UCLouvain

École polytechnique de Louvain

Automatic segmentation of the lumbosacral spinal cord

Author: **Nikita DE BROUX**
Supervisors: **Benoît MACQ, Aleksandar JANKOVSKI**
Readers: **Christophe DE VLEESCHOUWER, Sylvain DEFFET**
Academic year 2020–2021
Master [120] in Mathematical Engineering

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

MASTER THESIS

Automatic segmentation of the lumbosacral spinal cord

Author:
Nikita DE BROUX

Supervisors:
Benoît MACQ
Aleksandar JANKOVSKI

A thesis submitted in fulfillment of the requirements for the degree of Master [120] in Mathematical Engineering

at

Applied Mathematics Departement
Ecole Polytechnique de Louvain

June 11, 2021

Declaration of Authorship

I, Nikita DE BROUX, declare that this thesis titled, "Automatic segmentation of the lumbosacral spinal cord" and the work presented in it are my own. I confirm that:

- I performed this work completely during in my Master degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Nikita de Broux

Date: 11 June 2021

UNIVERSITÉ CATHOLIQUE DE LOUVAIN

Abstract

Ecole Polytechnique de Louvain
Applied Mathematics Departement

Master [120] in Mathematical Engineering

Automatic segmentation of the lumbosacral spinal cord

by Nikita DE BROUX

Automatic segmentation of the spinal cord on Medical Resonance (MR) images is crucial in the accurate positioning of spinal cord injuries and the success of the associated treatments. Currently available segmentation algorithms have trouble with the lower part of the spinal cord: the lumbosacral part. This part is often badly segmented or not detected at all. With this work we are trying to close this loophole. Our goal is to create an automatic algorithm dedicated to the lumbosacral segmentation of the spinal cord. This algorithm will be based on an existing one which uses two deep learning models with U-net architecture. The first model is dedicated to the detection of the spinal cord centerline, the second one to the segmentation of images cropped around the detected centerline. By fine-tuning the existing segmentation model it becomes possible to create a U-net model dedicated to the lumbosacral segmentation. The creation of the training set is also part of the work. Manually segmented images are mandatory to have the best possible training. To support this task we have used and improved the CHARP platform developed by the UCLouvain and dedicated to the hosting of medical images and their manual segmentation. To optimize the fine-tuning, we have performed a series of tests to explore the different parameters impacting the training. Theses tests have highlighted the particularities of the U-net model in a fine-tuning process and the importance of the parameters that directly influence the fine-tuning architecture. With the segmentation results of our fine-tuned model, we have identified the two main areas of progress for the fine-tuning and the development of a segmentation algorithm dedicated to the lumbosacral spinal cord. First, we need to fine-tune the centerline detection model similarly to the segmentation model. Very few changes are needed on our current fine-tuning pipeline to support this activity. Second, with only 17 MR images, our training set is too small to ensure a qualitative fine-tuning. Our fine-tuned model provides promising results on some images but is not yet robust enough. Therefore, a larger training set is crucial to continue to develop the model. Our work provides all the bases and tools to complete our initial goal: the automatic creation of segmentation masks for lumbosacral spinal cord.

Acknowledgements

During the journey that led to this thesis. Many person have supported me. First, for his help, patience, availability and expertise with the CHARP platform, I would like to thanks Sylvain Deffet. It would never have been possible to deliver this work without him.

For his subject proposal, his medical expertise and the time spent at the manual segmenting of spinal cords, I am grateful to my supervisor Dr. Aleksandar Jankovski.

I would like to thanks a lot my supervisor Benoît Macq which was always dedicated to propose new investigations possibilities and give constructive feedbacks on my work.

Also thanks to Christophe De Vleeschouwer who has kindly accepted to read my thesis and be part of my jury.

Finally thanks to my family and friends for their moral support during the whole semester.

Nikita de Broux

Contents

List of Figures

List of Tables

List of Abbreviations

CNN	**C**onvolutional **N**eural **N**etwork
MRI	**M**edical **R**esonance **I**maging
SCI	**S**pinal **C**ord **I**njury
SCT	**S**pinal **C**ord **T**oolbox

Chapter 1

Introduction

1.1 The promises of spinal cord stimulation

Severe spinal cord injury (SCI) is a nightmare for the victim. It causes paralysis and disability of body functions under the lesion. For a long time those injuries have remained incurable. Recently, stimulation of the spinal cord as treatment has been a subject of discussion in the medical and scientific community. Indeed it has been discovered that stimulation of the lumbar spinal cord can generate a motor output like standing and stepping by accessing neural circuitry. It can be used for rehabilitation of motor functions on patients who suffered from chronicle SCI (Minassian and Hofstoetter, 2016). A long-term therapy involving a patient with chronic, motor complete SCI has already been completed. This therapy consists in the use of regular lumbosacral spinal cord epidural stimulation and activity based-training. It has lead to progressive recovery of leg movements and standing without stimulation (Rejc E., 2017). It is a breakthrough in the SCI cure and this field of research around lumbosacral spinal cord stimulation needs to be studied.

