure 2.1 shows a small example how a typical edge-cut is performed.

The *Ford-Fulkerson* algorithm finds the optimal set of edges to cut in order to separate the *source* and *sink* nodes from each other. This is achieved through the use of the *max-flow min-cut theorem*, which states that when the graph is considered as a flow-network, the maximal flow is equal to the capacity of the minimal cut. All edges in the problem are defined in Table 2.1.

The first type of edge $\{p,q\}$ indicate the weight for edges between neighbouring

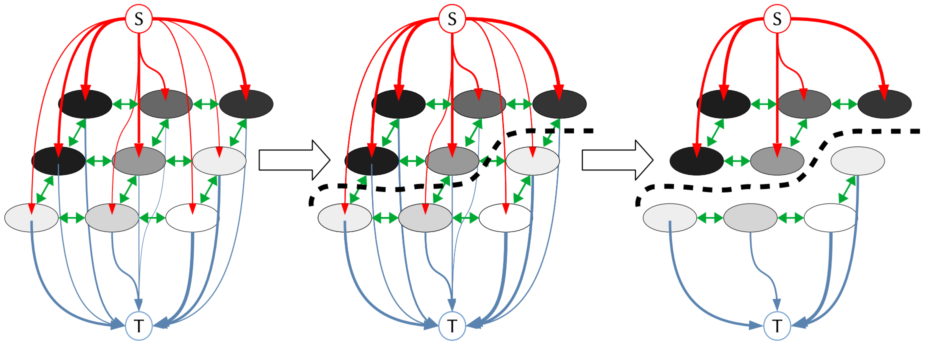


Figure 2.1: *Visual example of how a typical graph-cut segmentation of an image is done. The black line indicate the cut between the object and background. Blue and red edges gets their weights from the* region term *(2.4), while green edges gets their weights from the* boundrary term *(2.5).*

edge	weight	for
$\{p,q\}$	$B_{\{p,q\}}$	$\{p,q\} \in \mathcal{N}$
$\{p,S\}$	$\lambda \cdot R_p(\text{"bkg"})$	$p \notin \mathcal{O} \cup \mathcal{B}$
	K	$p \in \mathcal{O}$
	0	$p \in \mathcal{B}$
$\{p,T\}$	$\lambda \cdot R_p(\text{"obj"})$	$p \notin \mathcal{O} \cup \mathcal{B}$
	0	$p \in \mathcal{O}$
	K	$p \in \mathcal{B}$

Table 2.1: *Table of how edge-weights are calculated according to* Y. Y. Boykov and M-P Jolly 2001 *[4]. The table shows how the edge-weights should be calculated depending on the type of edge.*

pixel nodes. These are always taken from the *boundary term* (2.5). Edges defined as $\{p,S\}$ indicate the weight for edges between pixel nodes and the *source* S. These are defined differently depending on the whether the pixel node p is set by the user as a marker or not. If p is not a marker, the weight is calculated from the *region term* (2.5). The final type are edges between pixel nodes and the *sink* T where, as with edges between $\{p,S\}$, the weights are dependent on whether the user has marked them as either *object* or *background*. The special weight K is defined as

$$K = 1 + \max_{p \in \mathcal{P}} \sum_{\{p,q\} \in \mathcal{N}} B_{\{p,q\}}, \tag{2.6}$$

which is used to prevent an edge between a marked pixel and its corresponding segment to be cut. The cut is computed by finding the path with the maximum sum over the edges in the path.

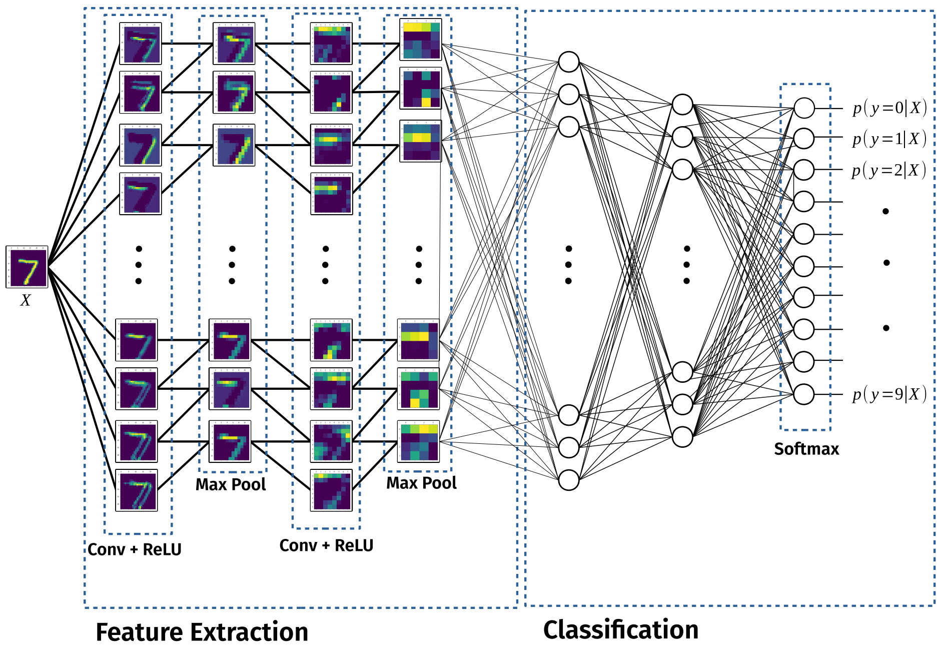


Figure 2.2: Example of a Convolutional Neural Network *(CNN) designed to perform digit recognition. The input is a* 28×28 *gray-scale image of a* 7*, the output is a vector of* 10 *elements where each element denotes the conditional probability of the input representing either of the numbers* $\{0, 1, 2, ..., 9\}$ *depending on the specific input image.*

2.2 Convolutional Neural Network

Convolutional Neural Networks (CNN) are a family of *deep learning* classification/regression models which employ *convolution* operations in some of its layers. As described in *I. Goodfellow et al. 2016*[10] this makes them significantly more effective on data where the spatial relationship between data points are important, such as time-series, image data, or volumetric data. Figure 2.2 shows a structural overview of a typical CNN which perform digit classification. The model consist of two *convolutional* layers with a *max-pool* layer after each other, followed by a *multi-layered perceptron* with a *softmax* output. The dataset used is the *MINST* [18] dataset.

The typical classification model is split into two parts. The first part, *Feature Extraction*, consists of a set of layers which transforms the input data into a higher dimensional feature space. One example is a transformation from pixel values into a feature space defined as a set of differently oriented edge-detections. The feature extraction is often implemented using *convolutional* and *pooling* layers, which are described in Section 2.2.1 and 2.2.4, respectively. The second part, *classification*, is often a regular *Multilayer Perceptron* (MLP) which is fed the new feature space, and produces the final classification.

The efficiency of CNNs comes from the use of shared weights in the convolutional layers, which drastically reduces the size of the model. This property helps with keeping the number of weights significantly smaller than a comparable MLP model, resulting in a more efficient model-fitting.

2.2.1 Convolutional Layers

Convolutional layers are the cornerstone and most prominent feature of a CNN. In this section, the structure of a convolutional layer , the *convolutional* operator for 1, 2, and 3 dimensions, and how the output is transformed into a higher dimensional feature-space, using a non-linear activation function, will be defined.

Convolution

In mathematics, the discrete version of the *convolution operator* is defined as the combination of two functions (f and g) on the form

$$(f * g)(t) = \sum_{x=-\infty}^{\infty} f(t-x)g(x), \tag{2.7}$$

where $f(t-x)$ denotes a weighted average of $f(x)$ at a given *offset* t, such as the value of a time-series at time t. The function g is often denoted *kernel* and the output is often denoted *feature map*. Convolution for two and three dimensions have the respective definitions

$$(f * g)(i,j) = \sum_x \sum_y f(i-x, j-y)g(x,y), \tag{2.8a}$$

$$(f * g)(i,j,k) = \sum_x \sum_y \sum_z f(i-x, j-y, k-z)g(x,y,z). \tag{2.8b}$$

When working with discrete convolutions in 2- and 3D space, specifically for image processing, the function can be interpreted as an operation on matrices. This is done by interpreting the offset t as the element offset in the matrix f where the transpose of the *kernel* g is applied. In practice, the convolution is implemented as a cross-correlation, i.e. the kernel is not transposed. How convolutions can be performed as matrix multiplication is described in Appendix B.

Border elements in f requires a policy about how to process sums of elements outside f, typical solutions involves *zero-padding*, *reflection*, or *border repetition*. For this thesis, it is enough to consider *zero-padding*, where the border of the input is padded with elements of value 0. The coefficients which represents the *kernel* are the values which are tuned in a CNN during the training phase. In addition to the kernel g, each convolutional node contains a bias-weight which acts as an offset on the feature map to shift the curve of the activation function either left or right.

2.2.2 Strided Convolution

Strided convolution is defined as

$$(f *_l g)(t) = \sum_x f(lt-x)g(x), \tag{2.9}$$

where l indicate the *stride* i.e. the size of the offset. When $l = 1$ the *strided convolution* is equivalent to a regular *convolution*. The output feature map is smaller than the input by a factor which is the same as the number of strides. As an example: an input of dimension 16×16 convolved with a kernel and a stride of 2 would result in an output feature map of size 8×8.

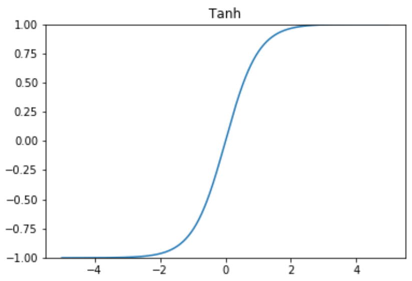
The term *dilated convolution* is sometimes used and differs from the strided convolution in that t is applied to x instead of t. If padding is added between elements in the input image, *fractional strided convolution* can be performed, resulting in a feature-map larger than the size of the input image.

2.2.3 Activation Function

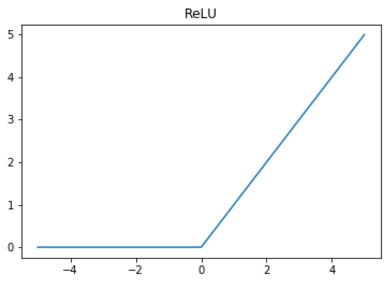
The activation function serves as a way to introduce non-linearity into the network. In the typical convolutional layer, the *activation function* is applied to the output feature map to provide a non-linear transformation to a higher dimensional feature space. The full function with both convolution and activation is

$$\sigma((f * g)(t)) = \sigma(\sum_{x=-\infty}^{\infty} f(t-x)g(x)), \tag{2.10}$$

where σ denotes the activation function. Traditionally a *sigmoid* function such as the *hyperbolic tangent* is used as activation function (Figure 2.3a). Two of the primary drawbacks when using a *hyperbolic tangent* is the computational cost and the *vanishing gradient* problem [3], in which, the tuning of a given weight is proportional to the partial derivative of said weight in the error function, resulting in weights of the first layers in a model, always being tuned less proportional to the weights in the final layers. *K. Jarrett et al. 2009* [14] show that the *rectified linear unit*(Figure 2.3b), abbreviated as *ReLU*, often preforms better for CNN models. Therefore the *sigmoid* is in many cases replaced with a *ReLU*, which has more desirable properties, both in therms of the *vanishing gradient* problem and computational complexity.



***(a)** Hyperbolic Tangent*



***(b)** Rectified Linear Unit*

***Figure 2.3:** Example of the two typical* activation functions *used for CNN.*

2.2.4 Pooling Layers

Pooling layers perform spatially constrained statistical summaries on the input feature map, reducing the output resolution, while at the same time preserving relevant features from the previous layer. *I. Goodfellow et al. 2016* [10] describe, in Section 9.3, how the typical pooling layer is constructed, and how it often is imposed between consecutive convolutional layers. This reduces the amount of parameters in a given model, and consequently the risk for overfitting.

The typical pooling layer have two parameters

1. Kernel size: Indicates the size of the kernel used during the pooling operation.
2. Stride: Dictates the number of elements the kernel jumps during computation.

The *kernel size* additionally provides the size of the bins for which the maximum element is picked from.

In the typical *max-pooling* (*Y. Zhou and R. Chellappa 1988* [30]) layer, only the largest value for each kernel placement is saved, resulting in a down-sampling of the input feature-map by the value of the *stride* parameter. For example, stride = 2 means the output feature-map will be half the size of the input. Figure 2.4 provides an example of how a typical max-pooling is performed.

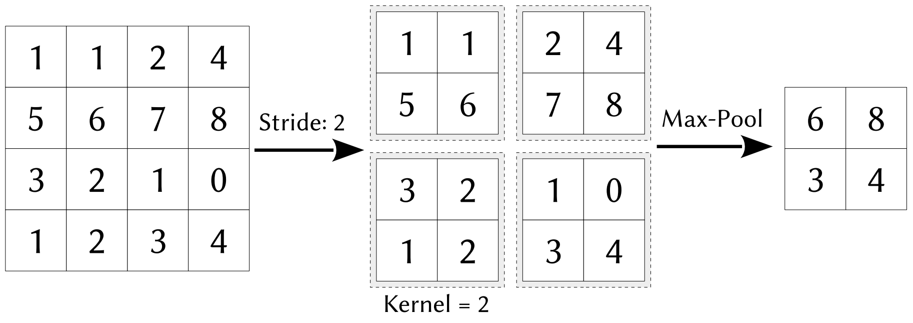


Figure 2.4: *Example of a max-pooling with kernel size* = 2 *and stride* = 2.

2.2.5 Loss Function and Learning Process

The loss function is used to measure how well the model predicted the output. It also serves as the base for the learning process, where the weights of the model are tuned to produce a lower loss score. The learning process is done through *back-propagation* using *gradient descent*, where a given weight in the model is tuned based on the partial derivative

$$\frac{\partial E}{\partial W}, \tag{2.11}$$

on the form $\Delta W = -N\frac{\partial E}{\partial W}$, where W is a given weight, N is a *learning rate*, and E is the *loss function*. For an in-depth explanation and a full example, see *I. Goodfellow et al. 2016* [10] Chapter 6.5.

When producing the final predicted class assignments for binary segmentation tasks, the *sigmoid* function

$$p(y = \text{'obj'}) = \text{sigmoid}(x; W) = \frac{1}{1 + e^{-W^T x}}, \tag{2.12}$$

where W are the weights for the output neuron, is often used.

When the prediction is done for a multi-class problem, i.e. the desired output is a vector of probabilities, the *softmax* function

$$p(y = n) = \text{softmax}(x; W) = \frac{e^{W_n^T x}}{\sum_{k=1}^{K} e^{W_k^T x}}, \tag{2.13}$$

where n dictate the probability for the *nth* class, is used instead. In the case of Figure 2.2, there would be 10 classes.

Cross-entropy is often used as loss function for the training process. This is done as it provides an approximative analogue for evaluation metrics such as *Dice score* or *Intersection over Union*, while being computationally simpler.

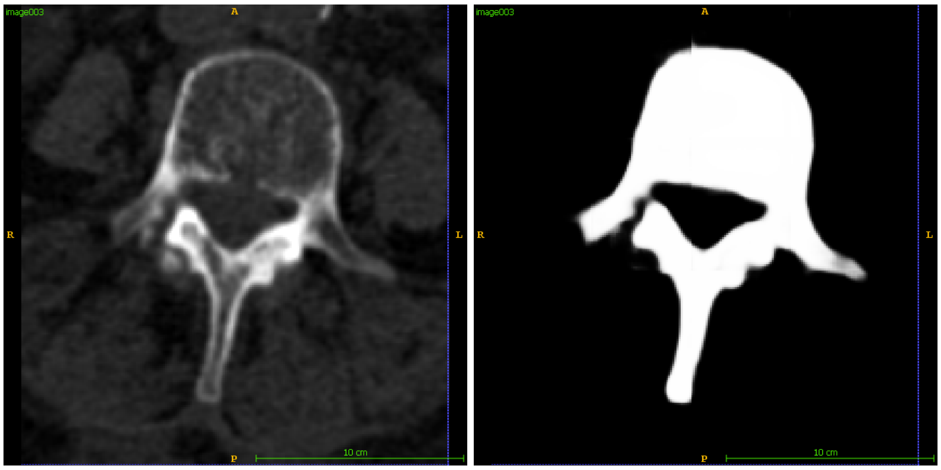
The accumulated cross-entropy is calculated for each input image as

$$-\frac{1}{N}\sum_{n=1}^{N} y_n log\hat{y}_n + (1 - y_n)log(1 - \hat{y}_n), \tag{2.14}$$

where $\hat{y}$ denotes the predicted class, N the length of the output vector, and y the annotated ground truth.

2.3 Segmentation using Deep Learning

In this section, segmentation with CNN models is described, by looking at three different network architectures. In a typical classification model an image is remapped to a single value describing the class assignment for the input. Segmentation models, on the other hand, try to to get a class assignment for each pixel in the input, not only indicating the class assignment for the entire image. In most cases, this is done by replacing the *classification* part of the model with additional convolutional layers. In general, this can be seen as the model outputs a higher-dimensional feature map, where each feature represents a probability for a given class-assignment for the corresponding input features. An example of how such a feature map looks like can be seen in Figure. 2.5, which shows an image of a lumbar vertebrae with the feature-map indicating the predicted class assignment.



(a) Image Slice *(b) Class Prediction*

Figure 2.5: *Example of an image slice with a corresponding* class prediction, *providing the predicted class for each element.*

2.3.1 Fully Convolutional Network

The *Fully Convolutional Network* (FCN) is one of first network architectures developed specifically for *semantic segmentation* tasks. It was presented by *J Long et al. 2015* [19] as a CNN model where the entire network architecture consists of convolutional and pooling layers. Then the network produces a feature map indicating probabilities for each pixels class assignments, but at a reduced resolution compared to the input. *J Long et al. 2015* [19] propose a way to remap the output to the input resolution involving interpolating the lower resolution feature map by either applying *fractionally strided convolutions* or performing up-sampling with e.g. *bilinear interpolation*. Figure 2.6 shows the stacking of layers in the model and how feature-maps from the down-sampling and up-sampling parts are combined in order to provide the final segmentation.

Given the nature of stacking pooling and convolutional layers, spatial locality is lost as the number of layers increase. To combat this, FCN combines outputs from shallow layers with up-sampled deeper feature maps to provide outputs of higher resolution. That is, given the segmentation problem at hand, the user may trade segmentation precision with the size of the model.

2.3.2 U-Net

U-Net was first presented by *O. Ronneberger et al. 2015* [23], and then extended into 3D by *Ö. Çiçek et al. 2016* [11]. U-Net was developed specifically with segmentation of medical images in mind, where the images consist of highly regular structures. The original paper by *O. Ronneberger et al. 2015* [23] used the model to segment cells in *neuronal structures* [2], placing first in the benchmarks.

