Automated bone segmentation in computed tomography using deep learning with distance maps

Julius Åkesson

Master's thesis
2020:E40

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Abstract

3D-models of bone structures from computed tomography (CT) data can be used for surgical planning, education and a wide range of other purposes. They can also be used in both digital and 3D-printed format. The currently used process to obtain such models consists of a combination of thresholding, morphological operations and manual adjustments. This can be very time-consuming. Deep Learning (DL) can be used to automatically segment organs such as bones in CT data, and can thus be used to automate such a process. Emerging data also suggests that including distance maps (DM) of ground truth segmentation masks when using DL for segmentation problems can yield improved performance. In this thesis project, a well-known neural network architecture for segmentation was modified in three different ways to include DM during training and prediction. The three modifications were inspired by three types of methods for DM-inclusion used in previous work, but simplified. Estimates of the generalization performances of the three modifications and the network in its original state were compared using both an in-house dataset and a publicly available dataset. The results showed that at least one of the modified networks outperformed the network in its original state in all the tested cases. This indicates that DL-methods for performing bone structure segmentation in CT data could benefit from an inclusion of DM during training and prediction. This was especially indicated when using a multi-task network to perform both segmentation and DM-regression in parallel. However, these results have to be further validated.

Acknowledgements

I would like to thank my supervisor Einar Heiberg for his guidance and provided discussions during the course of this project. I would also like to thank the Lund Cardiac MR Group for help with both technical and theoretical solutions.

Contents

1 Introduction

This introductory part of the report first describes the background and motivation behind the project (Section 1.1). Then, the aims and main research questions of the project are presented (Section 1.2), followed by a discussion about how the project was delimited (Section 1.3).

1.1 Background and motivation

3D-models of bone structures from CT-data can be used in both digital and 3D-printed format for a wide range of purposes. They can be used by physicians to perform surgical planning, which can lead to safer and more successful surgeries. Further, they can also be used for other purposes within the medical field, such as for education and as anatomical simulators for practicing surgery.

Currently, the process of going from a Hounsfield-valued CT-volume to a segmented 3D-model of the depicted bone structure is highly time-consuming. The procedure to obtain an adequately segmented model involves a combination of thresholding, morphological operations and manual adjustments. There are many possible approaches that one can use to reduce the time for this process. *Deep Learning* (DL) using Convolutional Neural Networks (CNN) has previously shown promising results in the area of general bone structure segmentation in CT-data [1] [2].

Recent studies in the areas of medical and non-medical image segmentation have also shown that including the *Distance Transform* (DT) of ground truth segmentation masks, also known as *Distance Maps* (DM), during training and prediction for CNN in various ways can improve segmentation results. The observed improvements include increased performance measured in several common metrics for segmentation performance in both 3D and 2D and for both multi- and single-organ segmentation, as well as non-medical segmentation problems [3] [4] [5] [6]. Hypotheses regarding the benefits of the inclusion of DM include that it helps *regularize* the training process by incorporating more *spatial* information [6] as well as more *shape* information about the target objects [4].

To the best of the author's knowledge, there exists no previous work assessing the benefits of including DM during training and prediction for CNN-based general (non-specific) bone segmentation. This was therefore performed in this project. There however exists previous work comparing different types of DM-inclusion on other segmentation problems. The motivation behind performing this assessment was partly that it could shed *further* light on what types of problems can benefit from the inclusion of DM. It was also partly to see if this simple method could prove to be helpful when applying a standard DL-method for segmentation to the difficult problem of segmenting bones in CT-data in practical implementations.

1.2 Aims

The principal aim of this project was to assess the benefits of including DM during training and prediction when performing bone segmentation in CT-data using a well-known CNN. The underlying goals of this were to evaluate the possibility to use such a method in a practical framework for bone segmentation as well as shedding more light on what type of segmentation problems can benefit from DM-inclusion. This could be done simply by observing the results obtained on this specific problem. Since this was done specifically by modifying a well-known CNN in *three* different simple ways inspired by general methods for DM-inclusion used in previous work, the aim was also to investigate which of these methods could yield the best performance.

To clarify, the main research question that this project aimed to shed more light on by the performed evaluations was thus whether simple modifications of a well-known CNN to include DM during training and prediction could help improve its performance for bone segmentation in CT-data. A secondary research question was simply *which* method for including DM could yield the best performance on the given bone segmentation problem.

1.3 Delimitations

The delimitations of this project include a restriction to the use of a single backbone network structure, a 2D *U-Net* (see Section 3.3), taking 2D image-data as input. 3D CT-volumes were thus seen as a set of 2D-slices throughout the project. This delimitation was set partly because existing 3D DL-methods require a training process that is highly computationally expensive [10], something that is not compatible with the limited amount of time and computational resources that were given. The use of a 2D U-Net allowed an easier assessment of the inclusion of DM. Another reason for this delimitation was that benchmark results on a publicly available dataset for bone segmentation were available using a similar 2D network structure [1] [7]. This allowed loose comparisons between results to be done.

This thesis project was also delimited to only assess the benefits of DM-inclusion. An initial additional aim of this project was to develop a functional tool for bone segmentation. This had to be reconsidered as a consequence of the limited amount of available data. An in-house dataset was created from 5 CT-volumes from SUS (Skåne University Hospital) and a small public dataset was found online (see Sections 4.2.1 and 4.2.2). These datasets are relatively small for DL, and do not fully represent all types of bones. This limits the possibilites of developing and evaluating the true robustness and functionality of a tool for general bone segmentation. Further limitations of this project are discussed in Section 6.1.

2 Previous work

This section gives an overview of some of the previous work that is related to the work of this project and that has contributed in some way to the used methodology. Firstly, a few recent methods for including DM in CNN are presented and roughly divided into three different categories (Section 2.1). Lastly, an overview of previous projects that have performed bone segmentation in CT-data is given (Section 2.2).

2.1 Distance maps in deep learning-based segmentation

DL has been widely used for medical image segmentation during the past few years. Emerging data also suggests that including DM in various ways during training and prediction can yield improvements to existing DL-methods for segmentation [3] [5] [6]. The recent work on DL-based segmentation that involves DM is in this project roughly divided into three categories. Each category could potentially be useful for the problem of bone segmentation. In the first category, *image-to-image regression* is performed to map an input image to a DM that is then internally post-processed in the network to yield a segmentation mask. The second category adds the regression of a DM as a *complementary task* to the task of learning a segmentation mask, creating multitask network. The third category performs pure image-to-image regression and outputs the DM without internal post-processing, leaving the need to post-process the DM-output in order to obtain a segmentation mask.

In the first category there exists the recent work of Xue et al. (2019). They propose a method where *signed* DM (SDM) are generated by a 3D U-Net and internally post-processed in the network using an approximated threshold operation (a modified sigmoid function) to create an output segmentation map. They include an intermediate output of the SDM in the used loss function. This showed improvements in most performance metrics as well as in smoothness and shape-preservation for both single- and multi-organ segmentation problems in 3D [3]. Similarly, Audebert et al. (2019) generates a DM, gives this as an intermediate output to be used in the loss function, and then performs internal operations in the network to output a segmentation mask. They apply their network to aerial images [6].

As a part of the second category, Navarro et al. (2019) observed that a multitask network with a backbone similar to the U-Net, using DM-regression as a complementary task to learning a segmentation mask could improve their segmentation results compared to baseline results from their U-Net on a 2D multi-organ segmentation problem. However, they suggest a final network using another auxiliary task performing *contour detection* as an addition to the DM-regression [4]. Similarly, Dangi et al. (2019) performed DM-regression as an auxiliary task to segmentation during training, using the DM output in the loss function to regularize the learning process. This yielded improvements in the segmentation results on 2D binary and mutli-class organ segmentation problems [5].

A model corresponding to the third category was used in preliminary experiments in [6], indicating that pure image-to-image regression of DM could not improve their results compared to baseline methods. Nevertheless, in the work of Naylor et al. (2018), 2D DM-regression was succesfully used to perform the segmentation of nuclei in histopathological data [8].

From the above mentioned projects, one can see indications that including DM in DL by using methods from these three categories seems to have potential to yield improved performance on segmentation problems. There are several hypotheses as to *why* DM could be beneficial in a training process. According to [3], DM (SDM) encodes richer information about structural features of the target segmentation objects by embedding e.g. contours in a higher dimensional (sub-pixel) space. They also state that small changes in a DM representation affects the values of many pixels, globally, while in a binary mask, only local changes occur when small changes are made. They state that learning this representation could lead to more continuity in the segmented shapes [3]. In [6], it is (roughly) stated that the use of DM assists the network in learning *spatial structures* of the segmentation maps, and observes that an inclusion of DM forces networks to better learn segmented objects that are closed shapes, giving sharper boundaries as well less holes in the objects. According to [4], including DM helps inferring geometric shape properties of the target segmentation objects into the learning.

This project aimed to see if the problem of segmenting bone structures in CT-images, which consist of many different shapes and sizes, could also benefit from this. Modifications corresponding to each of these three categories were performed and tested during the course of this project, especially inspired by [3], [4] and [8], but simplified. The implementations of these simple modifications are explained in detail in Sections 4.3.2-4.3.4. It should be noted that a similar study on how DM can boost segmentation CNNs for other problems than bone segmentation was released during this project [9]. The results from this study were however not taken into account in this project.

2.2 Bone segmentation in CT-images

The specific segmentation problem that this project investigates is the segmentation of bone structures in CT-data. This is done with the specific method of using CNN. According to a recent study [10], the segmentation of target objects with large differences in appearances like irregular shapes, sizes and positions is one of the big difficulties for medical image segmentation when using DL.

Nevertheless, this specific problem has been approached in previous work by using both 2D- and 3D-approaches. Klein et al. (2018) applied a slightly modified version of a U-Net on transversal 2D slices to segment bone structures in CT-data. They also tested a pseudo-3D approach where the network is trained and tested on transversal, coronal and sagittal slices, without showing improvements to their results. They trained and tested their network on a publicly available dataset [7], obtaining results that exceeded the results of previously tested methods (that were not based on DL) on the same dataset. This dataset is also used in this project. This was the most similar method to the one used in this project.

Similar problems have also been solved with DL using other approaches. Kvam et al. (2018) performed 3D bone-segmentation of full-body CT-scans of pigs by performing several 2D segmentation problems solved by using a U-Net [11]. Further, a 3D-approach for segmenting bones in *dual energy CT-images* (where CT-images of the same structure are obtained at different energies) was proposed by Sànchez et al. (2020). Here, two 3D CT volumes of the same structure obtained by using different energies were inserted into a slightly modified version of the 3D U-Net [21] for segmentation of bone structures [2]. These types of solutions were not used during this project.

3 Theory

This section describes the theoretical aspects behind the technical implementations that were used in this project. It also gives a theoretical background for the data that was used and the challenges that come with it. First, *Distance transforms* are defined and discussed (Section 3.1). Then, the basic concepts of *Deep learning* that were used in this project are explained (Section 3.2). In this section, some concepts are explained in detail, while other concepts that are less important are given a more synoptic explanation. Further, the concepts behind the *U-Net* are explained (Section 3.3) and the used performance measures of the project are defined and discussed (Section 3.4). After this, some used concepts for *model comparison* that were applied during testing are presented (Section 3.5). Finally, a brief explanation behind the characteristics and challenges of CT-data is given (Section 3.6).

3.1 Distance transforms

A distance transform (DT) is an alternative way to represent a digital image. This is most commonly used to transform *binary* images. In the DT of a binary image, also called a *distance map* (DM), each pixel contains the value of the distance between the pixel itself and the closest pixel that contains the value 1 [12]. There are several different types of *distance functions* that can be used to calculate the distances in a DT of a binary image. The two types that were considered in this project were the Euclidean Distance Transform (EDT) and the Chessboard Distance Transform (CDT).

For a more mathematical description, consider the binary image $b(x, y)$ as a function describing the elements of a binary image matrix, for which a *feature point*, a point describing an object, or *foreground* in the binary image matrix, is defined as $b(x, y) = 1$ and a non-feature point is defined as $b(x, y) = 0$. For a point (x, y), the nearest feature point to this can be given by the *nearest feature transform* of $b(x, y)$, defined as

$$[n_x(x, y), n_y(x, y)] \overset{\text{def}}{=} [x', y'] : b(x', y') = 1 \textit{ closest to } (x, y), \textit{ given } D \tag{1}$$

where $D = D[(x, y), (x', y')]$ is a *distance function* that gives the distance between two points (x, y) and (x', y') using some distance measure [12]. The notation states that $[x', y']$ is the feature point closest to (x, y) measured by D. The *distance transform* $d(x, y)$ of $b(x, y)$ for a point (x, y) can then be defined as

$$d(x, y) \overset{\text{def}}{=} D[(x, y), (n_x(x, y), n_y(x, y))] \tag{2}$$

for some distance function D [12], namely the distance D between the point and its closest feature point. The Euclidean distance function D_{EDT} that defines the EDT can be given using above notation as

$$D_{EDT}[(x, y), (n_x(x, y), n_y(x, y))] = \sqrt{(x - n_x(x, y))^2 + (y - n_y(x, y))^2}. \quad (3)$$

The Chessboard distance function (also known as the *Chebyshev* distance function) D_{CDT} that defines the CDT, uses an 8-connected neighbourhood for each pixel (as in a chessboard) [23], and can be given as

$$D_{CDT}[(x, y), (n_x(x, y), n_y(x, y))] = max(|x - n_x(x, y)|, |y - n_y(x, y)|). \quad (4)$$

In this project, the EDT was calculated using the fast algorithm described in Maurer et al. (2003) [24]. The CDT was given by using the sequential scanning algorithm given in Rosenfeld et al. (1966) [13]. In both cases, the built in MATLAB-function *bwdist* was used [23]. Examples of these two DT inversely applied to a binary image can be seen in Figure 1. The DT applied to the *inverse* of the binary image seen to the left describes the distance to the nearest background pixel for each object pixel. This can be called a *foreground* DM. This was the only type of DM used in this project.

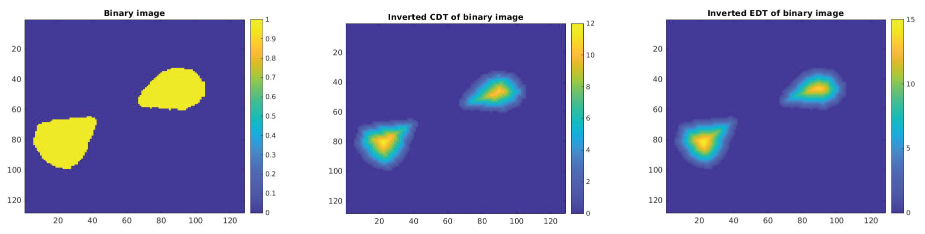

Fig. 1: This figure shows an example of a binary image (left). This binary image was inverted and transformed using the Chessboard Distance Transform (middle) and the inverted Euclidean Distance Transform (right).