It's common sense to say that the more we can stimulate precise zones of the lumbosacral spinal cord and position precisely the SCI, the more we are able to increase the accuracy and the diversity of the treatments. Unfortunately the lumbosacral spinal cord is a complex zone full of nerves and it is a very complex task to correctly delimit its different sections. Even detecting the lumbosacral spinal cord with the common medical image technologies such as Medical Resonance Imaging (MRI) can be tricky. In order to improve its detection, our thesis will focus on the creation of an automatic algorithm for the segmentation of the lumbosacral spinal cord on MR images.

1.2 Segmentation of the spinal cord

The spinal cord is long and thin structure made of nervous tissues which extend from the brainstem to the lumbar part of the vertebral column. The spinal cord and the brain make up the central nervous system. Its primary function is to transmit the nerve signals from the motor cortex to the body. The spinal cord controls also a lot of reflexes and contains spinal interneurons that make up neural circuits which are responsible for the motor control of rhythmic movements like walking. Injuries and cut in the spinal cord can lead to partial or total paralysis.

The spinal cord is contained in the spinal canal which is surrounded and protected by three layers of tissues called the meninges. A cross-section of the spinal cord is composed of white matter in his periphery and grey matter in a butterfly shape in

his center. The gray matter surrounds the central canal which contains cerebrospinal fluid. The cord is made of 31 segments. From each segment one pair of sensory nerve roots and one pair of motor nerve roots branch out. The nerve roots then merge into into bilaterally symmetrical pairs of spinal nerves .

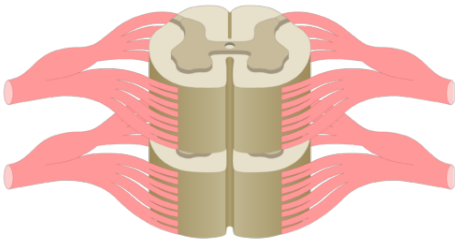

FIGURE 1.1: Cross-section of the spinal cord with the spinal nerves (GBS, 2017)

The first 30 segments are divided in 4 regions: cervical (8 segments), thoracic (12), lumbar (5) and sacral (5) region. The last segment is the coccygeal segment. Such numbering is also used for the vertebrae. Unfortunately since the sizes of the spinal cord and the vertebral column do not match (spinal cord stops between the L1 and L2 vertebrae) the positions of the spinal segments and the corresponding vertebrae are different. The difference is more and more visible when we go down in the spinal cord. In fact the spinal nerves exit the vertebral column at the level of the corresponding vertebra but their root is at the level of the spinal cord segment. It creates a bundle of nerves in the lower part of the spinal cord. This bundle is called the *cauda equina*. We observe this phenomenon on figure 1.2 where we can clearly seen the *cauda equina* at the end of the spinal cord (Wikipedia, 2021).

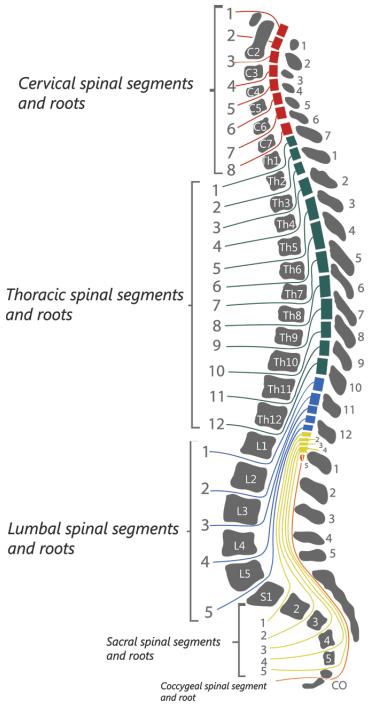

FIGURE 1.2: The spinal segments, spinal nerves and vertebrae (Young, 2019)

1.3 The challenge of finding segment location

1.3.1 A brief review of currently published approaches

Until now the focus concerning segment location has been made on the detection and the localisation of the vertebrae on MR images which is the common technique used in hospitals to visualize the spinal cord. An automatic algorithm based on detection of the intervertebral disks through 3D analysis of intensity profile has been developed for the detection and delimitation of the vertebrae along the entire spinal cord. The robustness of the detection is increased by using a template of vertebral distances created through a training set. The results of this algorithm are very accurate and ensure an almost perfect detection of the vertebrae (Ullmann et al., 2014). However, as we have seen before, the vertebral and spinal cord segment positions do not match. A study has been made on mouse's spinal cord to learn relation between spinal segment positions and vertebral landmarks. The study proposes clear relationships which allow to precisely position the spinal segment in function of the vertebrae. It will help to correctly predict the position of a SCI after an experimental procedure (Harrison et al., 2013).