The architecture of the model was initially based on the FCN model, but the inclusion of skip-connectors and a large amount of feature-channels in the up-

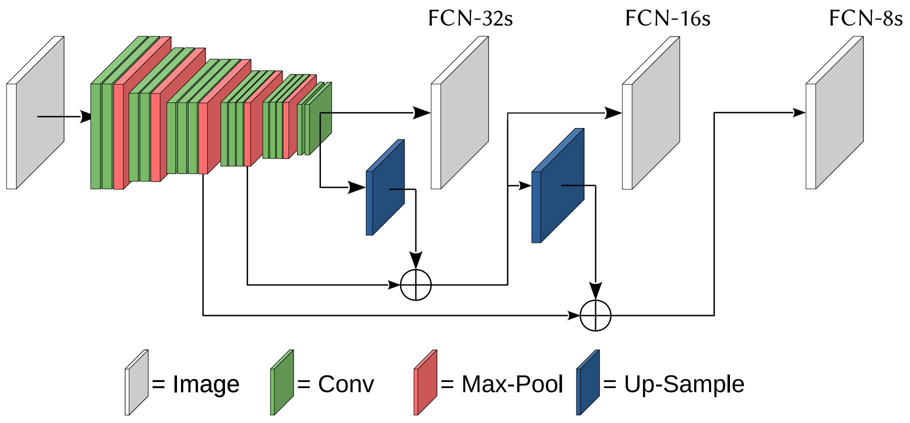


Figure 2.6: *Reference architecture for the typical FCN model. Note that three different outputs are generated where (*FCN-32s, 16s, and 8s*) where the number indicate the amount of up-sampling needed to match the input size.*

sample part, allows for information propagating from the different down-sample levels. In contrast to the regular encoder-decoder model where the up-sampling have to be done using only the bottleneck feature-map. Figure 2.7 provides a visual representation of the regular U-Net architecture. Note how smaller feature maps are concatenated in the up-sample stage, compared to the summation used in FCN.

O. Ronneberger et al. 2015 present a weighted cross-entropy loss function designed to punish erroneous segmentations in the border between cells. In addition to the modification of the architecture and the custom loss function, the paper also presents a set of model regularizing *data augmentation* methods which

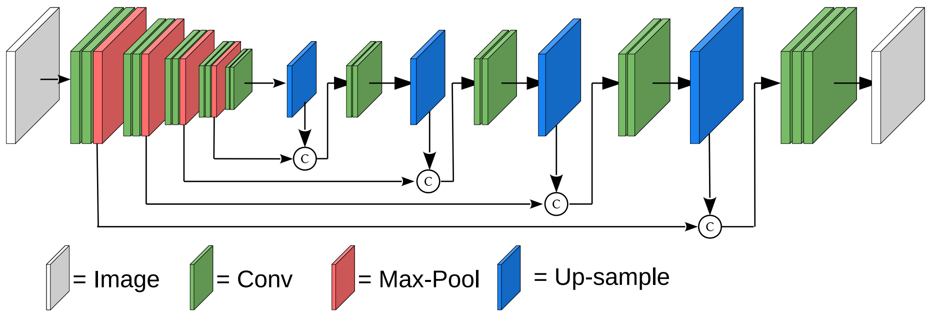


Figure 2.7: *Reference architecture for U-Net.* C *indicate channel-wise concatenation of feature maps.*

are applied to the training dataset. The authors puts heavy focus on the *elastic deformation* augmentation model where a given input sample is slightly distorted to prevent the model from memorizing exactly how the target segmentation should look like.

Ö. Çiçek et al. 2016 [11] present an extension of the U-Net model to 3 spatial dimensions to be used for segmentation of volumetric data, specifically through the use of 3D convolutions, up-sampling, and max-pooling. Additionally, the authors include a *Batch-Normalization* layer [12] before each convolutional layer to speed up training by reducing the internal covariance shift.

2.4 Deep Interactive Object Selection

Interactive object selection involves segmenting foreground object in an image using hints provided by the user. Gray-level segmentations techniques often struggle with such segmentation tasks and require multiple hints from the user to produce acceptable segmentations , especially when the objects have varying lightning, color, and texture. *N. Xu et al. 2016* [27] present a method of combining the classical interactive segmentation method *graph-cut optimization* with a semantic segmentation model such as FCN to produce accurate object segmentation with minimal user-provided hints. The model were tested on the *PASCAL VOC2012* segmentation dataset [8], which consists of 2D images with the goal of segmenting semantic classes such as *aeroplane* or *bottle* from the background.

To provide user hints to the FCN segmentation model, two additional image channels are appended to the input image. These channels consist of *distance maps* which indicate each pixel's euclidean distance to the closest point hint provided by the user. The first channel indicates the distance map to foreground points and the second channel indicates the distance map to background points. An overview of the model can be seen in Figure 2.8 in which an image slice is combined with the user-provided hints and segmented by first using an FCN model with the result refined though a graph-cut method.

N. Xu et al. point out that letting real users provide all hints for the training data is unrealistic, given the large number of samples used. As an alternative, a procedural hit placement method is described, which combines three placement strategies. All three variants give an acceptable model of user behaviour. The strategies are as follows:

1. Place background points in a band around the target object.
2. Place background points randomly on different objects in the image.
3. Place background points in a band around the target object but always maximize the distance for a new point to the ones already placed.

Combined, they result in a point placement strategy which is adequately close to how the typical user places points. Points indicating foreground pixels does not need any special strategy and can be placed randomly within the object bound.

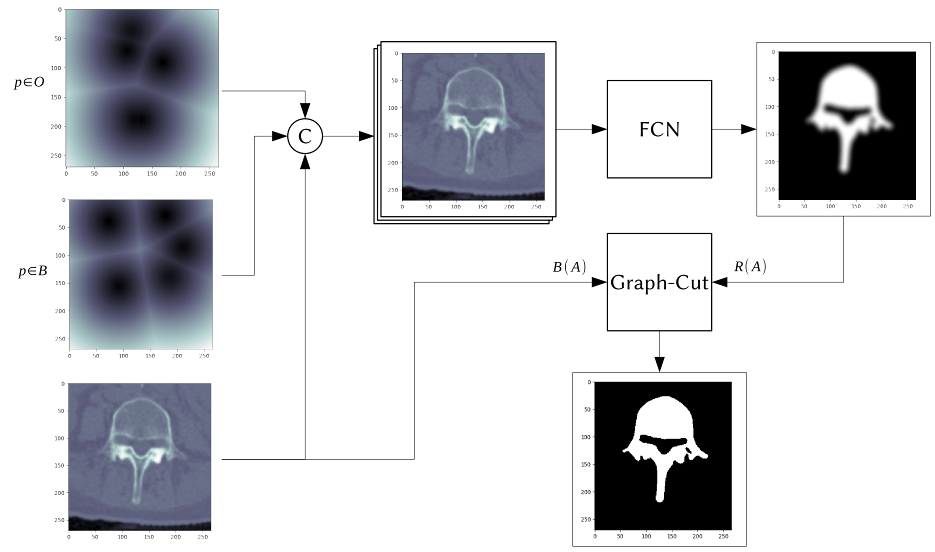


Figure 2.8: *Processing pipeline for Deep Interactive Object Selection. An image slice is segmented using an FCN model with the result refined in a graph-cut algorithm. User-provided hints are added to the model in the form of distance maps concatenated as channels on the input image.*

The predicted object segmentation from the FCN is incorporated into the graph-cut algorithm by replacing the typical *region properties term* (see Section 2.1) $R_p(A_p)$ with the *log probability* probability map q produced by the FCN:

$$R_p(A_p) = \begin{cases} -\log(q_p) & \text{if } A_p = \text{"obj"} \\ -\log(1 - q_p) & \text{otherwise} \end{cases} \tag{2.15}$$

where q denotes the feature-map produced by the segmentation model.

3 Method

This chapter provides an explanation of the model, dataset, and processing done to train and evaluate the model.

3.1 Dataset

As basis for the model training and evaluation, the *lumbar spine* CT dataset from *xVertSeg* [1] was used. The dataset provided 15 CT-volumes with the *lumbar vertebraes* annotated, and 10 CT-volumes without annotations. Each of the annotated CT-volumes contained the 5 lumbar vertebrae (L1 to L5), but consisted of varying resolution and scope. In Table 3.1, the different CT-volumes and their corresponding meta-information are given. Only the scans which had corresponding ground truth annotations were included.

The dataset was partitioned into two parts, one set consisting of 12 scans which were used for model training/fitting and the 3 remaining used for validation/performance evaluation. The partitioning of the dataset is shown in the last two columns of Table 3.1.

Sample	X	Y	Z	Δx	Δy	Δz	Training	Validation
image001	1024	1204	200	0.41362	0.41362	1.4506	x	
image002	1024	1024	250	0.43207	0.43207	1.2895	x	
image003	1024	1024	340	0.54070	0.54070	1.1924		x
image004	1024	1024	170	0.42254	0.42254	1.2837	x	
image005	1024	1024	181	0.49629	0.49629	1.8919	x	
image006	1024	1024	100	0.28860	0.28860	1.6686	x	
image007	1024	1024	180	0.47433	0.47433	1.7593		x
image008	512	512	218	0.80273	0.80273	1.1215	x	
image009	1024	1024	230	0.39381	0.39381	1.1052	x	
image010	1024	1024	200	0.36014	0.36014	1.1755		x
image011	512	512	351	0.62261	0.62261	1.3286	x	
image012	1024	1024	130	0.30626	0.30626	1.7025	x	
image013	1024	1024	110	0.32930	0.32930	1.8129	x	
image014	1024	1024	223	0.54411	0.54411	1.3069	x	
image015	1024	1024	190	0.39449	0.39449	1.1164	x	

Table 3.1: *Meta-information regarding the dataset.* Sample *indicates filename, X, Y, Z indicate the size of the volume, and $\Delta x, \Delta y, \Delta z$ indicate the distance between neighbouring voxels in mm.*

3.2 CNN-Model

The architecture was based on the *3D U-net* model presented by *R. Janssens et al.* [13], whose U-net model slightly differs from the original structure (Figure 2.7), by employing batch-normalization (BN) between each *convolution* and *activation* layer. The U-net model were chosen instead of FCN as it does not suffer from the same up-sampling problem as FCN. It as also shown promising results on many medical datasets. The model used in this thesis additionally have the final *softmax* output layer replaced by a *sigmoid* (eq. 2.12) with one node. This was done since the model only performed binary segmentation, since the inclusion of user-provided hints, provide a way for the model to segment the correct object. This is in contrast to [13], which perform multi-label segmentation for all vertebrae, without the ability to specify which vertebrae to segment. The architectural design of the model can be seen in Figure 3.1.

The method proposed by *R. Janssens et al.* [13] involves three steps. First the immediate region around the lumbar vertebrae is located using a CNN. The CNN outputs the minimum and maximum corners for an axis-aligned bounding-box, which is referred to as the *Region Of Interest* (ROI). Secondly, a U-net model is trained to segment the ROI with all vertebrae assigned the same label, this is referred to as the *pre-training* stage. Finally, the model is trained to segment each individual vertebrae i.e. each vertebrae is assigned an individual label. There are some differences between our method and R. Janssens et al. Primarily, the *interactive segmentation* part differs significantly from their method, in large part due to the inclusion of *distance-maps*. These differences are explained as the segmentation procedure is described in the following sections.

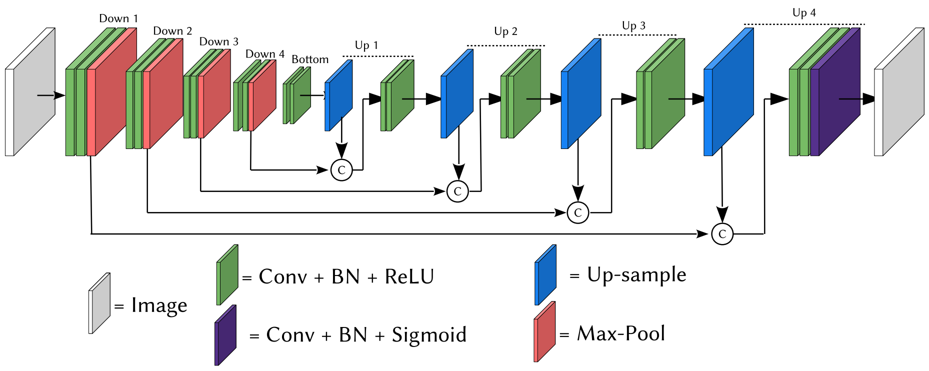


Figure 3.1: *Architectural reference for the segmentation model used.* C *indicate channel-wise concatenation of the feature-maps.*

The full model with corresponding hyperparameters can be found in appendix A.

As with *O. Ronneberger et al. 2015* [23], a weighted version of the *cross-entropy* function was used to calculate prediction loss. This was done since the output was predicted through a single *sigmoid*. Weighting of the function was done since there was a significant class imbalance between the *foreground* and the *background* voxels, applying sample weighting to the loss function reduces the likelihood that the segmentation model converges to outputting a single class for all voxels. Equation (3.1) shows how the typical *cross-entropy* loss function was modified to accommodate for sample weighting. The weighted cross-entropy loss function was implemented based on the *weighted cross-entropy with logits* from TensorFlow [1], adjusted for integration with Keras. The implemented function was defined as

$$-\frac{1}{\sum w}\sum_{n=1}^{N} y_n w_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n), \tag{3.1}$$

where y_n is the nth element in the ground truth matrix, $\hat{y}_n$ is the predicted class assignment, and

$$w = \frac{N(2y+1)}{\sum_{n=1}^{N} 2y_n + 1}, \tag{3.2}$$

where N is the number of elements in y, i.e. the number of voxels in the volume. This function provides the relative weight between the foreground and

[1]https://www.tensorflow.org/api_docs/python/tf/nn/weighted_cross_entropy_with_logits

background class calculated on a sample basis.

The built in *Adam* [16] optimizer was used to tune the model with a default learning rate of 10^{-4}.

3.2.1 Software

The model was implemented using *Keras* v2.2.2 with *Tensorflow* v1.11.0 as backend. House-keeping code and evaluation was done using Python 3.6.6, *Skimage* 0.14.0, *Scipy* 1.1.0, and *Numpy* v1.15.2.

3.3 Overview of the Segmentation Procedure

The segmentation procedure contains two parts, the *pre-training*, and the *interactive segmentation*. Both follow in large part the same procedure, with *interactive segmentation* requiring extra steps in the *pre-processing* and *post-processing* sections. These differences can be seen in Figure 3.2.

The segmentation for a single CT-volume is divided into three distinct steps. First, a volume is loaded and pre-processed for the segmentation model, the output from this step was a set of subvolumes. This step is referred to as the *pre-processing* stage. The second stage, took the list of subvolumes, ran them through the U-net, and recombined them into one volume afterwards. This step is refereed to as the *segmentation* stage. The third and final stage, called the *post-processing* stage, applied different morphological operations, to improve the segmentation. The flow-chart in Figure 3.2 shows the process. Each part of the figure is described in greater detail in the following sections.

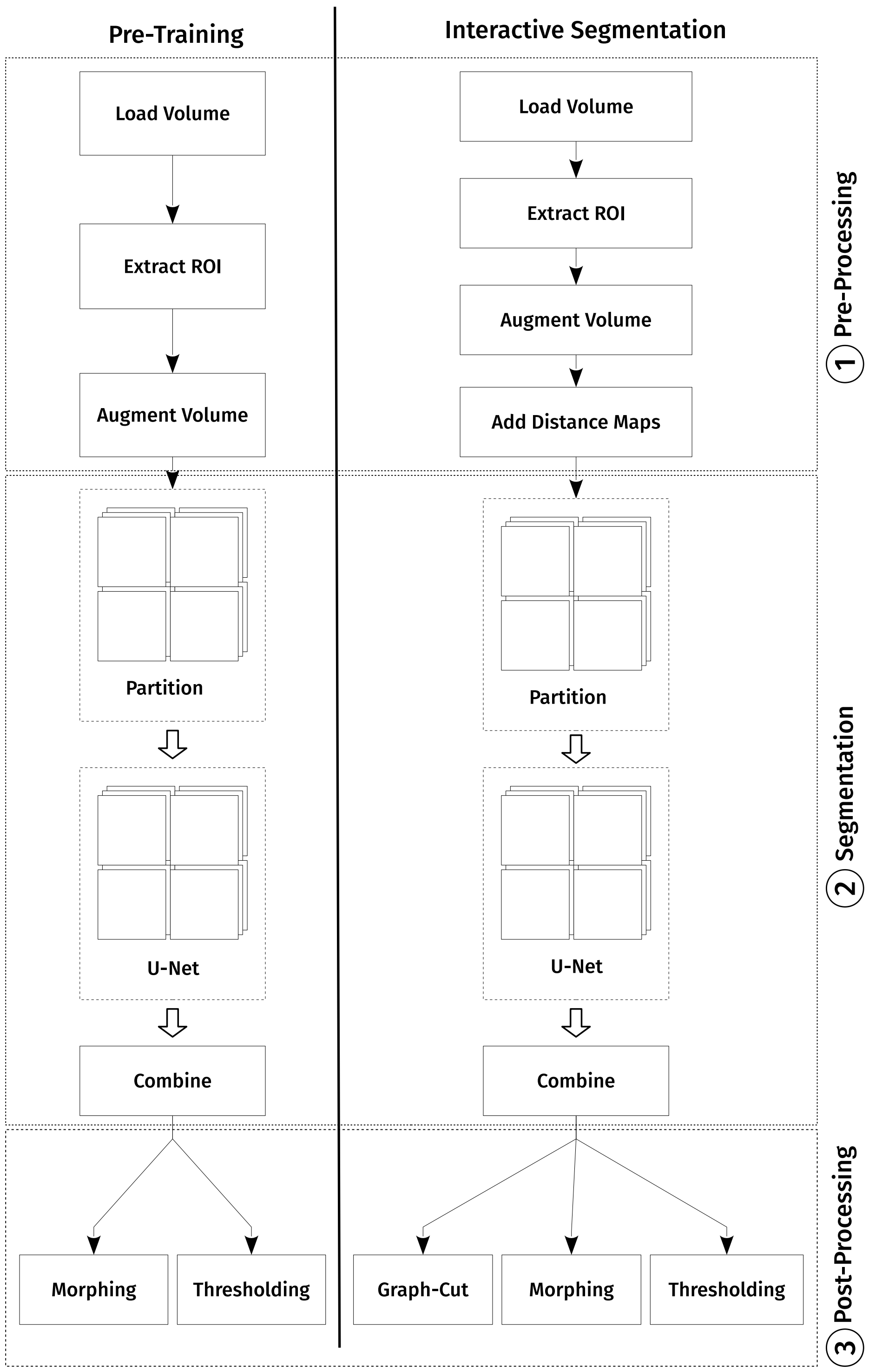


Figure 3.2: *Overview of the complete segmentation procedure for both pre-training and interactive segmentation.*

3.4 Pre-processing

This section explains the steps taken in order to transform a CT-volume into a format which can be fed to the U-net segmentation model.

3.4.1 ROI Extraction

Extraction of a ROI around the target anatomy is done to provide a higher ratio of relevant voxels to segment. If the model were to segment the entire CT-volume, it would both take more time and result in training the model on a severely class imbalanced dataset. Figure 3.3 shows the placement of a ROI indicating the lumbar spine.