Different *truncations* of the used DT:s were also tested during this project. A truncation is an upper distance limit that is set for the DT. Pixel distances of larger magnitude than this are clipped to this maximum value.

Amongst the mentioned projects in Section 2.1 that have included DM in DL for segmentation, the DM were processed in different ways. Some used *signed* DM [3] [6] and others used unsigned DM [8]. Signed DM additionally uses signed pixel values to indicate whether the pixel is inside or outside an object of interest. Some used the EDT [4] [6], some used the CDT [8] and others used a different DT of choice [3]. Some used truncation [6] and some used max-min normalization of the DM [3]. The deciding factor for choosing the DM-representation was in the case of this thesis that an integer representation such as is given by the CDT would simplify the storage and the processing of the data before being inserted into the networks, since this representation can be saved and treated as a grayscale image. The CDT, that yields values in discrete integer pixel units, was thus the DT of choice throughout the project. Truncation was used, in order to limit the numerical difference between DM that represents large and small objects.

3.2 Deep learning

3.2.1 Feed-forward neural networks

The methodology for solving the bone segmentation problem in this project was exclusively based on *deep learning*. This is a subdomain of *machine learning* that focuses on the use of *neural networks* (NN). A basic type of NN are *feed-forward neural networks* (FFNN). Such networks are characterized by not containing any feedback connections [14]. This is also the only type of network that was used in this project. The main objective of such a network is simply to perform a mapping between an input $\mathbf{x}$ and an output $\mathbf{y}$ by approximating a specific function $f_{true}(\mathbf{x})$. The network that performs this mapping can be seen as a *parametrized* function $\mathbf{y} = f(\mathbf{x}; \Omega)$, where Ω are learnable parameters of the network [14].

The target function f_{true} that the network aims to approximate through acquiring good values for the parameters Ω can be a function that performs a wide range of tasks. Two types of tasks were performed by NN in this project. The first one was *classification*. The classification performed in this project was done pixel-wise on an input image. The output of the NN for this task was an image of the same dimensions as the input image, but where each pixel has been assigned a class *label*. This operation is also known as *segmentation*, and the output image with class labels is known as a *segmentation mask*. The segmentation task in this project was binary, meaning that each pixel in the resulting segmentation mask was labeled as one of two possible classes, either bone (foreground) or background.

The second task performed in this project was *regression*. This was done in the form of *image-to-image* regression, where the output of the network is again an image of the same dimensions as the input image, but where each pixel is instead assigned a continuous, non-discrete value. In this case, these values corresponded to the values of a DM. The target function f_{true} was for this regression problem a function that generates a DM of the bone structures directly from an input image.

A FFNN generally consists of an assembly of *layers* that are connected in a specific way [14]. A layer can be seen as a sub-function of the entire parametrized network function f that either acts upon the input $\mathbf{x}$ (this is done by the *input layer*), or on the output of previous layers in the network. These yield outputs that are either used as inputs to other layers or as the output of the network (this is done by the *output layer*). The layers in between the input layer and the output layer are called *hidden layers* [14]. A principal model of a possible FFNN with two hidden layers can be seen in Figure 2. This model was drawn by the author of this report, but inspired by how a general FFNN is often presented in Deep Learning-literature. This figure illustrates that a layer is a collection of many parallell *units* [14]. It is also an example of a fully-connected network, since each unit is connected to every unit in the preceeding layer. The network used in this project contained other possible connectivites for a FFNN.

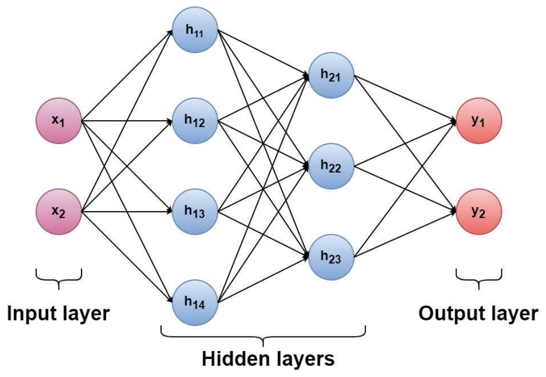

Fig. 2: This figure shows the principal structure of a possible type of FFNN. The network has an input layer with two units, a first hidden layer with four units, a second hidden layer with three units and an output layer with two units.

In a FFNN, most hidden- or output units take a set of inputs $\mathbf{z}$ (either from the input to the network or the output from other layers), multiplies these with a set of weights $\mathbf{w}$, adds a bias b and uses some *activation function* ϕ (see section 3.2.2) to calculate the output value of the unit, o [14]. This is done according to the expression in Equation 5. Such weights and biases of the units constitute the learnable parameters Ω of a FFNN that are updated during training, also called the *model parameters.*

$$o = \phi(\mathbf{w}^T\mathbf{z} + b) \tag{5}$$

3.2.2 Activation functions

Activation functions are used in most units of a FFNN to compute the value that they output. Activation functions are used in units in both the hidden layer and the output layer, and what type of output activation function is chosen for a NN is highly connected with the choice of *loss function* (see Section 3.2.5) that is used during training [14]. This section describes the activation functions that were used in this project.

The *sigmoid function* is an activation function that can be used as the activation function in the output layer for binary classification problems. It is defined for some input x as

$$\phi(x) = \frac{1}{1+e^{-x}}. \tag{6}$$

This function converts its input to a value in the interval $(0, 1)$ [25]. This value can be used to represent an unnormalized probability distribution (since it does not sum up to 1) [14]. This project also includes a modified version of the sigmoid function, which aims to approximate a differentiable version of the Heaviside step-function by using a factor k for scaling the slope and an offset factor o for choosing the threshold value. This was defined as

$$\phi(x) = \frac{1}{1+e^{-(x-o)\cdot k}}. \tag{7}$$

The *soft-max* function is a generalization of the sigmoid function for the multi-class case. This function can represent a probability distribution where the output can take on n possible values (classes). The soft-max function can, as an output unit, produce a vector $\hat{\mathbf{y}}$, with elements $\hat{y}_i = P(y = i|\mathbf{x})$ of probabilities (for the specific classes) that sums up to 1 [14]. Such an element is defined for a specific target index $i = 1, ..., n$ in such a vector as

$$\phi(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{-x_j}}. \tag{8}$$

Since this project only tackled a binary classification problem, the sigmoid-function could be used instead of the soft-max-function, since this is equivalent to a two-class soft-max function [25].

The *ReLU*-function is an activation function that is used as a hidden layer activation function in the U-Net in this project. The ReLU-function is defined as

$$\phi(x) = \max(0, x). \tag{9}$$

The plots in Figure 3 shows the two versions of the sigmoid function as well as the ReLU-function.

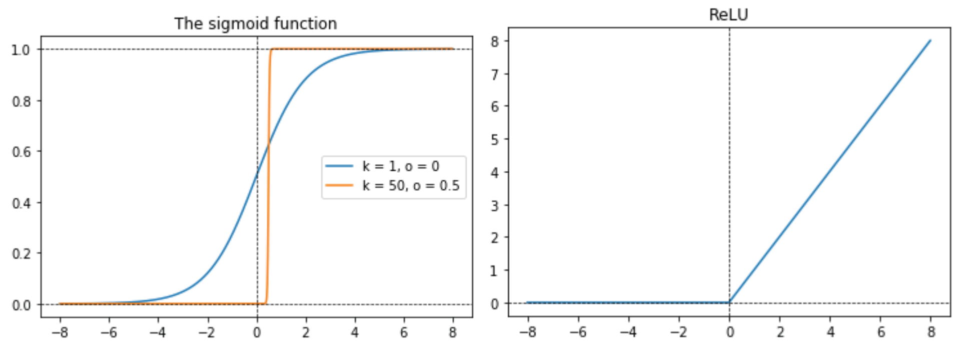

Fig. 3: This plot shows the two versions of the sigmoid function used in the output layers of networks during this project (left) and the ReLU activation function (right) used in the U-Net.

3.2.3 Basics of convolutional neural networks

A type of NN that is commonly applied to image analysis tasks (such as in this project), due to its ability handle image-like data by using sparse connectivity, is Convolutional Neural Networks (CNN). Such connectivity is given by *convolutions*. This section aims to give an overview of the basic concepts of CNN such that the description of the U-Net (a CNN) in Section 3.3 is understandable. A CNN is (roughly) defined in [14] as a NN that performs a convolution in at least one layer. A CNN can thus be a FFNN, but also other types of networks. A convolution is an operation that applies a *kernel* (or *filter*) to its input, yielding a *feature map* as output. An example of how an application of such a kernel to an input can compute one element in the output feature map is shown in Figure 4. This example involves no flipping of the kernel, as some mathematical definitions of convolutions do [14].

Practical implementations of convolutions also commonly involve *stride*, which is a hyperparameter that controls how the kernel is step-wisely applied across the input in each dimension. An example of how the feature map elements are given in a practical implementation of a 2D convolution operation can be seen in Figure 5, where the input image has the dimensions 4×4, the kernel has the dimensions 2×2 and the output feature map has the dimensions 3×3 as a result of using a stride of 1. Both figures 4 and 5 are slightly inspired by the figure on page 330 in [14].

The application of one kernel is only associated with one specific feature given at the different positions in the input. Since one often wants to detect

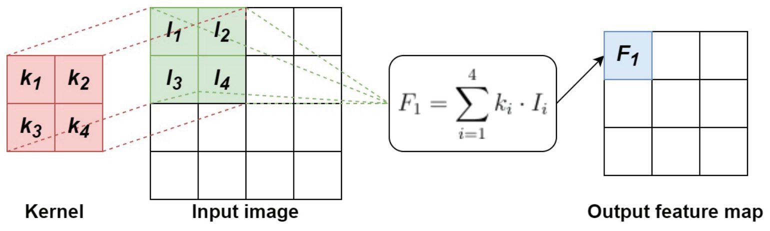

Fig. 4: This figure shows an example of how a 2D convolutional operation can compute one element F_1 in the output feature map.

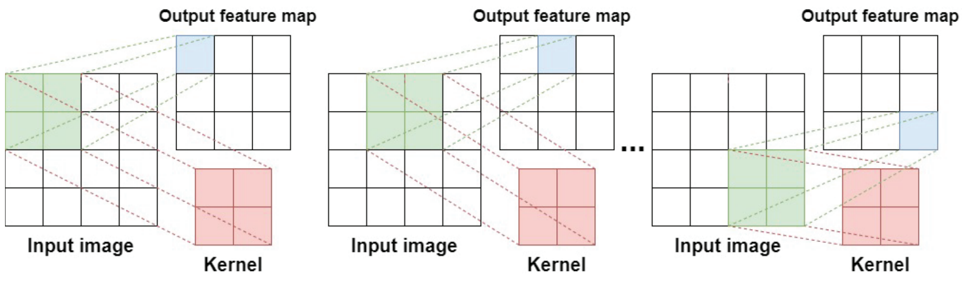

Fig. 5: This figure shows a 2D convolution operation in practice (3 of 9 steps). The kernel is with stride 1 applied to different positions of the input image to generate elements in the output feature map.

many different types of features of the input, it is common for a *convolutional layer* to contain *multiple* kernels, and thus perform multiple convolutional operations [14]. The number of separate feature maps that build up the resulting output feature map from such a layer then matches the number of kernels that has been applied to the input [18]. The input to a convolution can also have multiple *channels* (a depth larger than 1). The applied kernels can then have a depth that matches the input image that they are applied to [18].

The elements of the kernels in a convolutional layer are part of the learnable parameters (model parameters) of a CNN. The values of these elements are learned during training such that useful attributes can be extracted from the input image by application of the kernel. Commonly, a convolutional layer performing convolutional operations is followed by layers performing other operations [10]. A following layer might apply some activation function to each element of the feature map generated by the convolutions. After this, it is common for some kind of *pooling-operation* to be performed [10]. This is an operation that produces a downsampled output feature map by summarizing nearby elements. In the the U-Net in this project, *max-pooling* was used. This operation downsamples the feature map by finding the maximum element of a specified area of nearby elements [14]. An example of such an operation can be seen in Figure 6, where the max-pooling of a nearby pixel area of dimensions 2×2 is performed on a 3×3 feature map (that could be the output of the convolution in Figure 5 after an activation layer) using a stride of 1. This figure was drawn by the author of this report.

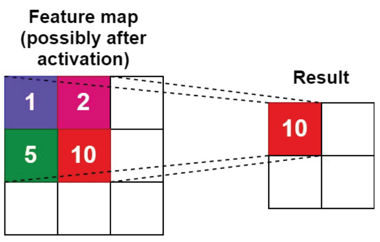

Fig. 6: This figure shows how an element is given in a 2×2 max-pooling operation on a feature map of dimensions 3×3 using stride 1. The resulting feature map has the dimensions 2×2.

3.2.4 Training a neural network

During the *training* process of a NN, the objective is usually to obtain the parameters Ω that optimizes (minimizes) some type of *loss function* L. The underlying objective of this is also usually to optimize some *performance measure*, which is indirectly optimized through minimizing the loss function [14], and that can measure in some way how well the NN solves the problem. This is often done by using a *gradient-based* optimization method, as for example *Adam* [16]. Gradient-based optimization methods use the gradient of the loss function with respect to the model parameters Ω, $\nabla_\Omega L(\Omega)$ in order to minimize it.

To compute this gradient, the *back-propagation* algorithm can be used. When training a FFNN, an input $\mathbf{x}$ is inserted into the network and forward-propagated through the network, giving an output $\mathbf{y}$ [14]. By comparing this to the corresponding *ground truth* $\hat{\mathbf{y}}$ (what the output should optimally be) by using the loss function, one can produce a scalar loss value $L(\Omega)$ [14]. The back-propagation algorithm uses this information and propagates it back through the network in order to compute $\nabla_\Omega L(\Omega)$. This computation is done inexpensively by using the chain rule in a recursive manner to compute gradients with respect to all parameters Ω in the network [14]. The computed gradient $\nabla_\Omega L(\Omega)$ is then used by the chosen optimization method to update the learnable parameters Ω. For a more detailed explanation of the back-propagation algorithm, the reader is referred to other literature.

This project used networks that yielded not only one, but two outputs $\mathbf{y_1}$ and $\mathbf{y_2}$. In these implementations, the total loss function $L_{total}(\Omega)$ was calculated as a sum of two different loss functions $L_1(\Omega)$ and $L_2(\Omega)$, each calculated from the two respective outputs, as

$$L_{total}(\Omega) = L_1(\Omega) + L_2(\Omega). \tag{10}$$

In the used framework for these networks, *Keras*, the implemented back-propagation computed a single $\nabla_\Omega L_{total}(\Omega)$ based on this *combined* loss function [25].