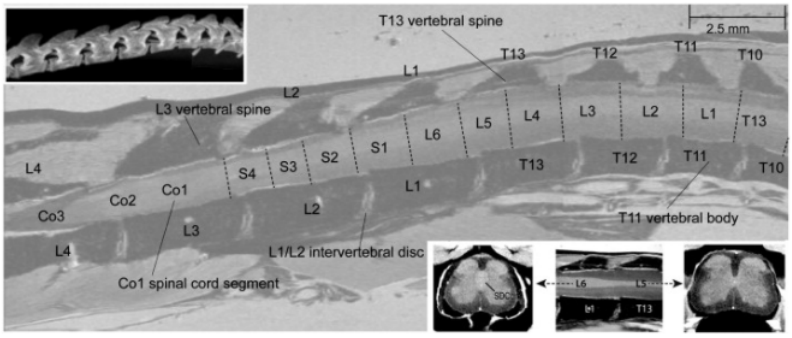

FIGURE 1.3: Relation between verterbrae and mouse spinal segments in the lumbar part (Harrison et al., 2013)

Unfortunately such study does not exist for human spinal cord. With the current technologies that we have, we are not able to reliably identify a specific segment of the spinal cord on MR images. Currently we use vertebral level to refer to spinal cord area. Obviously we can not use them to refer to the spinal segment position due to the difference in location between vertebrae and corresponding spinal segments. This difference increases caudally. When it comes to the lumbar part it become a real issue (Aleksandar Jankovski, 2021). Since the lumbosacral part is the region of interest for the treatments based on spinal cord stimulation, it has a real interest to position correctly the lumbosacral segments.

1.3.2 A new perspective to determine segments location from MR data

A recent study conducted by Dr. Aleksandar Jankovski, one of the principal investigators of the present work, has shown promises in locating segments in the spinal cord.

In his study, 32 post-mortem spinal cords have been dissected and measurements have been taken all along the spinal cords. To our knowledge, this study is the largest one in term of the spinal cord segment analysis through dissection. For every spinal cord the length, width and thickness of each segment have been measured. Using this measures, it has been possible to observe some patterns along the spinal cord. With those data and the relationship between them it is possible to construct a mathematical model which predicts the position of each segment. Unfortunately the length of the segments can not be used in practice since we can not observe the segment's limit on MR images. Width and thickness can be measured correctly (on MR images) and are the two measures used for the creation of such a model. The figures 1.4 and 1.5 refer to the average measures through all the dissected spinal cords for male and female patients (Aleksandar Jankovski, 2021).

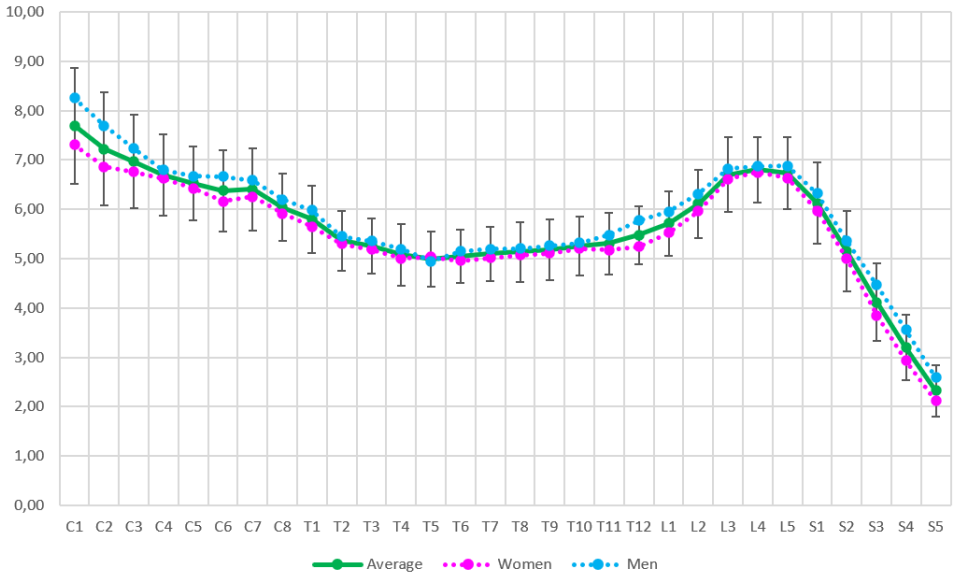

FIGURE 1.4: Comparison between male and female thickness of spinal cord (mm) (Aleksandar Jankovski, 2021)