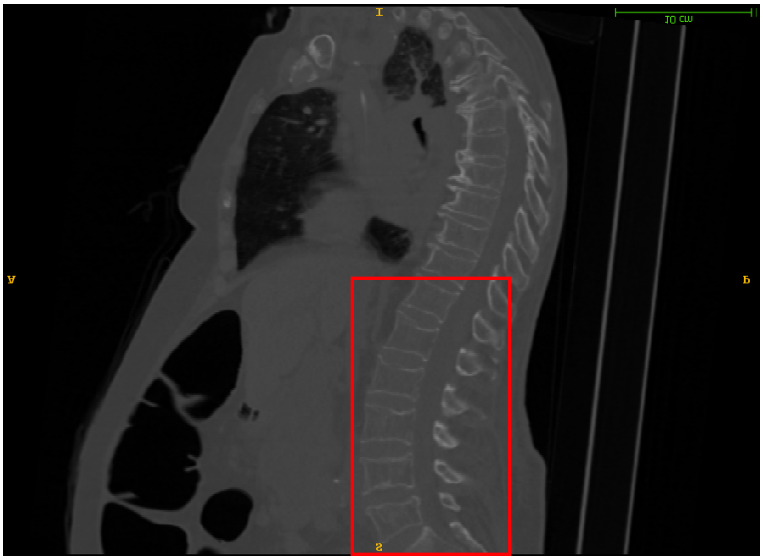


Figure 3.3: *Example of a ROI highlighted in red, notice that it covers the L1-L5 vertebrae but cuts-off the lowest thoracic vertebrae T12.*

For the purpose of this project, the localization of the ROI can be seen as a solved problem, see for example *A. Sekuboyina et al. 2017* [24], where the localization of vertebrae in the dataset was close to perfect with existing methods. As a result, the ROI was extracted by finding the *axis-aligned bounding box* which included all voxels annotated as foreground.

3.4.2 Data Augmentation

Once the CT-volume was cropped to the ROI, *data augmentation* was performed in order to produce a larger sample set and in doing so, acting as a regularization method, to prevent overfitting. To speedup the training process, data augmentation was applied after the volume was cropped to the ROI. For the purpose of this thesis the following three augmentation methods were used: elastic deformation, axial rotation, and ROI translation.

Elastic Deformation: Elastic deformation was used in the initial U-net paper [23] and seemed to contribute to the regularization of the model. In contrast to other implementations, our displacement volumes were created through the use of *Perlin noise*. *Perlin noise* was first presented by *K. Perlin 1985* [22] as a fast way to generate continuous noise. This method of generating noise was used as it drastically speed up the elastic deformation process. To produce the displacement volume, the Python wrapper *pyfastnoisesimd* over the *fastnoisesimd* [2] package was used to generate three noise volumes, one for each of the dimensions. These volumes were combined as a vector field and applied to the CT-volume, producing a smooth deformation of the original voxel data. Figure 3.4 shows how such a displacement field may look like in 2 dimensions.

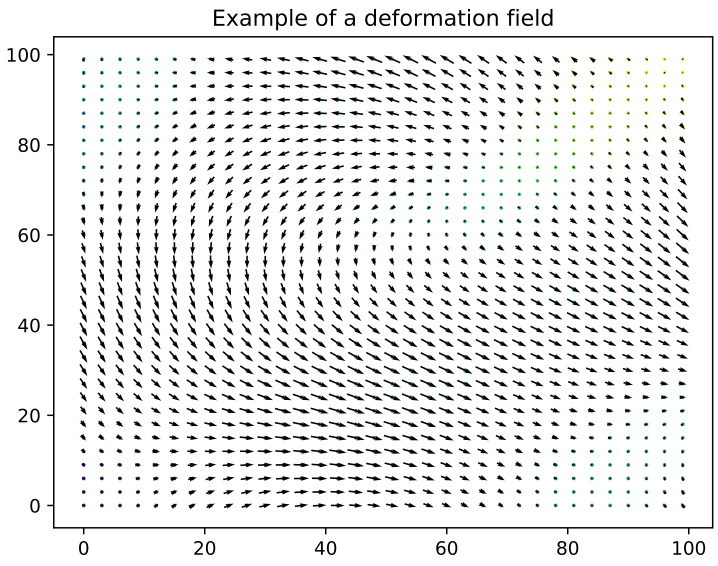


Figure 3.4: *Example of a elastic deformation field in two dimensions. Each arrow indicate a direction which the corresponding voxel should be displaced by.*

Voxel displacement was performed with assistance of *trilinear interpolation*. Each voxel position in the displaced volume D is computed by

$$\hat{x} = x + \vec{d}_x, \tag{3.3a}$$

$$\hat{y} = y + \vec{d}_y, \tag{3.3b}$$

where each resulting point $(\hat{x}, \hat{y})$ gets the intensity value of the interpolated value in the original image I. Interpolation is necessary since voxels may be displaced

[2]`https://github.com/Auburns/FastNoiseSIMD`

into sub-voxel locations. For the sake of simplicity *bilinear interpolation* is detailed below.

First interpolation is performed along the x-axis,

$$I(x, y_1) = \frac{x_2 - x}{x_2 - x_1} I(x_1, y_1) + \frac{x - x_1}{x_2 - x_1} I(x_2, y_1), \tag{3.4a}$$

$$I(x, y_2) = \frac{x_2 - x}{x_2 - x_1} I(x_1, y_2) + \frac{x - x_1}{x_2 - x_1} I(x_2, y_2), \tag{3.4b}$$

where $I(x_i, y_j)$ denotes the intensity value at coordinate (x_i, y_j). Then interpolation is performed along the y-axis by

$$I(x, y) = \frac{y_2 - y}{y_2 - y_1} I(x, y_1) + \frac{y - y_1}{y_2 - y_1} I(x, y_2). \tag{3.5}$$

Figure 3.5 provides a visual aid over the relation between the point (x, y) and its neighbours. As mentioned before, the interpolated intensity $I(x, y)$ will be assigned to the position $(\hat{x}, \hat{y})$ in the displacement volume.

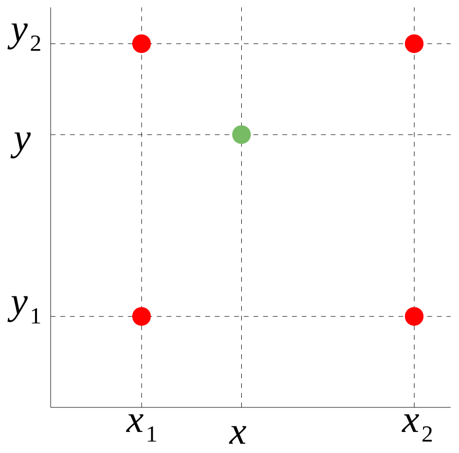


Figure 3.5: *Bilinear interpolation of the point x, y. Green point indicate $I(x, y)$ with the red points indicating the rectangular neighbourhood around (x, y).*

Axial Rotation: Given the structure of the human anatomy we could assume that the body was scanned in slices of the axial plane. Therefore the data was augmented by randomly rotating the volumes within the axial plane. The *rotation matrix* used for performing the rotation is

$$R_{axial} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.6}$$

where θ denotes the angle we want our slices rotated by. If empty voxels were rotated into the volume their values were set to 0. The rotation is done by

$$\begin{pmatrix}\hat{x}\\ \hat{y}\\ \hat{z}\end{pmatrix} = R_{\text{axial}} \cdot \begin{pmatrix}x\\ y\\ z\end{pmatrix}. \tag{3.7}$$

ROI Translation: The ROI was extended in the x and y directions with 15 voxels and in the z direction the method presented by *A. Sekuboyina et al. 2017* [24] was used. With the difference that the ROI had a random number in the interval $[-25, 25]$ instead of a value from the discrete set $\{5, 10, 15, 20, 25\}$ as offset. This was done to compensate for eventual slight imperfections in the ROI localization step, as well as not producing a tight clamp around the anatomy.

3.5 Segmentation

The segmentation was done by the semantic segmentation model described in Section 3.2. Since the ROI was too large to be segmented in one pass, the ROI was subdivided into overlapping volumes before the segmentation, and merged into one volume afterwards.

3.5.1 ROI Subdivision

The augmented ROI was subdivided into overlapping volumes before segmentation. This was done because of hardware restrictions as the GPU used for this thesis could not store the entire model in its VRAM unless the volume was provided in chunks. For this purpose, subdivisions of size $160 \times 128 \times 96$ were selected as it was the same size used in *R. Janssens et al.* [13]. The overlap was set to half the shape of the desired size i.e an offset of 80 for x, 64 for y, and 48 for z. These subvolumes were independently segmented by the model and merged using the mean voxel value over all relevant subvolumes as described in Section 3.5.2 before the post-processing step. Figure 3.6 provides a visual guide for how the partitions were extracted from the larger volume.

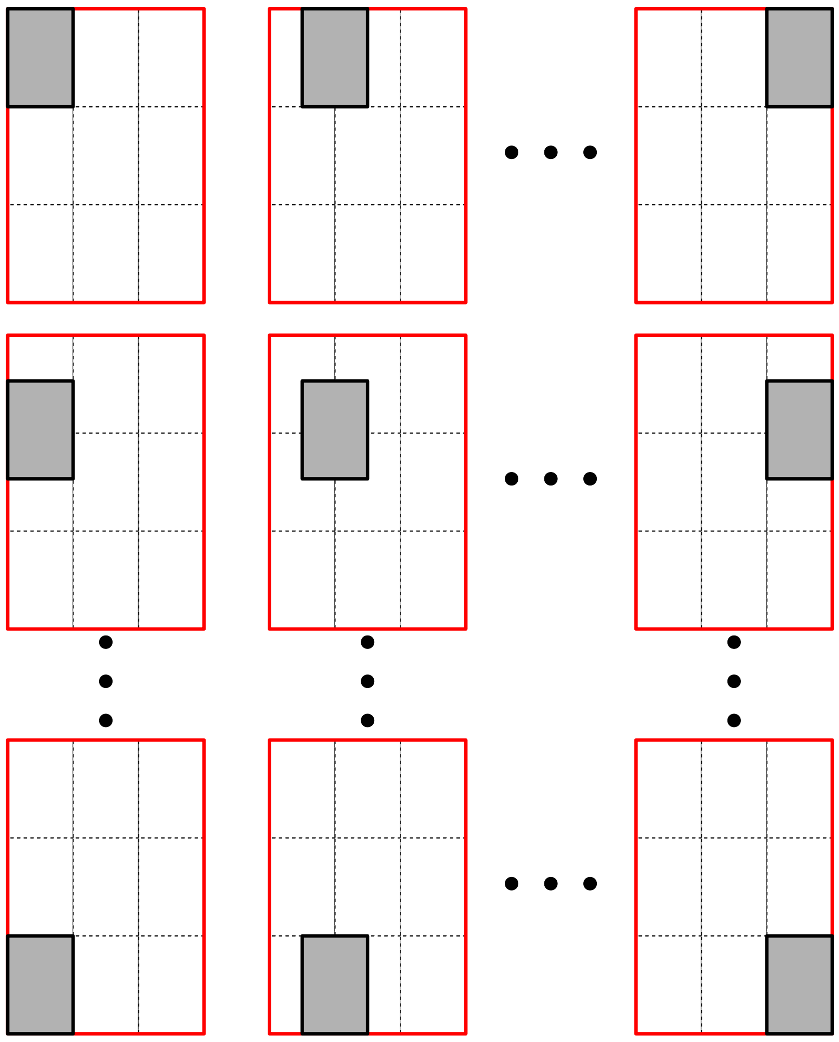


Figure 3.6: *Subvolume partitioning. Each gray tile indicates a partitions* $s_{ij} \in \mathcal{S}$ *with the size of* $160 \times 128 \times 96$ *. For the sake of demonstration the visual shows subdivision in 2D, the implementation was done in 3D.*

3.5.2 Volume stitching

Since the segmentation predictions were done on subvolumes, the final prediction volume q needed to be estimated by combining the subvolumes. Voxels which were contained in multiple subvolumes got the average value over all relevant subvolumes. Consequently,

$$q(x,y) = \frac{1}{\left|\mathcal{S}_{xy}\right|} \sum_{s \in \mathcal{S}_{xy}} s(x,y), \tag{3.8}$$

where $\mathcal{S}_{xy}$ denotes the set of subvolumes which contains voxel (x,y), and $s(x,y)$ denotes the segmentation prediction of voxel (x,y) in the given subvolume s.

3.6 Post-Processing

The raw output from the segmentation model often require some refinement. In this project three different methods were examined, *thresholding, morphing,* and *graph-cut*, see below.

Threshold: Perform a thresholding method on the prediction volume. If a given voxel had a predicted score of < 0.5 it was classified as *background* and if it had a score of ≤ 0.5 it was classified as *foreground*. An example is shown in Figure 3.7.

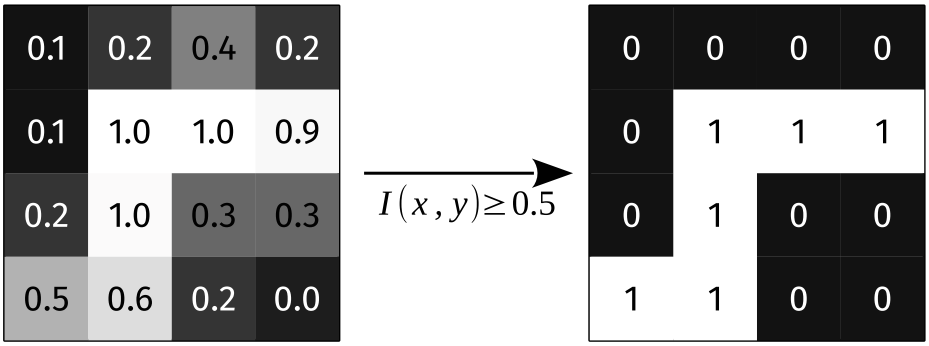


Figure 3.7: *Example of thresholding of a small region. The threshold is set to* 0.5.

Morphing: This procedure was based on the method presented in *A. Sekuboyina et al. 2017* [24], where the predicted volume was processed using a 3×3 *binary closing* on each *sagittal* (from anatomical left to right) slice followed by a removal of small (4-connected) *connected components* from the 3D volume. For this thesis a small *connected component* was chosen as any 4-connected group of segmented voxels with a voxel count of less than 125000. This value was decided based on exploratory testing of one of the results from the training set.

Graph-cut: This procedure performed the post-processing step presented by *N. Xu et al.* [27] to refine the output using a traditional *graph-cut* interactive segmentation model. The implementation was described in detail in Section 2.1.

3.7 Interactive Segmentation

Based on the work of *N. Xu et al.* [27], described in Section 2.4, this thesis examined if the segmentation could be improved by replacing the FCN with a 3D U-net, with additional post-processing methods, combined with segmentation hints placed by the user. This was done through a *graph-cut* optimization algorithm (explained in Section 2.1) with assistance of the prediction volume produced by the segmentation model.

Performing the graph-cut algorithm as described by *Y. Boykov and M-P Jolly 2001* [4] (also defined as eq 2.5) resulted in many integer overflows for the boundary term. Therefore, the boundary term was redefined as

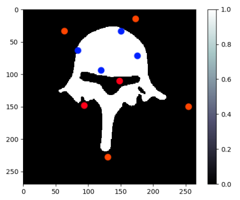
$$B_{\{p,q\}} = e^{-\frac{|I_p - I_q|}{2\sigma^2}}, \tag{3.9}$$

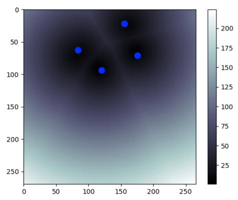
with σ set to 13 after a brute-force optimization in the range $[0, 100]$ using the *image001* as test volume. I_p describes the intensity value of voxel p in the volume I.

3.7.1 Distance Maps

Additionally, the *distance maps* which indicate background and foreground points placed by the user were created by placing the points in two volumes with the same size as the prediction volume and applying the *Euclidean distance transform*. This produced two dense volumes which could be concatenated to the initial volume as channels to provide context for the segmentation method. Example of distance maps for foreground and background can be seen in Figure 3.8 with the corresponding image slice.

The hints indicating the foreground object were placed uniformly on random voxels annotated as *object* in the dataset. Hints indicating background were placed using the strategies described in Section 2.4. The first placement strategy places points around the target object with a maximum distance of 5 voxels from the edge of the target vertebrae. The second strategy places points randomly on other lumbar vertebraes in the volume. The third strategy places points around the target vertebrae as the first strategy, but also tries to maximize the distance between each point, to provide greater cover around the segment boundary.


(a) GT mask.


(b) Foreground distance map.

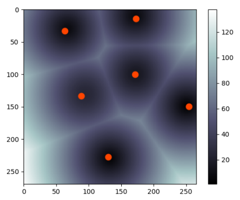

(c) Background distance map.

Figure 3.8: *Example of an annotated image slice with foreground (red) and background (blue) points placed. 3.8b and 3.8c provides the corresponding distance maps.*

3.8 Performance Metrics

This section describes the evaluation metrics used for quantitative analysis of the segmentation method. *Dice Score* and *Intersection over Union* were selected for their appearance in multiple segmentation challenges as well as other papers presenting segmentation methods, see e.g. *xVertSeg challenge* [1]. *Precision* and *Recall* were selected as well to provide insight into the type of segmentation errors the model produces, and to assist in the performance analysis.

Precision, also called the *positive predicted value*, indicate the amount of correctly predicted foreground voxels compared to all voxels classified as foreground, close to 1 would indicate good results and close to 0 bad ones. *Precision* can be represented using a binary confusion matrix as

$$\frac{TP}{TP + FP}, \tag{3.10}$$

where TP indicate correctly segmented foreground voxels and FP incorrectly segmented foreground voxels.

Recall, also called the *true positive rate*, indicate the ratio between correctly predicted foreground voxels compared to all voxels which should have been classified as foreground. A score close to 1 would indicate a good result while a score close to 0 would indicate bad ones. *Recall* can be described using a confusion matrix as the equation

$$\frac{TP}{TP + FN}, \tag{3.11}$$

where FN indicate voxels incorrectly segmented as background.

Sørensen–Dice coefficient, also known as F_1*-Score*, or simply *Dice score*, is defined as the harmonic mean between the precision and recall scores. In some implementations a weight dictate if the score should *skew* towards one of the metrics depending on which is most relevant for the problem. This implementation only takes the harmonic mean. Mathematically, F_1*-Score* can be written as

$$\frac{2TP}{2TP + FP + FN}, \tag{3.12}$$

which is useful when both *precision* and *recall* are of interest for the comparison.

Intersection over Union (IoU), also known as the *Jaccard index* is often used when comparing the relative overlap between ground truth segmentation and the predicted one. Often used in object detection and localization to compare bounding-boxes but has seen significant use within segmentation problems such as the *Cityscapes* dataset [6]. A score close to 1 would indicate a good result while a score close to 0 would indicate a bad result. IoU can be written using a confusion matrix as

$$\frac{TP}{TP + FP + FN}. \tag{3.13}$$

4 Results

This chapter provides the different scores and metrics presented in Chapter 3 based on the training and evaluation data. The chapter is split up into three primary parts, Pre-Training 4.1, Interactive Segmentation 4.2, and Visual Inpsection 4.3.