In this project, the chosen gradient-based optimization algorithm for updating the parameters Ω to minimize $L(\Omega)$ was *Adam*. This method is a modified version of *gradient descent* (GD). GD minimizes a loss function by iteratively updating the learnable parameters of the network, Ω in each iteration step t by moving slightly in the negative direction of the gradient with respect to the current set of parameters. The formula for updating the parameters in GD can be given as

$$\Omega^{(t+1)} = \Omega^{(t)} - \eta \cdot \nabla_\Omega L(\Omega^{(t)}) \tag{11}$$

where η is the *learning rate* that determines the size of the step in the negative direction of the gradient [20]. This method can be stochastic in the sense that it can use randomly selected *mini-batches*, i.e. small parts (of specified size) of the training data to compute the gradient.

The Adam algorithm offers improvements to GD by using adaptive and decaying learning rates individually for *each* learnable parameter [16]. For a more in-depth explanation of the Adam algorithm than the one given here, see [16]. The individual learning rates for each parameter are calculated from running averages of the gradients from previous iterations that are iteratively updated at each step [16]. This causes the algorithm to adapt to the behavior of the gradients. Further, there are hyperparameters that controls how fast these averages will decay exponentially (causing the learning rate to decay), and another hyperparameter that functions like η in Equation 11 [16], often denoted as the *initial* learning rate. This terminology was used during this project.

3.2.5 Loss functions

Since two separate problems were tackled in this project, namely pixel-classification and image-to-image regression, different loss functions had to be used for each problem. This section describes the two loss functions that were used in this project.

For the binary classification (segmentation) output, *Dice-loss* was used. This is a loss function based on the *Sørensen-Dice coefficient*, which is a commonly used performance metric for segmentation problems (see section 3.4). There are several different existing definitions of this loss function [3] [4] [21] that also extend to multiclass segmentation problems. In this project, the definition of the Dice-loss for binary classification, similar to [21], was used. This definition was given as

$$L_{Dice} = 1 - \frac{2\sum_{i=1}^{N} y_i \hat{y}_i}{\sum_{i=1}^{N} y_i^2 + \sum_{i=1}^{N} \hat{y}_i^2} \tag{12}$$

where it is assumed that the predicted output from the network, $\mathbf{y} = (y_1, y_2, ..., y_N)$ and the ground truth targets $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_N)$ are equidimensional volumes with N corresponding elements (pixels), in this project given by a mini-batch of 2D image-slices. Here, $\hat{\mathbf{y}}$ contains annotated binary values, i.e. $\hat{\mathbf{y}} \in \{0, 1\}$ and the values of $\mathbf{y}$ are given by a pixel-wise sigmoid in the output layer (see section 3.2.2), which yields outputs in the range $\mathbf{y} \in (0, 1)$ [17].

The 1 is added to yield positive loss values, as opposed to the form in [17]. Commonly, a factor ϵ is added in both the denominator and the numerator to avoid zero-division when $\mathbf{y} = \hat{\mathbf{y}} = \mathbf{0}$. However, the training data contained no fully zero-valued images, which causes this situation to never occur. This loss function makes up for class imbalances in the data, such as the imbalance between foreground (bone) and background pixels in this project [21].

For the regression output, the common *mean squared error* (MSE) loss was used. This is defined as

$$L_{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{13}$$

where again the output $\mathbf{y} = (y_1, y_2, ..., y_N)$ and the ground truth targets $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_N)$ are equidimensional volumes with N elements corresponding to mini-batches of 2D image slices. In this project, the elements in $\mathbf{y}$ and $\hat{\mathbf{y}}$ are DM-values.

3.2.6 Regularizing a neural network

During the process of minimizing a loss function $L(\Omega)$ such as the ones given in the previous section, a problem called *overfitting* can occur. Briefly explained, this is when the NN starts to tend too much to details in the training data, decreasing its ability to generalize on unseen data, thus decreasing what can be called the *generalization performance*. During a training process, this can be observed when the loss evaluated on the *training* data keeps decreasing while the loss evaluated on the *validation* data starts to increase or diverge. An example of such a training process from this very project can be seen in Figure 15.

To cope with this problem, *regularization* can be used. Regularization is (roughly) defined in [14] as any type of modification to a network that is performed with the intention to increase the generalization performance without decreasing the performance on the training data. This is often done by adding a *penalty* term to the loss function. One of the most basic examples of this is L_2*-regularization*. For a loss function $L(\Omega)$, with $\mathbf{w}$ being the *weights* in Ω (that also contains biases), this can be written as

$$\hat{L}(\Omega) = L(\Omega) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{14}$$

for a specific regularization coefficient (hyperparameter) λ [20].

In essence, L_2-regularization is used to force the magnitude of the weights to stay relatively low, since large weight values penalize the loss function more [20], and can improve the generalization performance. In a similar way, one can add other types of penalty terms to the loss function to improve generalization performance. There is a clear similarity between Equations 10 and 14. Adding multiple loss functions based on different outputs from the network can also be seen as a means of regularization, and has indicated the ability to improve generalization performance, as e.g. in [5]. Using a network that performs multiple tasks with some parameters of the models being shared between the two (or even more) tasks is also known as *multitask learning*, as previously mentioned in category two in Section 2.1. Solving multiple, related tasks can improve the network's ability to generalize. The parameters shared between the tasks are forced to take on values that allows it to generalize well enough to be able to perform all tasks during training [14].

Regularization is however not only done by adding penalty terms. A non-penalty regularization technique that is included in this project is *data augmentation*, which aims to increase the variance of a dataset by performing random alterations to existing examples in the dataset in order to create new examples. Data augmentation is done using several techniques in this project. These are presented in Section 4.2.4.

3.3 The U-Net

In this project, all network models were implemented by modifying a network architecture similar to the *U-Net*. The U-Net is a well-known CNN-architecture that was established in Ronneberger et al. (2015). It has been modified by others and used for a wide range of segmentation tasks. This network is built up by a contractive path that *downsamples* an input image and an expansive path that *upsamples* the downsampled data into an output image. Figure 7 shows the U-Net as defined in [15]. This network architecture is said to be able to apprehend context through the contraction and get precise localizational information through the expansion [15]. This has shown great results in medical image segmentation, being one of the most well-known architectures for this purpose [10].

The contractive path is built up by several repetitions of the process of applying two successive convolutional layers with kernels of dimensions 3×3. These are both succeeded by a *ReLU*-activation layer as well as a layer performing 2×2 *max-pooling* on the output feature maps. This uses a *stride* of 2. In the first such process, the channels for the output feature map (the number of filters for the convolutional layers) are set to 64. In each following process, the number of channels are increased by a factor of 2. In each contracting step, a skip-connection sends the current feature map to the corresponding level in the contractive path [15]. *Dropout* is used in end of the contractive path. This technique randomly drops out elements of the current feature map by setting them to zero. This can be seen as a way to augment a feature map [15].

The expansive path performs similar sub-processes as the contractive path but uses *up-convolutions* (denoted *up-conv* 2×2 in Figure 7) instead of max-pooling. This operation has the ability to up-sample its input to a larger dimensionality. For more information about specific implementations of this, the reader is referred to [18]. The up-convoluted feature map is concatenated in the depth-dimension with the feature map given from the corresponding skip-connection in the contractive part of the network. Two convolutions with 3×3 kernels and *ReLU*-activation layers follows this. The number of channels is reduced by a factor of 2 in each such process. When the data has been upsampled by the expansive path, a feature map of depth of 64 remains (see Figure 7) [15].

The network up until this point is used as the backbone network in this project. This means that all modifications that were done in this project were done to the layers following this corresponding position in the used U-Net implementation (see Section 4.3).

In the original implementation, the final output is obtained by using convolutions with 1×1 kernels to transform each pixel, at this point given as a feature vector of depth 64, to a depth corresponding to the wanted number of classes for classification, then applying a soft-max activation function to yield a channel-wise probability vector for each pixel (as explained in Section 3.2.2) [15].

In the case of a binary classification, a sigmoid activation can correspondingly be applied to get an output segmentation map with a depth of 1 where the value of each pixel, in the interval $(0, 1)$, can be seen as a probability of belonging to one of the two classes. This is the method that is used in the implementations in this project, since only a binary classification problem was approached.

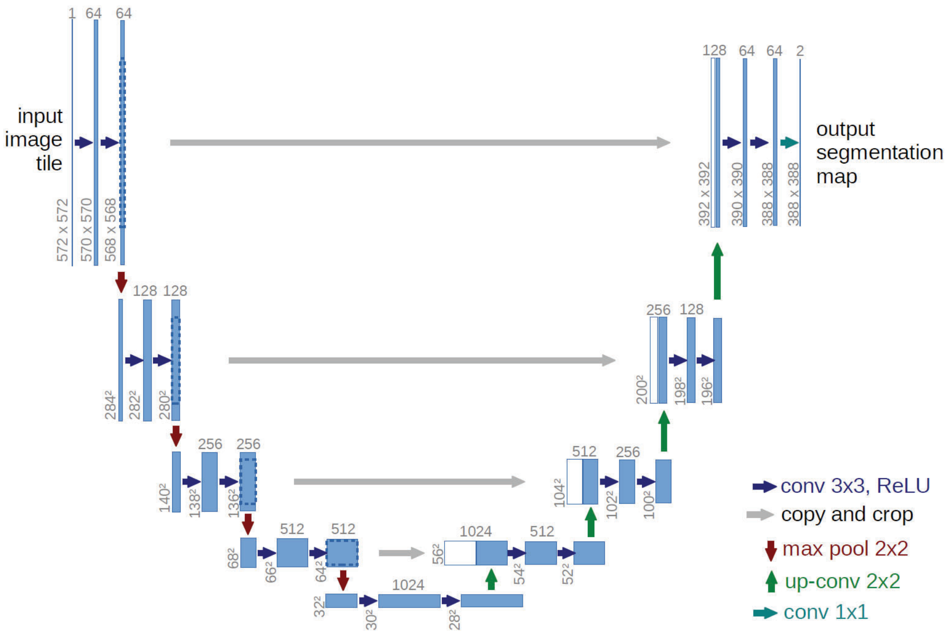

Fig. 7: This figure shows the U-Net architecture introduced in Ronneberger et al. (2015). The image is taken (with permission) from the original paper [15]. The data dimensions in this differ from the ones used in this project.

3.4 Measuring performance

To assess how well the the tested networks performed the bone segmentation, performance metrics that could satisfactorily measure this had to be used. This section describes the measurements that were used. All well-known performance metrics for evaluating segmentation results are, naturally, based on segmentation masks. A DM can adequately be transformed into a binary segmentation mask by a thresholding operation at a specific distance value. For models that yielded a DM-output, the resulting DM were thus thresholded and evaluated as binary segmentation masks. The evaluation metrics that were used in this project were defined for binary objects, regardless of the fact that some model outputs were only approximately binary (given by a sigmoid output activation).

When bone segmentations are 3D-printed at SUS, the digital representation that is used is an *isosurface*. An isosurface is created by connecting pixels of a specific (pre-set) value (isovalue) on a sub-pixel level [23]. Creating an isosurface directly from a DM can adequately be approximated by creating an isosurface from a thresholded DM. Figure 8 shows isosurfaces created from the two respective representations of a sphere. There is thus little apparent benefit of using an un-thresholded DM-representation for 3D-printing.

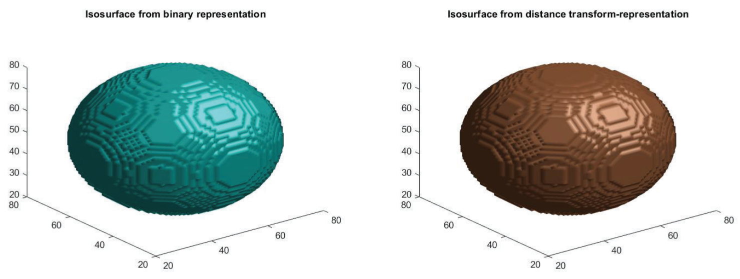
Isosurface from binary representation
Isosurface from distance transform-representation

Fig. 8: This figure shows isosurfaces of a sphere from a binary representation (left) and a (CDT) DM-representation (right) using the isovalue 0 in both cases.

The main performance metric used to evaluate segmentation results in this project was the *Sørensen-Dice coefficient*. This is the metric that the Dice loss described in Equation 12 is based on. The definition of the *Sørensen-Dice coefficient* as given in [21] was used. This can be seen in Equation 15, where $\mathbf{y} = (y_1, y_2, ..., y_N)$ and $\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_N)$ are binary volumes, both containing N binary elements.

$$Dice(\mathbf{y}, \hat{\mathbf{y}}) = \frac{2\sum_{i=1}^{N} y_i\hat{y}_i}{\sum_{i=1}^{N} y_i^2 + \sum_{i=1}^{N} \hat{y}_i^2}. \tag{15}$$

The *Jaccard similarity coefficient* can be defined, with the same prerequisites, as in Equation 16.

$$Jaccard(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_{i=1}^{N} y_i\hat{y}_i}{\sum_{i=1}^{N} y_i^2 + \sum_{i=1}^{N} \hat{y}_i^2 - \sum_{i=1}^{N} y_i\hat{y}_i}. \tag{16}$$

Both of these metrics are measures of spatial overlap for binary objects [22]. One can also define the both the Sørensen-Dice coefficient and the Jaccard index for measuring the similarity between two binary objects in terms of the measures TP, TN, FP and FN as defined in Figure 9 for two binary objects. This figure is inspired by the table given in [22]. These alternative definitions are shown in Equations 17 and 18. They are equivalent to Equations 15 and 16, respectively, that describe the practical implementations of said metrics in this project. The subtracted term in the denominator in Equation 16 is added to avoid two instances of the TP, as in the denominator in the *Sørensen-Dice coefficient*.

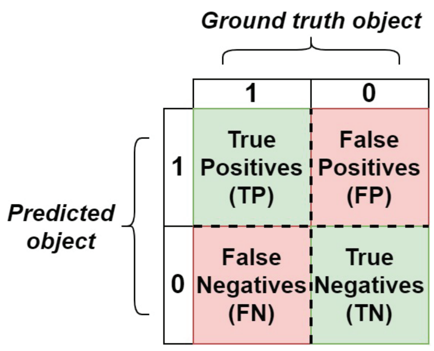

Fig. 9: This figure shows a table defining the binary overlap measures that can be used to alternatively define Sørensen-Dice coefficient and the Jaccard index [22].

$$\textit{Sørensen-Dice coefficient} = \frac{2 \cdot \mathbf{TP}}{2 \cdot \mathbf{TP} + \mathbf{FP} + \mathbf{FN}} \tag{17}$$

$$\textit{Jaccard Index} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FP} + \mathbf{FN}} \tag{18}$$

3.5 Comparing models

To be able to measure a potential gain in performance given by the performed modifications of the U-Net as well as the differences in performance between the different modifications, some type of *model comparison* procedure had to be used in this project. To compare different models, one can obtain estimates of their generalization performances. Obtaining such estimates can however be a difficult task, since they are often *biased*. Bias can be defined as the difference between the *expected* value of the *estimation* of a parameter (such as the generalization performance) and the true value of the parameter [19]. Bias can be added to such estimations both positively and negatively.

K-fold cross-validation is a method commonly used to estimate the generalization performance of a model on a specific, typically smaller dataset. Since both datasets used in this project were relatively small, this was an appropriate method to use. This method is based on dividing the dataset into k *folds*. During the process of the method, networks are trained on $k-1$ folds and evaluated on the remaining fold of the dataset [14]. This is done for all k folds. This means to essentially train k different networks. Each network is evaluated on the remaining fold using a performance metric of choice. By computing the average performance as well as the standard deviation over folds, one can obtain an estimate of the generalization performance of the NN.

For many NN, there is a set of *hyperparameters* that needs to be *tuned*. In contrast to the model parameters mentioned in Section 3.2.1, these are parameters that affect how the training procedure is carried out, and that have to be specified manually or by an external procedure before training [19]. These parameters can affect the model that is obtained from the training procedure. If one uses the same test set to both tune the hyperparameters and estimate the generalization performance of the model, a positive bias can be added to the estimate [19].

Nested k-fold cross validation can be used to obtain an almost entirely unbiased estimate of the generalization performance of a model when the tuning of hyperparameters is needed. This finds the best hyperparameters in an *inner* cross-validation loop for each training fold by using a hyperparameter optimization method (such as e.g. *grid search*) and then uses the found hyperparameters to train the model on the current training set before testing it on the current test fold. The inner loop thus performs the hyperparameter tuning. This gives a more unbiased estimate of the generalization performance, since information leakage from the test set to the training set is no longer possible.

However, when only performing model or algorithm comparisons, such a process may not be needed, since one is commonly only interested in the *relative* performance of the different models. An equal (positive or negative) bias between the different models will thus not affect the model comparison procedure negatively [19]. Nevertheless, it might be hard to tell if the added bias is equal for the different models.

Since the different models in this project were essentially the same model, but with the final layers modified, the common hyperparameters were set to the same fixed values when performing the k-fold cross-validation for comparison. This was done because the aim was to observe the direct impact in performance given by the modifications. The common hyperparameter values were found by using part of the training set in each fold as a validation set to find a set yielding *converging* validation and training set losses for all models. The term *convergence* is used in this project to denote when the loss value approaches a local or global minimum of the loss function and ceases to notably decrease or increase. More details regarding how the final evaluations were carried out is given in Section 4.5 and the possible limitations of this are discussed Section 6.

3.6 Computed Tomography (CT) data

The data in the two datasets that were used this project was generated by *Computed Tomography* (CT). CT is a medical imaging modality that produces images where different types of tissue are assigned different values according to the *Hounsfield scale*. The Hounsfield scale ranges from values of -1000, which corresponds to air, to $+3000$, which corresponds to *dense* bones [27].

Many difficulties for bone-segmentation in CT-data arrive from the fact that bones consist of several different types of tissue that result in very different Hounsfield values during a CT-scan, among which cortical bone (compact bone), cancellous bone (spongy bone) and bone marrow are examples [7].

Two example problems for bone segmentation in CT-data that were observed in this project can be seen in Figure 10. The problem denoted as *P1* arises due to some bones being very closely situated, making separation difficult due to the limited resolution of the CT-image. The problem denoted as *P2* arises directly from the fact that bones consist of different types of tissues. Some bones only have a thin, outer layer of cortical bone and a large inner cavity of bone marrow or cancellous bone. If one performs a thresholding operation, these may not result in fully enclosed objects, which one usually wants. This is also due to restrictions in resolution.

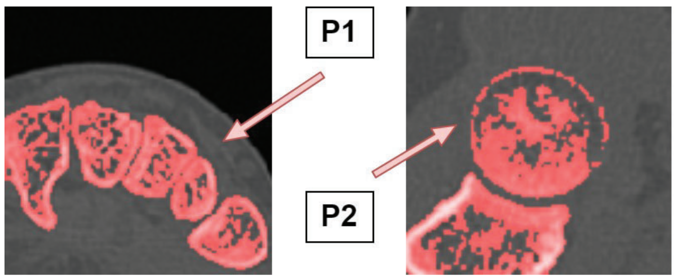

Fig. 10: This figure shows the example problems *P1* (left) and *P2* (right).

4 Methodology

This project included a lot of practical work. This section first describes the used hardware, and then two datasets that were used in this project and how they were partitioned and pre-processed (Sections 4.2.1-4.2.4). Then, the backbone U-Net implementations (Section 4.3.1) and the modifications that were performed to them to include DM in three different ways (Sections 4.3.2-4.3.4) are presented in detail. Further, the tests that were carried out are described and rationalized in two different parts. First, the less structured preliminary testing that was performed, the results that were obtained from this and the conclusions that were drawn are presented (Section 4.4). The reason for presenting these results in the section describing the methodology were that they were used during the development of the methods that were utlized for the final evaluations. Finally, descriptions of the evaluations that were performed to get the final results for comparing the different models are given (Section 4.5).

4.1 Hardware

The training and prediction that was performed during this project (for all models) used either one or two NVIDIA Titan RTX GPU:s.

4.2 Data

Two different datasets were used in this project. This section describes these datasets, how they were partitioned during training and testing and how they were pre-processed before being used.

4.2.1 In-house dataset

The in-house data available at Skåne University Hospital (SUS) consisted of five volumetric CT-images with values within the standard Hounsfield scale and corresponding segmentation masks. One volume depicted a set of arms (and hands) and had the dimensions $398 \times 512 \times 945$, two volumes depicted legs (and feet) and had the dimensions $288 \times 710 \times 648$ and $325 \times 718 \times 899$. The remaining two volumes depicted pelvises, and were of the dimensions $327 \times 454 \times 448$ and $327 \times 454 \times 565$. Ground truth segmentation masks were annotated through manual segmentation by the author of this report using a segmentation software.

4.2.2 Public dataset

The publicly available dataset from Peréz-Carrasco et al. (2017) consisted of 27 CT-volumes of size $512 \times 512 \times 10$ with corresponding segmentation masks from 20 different patients, and depicted a wide range of body parts, such as spines, skulls and arms, resulting in a large variety of shapes and sizes of the depicted bones [7]. In contrast to the in-house dataset, this dataset had been scaled and offset from the standard Hounsfield scale in a way that was unknown, with values ranging from -2000 to $3000+$. The dataset was used with permission from the author of [7] and administrator of the dataset.

4.2.3 Data partitioning

In the initial phase of this project, only three CT-volumes were available from the in-house dataset. This data was only used for initial testing of the backbone U-Net (MATLAB). The dataset was divided into training, testing and validation data by assigning the two first volumes as training data, and the last volume as validation and testing data (60% testing and 40% validation). However, there was a clear *shape overlap* of bones within volumes. To avoid the risk of inducing bias if data from the same volume was used across partitions, the splitting of the data was reconsidered. It was decided that a *volume-wise partitioning* of the data was the best way to completely avoid this type of bias. The volume-wise partitioned data is suitable for performing a k-fold cross-validation. This was the method of choice for evaluating the models. When all 5 CT-volumes were acquired for the in-house dataset, they were split volume-wise in a 5-fold cross-validation in both the preliminary and final evaluations.

The public dataset was also split volume-wise, but since the CT-volumes were smaller in this dataset, the folds had to contain multiple volumes. The same evaluation method and data partitioning as used in [1] was performed, in order to also make comparisons between results possible. The used evaluation method for the models on this dataset was *3-fold* cross-validation in both the preliminary and final evaluations. The used partitioning for this was 15 CT-volumes for training, 9 CT-volumes for testing and 3 CT-volumes for validation [1].

4.2.4 Pre-processing the data

The pre-processing that was used on the two datasets was similar. In both datasets, the CT-volumes were split and saved as 2D CT-slices from the transversal view (feet to head). As an initial pre-processing step, the CT-slices in the in-house dataset were clipped to the scale $[-300, 500]$, with the motivation that this showed good contrast between bones and background. The slices were then max-min normalized to the range $[0, 1]$, multiplied by 255 and saved as grayscale images. The CT-slices in the public dataset dataset were instead clipped to the scale $[800, 1800]$, with the same motivation as above, and then processed and saved in the same manner.

For the public dataset, data-augmentation of the CT-slices and the corresponding segmentation masks was performed. The used augmentation was inspired by the augmentation used in [1]. It consisted of random rotations of degrees in the range $[-10, 10]$, random reflections around the vertical axis, random scaling between 60% and 140% as well as random *shearing* in both the x and y-direction of degrees in the range $[-10, 10]$. The data augmentation function *augment* in MATLAB was used to perform all augmentation [23]. For each transversal image slice, 5 image slices with different random augmentations were obtained. The amount of data from each CT-volume was consequently increased by a factor of 5. A slight drop in quality for the segmentation mask boundaries was observed after augmentation. Nevertheless, it was still decided to use the augmented data.

For the in-house dataset, random *patches* of size 128×128 were extracted from all images and the corresponding segmentation masks. The use of patches was initially motivated by the fact that there was a wide range of dimensions of the transversal slices in the in-house dataset. To cope with the imbalance between bone and background, patches where the segmentation mask contained at least 1000 pixels of annotated bone were saved, and the patches that did not fulfill this criterion were not saved.

For the preliminary testing on the in-house dataset (see Section 4.4), 50 patches from random positions in the image were extracted. This resulted in the smallest amount of patches extracted from a single volume being 4088. Since the data was split CT-volume-wise, this smallest number of patches was utilized as a threshold for the number of patches that was used from each volume during training. The entire in-house dataset thus consisted of $5 \cdot 4088 = 20440$ patches during training.

For the final evaluations on the in-house dataset, new patches were generated. Due to limited computational resources and time, a reduction in the amount of patches had to be done. The amount of random patches from each image was significantly lowered to 6 patches, resulting in the smallest number of patches from a single volume being 806. This yielded a training set of 3224 images and a test set of 806 images for each fold. The networks were in both cases trained and tested on these randomly extracted patches.

For the public dataset, a similar procedure for patch extraction was used. The smallest number of randomly extracted patches from a single volume in this dataset was 85. Since the partitioning was 15 volumes for training, 3 volumes for testing and 9 volumes for testing, the network was thus trained on 1275 patches and validated on 255 patches in each fold. For the public dataset, the network was tested on 128×128 patches that were extracted from the 9 testing volumes in each fold by using a *sliding window* approach. This means that from each 512×512 transversal slice in each testing volume, 16 separate, non-overlapping patches were extracted and used as test data. This yielded $16 \cdot 10 \cdot 9 = 1440$ testing patches for each fold.

For both datasets, the extracted patches for training and testing consisted of corresponding image patches and mask patches. To obtain the corresponding DM-represenions for these patches, the MATLAB-function *bwdist* [23] was applied to the *inverted mask* patches. Because of the fact that the ground truth DM were obtained from the segmentation masks, and misrepresentations could arise if augmenting a DM using standard techniques, the data augmentation had to be performed jointly on the images and masks prior to generating the DM-representations of the masks.

Figure 11 shows each step of the pre-processing pipeline for the public dataset for a CT-image and segmentation mask pair that is transformed into training, testing and validation data. It should be noted that such a pair was of course used as training, validation and testing data for different folds.

4.3 Models and modifications

The four models used in this project all utilized a U-Net very similar to the one described in Section 3.3 as a backbone. The implementations of this are both called *U-Net* in this project, and are presented in Section 4.3.1. The final layers of this model were modified into three different models, all of which are inspired by previous work, but simplified. The rationale behind each performed modification of the U-Net are described in detail in Sections 4.3.2-4.3.4.

4.3.1 U-Net implementations

Two different implementations similar to the U-Net were used as backbones for modifications. The first implementation was acquired in MATLAB by using the MATLAB-function *unetLayers* [23]. The second implementation was found in the git-repository [26], implemented in Keras using the *Keras functional API*. This repository allows copying and modification of the code [26]. Since operations in MATLAB and Keras might yield different numerical results, full correspondence between the two networks could not be guaranteed.

The differences between the two backbone implementations include the fact that in MATLAB, the loss function for the network is defined in the final layer of the network, the *output layer*, which in the used original implementation is a *pixelClassificationLayer*, which applies a pixel-wise soft-max and a *cross-entropy loss* [23]. The final convolutional layer then has to use *two* 1×1 filters, in correspondence with the method used in [15], to yield the needed $128 \times 128 \times 2$ output for a binary soft-max. The backbone U-Net in

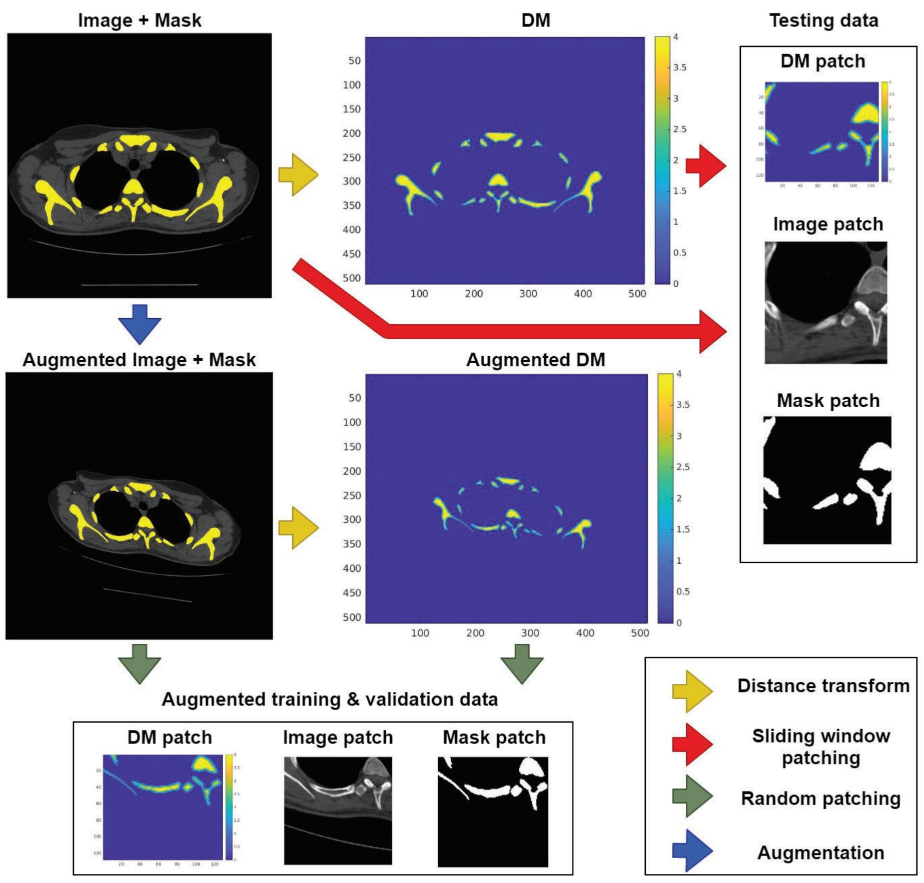

Fig. 11: This figure shows the pre-processing pipeline for the public dataset. The CT-image and its corresponding segmentation mask is denoted *Image + Mask* and is shown as a combined image where the yellow areas correspond to the segmentation ground truth for bone. The example images are taken from the public dataset [7].