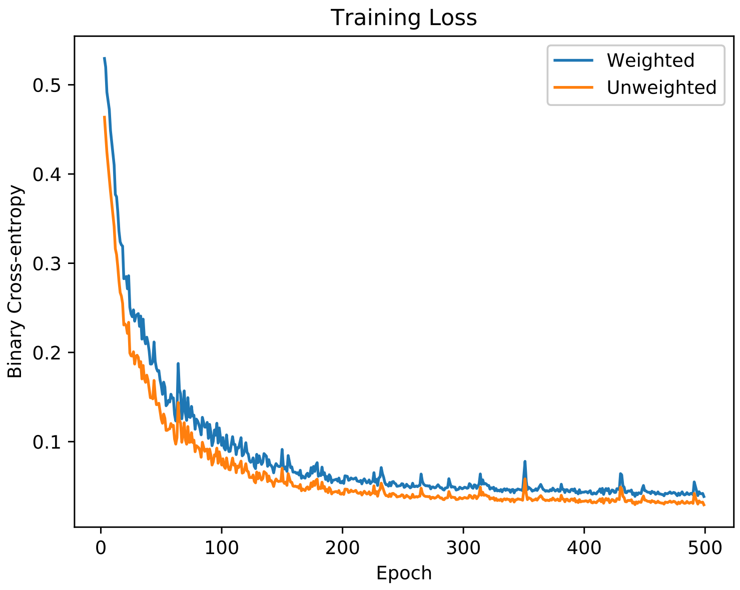
4.1 Pre-Training

Following the method presented in Section 3.3 and Figure 3.2, the model was pre-trained by segmenting all vertebrae as the same label and without distance maps.

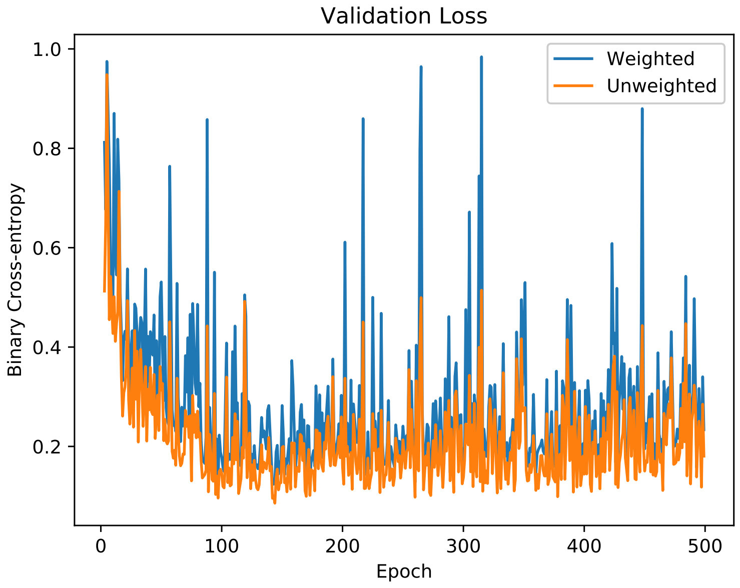
4.1.1 Training

The model architecture was identical to Figure 3.1 with a learning rate of 10^{-4}, number of epochs 500 and samples per epoch of 80. The number of samples per epoch was chosen to allow for all partition of the entire dataset to be evaluated atleast once each epoch. Since the pre-training segmentation is done with all vertebrae being assigned the same label, the distance maps on the input is set to 0. The dataset and its corresponding split into training/validation sets can be seen in Table 3.1. Three volumes were used for validation/evaluation and 12 volumes were used for training.

Figure 4.1a and 4.1b shows the *cross-entropy* loss over training epochs. For loss over training samples (Figure 4.1a), the loss function seem to converge over the epochs. For the validation loss (figure 4.1b), there is significantly more variance but some convergence can be seen between the 0th and $100th$ epoch.



(a) *Training loss of 12 augmented volumes*



(b) *Validation loss of 3 augmented volumes*

Figure 4.1: *Training and Validation loss for binary segmentation. The Blue line indicate weighted cross-entropy and the orange indicate the default non-weighted cross-entropy from Tensorflow. Values closer to 0 are better and there is no upper bound. Note how the training loss is similar to the typical inverse square function and how validation loss has significantly more variance. Dataset can be seen in Table 3.1.*

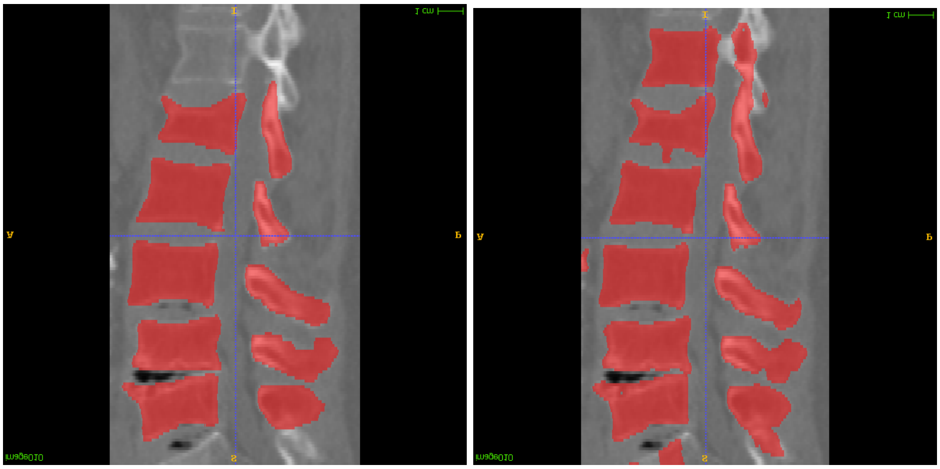
4.1.2 Difference between Post-Processing Methods

Table 4.1 shows mean and standard deviation over the two post-processing methods used for complete lumbar bone segmentation. *Morphing* post-processing produced slightly better results compared to *thresholding*.

Post-Processing	Dice	Precision	Recall	IoU
Morphed	**0.923±0.036**	**0.945±0.024**	**0.903±0.062**	**0.858±0.063**
Threshold	0.915±0.026	0.943±0.025	0.890±0.046	0.844±0.044

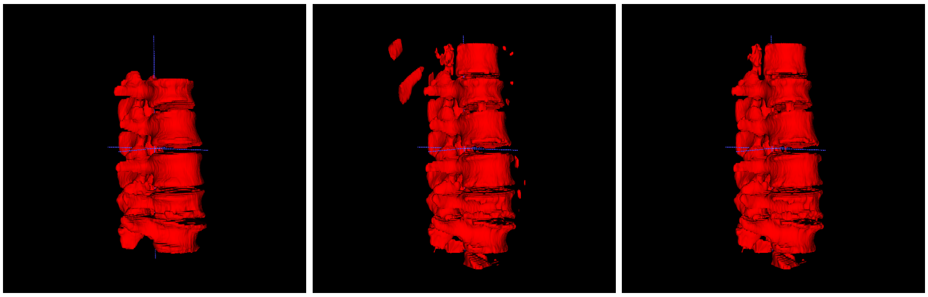
Table 4.1: *Pre-training segmentation results over the validation set seen in Table 3.1. Mean and standard deviation for the different metrics over post-processing methods. The 3 CT-volumes in the validation set was used for evaluation. Note that* morphed *slightly outperforms* threshold *by around 1% for each metric.*

Figure 4.2 provides a view of the volume *image010* when segmented during the binary pre-training phase, with thresholding post-processing. Figure 4.3 shows the same segmentation in 3D, with the inclusion of the *morphed* post-processing method as well. In addition to the lumbar spine, parts of the ribs, T12, and S1 were segmented as well.



*(**a**) Reference Mask* *(**b**) Predicted Segmentation (IoU* 0.80*)*

Figure 4.2: *Pre-training segmentation performed on image010 using the threshold post-processing method. Red color indicate voxels in the segment. Note how the lowest* thoracic vertebrae *T12, as well as the highest* sacral *S1 was segmented to a large part as well.*



(**a**) Reference Mask 3D (**b**) Threshold 3D (IoU 0.80) (**c**) Morphed 3D (IoU 0.81)

Figure 4.3: *Pre-training segmentation in 3D performed on image010 using both thresholding and morphing post-processing. Red color indicate voxels in the segment. Note how parts of the rib was segmented as well when* threshold *was used but removed in the* morphed *segmentation.*

Figure 4.4 shows a scatter plot over IoU and Dice score for the pre-training evaluation. Post-processing with *morphing* produces a slightly better segmentation on *image003*.

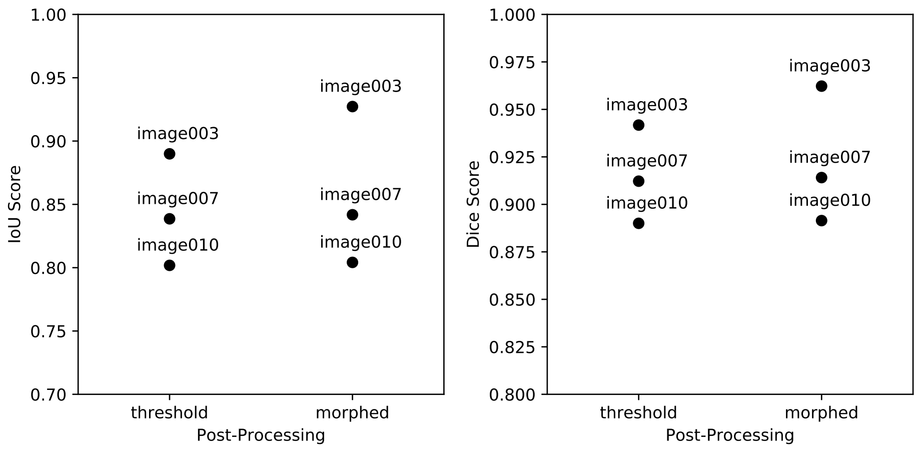


Figure 4.4: *IoU and Dice score over post-processing methods for the pre-training evaluation. Note how* morphing *has a slightly better best-case.*

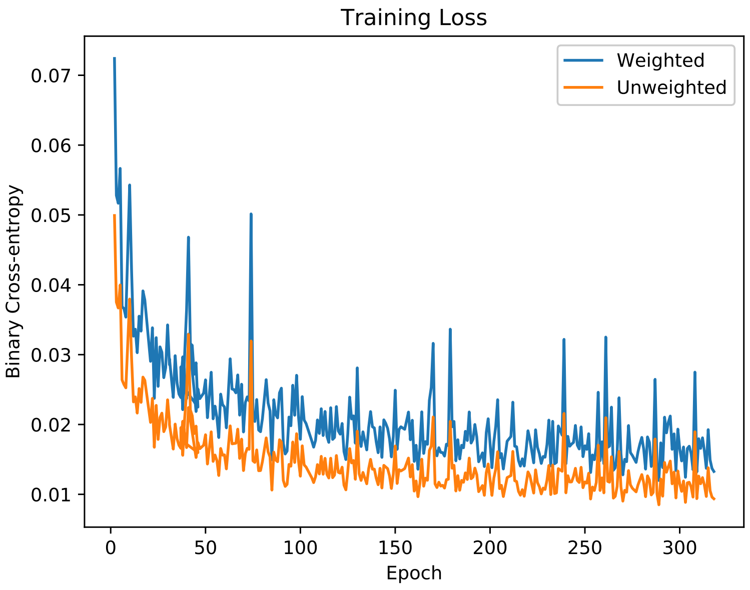
4.2 Interactive Segmentation

This section presents the results of providing segmentation guides by the user. The model trained during the *pre-training* phase is used for interactive segmentation with the same volumes for training and validation. As with pre-training, a learning rate of 10^{-4} was used. Number of epochs was set to 350 and samples per epoch set to 240, again to allow for all partitions of the entire dataset to be evaluated at least once each epoch..

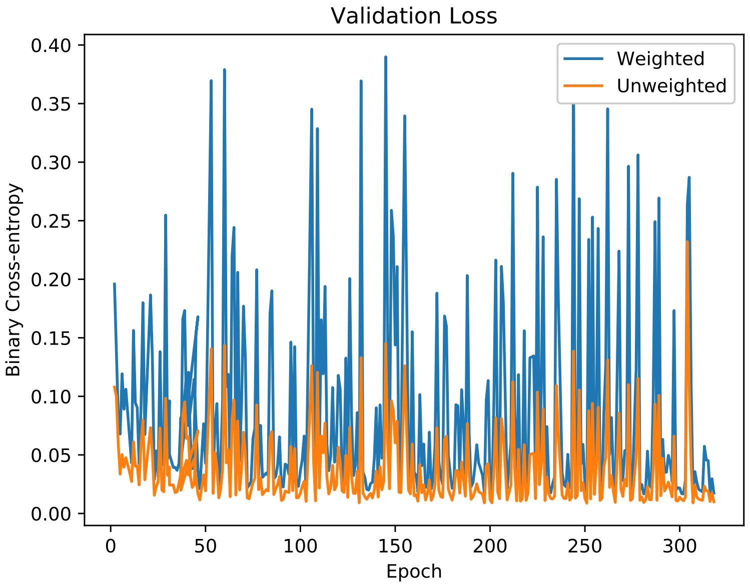
The training follows the one presented to the right in Figure 3.2. It was done by selecting a single vertebrae for each sample. A random amount of points were placed in accordance with the strategies presented by *N. Xu et al. 2016* [27], described in Section 2.4.

4.2.1 Training

Below in Figure 4.5a and 4.5b the respective *training* and *validation* loss can be observed. The same *loss functions* as with the pre-training was used. In Figure 4.5a, the initial loss is significantly lower compared the the pre-training, possibly caused by the weights in the pre-training only requiring fine-tuning, as most image-features would be similar. No discernible convergence can be observed in the validation loss (Figure 4.5b).



***(a)** Training loss*



***(b)** Validation loss*

***Figure 4.5:** Training and validation loss for interactive segmentation. The blue line indicate sample-weighted cross-entropy and the orange indicate standard non-weighted cross-entropy from Tensorflow. Values closer to 0 are better and there is no upper bound. Note that training loss converges faster than for the pre-training while validation loss does not converge to any significant degree.*

4.2.2 Effects from Point Placement

Tables 4.2, 4.3, and 4.4 shows the different metrics over number of user-provided hints, with mean and standard deviation. Figure 4.6 shows heatmaps over IoU and Dice scores based on number of user-placed points and placement strategy. The placement strategies can be found under Section 2.4.

Looking at the heat-maps in Figure 4.6, the number of foreground points seems to affect the result much more than background points, with a convergence slightly above 8 foreground points. The difference between background point placement strategies is negligible.

	Dice	Precision	Recall	IoU
Points				
1	0.285 ± 0.262	0.199 ± 0.204	0.924 ± 0.180	0.197 ± 0.201
2	0.562 ± 0.320	0.466 ± 0.298	**0.950 ± 0.067**	0.454 ± 0.290
4	0.810 ± 0.213	0.751 ± 0.230	0.946 ± 0.044	0.719 ± 0.221
8	0.930 ± 0.053	0.921 ± 0.080	0.945 ± 0.020	0.874 ± 0.076
16	0.952 ± 0.015	0.964 ± 0.021	0.940 ± 0.021	0.908 ± 0.026
32	**0.954 ± 0.013**	**0.969 ± 0.014**	0.939 ± 0.022	**0.912 ± 0.023**
64	0.953 ± 0.013	0.968 ± 0.014	0.939 ± 0.022	0.911 ± 0.023
128	0.952 ± 0.012	0.965 ± 0.015	0.939 ± 0.021	0.909 ± 0.022

Table 4.2: *Mean and standard deviation over validation samples for the different metrics over a number of foreground points.* Background points *vary between* 1 *and* 128. *Note how the number of foreground points significantly affect on the scores.*

	Dice	Precision	Recall	IoU
Points				
1	0.761 ± 0.318	0.728 ± 0.342	**0.952 ± 0.048**	0.696 ± 0.323
2	0.774 ± 0.307	0.744 ± 0.334	0.943 ± 0.083	0.709 ± 0.315
4	0.795 ± 0.292	0.771 ± 0.317	0.940 ± 0.067	0.731 ± 0.298
8	0.806 ± 0.280	0.782 ± 0.309	0.941 ± 0.059	0.741 ± 0.290
16	0.811 ± 0.277	0.789 ± 0.305	0.940 ± 0.054	0.747 ± 0.285
32	0.803 ± 0.281	0.781 ± 0.311	0.938 ± 0.058	0.739 ± 0.291
64	0.823 ± 0.261	0.802 ± 0.292	0.938 ± 0.055	0.758 ± 0.272
128	**0.826 ± 0.257**	**0.806 ± 0.288**	0.933 ± 0.077	**0.761 ± 0.268**

Table 4.3: *Mean and standard deviation over validation samples for the different metrics over a number of background points.* Background points *vary between* 1 *and* 128. *Note how the number of background points seem to have a very small effect on the scores.*

Strategy	Dice	Precision	Recall	IoU
1	0.798 ± 0.290	0.776 ± 0.317	0.936 ± 0.086	0.735 ± 0.298
2	0.796 ± 0.287	0.769 ± 0.316	**0.945** ± **0.051**	0.731 ± 0.296
3	**0.805** ± **0.279**	**0.781** ± **0.308**	0.941 ± 0.046	**0.740** ± **0.288**

Table 4.4: *Mean and standard deviation over validation samples for the different metrics over a number of background placement strategy. Note how only minor differences can be observed between the strategies.*

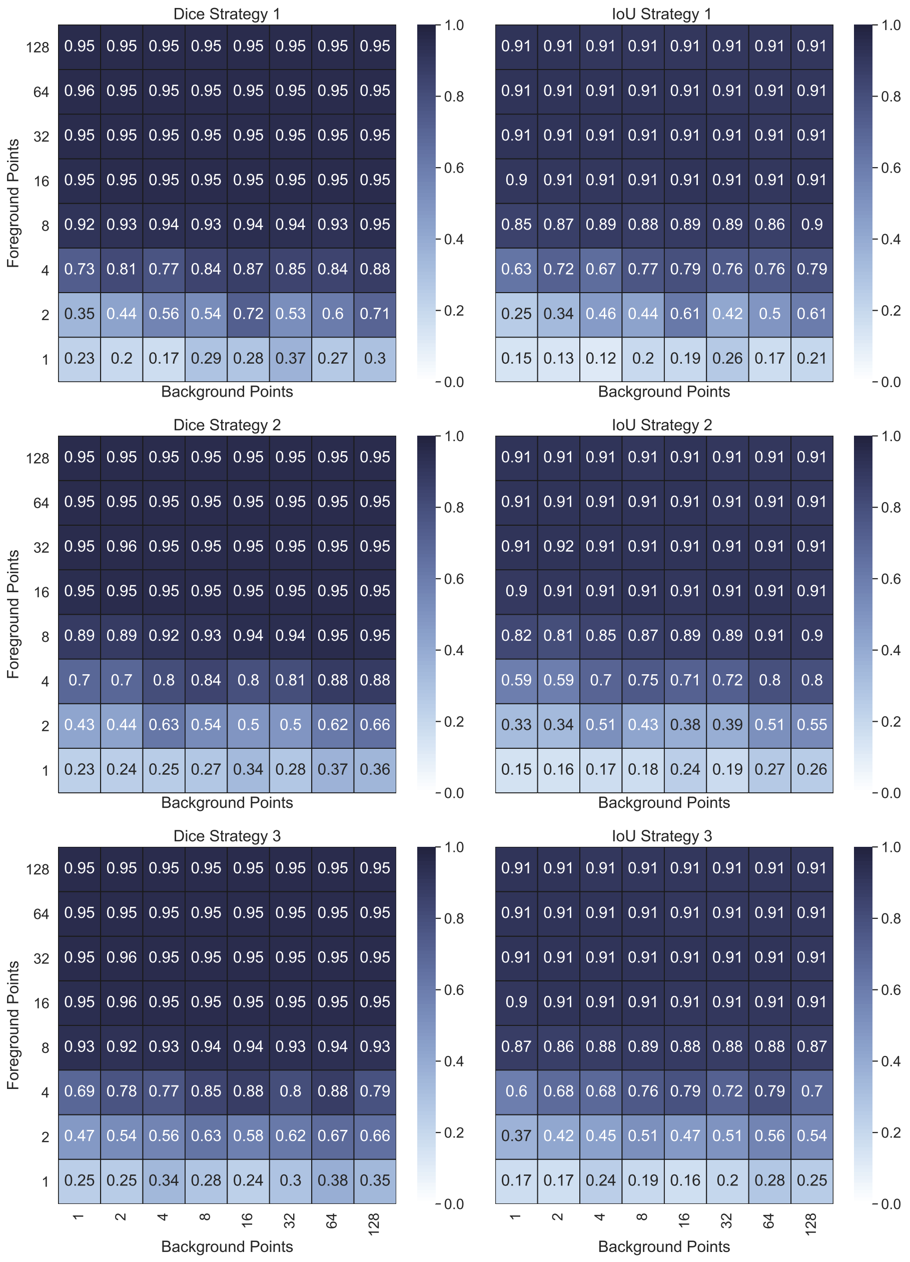


Figure 4.6: *Heat-maps showing mean Dice and IoU score over Point placement. Darker colour indicate higher mean value. Note how the* number of background points *does not seem to significantly affect the result compared* to number of foreground points, *which converges around 8 placed points.*

4.2.3 Vertebrae Segmentation

Table 4.5 and Figure 4.7 presents the difference in segmentation between vertebrae label. Table 4.5 indicate that the *L3* vertebrae were the easiest to segment, while *L5* was the hardest.