MATLAB was only used for preliminary testing. It is shown in Figure 19 in Appendix A.

The Keras implementation of the backbone U-Net was slightly modified from its original state found in [26] to more closely correspond to the MATLAB implementation. The final layer of the network was modified to be given by a layer that performs 1×1 convolution using a single filter on the $128 \times 128 \times 64$ output from the final up-convolution process, and applies a pixel-wise sigmoid activation function, resulting in an output segmentation mask of dimensions $128 \times 128 \times 1$. In Keras, the loss function is not specified when specifying the output layer, but instead when compiling the network [25]. This network is shown in Figure 20 in Appendix A. This network was evaluated for baseline results in the final evaluations.

The weights in both backbone implementations were initialized by using the 'He' weight initialization method [23]. All modifications that were done to the backbone U-Net were performed after the final process in the expansive path, or in other words when the feature map has a depth of 64 (see Figure 7). All operations up until this layer were considered the backbone U-Net in this project.

4.3.2 DTR-Net

The DTR-Net (Distance Transform Regression-Network) was a simple modification to the backbone U-Net that allowed it to perform *image-to-image regression*, where the target regression output is a DM. This type of model is mentioned in section 2.1 as a part of the *third category*.

The modification that was performed to the backbone U-Net (MATLAB) was to modify the final 1×1 convolutional layer to only yield one output channel resulting in an output of $128 \times 128 \times 1$, and then applying a linear activation function. Since working in MATLAB requires the loss function to be specified by the output layer, a *regressionLayer* was chosen [23]. This layer calculates the *half* MSE-loss between the output DM and the ground truth DM. This simply means that the MSE-loss is multiplied by a factor $\frac{1}{2}$.

The same implementation was later performed for the U-Net (Keras), instead using a regular MSE-loss. This was done for the final evaluations described in Section 4.5. The principal structure of the DTR-Net can be seen in Figure 12.

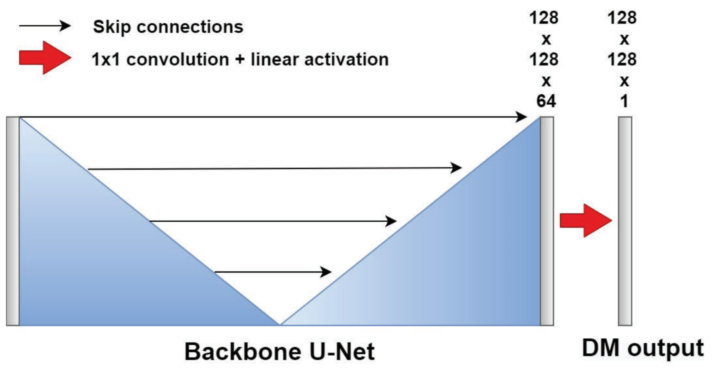

Fig. 12: This figure shows the structure of the DTR-Net with respect to the backbone U-Net.

4.3.3 MT-Net

The MT-Net (Multitask-Network) was a modification to the U-Net (Keras) that allowed it to perform image-to-image regression of a DM as a complementary task to the segmentation performed by the U-Net, in a multitask framework. This approach was inspired by [4], but simplified. In Section 2.1, this was mentioned as part of the *second category*. This architecture removes the need of the post-processing that is needed on the DTR-Net output in order to create a segmentation mask, but allows the network to still possibly be regularized by the DM-output during backpropagation.

As for the practical implementation, the same technique as for the DTR-Net was used in order to perform the image-to-image regression task. This model can be seen as a combination of the U-Net and the DTR-Net. The two branches simply consisted of two separate convolutional layers performing 1×1 convolutions added to the backbone. The branch for DM-regression used linear activation, and the branch for segmentation used sigmoid activation.

The loss function that was used for the MT-Net was a joint loss function (see section 3.2.4) of Dice-loss and MSE-loss. The Dice-loss L_{Dice} was evaluated on the segmentation mask-output, y_{seg}. The MSE-loss L_{MSE} was evaluated on the DM-output y_{DM}. The total loss was given as in Equation 19. The principal structure of the MT-Net can be seen in Figure 13.

$$L_{total} = L_{Dice}(y_{seg}) + L_{MSE}(y_{DM}). \tag{19}$$

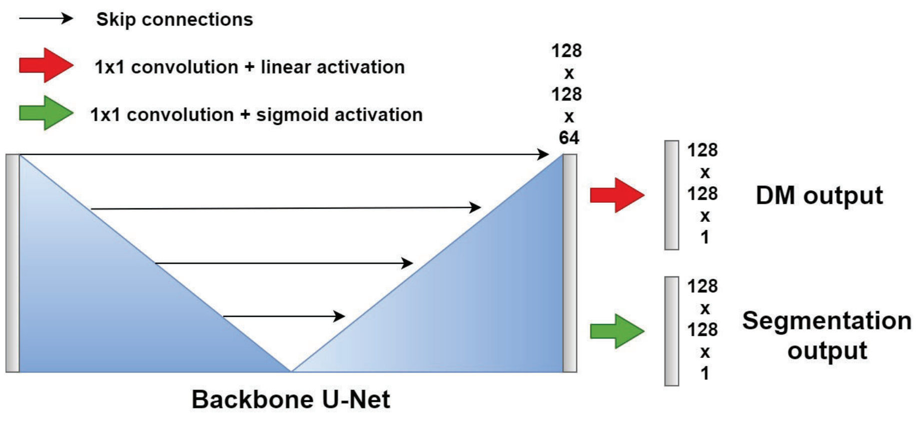

Fig. 13: This figure shows the structure of the MT-Net with respect to the backbone U-Net.

4.3.4 IO-Net

The IO-Net (Intermediate Output-Network) was a modification to the U-Net (Keras) that was similar to the MT-Net, but instead of adding a separate branch to learn the segmentation mask and DM separately, the network first learns a DM-representation, gives this as an intermediate output, and then performs an approximated thresholding operation by applying the modified sigmoid function described in Section 3.2.2 and Figure 3 with nonzero k and o to obtain the output segmentation mask. This is inspired by the approach used in [3], previously mentioned in section 2.1 as a part of the *first category*.

This was implemented in practice by again attaching a 1×1 convolutional layer with a *linear* activation function to the backbone U-Net to yield the DM, giving this as an intermediate output. Then, the resulting DM is inserted into an activation layer performing the modifed sigmoid activation. During the final evaluations, the parameters $k = 50$ and $o = 0.5$ were used, based on preliminary testing. The total loss function $\mathbb{L}_{total}$ for the IO-Net was implemented identically to equation 19. The principal structure of the IO-Net can be seen in Figure 14.

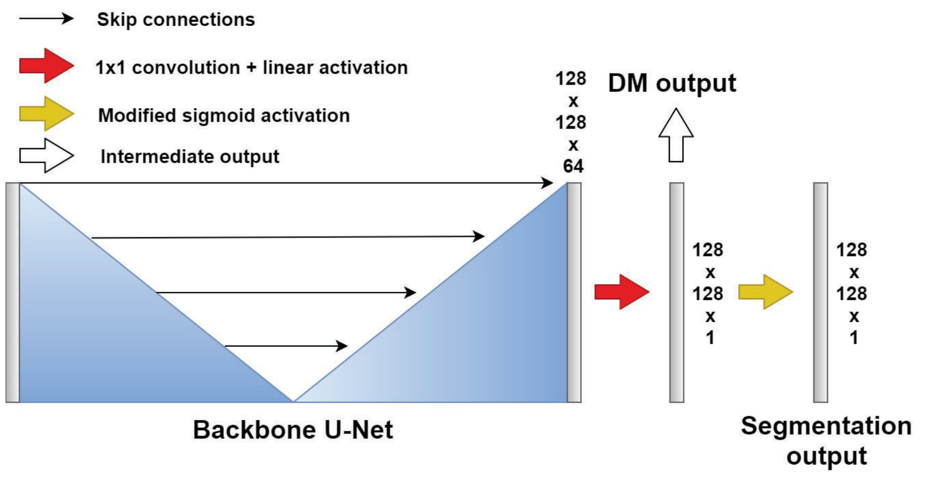

Fig. 14: This figure shows the structure of the IO-Net with respect to the backbone U-Net.

4.4 Preliminary tests and results

In the initial phase of the project, less structured preliminary tests were performed with the purpose of assessing the data, the possible difficulties of the bone segmentation problem and the performance of the basic network structures. Several hyperparameters and factors were decided from the results. This section describes these tests, their results and the decisions that were made based on them.

An initial assessment of the U-Net (MATLAB) was done by training and testing using the in-house dataset consisting of 3 CT-volumes (see Section 4.2.3). During this initial training, clear signs of overfitting on the training set were observed. An example of this can be seen in Figure 15. The overfitting occured very early in the training process for almost every combination of hyperparameters that was tested. The U-Net initially had more down-sampling (and up-sampling) steps, but in order to tackle the overfitting problem, the complexity of the model was reduced by decreasing the depth of the network to the final size (see Appendix B). The network converged and a Sørensen-Dice coefficent of **89.53 %** was obtained on the test set. This result is biased due to information leakage between the validation and testing sets, but showed potential for future modifications of the U-Net backbone.

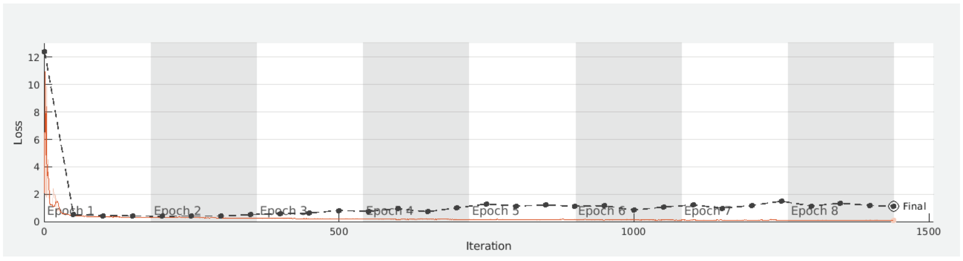

Fig. 15: This figure shows an example of the overfitting that consistently occured during the first epochs of the initial testing of the backbone U-Net (MATLAB). The orange curve represents the loss evaluated on the training set and the black, dotted curve represents the loss evaluated on the validation set, that clearly starts to increase during the second epoch.

The succeeding preliminary test was an evaluation of the DTR-Net (MATLAB) and its required post-processing. This was done by using the full in-house dataset. Since overfitting was a problem during the evaluation of the U-Net, it was concluded that hyperparameter tuning could be needed for the validation loss to converge during training. By writing a script performing volume-wise *5-fold nested* cross-validation (see section 3.5), the generalization performance of the network could be estimated. This script was implemented to perform grid search (brute-force testing of all combinations of a set of given hyperparameters) in its *inner loop* (performing 4-fold cross validation) to find the best performing combination of hyperparameters. The only two hyperparameters that were tuned were the *initial learning rate* and the *mini-batch size.* Two truncation factors for the DM; 8 and 4, were tested. Each network was trained for 500 epochs each. The results after post-processing the DM by thresholding at 0, averaged across folds, can be seen in Table 1.

Tab. 1: This table shows the results (across folds) given for the DTR-Net for the 5-fold nested cross-validation, where t denotes the truncation threshold that was used on the DM-representation of the ground truth.

Network	Sørensen–Dice coefficient
DTR-Net ($t = 4$)	0.41 ± 0.15
DTR-Net ($t = 8$)	0.34 ± 0.09

Through troubleshooting, it was found that the results in Table 1 were affected by the fact that the used (half) MSE-loss yielded small fluctuations (noise) around it's limits (0 and 4) in the predicted DM, making a threshold close to 0 yield a bad segmentation map. This is probably due to the fact that the used loss function does not give enough punishment to the network for noise-like fluctuations. This problem is illustrated in Figure 16. This figure shows how an input CT-slice inserted into the DTR-Net generates a prediction of a noisy DM that contains values above 4 and below 0, giving a very noisy segmentation mask when thresholding at 0. It also shows that setting the threshold to 0.5 results in a segmentation mask more similar to the ground truth, since this is more robust to noise.

Although it yields a good estimation of the generalization performance, the nested cross-validation that was used in this evaluation was not a computationally feasible way to perform further estimations of network performances in this project, since a limited amount of resources for computation were available.

Further preliminary evaluations of the DTR-Net (MATLAB) were performed using a script performing *3-fold* cross-validation on the *public dataset*, post-processing the results using a threshold at 0.5. This was implemented from scratch, utilizing the data partitioning described in Section 4.2.3, and is described in more detail in Section 4.5, since it was also used for the final evaluation. The training was performed for 500 epochs using an initial learning rate of 10^{-4} and a mini-batch size of 16. These hyperparameters were determined by observing that convergence of the validation loss was given for each fold (or rather non-divergence). A DM-truncation at 4 was used. The results can be seen in Table 2. These results confirmed that the use of the DM representation was possible for segmentation and that the used pre- and post-processing was usable. The DM-truncation at 4 and the post-processing threshold of 0.5 were used for the rest of the project, as well as the chessboard distance transform (see section 3.1) for the DM.

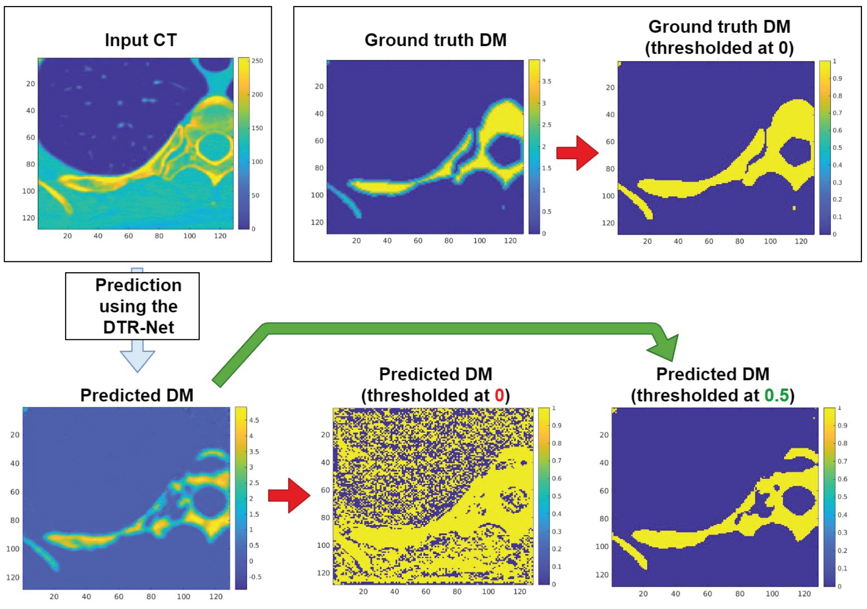
Input CT
Ground truth DM
Ground truth DM (thresholded at 0)
Prediction using the DTR-Net
Predicted DM
Predicted DM (thresholded at 0)
Predicted DM (thresholded at 0.5)

Fig. 16: This figure shows the thresholding issue for the post-processing of the output of the DTR-Net. The example image is taken from the public dataset [7].

Tab. 2: This table shows the results given by the DTR-Net (MATLAB) for the 3-fold cross-validation using the public dataset.

Network	Sørensen–Dice coefficient	Jaccard Index
DTR-Net	0.82 ± 0.01	0.69 ± 0.01

4.5 Final evaluations

The aim of this project was to see if simple modifications of the U-Net to include DM in the training and prediction processes could yield improved performance for bone segmentation in CT-data. Another aim was to determine which of these modifications yielded the best performance if the previous aim proved to be true. To be able to meet these aims, controlled evaluations that allowed a fair comparison between the performance estimations of the different modifications had to be performed.

To do this, only the *Keras-implementations* of the U-Net and its modifications described in Sections 4.3.1-4.3.4 were used in the final evaluation. As previously described, these were all implemented using the same backbone network and either using Dice-loss, MSE-loss or the *combination* of these as a loss function. The comparative evaluation was carried out using the two different datasets. Each dataset used a different method for estimating the performances of the models. The scripts for training and testing were also implemented in Keras, and the same scripts were used across networks to ensure control.

For evaluation on the public dataset, a 3-fold cross-validation was implemented using the training, testing and validation partitioning described in Section 4.2.3. The evaluation pipeline for the 3-fold cross-validation is shown in Figure 17.

A 3-fold cross-validation using a part of the training data as a validation set is of course not optimal, since it lowers the amount of training data that is used. However, the validation set was used to find an *initial learning-rate* for the Adam-optimizer and a *mini-batch size* such that the backbone U-Net, the DTR-Net, the MT-Net and the IO-Net all seemed to yield validation and training losses that did not diverge. The validation set in each fold was thus used for tuning these common hyperparameters. The used hyperparameters were a mini-batch size of *4* and a learning rate of 10^{-4}.

All networks during this test were trained for 800 epochs. For reproducibilty and control, training was performed using a fixed random seed. In order to further evaluate the robustness to different hyperparameters of the different modifications and to be able to show how this could affect the results, combinations of mini-batch sizes 4 and 2 and initial learning rates 10^{-4} and 10^{-3} were also tested. All other common hyperparameters used preset values. These results can be found in Table 4.

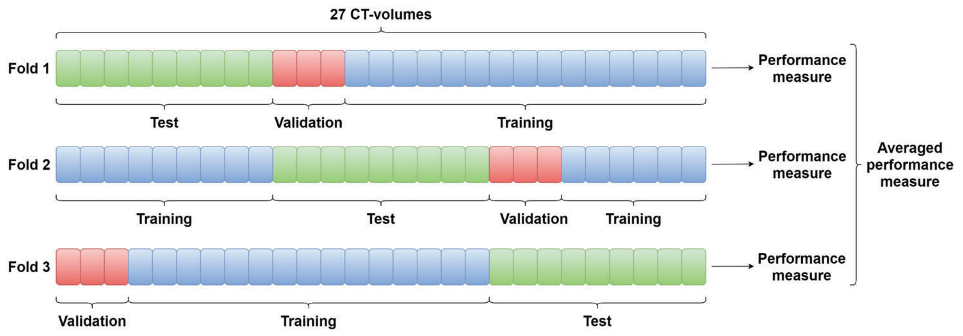

Fig. 17: This figure shows the partitioning of the 27 CT-volumes in the public dataset for the different folds in the 3-fold cross validation as well as the pipeline for obtaining the performance measure averaged across folds.

For the in-house dataset, a volume-wise 5-fold cross-validation was performed for all modifications. A schematic image of this pipeline for evaluation can be seen in Figure 18. Due to observations of an oscillating training loss during a first run where the same common hyperparameters as for the 3-fold cross-validation were used, the initial learning rate was lowered to 10^{-5}. The results for both runs are reported below. This test was run for 200 epochs for all models, using a batch size of 4. All other common hyperparameters used preset values.

The results from both these tests can be seen in Section 5. Examples of resulting segmentations from each of the three test-folds by the respective networks during the 3-fold cross-validation can be seen in Figure 21.

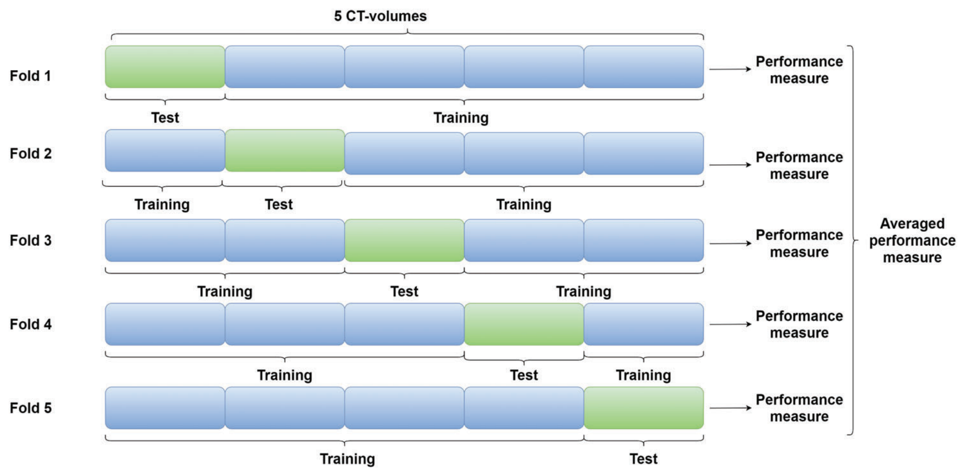

Fig. 18: This figure shows the partitioning of the 5 CT-volumes in the public dataset for the different folds in the 5-fold cross-validation as well as the pipeline for obtaining the performance measures (Dice and Jaccard), averaged across folds.

5 Results

The results given by the 3-fold cross-validation on the public dataset for the combination of hyperparameters that allowed the training loss of all models to converge can be seen in Table 3. Plots of the training and validation losses during training for each model to obtain these results can be seen in Figure 22 in Appendix B. Table 4 shows the results of the 3-fold cross validation for all combinations of the common hyperparameters *initial learning-rate* (with values 10^{-4} or 10^{-3}) and *mini-batch sizes* of 4 or 2. Example segmentation predictions by each network for each fold of the 3-fold cross-validation can be seen in Figure 21 in Appendix B. The results from the 5-fold cross-validation using the in-house dataset for two different initial learning rates can be seen in Table 5.

Tab. 3: This table shows the averages and standard deviations of the results for all models for the 3-fold cross-validation on the public dataset, using an initial learning rate of 10^{-4} and a mini-batch size of 4. The averages and the standard deviations are computed across folds. For reference, it also shows the results obtained by [1] for the similar test. *Dice* denotes the Sørensen–Dice coefficient and *Jaccard* denotes the Jaccard Index. The *Loss*-column describes the used loss functions for the different models, either the Dice-loss, the Mean Squared Error (MSE) loss or a combination.

Network	Loss	Dice	Jaccard
MT-Net	Dice + MSE	0.79 ± 0.05	0.66 ± 0.07
IO-Net	Dice + MSE	0.78 ± 0.10	0.65 ± 0.13
U-Net	Dice	0.76 ± 0.06	0.62 ± 0.07
DTR-Net	MSE	0.74 ± 0.11	0.60 ± 0.14
Klein et al. (2018)	-	*0.92 ± 0.05*	*0.85 ± 0.08*

Tab. 4: This table shows the results for all models for the 3-fold cross-validation on the public dataset for all different combinations of initial learning rates (ILR) and mini-batch sizes (MBS). The given average scores and standard deviations are computed across folds. Entries with a dash (-) indicates that the network diverged during training for more than one of the 3 folds. The *Loss*-column describes the used loss functions for the different models. The results in bold were used in Table 3.

Network	Loss	ILR	MBS	Dice	Jaccard
MT-Net	Dice + MSE	10^{-4}	4	**0.79 ± 0.05**	**0.66 ± 0.07**
			2	0.79 ± 0.13	0.67 ± 0.16
		10^{-3}	4	-	-
			2	-	-
IO-Net	Dice + MSE	10^{-4}	4	**0.78 ± 0.10**	**0.65 ± 0.13**
			2	0.76 ± 0.13	0.63 ± 0.16
		10^{-3}	4	-	-
			2	0.41 ± 0.31	0.31 ± 0.25
U-Net	Dice	10^{-4}	4	**0.76 ± 0.06**	**0.62 ± 0.07**
			2	-	-
		10^{-3}	4	-	-
			2	-	-
DTR-Net	MSE	10^{-4}	4	**0.74 ± 0.11**	**0.60 ± 0.14**
			2	0.75 ± 0.09	0.61 ± 0.11
		10^{-3}	4	0.62 ± 0.14	0.46 ± 0.16
			2	0.69 ± 0.11	0.54 ± 0.13

Tab. 5: This table shows the results for all models for the 5-fold cross-validation on the in-house dataset. The given average scores and standard deviations are computed across folds. The results are shown for two different initial learning rates for the Adam-optimizer, denoted by *ILR*. The numbers in bold indicate the results for the initial learning rate with which the training loss converged for all networks and folds. *Dice* denotes the Sørensen–Dice coefficient and *Jaccard* denotes the Jaccard Index. The *Loss*-column describes the used loss functions for the different models, either the Dice-loss or the Mean Squared Error (MSE) loss.

Network	Loss	ILR	Dice	Jaccard
MT-Net	Dice + MSE	10^{-5}	**0.93 ± 0.04**	**0.88 ± 0.06**
		10^{-4}	0.96 ± 0.01	0.93 ± 0.02
IO-Net	Dice + MSE	10^{-5}	**0.86 ± 0.18**	**0.79 ± 0.23**
		10^{-4}	0.96 ± 0.02	0.93 ± 0.03
U-Net	Dice	10^{-5}	**0.88 ± 0.13**	**0.81 ± 0.18**
		10^{-4}	0.56 ± 0.45	0.52 ± 0.42
DTR-Net	MSE	10^{-5}	**0.85 ± 0.20**	**0.79 ± 0.24**
		10^{-4}	0.96 ± 0.02	0.92 ± 0.04

6 Discussion

The results from the 3-fold cross-validation in Table 3 shows a clear increase in the *Sørensen–Dice coefficent* and the *Jaccard index* for both the *MT-Net* and the *IO-Net* in comparison with the *U-Net*. The MT-Net also shows a slight decrease in the standard deviation across folds in relation to the U-Net. The DTR-Net shows the lowest Sørensen–Dice coefficient and Jaccard index and the highest standard deviation across folds.

Table 4 that presents results for the 3-fold cross-validation for slightly different combinations of the initial learning-rate and mini-batch size indicate the same internal order of performance. The MT-Net shows better performance than the IO-Net and the DTR-Net for the two cases where these three models' training losses converged. However, the DTR-Net seems to be the most robust to slight perturbations of the hyperparameters, showing convergence for the validation and training losses for all folds and for all combinations of hyperparameters.

This table also shows that the U-Net only reached training- and validation loss convergence on all folds for a single tested combination of hyperparameters. The observed robustness during training of the DTR-Net indicates that the added DM-regression for the MT-Net and the IO-Net might be the factor that allowed them to be slightly more robust during training for different hyperparameters than the U-Net. This could possibly be due to the DM-inclusion having a regularizing effect during the training process (see Section 3.2.6).

Table 5 shows that for the volume-wise 5-fold cross-validation on the in-house dataset, the MT-Net again yields the highest Sørensen–Dice coefficent and Jaccard index. This is observed for both tested initial learning rates. As for the U-Net, the validation loss diverged for two of the folds when using the initial learning rate 10^{-4}. When converging for all folds (using the initial learning rate 10^{-5}), the U-Net shows a significantly lower performance and a higher variance across folds than the MT-Net. The DTR-Net and the IO-Net both obtain worse results than the U-Net for this setting.

It is important to note that the principal aim of this project was to evaluate if the inclusion of DM could improve the performance of the U-Net for bone segmentation. The results show that for both datasets and for all tested combinations of hyperparameters, at least one of the three modifications that includes DM outperforms the U-Net. The MT-Net seems to yield the overall best performance when converging, exhibiting the best performance in all such cases. Due to the controlled circumstances during training, with all common hyperparameters being fixed, the data being the same and the only differences being the slight modifications to include DM, one can argue that these results indicate that the inclusion of DM could actually help the

U-Net obtain better performance. At least in all observed cases during this project.

It is also important to stress the simplicity of the modifications that were made to the U-Net. One can see the different methods for including DM as simply forcing the network to give slightly more information about the surroundings of each object (bone) pixel in the input image when predicting the output. There was no noticeable difference in training or prediction time between the U-Net and its modifications. This allows one to see these modifications as possible *additions* or *alternatives* to tuning other hyperparameters or performing other modifications, if one wishes to obtain better performance for general bone segmentation.

This view of the modifications can also be used to motivate why no separate tuning of hyperparameters was used to evaluate the differences in performances. One can argue that this could obscure the actual impact of the modifications. Using a fixed set of common hyperparameters for comparing the different model performances perhaps does not allow one to find the optimal performances of the models, but it allows one to slightly decrease the risk that better tuned hyperparameters for a specific model could be the cause of that model performing better than the others. The testing of a few different combinations of hyperparameters was done to further decrease this risk. The method of using a fixed set of common hyperparameters when comparing different modifications has also been used in other similar projects, including [4].