Vertebrae	Dice	Precision	Recall	IoU
L1	0.792 ± 0.292	0.777 ± 0.320	0.925 ± 0.077	0.726 ± 0.296
L2	0.829 ± 0.264	0.801 ± 0.290	**0.953** ± **0.062**	0.769 ± 0.275
L3	**0.838** ± **0.252**	**0.815** ± **0.283**	0.949 ± 0.050	**0.778** ± **0.266**
L4	0.791 ± 0.291	0.764 ± 0.320	0.941 ± 0.065	0.726 ± 0.301
L5	0.749 ± 0.315	0.719 ± 0.343	0.934 ± 0.058	0.678 ± 0.319

Table 4.5: *Mean and standard deviation over validation samples for the different metrics over lumbar vertebrae, Close to 1 is good and close to 0 is bad. Note how segmentation of L3 produces the highest Dice and IoU scores while L5 have significantly lower IoU. The number of foreground/background points were varied between* 1 *and* 128*.*

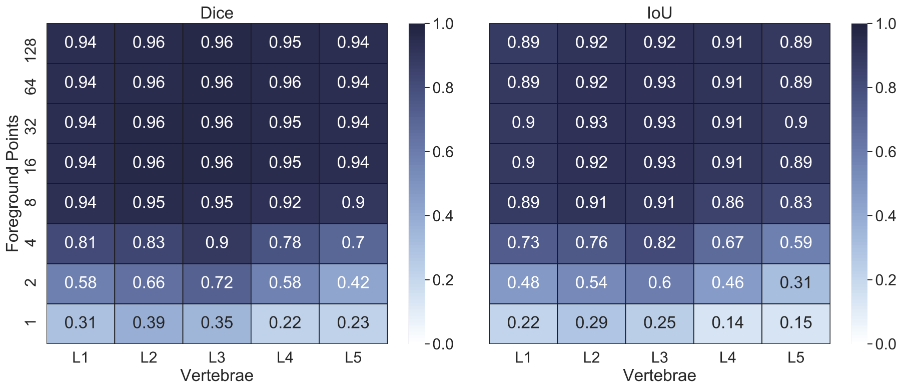


Figure 4.7: *Heatmaps showing Dice and IoU score Vertebrae partitioned over* number of foreground points. *Darker colour indicate higher mean value.*

4.2.4 Difference between Post-processing Methods

Table 4.6 and Figure 4.8 show how the different post-processing methods affected the evaluation result. *Thresholding* and *graph-cut* performed better, with *morphing* performing poorly on low amount of foreground points. If more than 8 foreground points were provided, all methods had similar performance.

Processing	Dice	Precision	Recall	IoU
Graphcut	**0.831** ± **0.219**	**0.797** ± **0.270.**	0.941 ± 0.074	**0.757** ± **0.251**
Morphed	0.738 ± 0.379	0.732 ± 0.383	**0.944** ± **0.020**	0.693 ± 0.362
Threshold	**0.830** ± **0.218**	**0.798** ± **0.270**	0.938 ± 0.074	**0.756** ± **0.251**

Table 4.6: *Mean and standard deviation over validation samples for the different metrics over post-processing methods. Note how* Graphcut *have scores close to* Threshold *but* Morphed *have significantly lower score.*

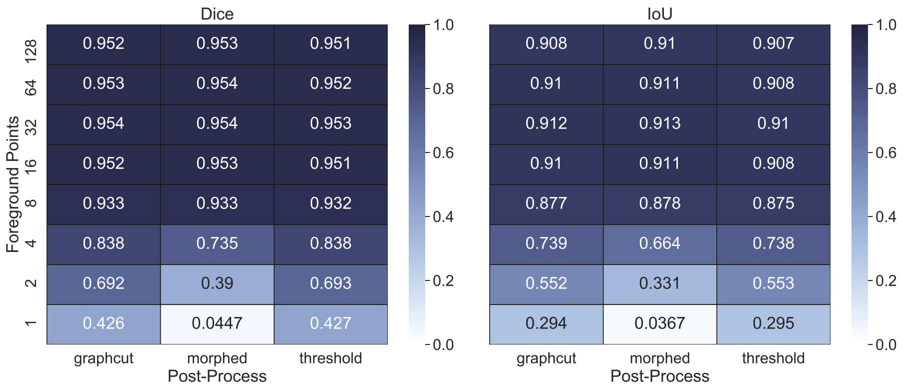


Figure 4.8: *Heat-maps showing Dice and IoU score over post-processing methods. Darker colour indicate higher mean value. Note how* Morphed *performs significantly worse on low* number of foreground points *compared to other methods but slightly better at higher* number of foreground points.

4.2.5 Difference between Samples

Table 4.7 and Figure 4.9 shows the evaluation difference between the three validation samples.

Sample	Dice	Precision	Recall	IoU
image003	**0.831** ± **0.276**	**0.809** ± **0.297**	**0.955** ± **0.054**	**0.777** ± **0.283**
image007	0.774 ± 0.300	0.756 ± 0.334	0.924 ± 0.072	0.706 ± 0.307
image010	0.794 ± 0.277	0.761 ± 0.306	0.943 ± 0.061	0.723 ± 0.286

Table 4.7: *Mean and standard deviation for the different metrics over validation samples.*

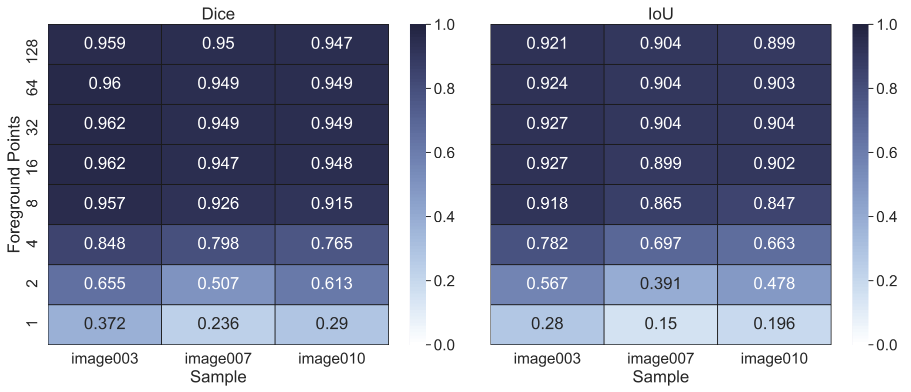
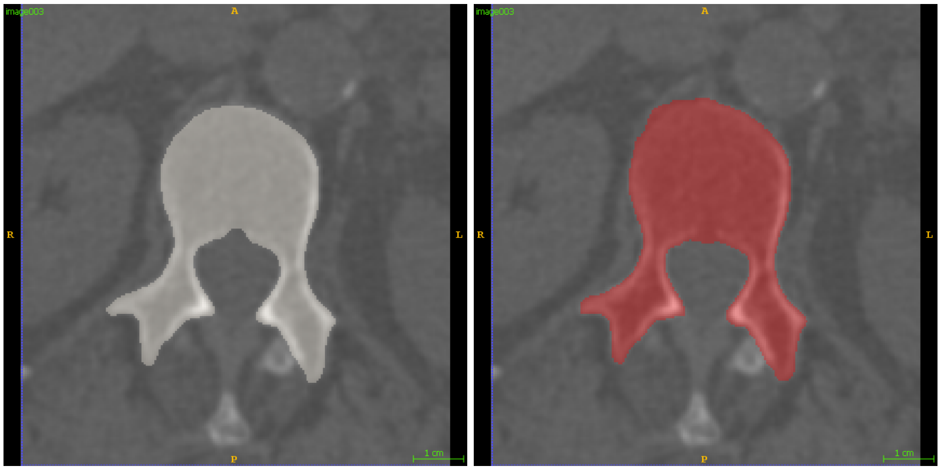


Figure 4.9: *Heat-maps showing Dice and IoU score over validation samples. Darker colour indicate higher mean value. Note how the samples have different scores when few foreground points are placed but similar when many points are provided.*

4.3 Visual Inspection

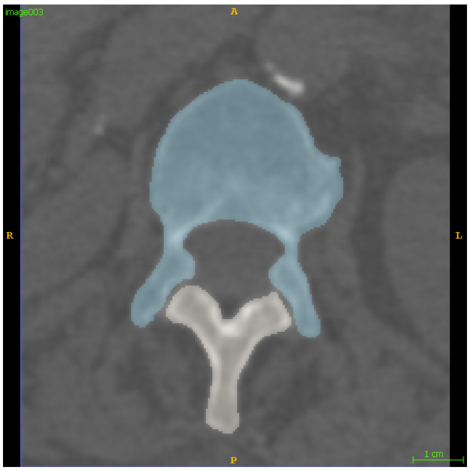
This section presents visual references to determine the quality and usability of the segmentation method.

Figure 4.10, 4.11, 4.12, 4.13 and 4.14 shows examples of each lumbar vertebrae in the *image003* CT-volume. The slices are presented with corresponding reference masks.

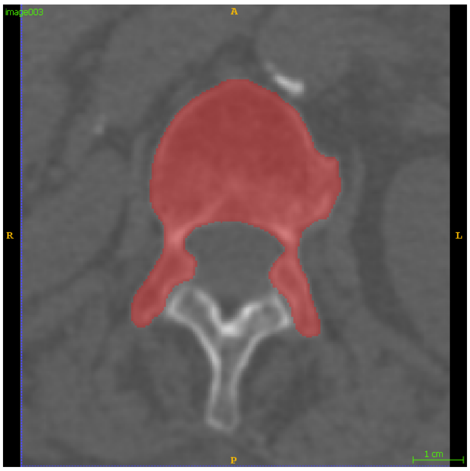


*(**a**) Reference Mask* *(**b**) Segmentation (Dice* 0.96*)*

Figure 4.10: *Segmentation and reference for the L1 vertebrae in* image003 *with 16 provided hints.*

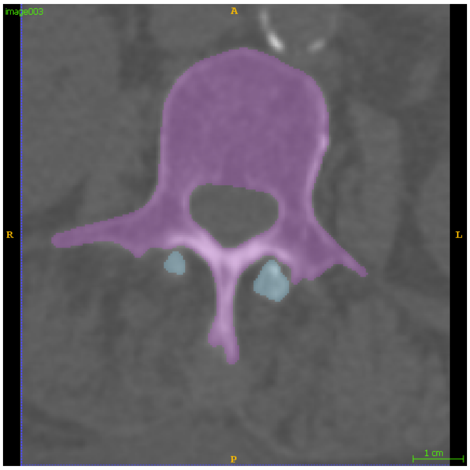


*(**a**) Reference Mask*

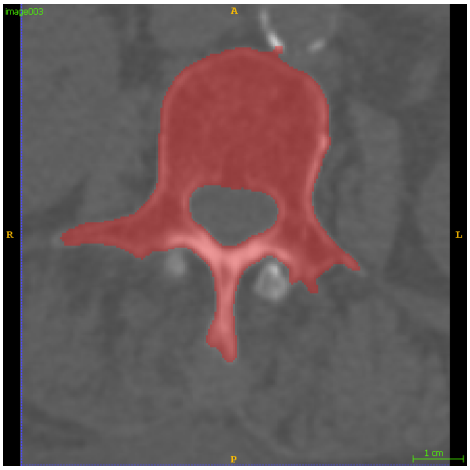


*(**b**) Segmentation (Dice* 0.95*)*

Figure 4.11: *Segmentation and reference mask for the L2 vertebrae in* image003 *with 16 provided hints.*

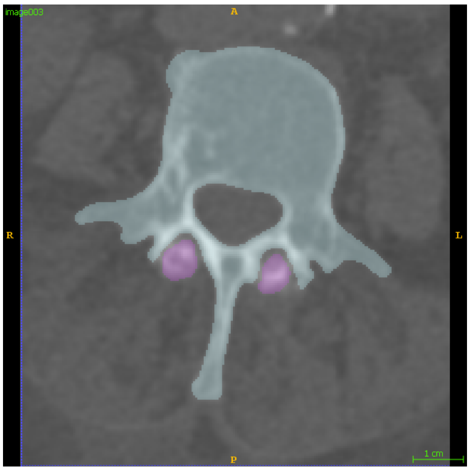


*(**a**) Reference Mask*

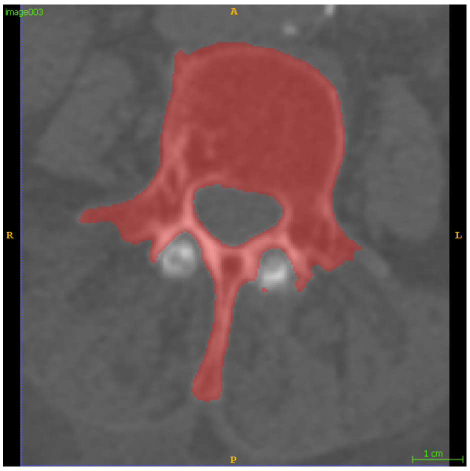


*(**b**) Segmentation (Dice* 0.95*)*

Figure 4.12: *Segmentation and reference mask for the L3 vertebrae in* image003 *with 16 provided hints.*

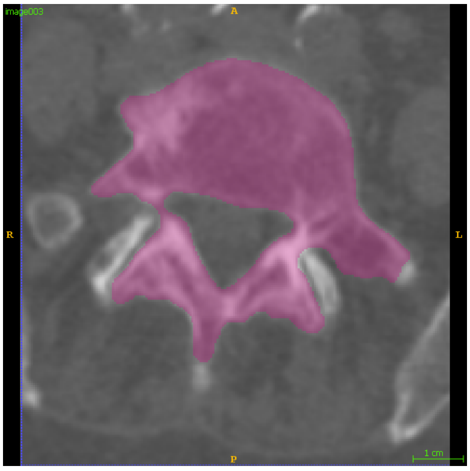


(a) Reference Mask

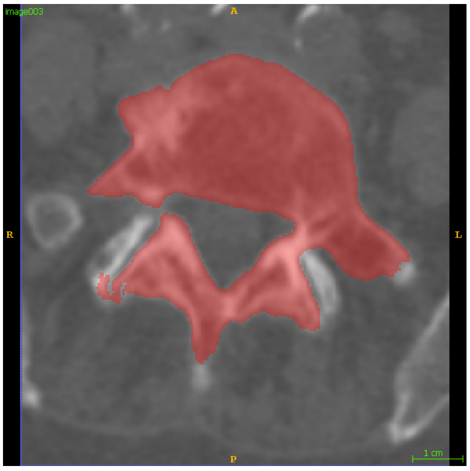


(b) Segmentation (Dice 0.96*)*

Figure 4.13: *Segmentation and reference mask for the L4 vertebrae in* image003 *with 16 provided hints.*



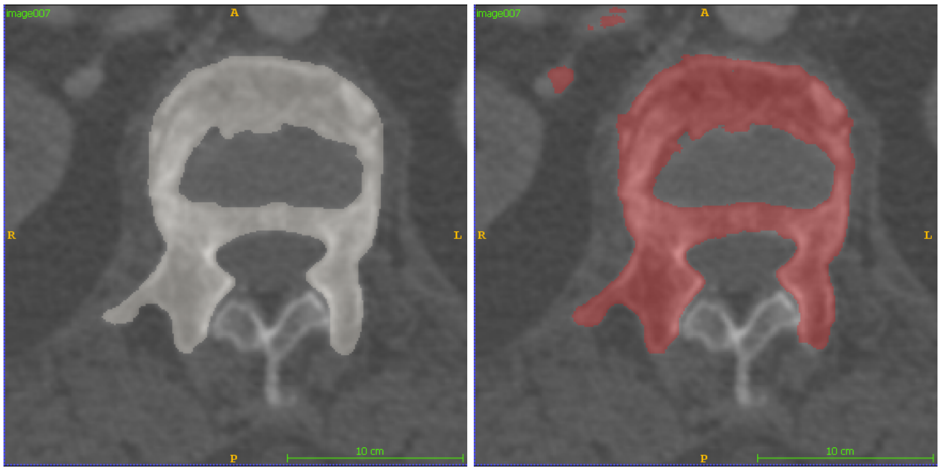
(a) Reference Mask



(b) Segmentation (Dice 0.94*)*

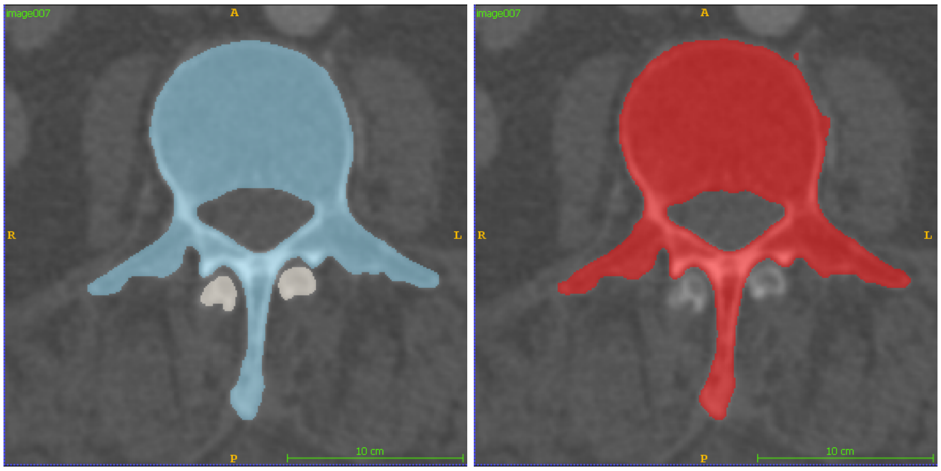
Figure 4.14: *Segmentation and reference mask for the L5 vertebrae in* image003 *with 16 provided hints.*

Figure 4.15, 4.16, 4.17, 4.18 and 4.19 shows examples of each lumbar vertebrae in the *image007* CT-volume. The slices are presented with corresponding reference masks.



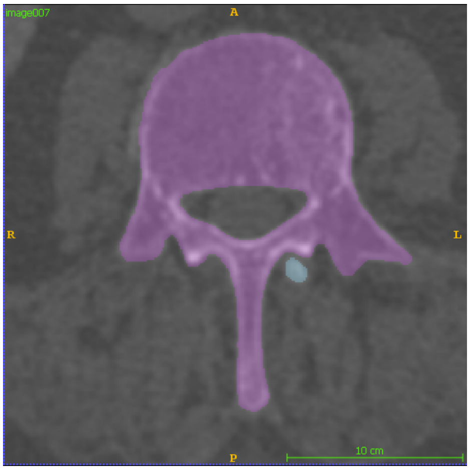
*(**a**) Reference Mask* *(**b**) Segmentation (Dice* 0.92*)*

Figure 4.15: *Segmentation and reference mask for the L1 vertebrae in* image007 *with 16 provided hints.*

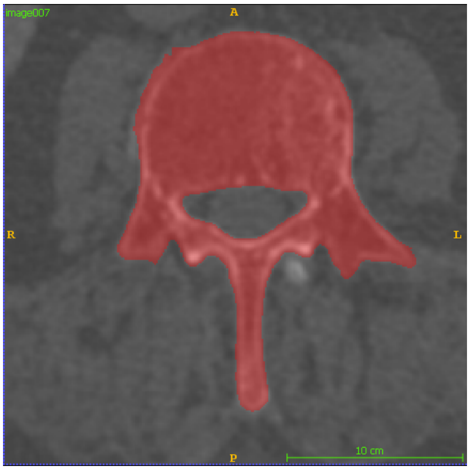


*(**a**) Reference Mask* *(**b**) Segmentation (Dice* 0.96*)*

Figure 4.16: *Segmentation and reference mask for the L2 vertebrae in* image007 *with 16 provided hints.*

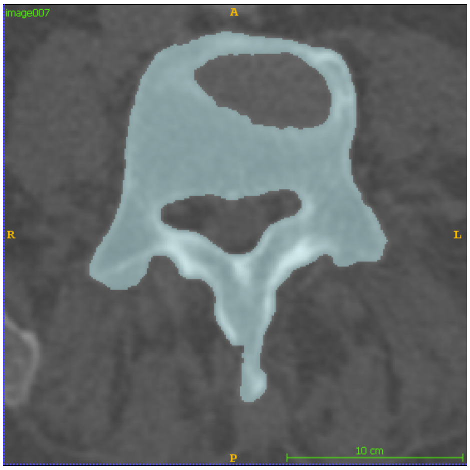


*(**a**) Reference Mask*

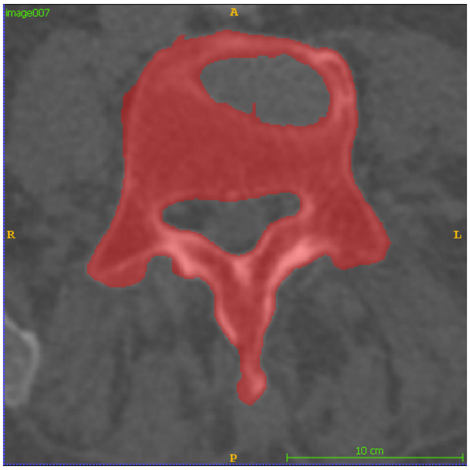


*(**b**) Segmentation (Dice* 0.95*)*

Figure 4.17: *Segmentation and reference mask for the L3 vertebrae in* image007 *with 16 provided hints.*

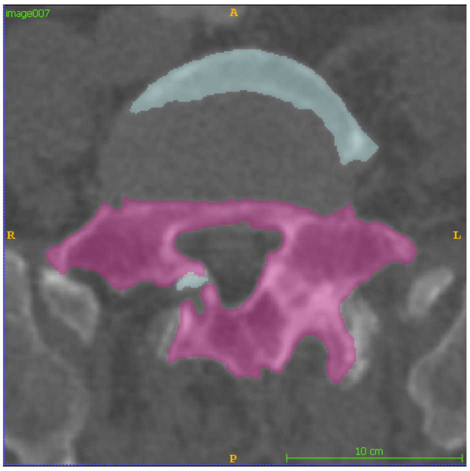


*(**a**) Reference Mask*

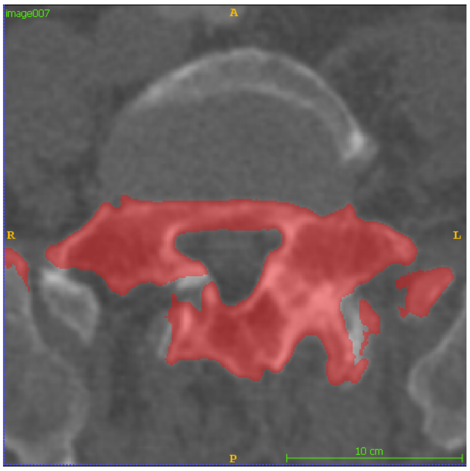


*(**b**) Segmentation (Dice* 0.93*)*

Figure 4.18: *Segmentation and reference mask for the L4 vertebrae in* image007 *with 16 provided hints.*



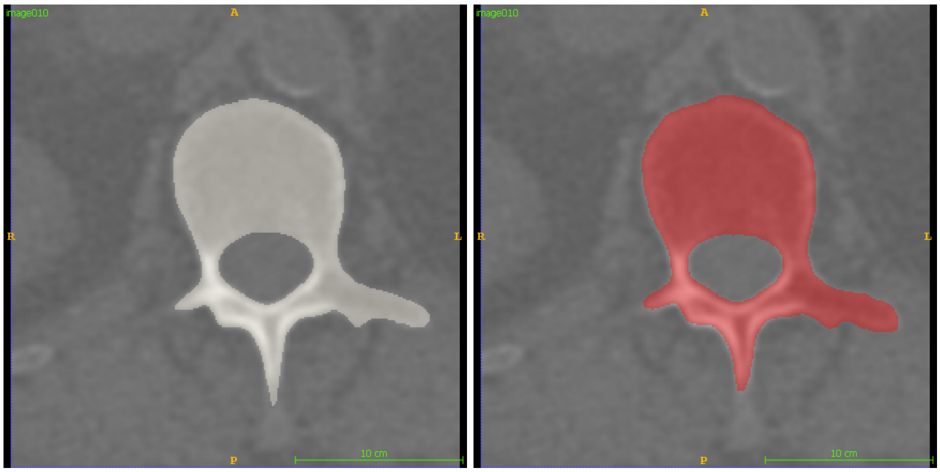
(a) *Reference Mask*



(b) *Segmentation (Dice* 0.91*)*

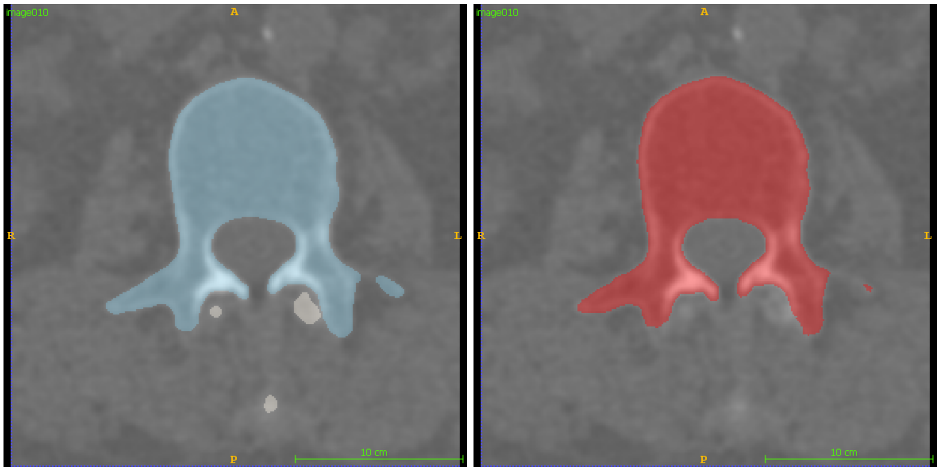
Figure 4.19: *Segmentation and reference mask for the L5 vertebrae in* image007 *with 16 provided hints.*

Figure 4.20, 4.21, 4.22, 4.23 and 4.24 shows examples of each lumbar vertebrae in the *image010* CT-volume. The slices are presented with corresponding reference masks.



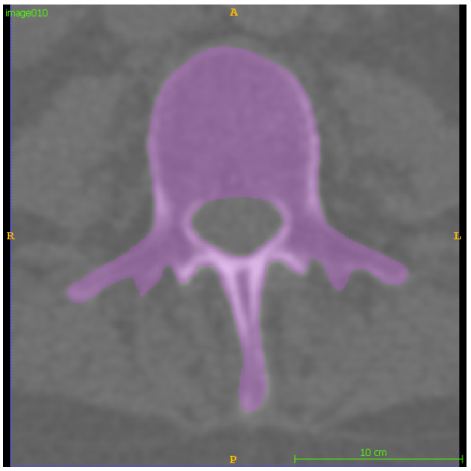
*(**a**) Reference mask* *(**b**) Segmentation (Dice* 0.94*)*

Figure 4.20: *Segmentation and reference mask for the L1 vertebrae in* image010 *with 16 provided hints.*

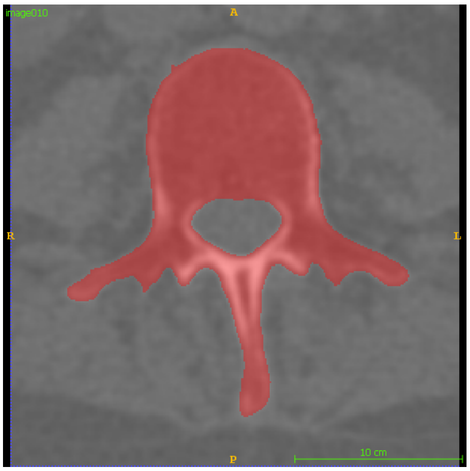


*(**a**) Reference mask* *(**b**) Segmentation (Dice* 0.95*)*

Figure 4.21: *Segmentation and reference mask for the L2 vertebrae in* image010 *with 16 provided hints.*

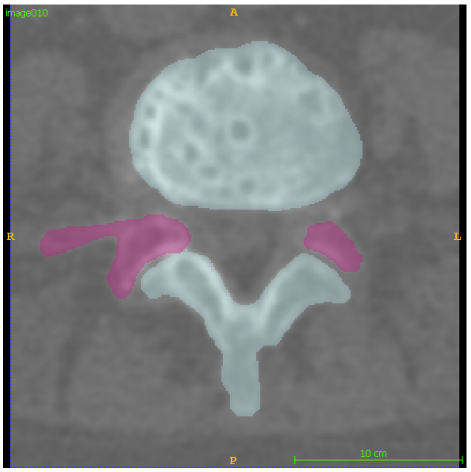


(a) *Reference mask*

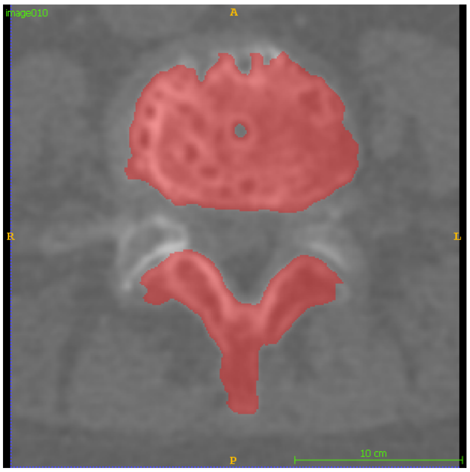


(b) *Segmentation (Dice* 0.93*)*

Figure 4.22: *Segmentation and reference mask for the L3 vertebrae in* image010 *with 16 provided hints.*

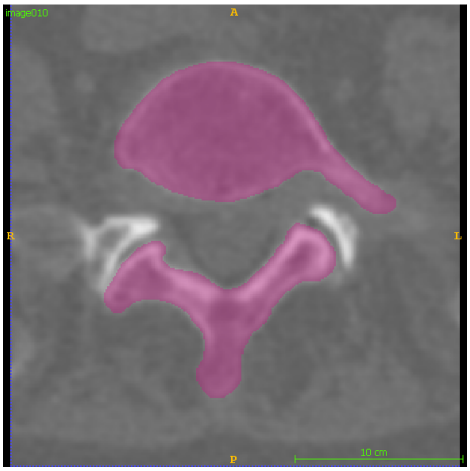


(a) *Reference mask*

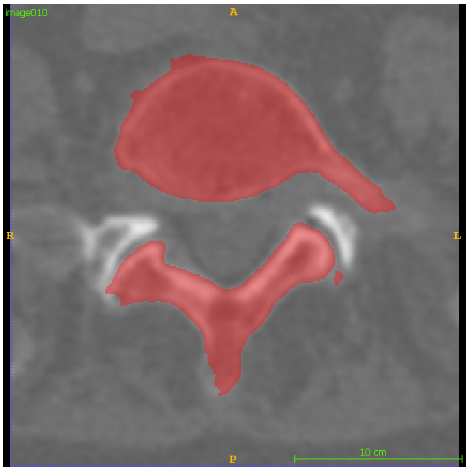


(b) *Segmentation (Dice* 0.91*)*

Figure 4.23: *Segmentation and reference mask for the L4 vertebrae in* image010 *with 16 provided hints.*



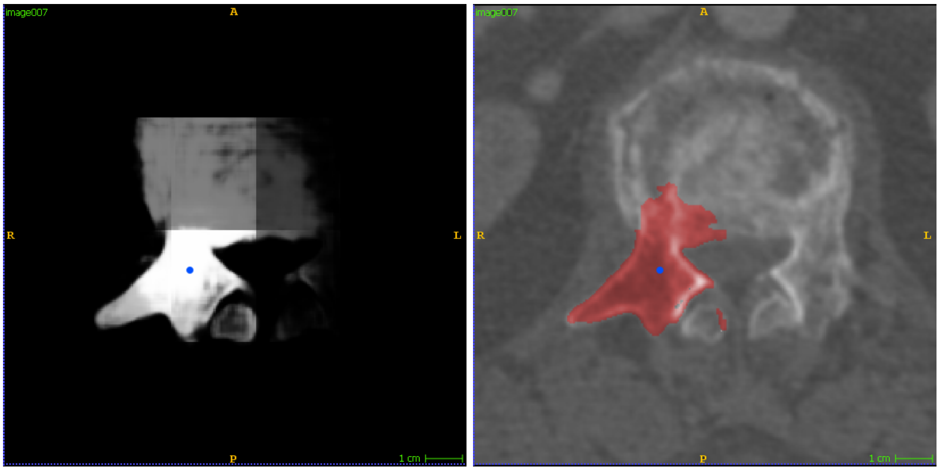
*(**a**) Reference Mask*



*(**b**) Segmentation (Dice* 0.90*)*

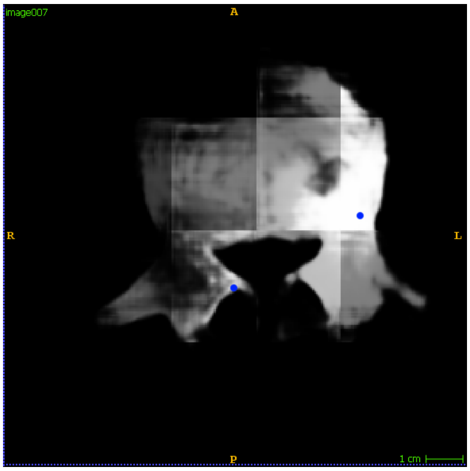
Figure 4.24: *Segmentation and reference mask for the L5 vertebrae in* image010 *with 16 provided hints.*

Figure 4.25, 4.26, 4.27 and 4.28 presents the raw prediction produced by the segmentation model and the final segmentation with foreground points indicated as blue dots. Each case represents the same slice of a CT-volume but the points are randomized for each case. Hints far away from the given slice are removed as to not cause confusion. The images shows how the model perform better as more hints are provided. Additionally, the partitions are clearly visible even after the combination, with partitions close to hints containing higher confidence. The brighter the colour, the more certain is the model is of its prediction.