There is a clear difference between the results obtained by [1], as seen in Table 3, and the results given by the best-performing network, the MT-Net. This shows that by applying other modifications to the U-Net and perhaps using better hyperparameters, even better results could be obtained. The aim of this project was however not to achieve optimal performance on the dataset, but rather to see if the simple modifications that were done could yield an *improved* performance for the specific problem. It is possible that by combining the two methods, even better results could be obtained. There were however also many differences between the test in this project and in [1], including differences in pre-processing, augmentation and training time. One can thus argue that the results are not really comparable.

By observing the training plots in Figure 22, one can see that for the U-Net, the validation loss oscillated throughout the training. One should note that the plots were created with adaptive scaling, causing the axes to (suboptimally) be of different scales. Although difficult to say with the difference in scaling, it seems like the computed Dice-loss for the MT-Net and the IO-Net shows less oscillations for the validation loss curve. This might be due to regularization from the added DM-regression (see Section 3.2.6).

From the example images in Figure 21, one can note that in the example images from Fold 1, the MT-Net is the most sucessful in filling the bone in the upper left corner. This indicates that it could be better than the other models at solving P2 (see Section 3.6). In the example images from Fold 2, there is visible salt- and pepper noise for both the U-Net and the DTR-Net. No such noise is given in the corresponding example images for the MT-Net and the IO-Net. In the example images from Fold 3, one can note and speculate that the three modifications are slightly better at predicting the shape of the bone in the upper right corner than the U-Net, which might be due to some awareness in shape enforced by making the network predict a DM.

Regarding the hypotheses of the benefits of DM from previous projects, discussed in Section 2.1, the results indicate that even though bones in CT-images contain such a wide variety of shapes, sizes and positions, a DL framework could benefit from including DM in the training and prediction processes when approaching this problem. However, since the actual potential benefits are obscured when only observing a few example predictions and the difference in performance measures, further investigation has to be done to note the actual benefits.

It should be further noted that the differences in results between the two datasets possibly arise due to the difference in pre-processing or the fact that the in-house dataset was tested on patches where no patch contained only background, as opposed to the public dataset.

6.1 Limitations

There are many limitations to this project that affect what conclusions can be drawn from the results. Firstly, since both datasets were limited to depicting only a few types of bones, one cannot say that this project fully assessed the models' abilities to perform *general* bone segmentation. Rather, the abilities of these models to perform bone segmentation for specific and limited datasets were assessed.

Further, the resulting estimates of the generalization performances that were obtained for comparing the models on these datasets by using the two cross-validation techniques could be misleading in several ways. Only a limited number of sets of common hyperparameters were tested and reported before finding a set that allowed the validation and training loss to converge for every fold and model. This choice could of course induce a more positive bias for one model than for the other models. This could possibly cause misleading results. There are better methods to estimate and compare the true unbiased generalization performances for the models which could have given more accurate and convincing results. These methods were not used due to a limited amount of computational resources and time.

The estimates of the model performances were thus only compared for a limited, fixed number of situations. This only allows one to say that that the MT-Net was observed to have the best estimated generalization performance for this limited set of tested situations. To be able to say that a general increase in performance is given for the U-Net for general bone segmentation when modifying it to become the MT-Net, more sophisticated tests would have to be carried out. Such tests would have to take into account the stochasticity of the training process by performing a large number of evaluations. Many other aspects of the training and the networks would also have to be taken into account.

Furthermore, the project itself is limited to evaluating and modifying a backbone U-Net that uses sigmoid output activations and a Dice-loss implemented in the specific way as described in Section 3.2.5. This might not be the most commonly used implementation of a U-Net, and there might be more optimal implementations of the U-Net for this problem that perhaps would not benefit from DM-inclusion.

Regarding the project's connection to 3D-modelling and 3D-printing, the used approach with 128×128 patches of 2D CT-slices might not be optimal for this purpose. The DM representation might cause problems when bones are situated in the boundaries of the patches, since predicting DM values for such bone pixels is difficult due to information shortage about adjacent patches. This could however be solved by using more sophisicated algorithms for reconstructing images from patches. This was however not the focus of this project.

6.2 Future work

In order to be able to further assess the benefits of the tested modifications, one could perform a wider range of tests, making it possible to investigate if the observed increase in performance is statistically significant. Another possibility is to further research the benefits of the inclusion of DM by using other performance measures that capture other aspects of the segmentation, such as the separability of adjacent objects (P1 in Section 3.6).

7 Conclusion

This project has shown that by performing simple modifications to a U-Net to make use of DM during training and prediction, indications of increased performance can be given when performing bone segmentation in CT-data. Especially, a multitask approach showed the most promising increasement in performance and an improved robustness to changes in hyperparameters during training. Pure DM-regression showed less of an increase in performance, but more robustness in training. However, these indications have to be further validated using more sophisticated methods before general conclusions can be drawn and before being used in a practical framework for bone segmentation. Nevertheless, this thesis sheds light on the possibility that DM-inclusion can also be beneficial for segmentation problems that consist of target objects with a wide range of shapes, sizes and positions.

References

[1] Klein, André & Warszawski, Jan & Hillengass, Jens & Maier-Hein, Klaus. (2018). Automatic bone segmentation in whole-body CT images. International Journal of Computer Assisted Radiology and Surgery. 14. 10.1007/s11548-018-1883-7.. Available at: https://link.springer.com/content/pdf/10.1007/s11548-018-1883-7.pdf

[2] Sánchez, José & Magnusson, Maria & Sandborg, Michael & Carlsson Tedgren, Asa & Malusek, Alexandr. (2020). Segmentation of bones in medical dual-energy computed tomography volumes using the 3D U-Net. Physica Medica. 69. 241-247. 10.1016/j.ejmp.2019.12.014. Available at: https://www.researchgate.net/publication/338455266_Segmentation_of_bones_in_medical_dual-energy_computed_tomography_volumes_using_the_3D_U-Net

[3] Xue, Yuan & Tang, Hui & Qiao, Zhi & Gong, Guanzhong & Yin, Yong & Qian, Zhen & Huang, Chao & Fan, Wei & Huang, Xiaolei. (2019). Shape-Aware Organ Segmentation by Predicting Signed Distance Maps. p. 2. Available at: https://arxiv.org/pdf/1912.03849.pdf

[4] Navarro, Fernando & Shit, Suprosanna & Ezhov, Ivan & Paetzold, Johannes & Gafita, Andrei & Peeken, Jan & Combs, Univ.-Prof. Dr. Stephanie & Menze, Bjoern. (2019). Shape-Aware Complementary-Task Learning for Multi-organ Segmentation. Available at: https://arxiv.org/pdf/1908.05099v1.pdf

[5] Dangi, Suresh & Linte, Cristian & Yaniv, Ziv. (2019). A Distance Map Regularized CNN for Cardiac Cine MR Image Segmentation. Medical Physics. 46. 10.1002/mp.13853. Available at: https://arxiv.org/pdf/1901.01238.pdf

[6] Audebert, Nicolas & Boulch, Alexandre & Saux, Bertrand & Lefèvre, Sébastien. (2019). Distance transform regression for spatially-aware deep semantic segmentation. Computer Vision and Image Understanding. 189. 102809. p. 15. Available at: https://arxiv.org/pdf/1909.01671.pdf

[7] Pérez-Carrasco, José & Begoña, Acha & Cristina, Suárez-Mejías & Luis, López-Guerra & Carmen, Serrano. (2017). Joint Segmentation of Bones and Muscles Using an Intensity and Histogram-Based Energy Minimization Approach. Computer Methods and Programs in Biomedicine. 156. 10.1016/j.cmpb.2017.12.027.

[8] Naylor, Peter & Laé, Marick & Reyal, Fabien & Walter, Thomas. (2018). Segmentation of Nuclei in Histopathology Images by Deep Regression of the Distance Map. IEEE Transactions on Medical. Available at: https://www.researchgate.net/publication/327065839_

Segmentation_of_Nuclei_in_Histopathology_Images_by_Deep_Regression_of_the_Distance_Map

[9] Jun Ma and Zhan Wei & Yiwen Zhang & Yixin Wang & Rongfei Lv & Cheng Zhu and Gaoxiang Chen & Jianan Liu and Chao Peng and Lei Wang and Yunpeng Wang and Jianan Chen. (2020). How Distance Transform Maps Boost Segmentation CNNs: An Empirical Study. Medical Imaging with Deep Learning. Available at: https://openreview.net/forum?id=gsKS0baY1B

[10] Hesamian, Mohammad Hesam & Jia, Wenjing & He, Xiangjian & Kennedy, Paul. (2019). Deep Learning Techniques for Medical Image Segmentation: Achievements and Challenges. Journal of Digital Imaging. 32. 10.1007/s10278-019-00227-x. Available at: https://link.springer.com/article/10.1007/s10278-019-00227-x#Sec29

[11] Kvam, Johannes & Gangsei, Lars & Kongsro, Jorgen & Solberg, Anne. (2018). The use of Deep Learning to automate the Segmentation of the Skeleton from CT volumes of Pigs. Translational Animal Science. 2. 10.1093/tas/txy060.

[12] Paglieroni, D. W. (1992). Distance transforms: Properties and machine vision applications. CVGIP: Graphical models and image processing, 54(1), 56-74.

[13] Rosenfeld, Azriel & Pfaltz, John. (1966). Pfalz, J.L.: Sequential Operations in Digital Picture Processing. Journal of the ACM 13(4), 471-494. J. ACM. 13. 471-494. Available at: https://www.researchgate.net/publication/220430836_Pfalz_JL_Sequential_Operations_in_Digital_Picture_Processing_Journal_of_the_ACM_134_471-494

[14] Goodfellow, Ian & Bengio, Yoshua & Courville, Aaron. (2016). Deep Learning. MIT Press. p. 117, 121, 164-165, 177, 179, 181, 189, 200, 204, 241, 271-272, 326, 330, 342-344. Available at: http://www.deeplearningbook.org

[15] Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. LNCS. 9351. 234-241. Available at: https://arxiv.org/pdf/1505.04597.pdf

[16] Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations. Available at: https://arxiv.org/pdf/1412.6980.pdf

[17] Perone, Christian & Cohen-Adad, Julien. (2018). Deep semi-supervised segmentation with weight-averaged consistency targets. p. 3. Available at: https://arxiv.org/pdf/1807.04657.pdf

[18] Dumoulin, Vincent & Visin, Francesco. (2016). A guide to convolution arithmetic for deep learning. p. 8-9. Available at: `https://arxiv.org/pdf/1603.07285v1.pdf`

[19] Raschka, Sebastian. (2018). Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning. p. 45-46. Available at: `https://arxiv.org/pdf/1811.12808.pdf`

[20] Bishop, Christopher. (2006). Pattern Recognition and Machine Learning. p. 144, 240, 257. 10.1117/1.2819119.

[21] Milletari, Fausto & Navab, Nassir & Ahmadi, Seyed-Ahmad. (2016). V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. p. 5-6. 565-571. Available at: `https://arxiv.org/pdf/1606.04797v1.pdf`

[22] Cox, M. A., & Cox, T. F. (2008). Multidimensional scaling. In Handbook of data visualization (pp. 315-347). Springer, Berlin, Heidelberg. p. 317-318. Available at: `https://link.springer.com/content/pdf/10.1007%2F978-3-540-33037-0.pdf`

[23] MATLAB. (2018). 9.7.0.1190202 (R2020a). Natick, Massachusetts: The MathWorks Inc. Available at: `https://se.mathworks.com/help/images/ref/bwdist.html` (for documentation regarding distance transforms).

[24] Maurer, Calvin, Rensheng Qi, & Vijay Raghavan. (2003). A Linear Time Algorithm for Computing Exact Euclidean Distance Transforms of Binary Images in Arbitrary Dimensions IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 25, No. 2, February 2003, pp. 265-270.

[25] Chollet, François and others. (2015). Keras. Available at: `https://keras.io` and `https://www.tensorflow.org/api_docs/python/tf/keras/Model#fit` (for documentation regarding multiple outputs) and `https://keras.io/api/layers/activations/` (for documentation regarding sigmoid and soft-max similarity and ReLU).

[26] zhixuhao, unet, (2018), GitHub-repository. Available at: https://github.com/zhixuhao/unet. Accessed on: 2020-04-20.

[27] Smith, N., & Webb, A. (2010). Introduction to Medical Imaging: Physics, Engineering and Clinical Applications (Cambridge Texts in Biomedical Engineering). Cambridge: Cambridge University Press. doi:10.1017/CBO9780511760976

A Appendix

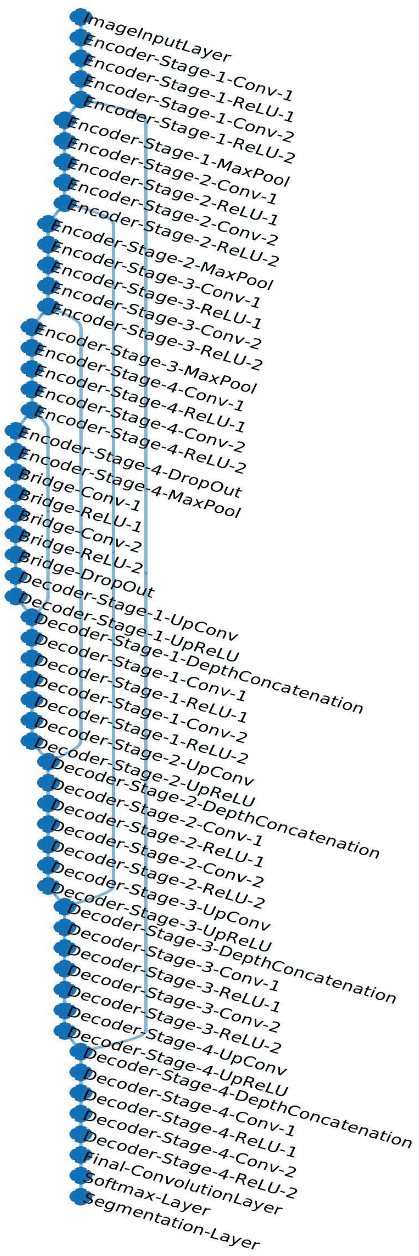

Fig. 19: This figure shows the implementation of the U-Net backbone in MATLAB.