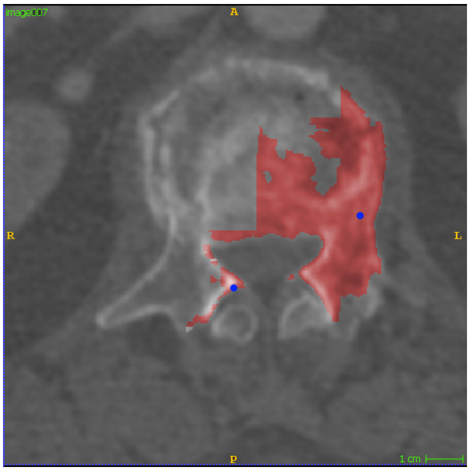


(a) Raw prediction *(b) Segmentation (Dice 0.51)*

Figure 4.25: *Segmentation and reference mask for the L1 vertebrae in image007 with 2 hint provided. Note how different partitions are clearly distinguishable, as well as the spatial relationship between the point and the model confidence.*

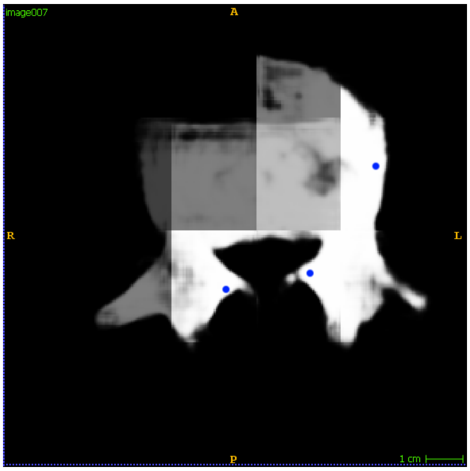


(a) Raw prediction

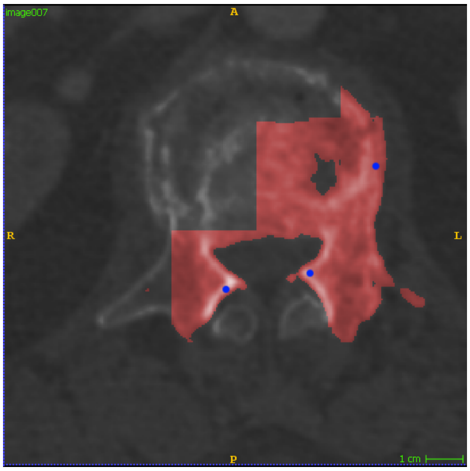


(b) Segmentation (Dice 0.70)

Figure 4.26: *Segmentation and reference mask for the L1 vertebrae in image007 with 2 hints provided. Note how different partitions are clearly distinguishable, as well as the spatial relationship between the point and the model confidence.*

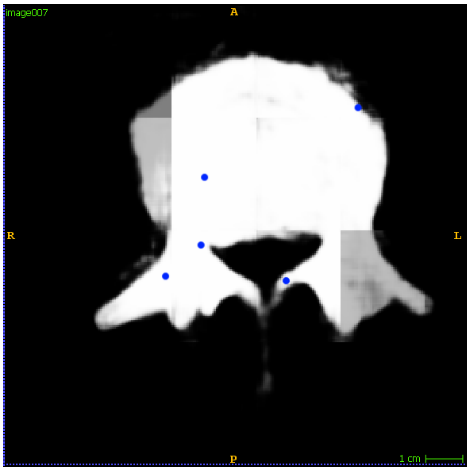


(a) Raw prediction

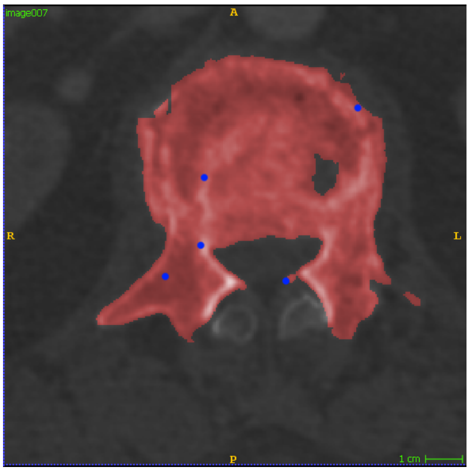


(b) Segmentation (Dice 0.77)

Figure 4.27: *Segmentation and reference mask for the L1 vertebrae in image007 with 4 hints provided. Note how different partitions are clearly distinguishable, as well as the spatial relationship between the point and the model confidence.*



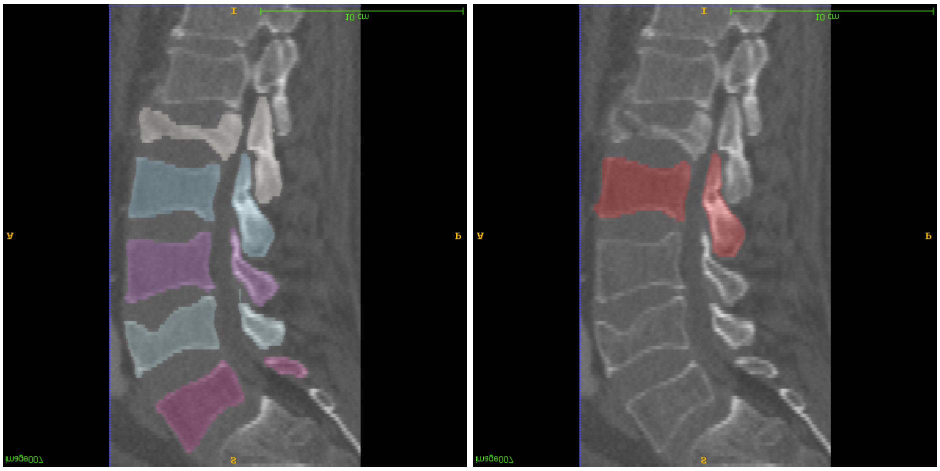
(a) Raw prediction



(b) Segmentation (Dice 0.95).

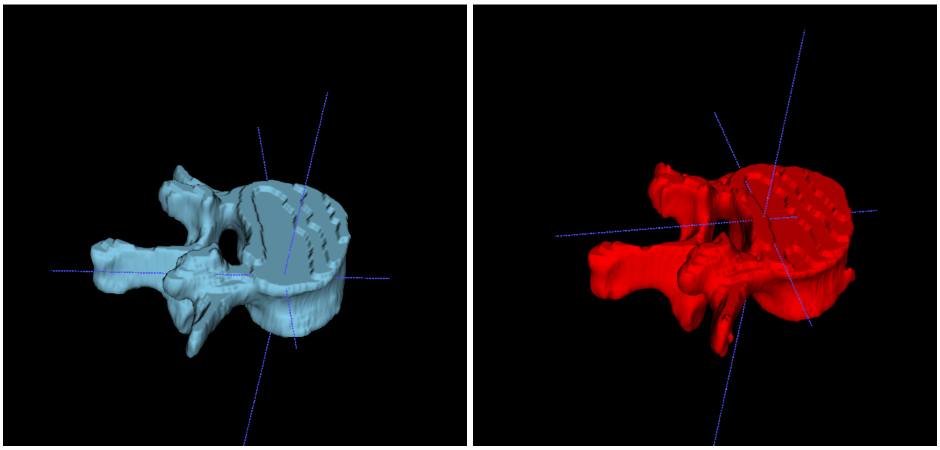
Figure 4.28: *Segmentation and reference mask for the L1 vertebrae in* image007 *with* 8 *hints provided. Note how different partitions are clearly distinguishable, as well as the spatial relationship between the point and the model confidence.*

Figures 4.29 shows a sagittal profile slice of a segmented L2 in the *image007* CT-volume. Figure 4.30 contains the full 3D segmentation of the same vertebrae.



(a) *Reference mask* **(b)** *Predicted segmentation*

Figure 4.29: *Example segmentation of a L2 vertebrae in the sagittal plane of the* image007 *CT-volume.*



(a) *Ground Truth* **(b)** *Segmentation*

Figure 4.30: *Example 3D segmentation of a vertebrae. The figure represents the same vertebrae and sample as Figure 4.29. Note how the shape is very similar to the reference mask.*

5 Discussion

This section presents discussion and analysis regarding the method proposed and the results produced in Chapter 4. The results are additionally compared in relation to the reference literature as well as some similar studies. Finally, the method and thesis work will be examined in terms of difficulty and quality.

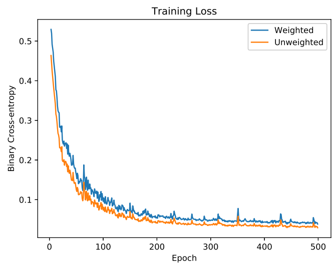
5.1 Result Discussion

This section examines the different results procured and possible explanations of what the results may entail. The section is split into roughly the same structure as in Chapter 4.

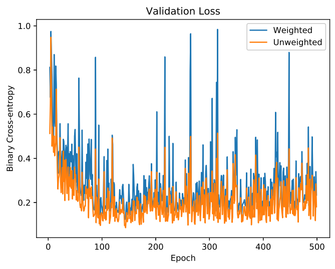
5.1.1 Model Training

Model training involved two steps, first the model was pre-trained without any hints provided by the user and all lumbar vertebraes assigned to the same segment.

The *training loss* (Figure 4.1a) follows the shape one would expect with the loss decreasing similar to an inverse-exponential until it converges. This is a good sanity-check indicating that the model manages to find some signatures in the data which can be learned. The *validation loss* (Figure 4.1b) on the other hand is not as straight-forward, some convergent trend can be observed but with a large amount of spikes and noise. This is not necessarily what would be expected but could be explained in multitude of ways. The first and most straightforward explanation is that there is a small number of validation samples which results in uncommon features such as damaged vertebraes contributing to higher variance. Additionally, looking at the result it can be observed that large part of non lumbar vertebraes, specifically T12 is segmented as well.



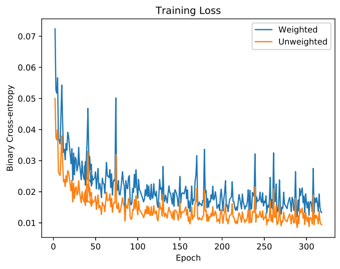
*(**a**) Training loss*



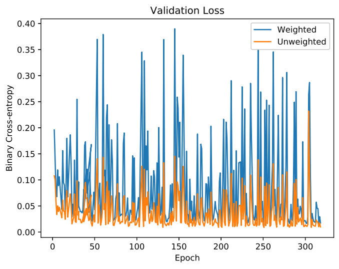
*(**b**) Validation loss*

Figure 5.1: *Copy of the training and validation loss plots for pre-training (Figure 4.1).*

Looking at the loss for interactive segmentation (Figure 4.5a and 4.5b) we note a faster convergence in the *training loss* but no clear trend in the *validation loss*, here the spikes in the validation set are significantly more pronounced. Examining how the number of points affected the result, a possible explanation is that spikes are caused by samples where few points were provided, resulting in a significantly lower score, which would translate to higher loss. The number of epochs for both pre- and interactive model fitting were initially set to 500 each, but time constraints and the non-convergence of the validation set resulted in the interactive training-phase being terminated early.



*(**a**) Training loss*



*(**b**) Validation loss*

Figure 5.2: *Copy of the training and validation loss plots for interactive segmentation (Figure 4.5).*

5.1.2 Post-processing

For the pre-training both post-processing methods performed about as well (Table 4.1, Figure 4.4) with the *Morphed* method having a slightly higher best-case. This could be explained by looking at the visual result (Figure 4.3) where a number of small connected components can be observed, these are removed by the *morphed* post-processing method.

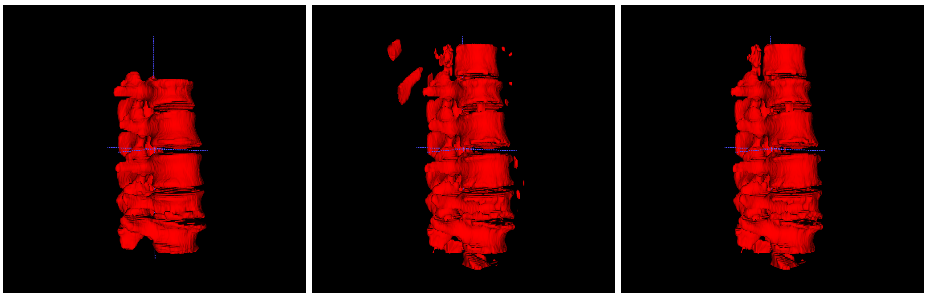

(a) Reference Mask 3D *(b) Threshold 3D* *(c) Morphed 3D*

Figure 5.3: *Copy of pre-training segmentation (Figure 4.3).*

For the interactive segmentation the post-processing methods performed slightly different (Table 4.6 and Figure 4.8). *Graphcut* and *Threshold* performed very close to each other but *Morphed* had a lower score. This is an interesting result since the pre-training would indicate that the *Morphed* would perform better than *Threshold*. Looking at the performance over foreground points (Figure 4.8) this discrepancy can be explained by the amount of foreground points. Low amount of points resulted in the connected component removal process in *Morphed* removing the entire foreground object, when multiple points are present, similar improvements over the other post-processing methods can be observed as in the case of the pre-training. Figure 5.4 shows the final segmentation of a single lumbar vertebrae.

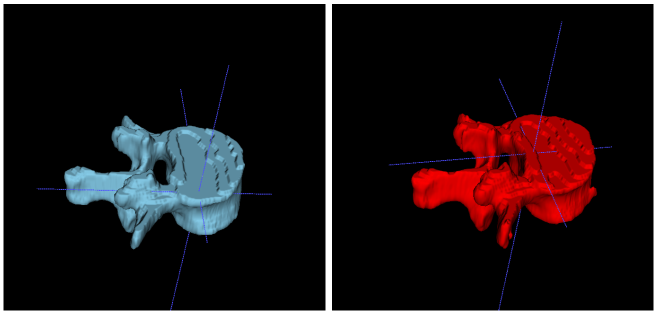

(a) Ground Truth *(b) Segmentation*

Figure 5.4: *Example 3D segmentation of a vertebrae. Note some slight bleed-out at the front (Figure 4.30).*

It would be expected that the *Graphcut* method would outperform the others as it is the most computationally complex of the methods. A possible explanation for this discrepancy could be found by observing the *Precision* column of Table 4.6, where *Graphcut* and *Threshold* have significantly higher *Precision* compared to *Morphed*. This means that about 8% more cases of false positives, with a high degree of certainty ($q_{ij} > 0.5$). *Graphcut* and *Threshold* were also very close to each other.

Visual inspection such as Figures 4.13, 4.15, 4.19, 4.23 and 4.14 would indicate that incorrect predictions often occur either right on the border of the correct object, or as isolated clusters. This would result in the *boundary term* $B(A)$ ignoring these regions during the *Graphcut* post-processing step, as $R(A)$ would be the same in the surrounding neighbourhood. As a result, the discontinuity function (eq. 2.3) causes these regions to only account for $R(A)$, which caused the incorrect predictions in the first place. A possible solution could be to modify the *region term* $R(A)$ as an ensemble method, such as combining the CNN with histogram matching or similar method to improve the reliability of the term.

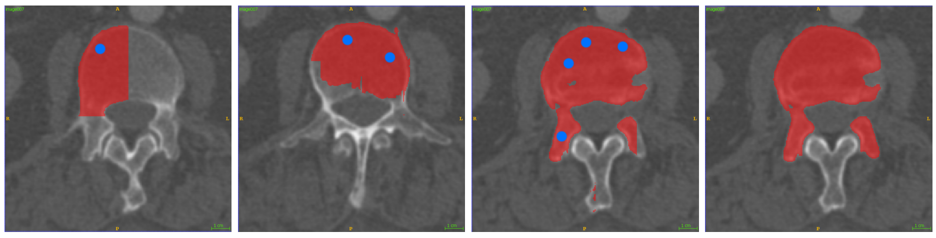
5.1.3 Point-Placement

The number of background points (Table 4.3 and Figure 4.6) does not seem to significantly affect the quality of the result, as a consequence, the placement strategies for background points does not affect the result. The number of foreground points (Table 4.2 and Figure 4.6) does on the other hand affect the quality of the result, with the quality converging from 8 points.

Possible explanations for the fact that only foreground points significantly affect the result could include properties such as obvious background voxels being easy to segment as they often consists of different tissues while there are many parts of a given volume which have the same signature as the foreground, meaning that the model has to adhere to the foreground points for context.

Looking at the effect of number of points placed in relation to the paper *N. Xu et al. 2016* [27] they needed around $6 - 8$ points in order to reach the target IoU for a number of datasets, although not volumetric medical images and a FCN instead of a U-Net. This is the same order of magnitude of points our method required to reach comparable scores.

Figure 5.5 provides a visual example of how the number of placed foreground points affect the segmentation.

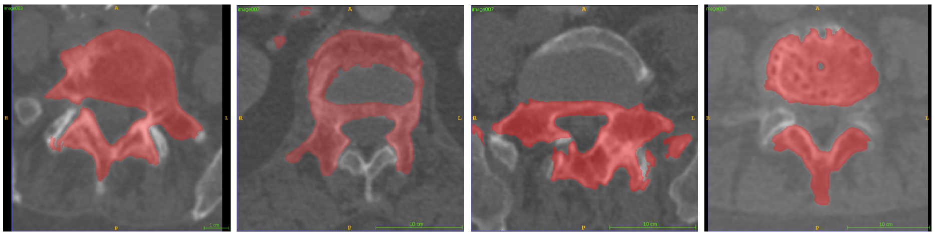


(a) *1 Point* ***(b)*** *2 Points* ***(c)*** *4 Points* ***(d)*** *32 Points*

Figure 5.5: *Visual example of segmentation results based on number of foreground points. Blue points indicate foreground placements. Note how placed points are always included in the resulting segmentation and how the volume stitching is visible, especially in 5.5a.*

As can be observed in Figures 4.25, 4.26, 4.27, and 4.28, also indicate how both the point placement and volume partitioning have a significant effect on the segmentation quality.

Figures 4.14, 4.15, 4.19 and 4.23 shows how damaged/strangely shaped vertebraes causes poorer segmentation. The above referenced figures can be seen collected in Figure 5.6.



(a) *Image003 L5* ***(b)*** *Image007 L1* ***(c)*** *Image007 L5* ***(d)*** *Image010 L4*

Figure 5.6: *Copies of Figures 4.14, 4.15, 4.19 and 4.23.*

5.1.4 Validation Samples

There were significant difference between the score of the respective validation samples, visual inspection shows that some of the vertebraes in the validation set suffered from significant deformities. Papers such as *R. Janssens et al.* [13] performed 3-fold cross validation in order to get more robust results which means that any comparison require some consideration. Due to time constraints, the cross validation was omitted from the project.

5.1.5 Vertebrae Segmentation

In comparison to *R. Janssens et al.* [13] the average score for the different vertebrae (Table 4.5 and Figure 4.7) differs slightly. Both ours and their results found that the $L5$ vertebrae produce lower scores than the others, but their results would indicate that the remaining vertebrae had similar performance. In our case it seemed that $L2$ and $L3$ were slightly easier to segment compared to the rest, perhaps explained by the samples picked for validation or that they were less affected by any ROI augmentation. Additionally they perform multi-class segmentation where the model can internalize pairs of vertebrae as anchor points for their segmentation.

5.2 Method Discussion

The method used during the project was designed based on a number of papers, some of which did not entirely explain their methods in a reproducible manner. As a result, some qualified assumptions had to be made in order to produce a concrete method. There were three primary parts of the method which lacked significant support and therefore were either motivated by alternative sources, or by factually grounded assumptions.

Comparing the result with other papers, the model reaches a score similar to some of the best methods presented (*R. Janssens et al.* [13]), which produced a *Dice score* of 0.95 compared to our 0.94 and an *Intersection over Union* of 0.92 compared to our 0.91. The aim of introducing user-provided hints into the model was to provide a more general end-to-end segmentation, which could be extended into additional anatomical areas, such as knee-joints. The result shows that this may be a possibility as we reach comparable performance with a more general method.

The first significant change in the method was the *data augmentation* (Section 3.4.2) as *R. Janssens et al.* [13] performed some additional regularization techniques, such as gray-level augmentation which was not described in the paper. The original method additionally applied random scaling of the volumes. Volume scaling was omitted during this project since the *elastic deformation* technique modifies the volume in a similar way. Additionally, there were resolution differences already present in the dataset, one of the extreme examples being image008, which only had half of the slice-resolution compared to the other volumes. A more consistent approach may be to work with the volumes in metric, as opposed to the discrete image size.

The second change is the *Morphed* post-processing method. *R. Janssens et al.* [13] did not specify what modifications were involved, resulting in the one used during this thesis being taken from a different paper (*A. Sekuboyina et al. 2017* [24]) which provided a base for the original method.

Initially, the segmentation model developed during this thesis downscaled the volume instead of extracting overlapping sub-volumes. This approach was motivated as a way to produce a larger receptive field, but significantly reduced the segmentation quality. This approach was abandoned both for the reduced resolution of the output and the inconsistency of comparing down-scaled results with alternative methods.

During the evaluation of the segmentation model the dataset was only split into two parts and no cross-validation was performed. This makes the results not as reliable as the reference studies but the concession was made as the time required for model training would make it hard to fit within the allotted thesis time-frame.

In Chapter 4, direct comparisons were limited to the papers providing the basis for the method. Methods using traditional segmentation techniques such as the paper by *M. Krčah et al. 2011* [17] reaches impressive results for general segmentation of bone tissue using a *graph-cut* based method. The *Probabilistic Watershed Transform* presented by *W. Shadid and A. Willis 2013* [25] managed a dice score of 0.94 ± 0.04 which is similar to ours given enough user-provided points. Worth pointing out is that their method worked on slices and therefore did not have the context of the entire volumetric data at hand.