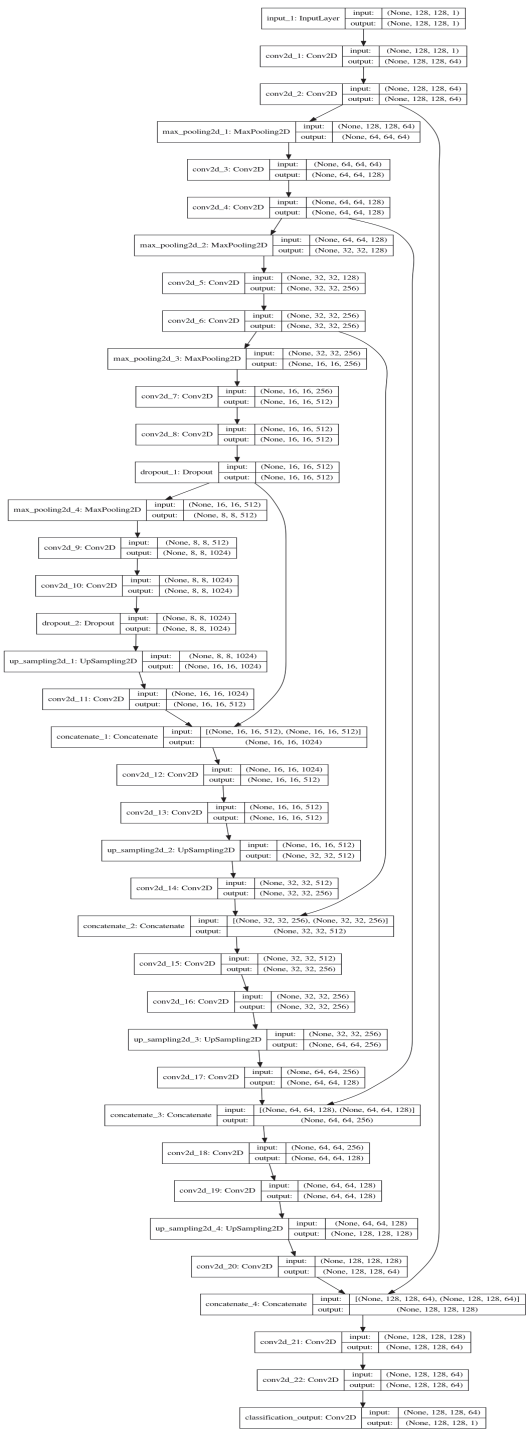
input_1: InputLayer | input: (None, 128, 128, 1) | output: (None, 128, 128, 1)
conv2d_1: Conv2D | input: (None, 128, 128, 1) | output: (None, 128, 128, 64)
conv2d_2: Conv2D | input: (None, 128, 128, 64) | output: (None, 128, 128, 64)
max_pooling2d_1: MaxPooling2D | input: (None, 128, 128, 64) | output: (None, 64, 64, 64)
conv2d_3: Conv2D | input: (None, 64, 64, 64) | output: (None, 64, 64, 128)
conv2d_4: Conv2D | input: (None, 64, 64, 128) | output: (None, 64, 64, 128)
max_pooling2d_2: MaxPooling2D | input: (None, 64, 64, 128) | output: (None, 32, 32, 128)
conv2d_5: Conv2D | input: (None, 32, 32, 128) | output: (None, 32, 32, 256)
conv2d_6: Conv2D | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
max_pooling2d_3: MaxPooling2D | input: (None, 32, 32, 256) | output: (None, 16, 16, 256)
conv2d_7: Conv2D | input: (None, 16, 16, 256) | output: (None, 16, 16, 512)
conv2d_8: Conv2D | input: (None, 16, 16, 512) | output: (None, 16, 16, 512)
dropout_1: Dropout | input: (None, 16, 16, 512) | output: (None, 16, 16, 512)
max_pooling2d_4: MaxPooling2D | input: (None, 16, 16, 512) | output: (None, 8, 8, 512)
conv2d_9: Conv2D | input: (None, 8, 8, 512) | output: (None, 8, 8, 1024)
conv2d_10: Conv2D | input: (None, 8, 8, 1024) | output: (None, 8, 8, 1024)
dropout_2: Dropout | input: (None, 8, 8, 1024) | output: (None, 8, 8, 1024)
up_sampling2d_1: UpSampling2D | input: (None, 8, 8, 1024) | output: (None, 16, 16, 1024)
conv2d_11: Conv2D | input: (None, 16, 16, 1024) | output: (None, 16, 16, 512)
concatenate_1: Concatenate | input: [(None, 16, 16, 512), (None, 16, 16, 512)] | output: (None, 16, 16, 1024)
conv2d_12: Conv2D | input: (None, 16, 16, 1024) | output: (None, 16, 16, 512)
conv2d_13: Conv2D | input: (None, 16, 16, 512) | output: (None, 16, 16, 512)
up_sampling2d_2: UpSampling2D | input: (None, 16, 16, 512) | output: (None, 32, 32, 512)
conv2d_14: Conv2D | input: (None, 32, 32, 512) | output: (None, 32, 32, 256)
concatenate_2: Concatenate | input: [(None, 32, 32, 256), (None, 32, 32, 256)] | output: (None, 32, 32, 512)
conv2d_15: Conv2D | input: (None, 32, 32, 512) | output: (None, 32, 32, 256)
conv2d_16: Conv2D | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
up_sampling2d_3: UpSampling2D | input: (None, 32, 32, 256) | output: (None, 64, 64, 256)
conv2d_17: Conv2D | input: (None, 64, 64, 256) | output: (None, 64, 64, 128)
concatenate_3: Concatenate | input: [(None, 64, 64, 128), (None, 64, 64, 128)] | output: (None, 64, 64, 256)
conv2d_18: Conv2D | input: (None, 64, 64, 256) | output: (None, 64, 64, 128)
conv2d_19: Conv2D | input: (None, 64, 64, 128) | output: (None, 64, 64, 128)
up_sampling2d_4: UpSampling2D | input: (None, 64, 64, 128) | output: (None, 128, 128, 128)
conv2d_20: Conv2D | input: (None, 128, 128, 128) | output: (None, 128, 128, 64)
concatenate_4: Concatenate | input: [(None, 128, 128, 64), (None, 128, 128, 64)] | output: (None, 128, 128, 128)
conv2d_21: Conv2D | input: (None, 128, 128, 128) | output: (None, 128, 128, 64)
conv2d_22: Conv2D | input: (None, 128, 128, 64) | output: (None, 128, 128, 64)
classification_output: Conv2D | input: (None, 128, 128, 64) | output: (None, 128, 128, 1)

Fig. 20: This figure shows the implementation of the U-Net backbone in Keras.

B Appendix

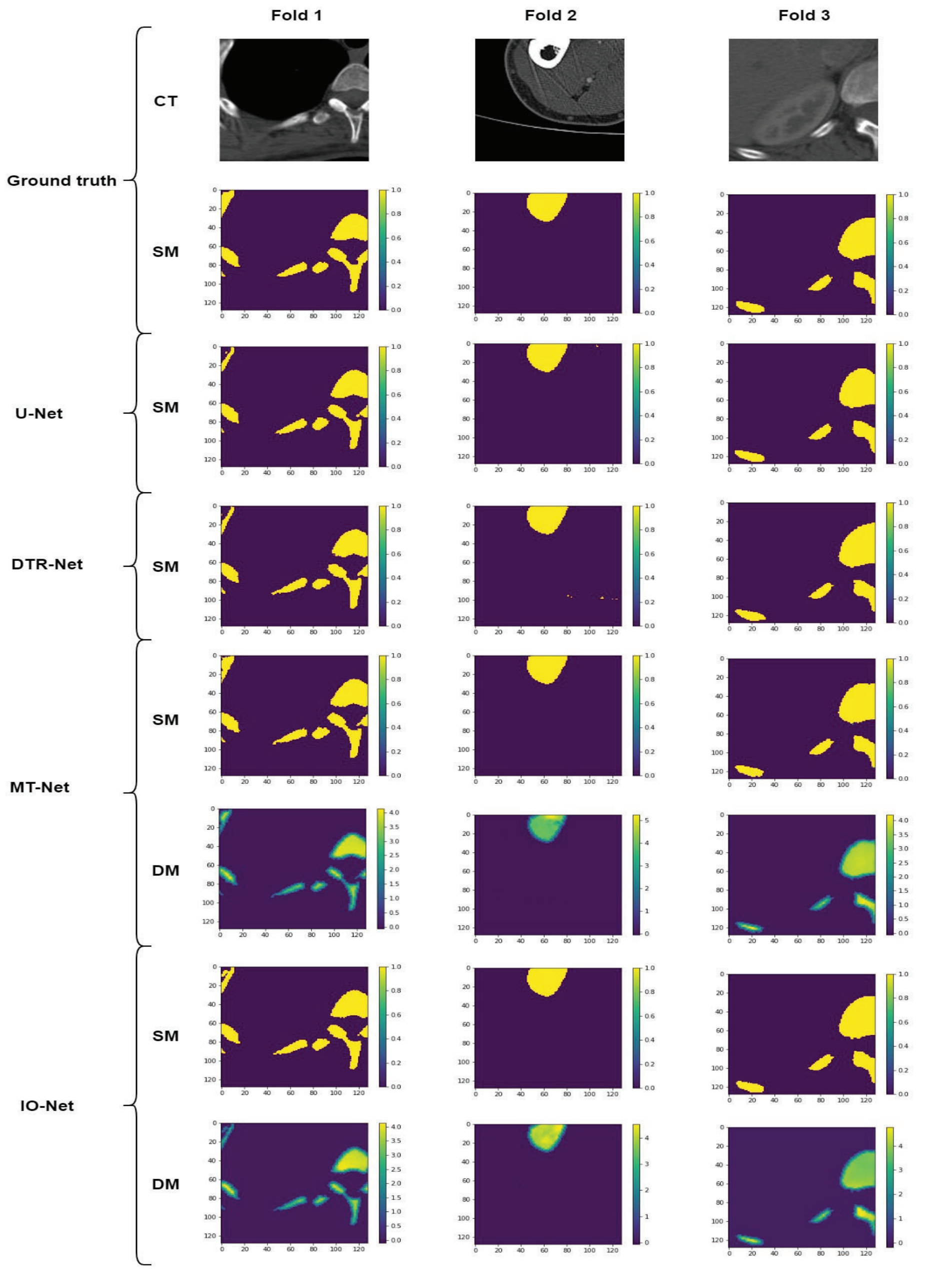

Fig. 21: This figure shows example images from each test fold of the 3-fold cross-validation given by each model on the public dataset with the initial learning rate 10^{-4} and the mini-batch size 4. *CT* denotes the input CT-image, *SM* denotes a segmentation mask and *DM* denotes the distance map.

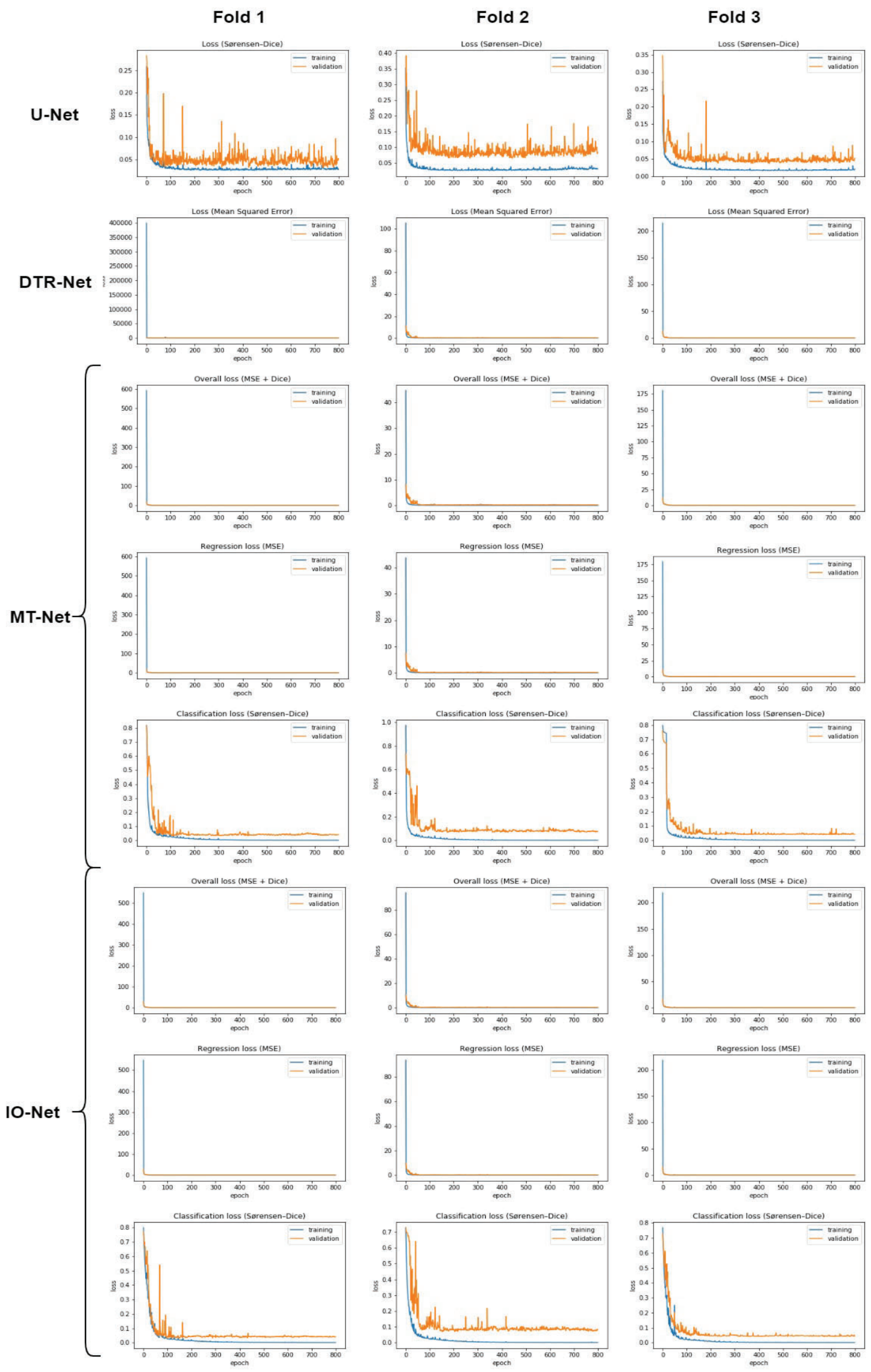

Fig. 22: This figure shows the training losses (blue) and validation losses (orange) from each test fold of the 3-fold cross-validation during training for each model on the public dataset with the initial learning rate 10^{-4} and the mini-batch size 4. For the MT-Net and the IO-Net, three plots are shown. These correspond to the overall loss, the Dice-loss and the MSE-loss, where the overall loss is a combination of the latter. The plotting was done using adaptive scaling on the axes, and was not possible to redo using better scaling.

Master's Theses in Mathematical Sciences 2020:E40
ISSN 1404-6342

LUTFMA-3420-2020

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

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