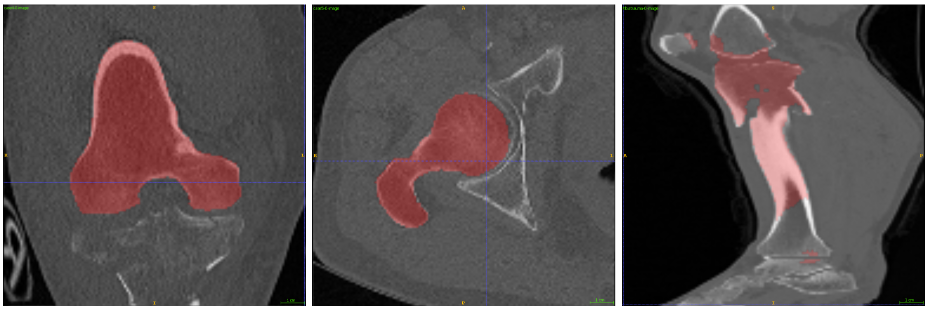
5.3 Thesis Reflection

Deep learning methods for segmentation, classification and regression has seen enormous progress within the field of medical imaging, but the task of segmenting bone-tissue is in the clinical world often done through traditional digital image processing and gray-level segmentation. At the start of this thesis, this seemed to be caused by a reliability on old pipelines and traditional methods, which I am certain is still true to some extent. What I discovered during this thesis was that deep learning based semantic segmentation algorithms are highly dependent on the data provided for training, but clinical use often consists of segmentation tasks on wildly varying anatomies and different quality scans/intensity ranges. As a result there is a high expectation for reliability and the desire to be able to provide worst-case guarantees, when considering segmentation methods. As no large-scale dataset for the domain of segmenting bones in CT-volumes is available at the present, these methods may struggle to reach the same reliability traditional methods provides.

The time required to build and test the model was significantly underestimated and required at least two complete re-works, as concessions for the original hardware had to be accounted for. Once I got access to a better GPU the implementation could go back to focusing on reproducing the methods presented in the reference literature.

5.3.1 Additional Dataset

From the start of the thesis a dataset of 10 CT-volumes, provided by Region Östergötland, of knee-joints was designated for training/evaluation. Significant time was put on segmenting this set of CT-scan by hand which in the end turned out to be too small to produce reliable results, resulting in the model memorizing the training volumes and performing poorly on any other scans. Figure 5.7 shows the relative quality between the training and validation sets for the private dataset. Note how the segmentation of the training set is close to perfect while the validation set is unusable.



(a) *Training sample* *(b)* *Training sample* *(c)* *Validation sample*

Figure 5.7: *Example segmentations produced for the private dataset.*

Because of the inconsistent results, the project moved to using the lumbar vertebrae dataset which provided the added benefit of being open and providing a direct comparisons with other studies.

6 Conclusions and Future Work

This section presents the final summary of the project and what can be improved/explored in the future.

6.1 Conclusions

The aim for this thesis was to look at the performance of an interactive CNN-based segmentation model, with the added goal of examining how user-provided hints affect the result. In conclusion, the model achieves results comparable with the reference studies, perhaps meriting further studies, as *N. Xu et al. 2016* [27], and the follow-up *N. Xu et al. 2017* [28] shows that interactive semantic segmentation is a field with much potential. But for the implementation examined in the thesis, the model lacks the verifiability of classical segmentation methods, resulting in the model being too unreliable to be used independently for clinical tasks, as there is no minimal guaranteed quality.

6.1.1 Research Questions

Below are the research questions which provided the foundation for this thesis with the answers found during the project:

1. **How well does the model perform segmentation of bone-tissue in regards to *Intersection over Union*?**

   When adequate number of hints (≥ 8) are provided by the user we achieve a IoU of 0.91. With a random number of segmentation hints, we achieve a mean IoU of 0.72.

2. **How well does the model perform segmentation of bone-tissue in regards to *Dice Score*?**

When adequate number of hints (≥ 8) are provided by the user we achieve a Dice score of 0.95. With a random number of segmentation hints, we achieve a mean Dice score of 0.79.

3. **Does the number of user-provided points affect the segmentation performance and in what way?**

   The number of points does affect the result. The number of foreground points affect the result where the segmentation quality converges above 8 points. Background points does not significantly affect the result but some percentage improvement can be seen up to 8 points where the improvement stagnates. Placement strategy for the background points does not affect the result at all.

6.2 Future Work

Many avenues were explored during this project but did not make the cut for the report. Below are some of these topics which may merit further studies:

Is it possible to get better/equivalent performance on other medical imaging segmentation problems? This thesis has exclusively looked at segmentation of bone tissue in CT-scans, but the procedure developed can be directly applied to any other segmentation problem within medical imaging. It may interesting to examine if the model is evaluated on different problems, such as organ, or other soft-tissue segmentation problem.

Could the model be used in ensemble with other interactive segmentation models to produce more reliable results? The model produces on average very good results, but it lacks worst-case guarantees and any incorrect segmentation is hard to infer from the model. Therefore, it may be interesting to examine if the model can be used as a weak classifier in ensemble with other models, which are better understood.

Does different post-processing methods produce more desirable results? During this thesis, we examined three different post-processing methods, these showed varying properties and results, perhaps there are other refinement methods which produces higher quality results.

Can the distance maps be generated through autonomous methods? The manual work involved in this method places it in a direct disadvantage compared to the related methods, which are completely automatic. Therefore, it would be interesting to examine way to automatically produce the distance-maps.

Appendix

A
Model Architecture

This appendix presents the implementation details for the segmentation model used during this project. Figure A.1 shows the U-Net segmentation model and Table A.1 shows the corresponding model parameters. Note that concatenation layers concatenate the output from the *Down* group with the corresponding *Up* group on the channel axis.

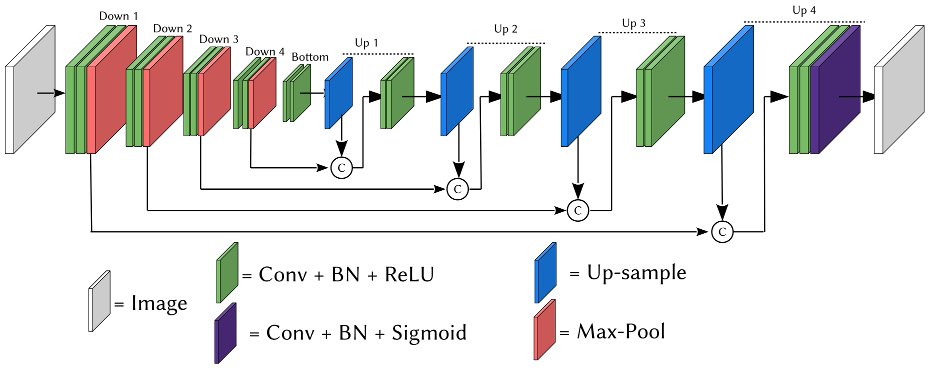


Figure A.1: *Annotated model reference*

Layers						
Group	Layer Type	# Size	#Nodes	Kernel	Activation	Padding
Input		160 x 128 x 96	3			
Down 1	Conv + BN	160 x 128 x 96	16	3	ReLU	Same
	Conv + BN	160 x 128 x 96	32	3	ReLU	Same
	Max-Pool	80 x 64 x 48	32	2		
Down 2	Conv + BN	80 x 64 x 48	32	3	ReLU	Same
	Conv + BN	80 x 64 x 48	64	3	ReLU	Same
	Max-Pool	40 x 32 x 24	64	2		
Down 3	Conv + BN	40 x 32 x 24	64	3	ReLU	Same
	Conv + BN	40 x 32 x 24	128	3	ReLU	Same
	Max-Pool	20 x 16 x 12	128	2		
Down 4	Conv + BN	20 x 16 x 12	128	3	ReLU	Same
	Conv + BN	20 x 16 x 12	256	3	ReLU	Same
	Max-Pool	10 x 8 x 6	256	2		
Bottom	Conv + BN	10 x 8 x 6	256	3	ReLU	Same
	Conv + BN	10 x 8 x 6	512	3	ReLU	Same
Up 1	Up-Sample	20 x 16 x 12	512	2		
	Concatenate					
	Conv + BN	20 x 16 x 12	256	3	ReLU	Same
	Conv + BN	20 x 16 x 12	256	3	ReLU	Same
Up 2	Up-Sample	40 x 32 x 24	256	2		
	Concatenate					
	Conv + BN	40 x 32 x 24	128	3	ReLU	Same
	Conv + BN	40 x 32 x 24	128	3	ReLU	Same
Up 3	Up-Sample	80 x 64 x 48	128	2		
	Concatenate					
	Conv + BN	80 x 64 x 48	64	3	ReLU	Same
	Conv + BN	80 x 64 x 48	64	3	ReLU	Same
Up 4	Up-Sample	160 x 128 x 96	64	2		
	Concatenate					
	Conv + BN	160 x 128 x 96	32	3	ReLU	Same
	Conv + BN	160 x 128 x 96	32	3	ReLU	Same
	Conv + BN	160 x 128 x 96	1	1	Sigmoid	Valid

Table A.1: *Grouping and hyper-parameters for the U-Net Segmentation Model.*

B Convolution as Matrix Multiplication

In practice, the convolution operation is implemented as a matrix multiplication for computation efficiency. The operation is represented by a *Toeplitz matrix* denoting the kernel multiplied with an vector representing the input. The following description is based on the explanation by *V. Dumoulin and F. Visin 2016* [7] and *Z. Zhang 2016* [29].

$$y = f * g = \underbrace{\begin{bmatrix} g_1 & 0 & \cdots & 0 & 0 \\ g_2 & g_1 & \cdots & \vdots & \vdots \\ g_3 & g_2 & \cdots & 0 & 0 \\ \cdots & g_3 & \cdots & g_1 & 0 \\ g_{m-1} & \vdots & \cdots & g_2 & g_1 \\ g_m & g_{m-1} & \vdots & \vdots & g_2 \\ 0 & g_m & \cdots & g_{m-2} & \vdots \\ 0 & 0 & \cdots & g_{m-1} & g_{m-2} \\ 0 & 0 & 0 & \cdots & g_m \end{bmatrix}}_{\mathbf{H}} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}. \tag{B.1}$$

When extending the *Toeplitz matrix* for 2D convolutions it is referred to as a *doubly block circulant matrix* i.e a *block circulant matrix* consisting of *circulant topelitz matrices* (H_i) generated from the rows in the kernel g. Below is an example of how a convolution in 2D can be represented as a matrix multiplication problem.

$$f = \begin{bmatrix} f_{0,0} & f_{0,1} & f_{0,2} \\ f_{1,0} & f_{1,1} & f_{1,2} \end{bmatrix} \tag{B.2a}$$

$$g = \begin{bmatrix} 0 & 0 & 0 & 0 \\ g_{0,0} & g_{1,0} & 0 & 0 \\ g_{0,1} & g_{1,1} & 0 & 0 \end{bmatrix} \tag{B.2b}$$

Here, $\mathbf{H}_i$ is defined from the kernel, as seen in Figure B.1.

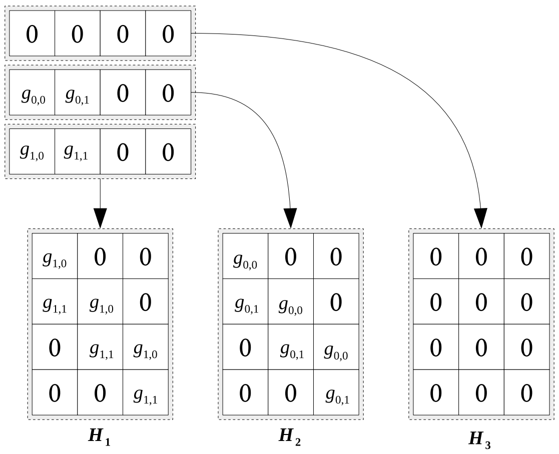


Figure B.1: *Conversion from the kernel g into Toeplitz matrices.*

which is then combined into the block matrix $\mathbf{H}$ by

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 & \mathbf{H}_3 \\ \mathbf{H}_2 & \mathbf{H}_1 \\ \mathbf{H}_3 & \mathbf{H}_2 \end{bmatrix}, \tag{B.3}$$

which defines the final operation as

$$y = \mathbf{H}f = \left[\begin{array}{ccc|ccc} g_{1,0} & 0 & 0 & 0 & 0 & 0 \\ g_{1,1} & g_{1,0} & 0 & 0 & 0 & 0 \\ 0 & g_{1,1} & g_{1,0} & 0 & 0 & 0 \\ 0 & 0 & g_{1,1} & 0 & 0 & 0 \\ \hline g_{0,0} & 0 & 0 & g_{1,0} & 0 & 0 \\ g_{0,1} & g_{0,0} & 0 & g_{1,1} & g_{1,0} & 0 \\ 0 & g_{0,1} & g_{0,0} & 0 & g_{1,1} & g_{1,0} \\ 0 & 0 & g_{0,1} & 0 & 0 & g_{1,1} \\ \hline 0 & 0 & 0 & g_{0,0} & 0 & 0 \\ 0 & 0 & 0 & g_{0,1} & g_{0,0} & 0 \\ 0 & 0 & 0 & 0 & g_{0,1} & g_{0,0} \\ 0 & 0 & 0 & 0 & 0 & g_{0,1} \end{array}\right] \begin{bmatrix} f_{1,0} \\ f_{1,1} \\ f_{1,2} \\ f_{0,0} \\ f_{0,1} \\ f_{0,2} \end{bmatrix}. \tag{B.4}$$

Note that this representation means we get the best of both worlds, the efficiency of the matrix multiplication with the weight-sharing of the convolutional operator. Additionally, the matrix representation of an convolution provides two desirable properties. The first is that the backward-pass can be computed by multiplying the *loss* with $\mathbf{H}^T$. The second property is that it allows for layers representing the *transposed convolution* by combining $\mathbf{H}^T$ with a strided kernel to produce feature maps larger than the input.

Bibliography

[1] Segmentation and classification of fractured vertebrae. http://lit.fe.uni-lj.si/xVertSeg/. Accessed: 2019-03-06. Cited on pages 17 and 29.

[2] Ignacio Arganda-Carreras, Srinivas C. Turaga, Daniel R. Berger, Dan Cireşan, Alessandro Giusti, Luca M. Gambardella, Jürgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M. Buhmann, Ting Liu, Mojtaba Seyedhosseini, Tolga Tasdizen, Lee Kamentsky, Radim Burget, Vaclav Uher, Xiao Tan, Changming Sun, Tuan D. Pham, Erhan Bas, Mustafa G. Uzunbas, Albert Cardona, Johannes Schindelin, and H. Sebastian Seung. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9:142, 2015. ISSN 1662-5129. doi: 10.3389/fnana.2015.00142. URL https://www.frontiersin.org/article/10.3389/fnana.2015.00142. Cited on pages 2 and 13.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994. Cited on page 10.

[4] Yuri Y Boykov and M-P Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112. IEEE, 2001. Cited on pages 2, 5, 6, 7, and 28.

[5] Travers Ching, Daniel S. Himmelstein, Brett K. Beaulieu-Jones, Alexandr A. Kalinin, Brian T. Do, Gregory P. Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M. Hoffman, Wei Xie, Gail L. Rosen, Benjamin J. Lengerich, Johnny Israeli, Jack Lanchantin, Stephen Woloszynek, Anne E. Carpenter, Avanti Shrikumar, Jinbo Xu, Evan M. Cofer, Christopher A. Lavender, Srinivas C. Turaga, Amr M. Alexandari, Zhiyong Lu, David J. Harris, Dave DeCaprio, Yanjun Qi, Anshul Kundaje, Yifan Peng, Laura K. Wiley, Marwin H. S. Segler, Simina M. Boca, S. Joshua Swamidass, Austin Huang, Anthony Gitter, and Casey S. Greene. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018. doi: 10.1098/rsif.2017.

0387. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsif.2017.0387. Cited on page 2.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. Cited on page 30.

[7] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016. Cited on page 71.

[8] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. Cited on page 15.

[9] Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2015. Cited on page 6.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. Cited on pages 8, 11, and 12.

[11] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. *arXiv:1606.06650 [cs]*, June 2016. URL http://arxiv.org/abs/1606.06650. arXiv: 1606.06650. Cited on pages 13 and 15.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167. Cited on page 15.

[13] Rens Janssens, Guodong Zeng, and Guoyan Zheng. Fully Automatic Segmentation of Lumbar Vertebrae from CT Images using Cascaded 3d Fully Convolutional Networks. *arXiv:1712.01509 [cs]*, December 2017. URL http://arxiv.org/abs/1712.01509. Cited on pages 18, 25, 61, and 62.

[14] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009. Cited on page 10.

[15] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015. Cited on page 2.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`. Cited on page 20.

[17] Marcel Krčah, Gábor Székely, and Rémi Blanc. Fully automatic and fast segmentation of the femur bone from 3d-ct images with no shape prior. In *Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on*, pages 2087–2090. IEEE, 2011. Cited on page 62.

[18] Yann LeCun and Corinna Cortes. 2010. Cited on page 8.

[19] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. Cited on page 13.

[20] Alexander Selvikvåg Lundervold and Arvid Lundervold. An overview of deep learning in medical imaging focusing on MRI. *CoRR*, abs/1811.10052, 2018. URL `http://arxiv.org/abs/1811.10052`. Cited on page 2.

[21] Eric N Mortensen and William A Barrett. Interactive segmentation with intelligent scissors. *Graphical models and image processing*, 60(5):349–384, 1998. Cited on page 6.

[22] Ken Perlin. An image synthesizer. 19:287–296, 1985. Cited on page 23.

[23] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. URL `http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a`. (available on arXiv:1505.04597 [cs.CV]). Cited on pages 13, 19, and 22.

[24] Anjany Sekuboyina, Alexander Valentinitsch, Jan S. Kirschke, and Bjoern H. Menze. A localisation-segmentation approach for multi-label annotation of lumbar vertebrae using deep nets. *CoRR*, abs/1703.04347, 2017. URL `http://arxiv.org/abs/1703.04347`. Cited on pages 22, 25, 27, and 62.

[25] Waseem Shadid and Andrew Willis. Bone fragment segmentation from 3d ct imagery using the probabilistic watershed transform. In *Southeastcon, 2013 Proceedings of IEEE*, pages 1–8. IEEE, 2013. Cited on pages 1 and 62.

[26] Chunliang Wang and Orjan Smedby. An automatic seeding method for coronary artery segmentation and skeletonization in cta. *The Insight Journal*, 43: 1–8, 2008. Cited on page 1.

[27] Ning Xu, Brian L. Price, Scott Cohen, Jimei Yang, and Thomas S. Huang. Deep interactive object selection. *CoRR*, abs/1603.04042, 2016. URL `http://arxiv.org/abs/1603.04042`. Cited on pages 2, 15, 27, 28, 35, 60, and 65.

[28] Ning Xu, Brian L. Price, Scott Cohen, and Thomas S. Huang. Deep image matting. *CoRR*, abs/1703.03872, 2017. URL `http://arxiv.org/abs/1703.03872`. Cited on page 65.

[29] Zhifei Zhang. Derivation of backpropagation in convolutional neural network (cnn). 2016. Cited on page 71.

[30] Yi-Tong Zhou and Rama Chellappa. Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks*, volume 1998, pages 71–78, 1988. Cited on page 11.