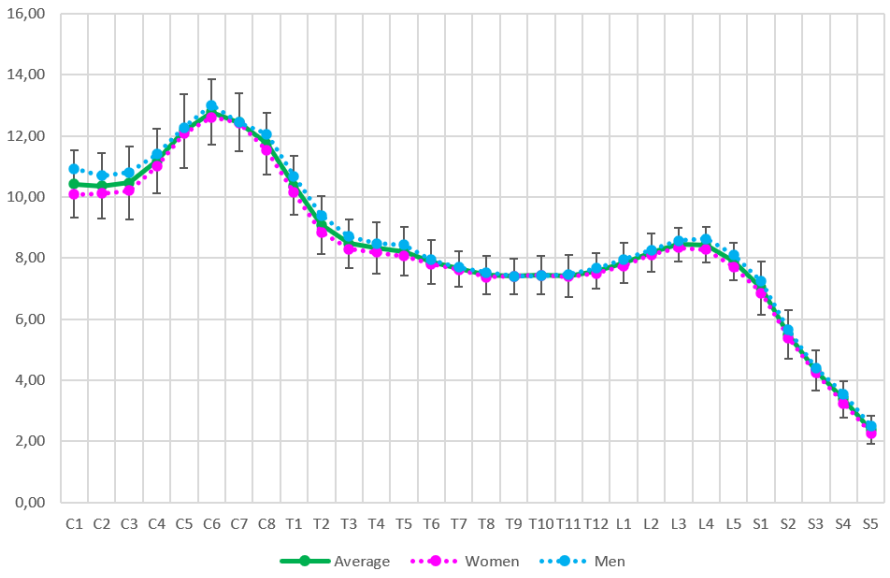

FIGURE 1.5: Comparison between male and female width of spinal cord (mm) (Aleksandar Jankovski, 2021)

1.4 Aim and overview of this Master thesis

The results disclosed by Jankovski et al. show great promises in accurately locating the segments of the spinal cord from MR images but two challenges remain unresolved. First, one has to build a mathematical model to link morphometric characteristics measured on MR images to segments location. Second, one has to build a tool to accurately measure those morphometric characteristics. My Master thesis

focuses on this second challenge.

In order to measure precisely the morphometric characteristics on MR images we need to precisely delimit the spinal cord on it. Since the spinal cord is very small, there is no MR image which only contains the spinal cord. We therefore need to construct an automatic mask of the spinal cord on MR images to take our measurements on the segmented spinal cord.

Segmentation of the spinal cord on medical image is a not a new problem and a lot of algorithms have been proposed during the last years. We will use some of those algorithms in this work. However, all have the same issue: they have difficulties to segment the lumbosacral part of the spinal cord properly. Obviously in our case an incomplete segmentation of the spinal cord has no interest since we need a full mask on the 3D image in order to take the needed measurements. Our thesis will therefore focus on the segmentation of the lumbosacral part of the spinal cord.

Before going into the heart of this work, one may ask why it is so difficult to segment the lower part of the spinal cord. The answer comes from the difference in length between the spinal cord and the vertebral column. The presence of a bunch of nerves around the lower part of the spinal cord leads to a smaller contrast between the spinal cord and the rest of the image. This generates "noise" and prohibits the spinal cord to be in a clean environment. As one can see on the following MR images it is clearly more difficult for human eyes to detect the spinal cord in the lumbosacral part than in the cervical one. While the spinal cord is clearly visible on the figure 1.6 (the grey dot surrounded by the white spinal canal) one can not not distinguish clearly its limit on the figure 1.7 and the manual segmentation on figure 1.8 is needed to clearly identify it. For automatic algorithms it is a true challenge and only a new algorithm dedicated to the lumbosacral segmentation can do it.

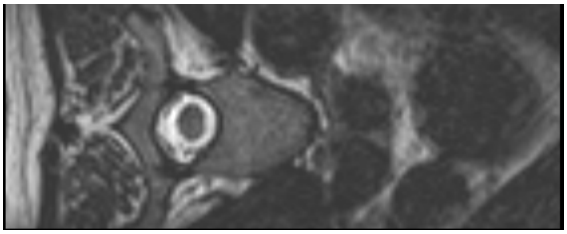

FIGURE 1.6: MRI axial slice in the cervical spinal cord

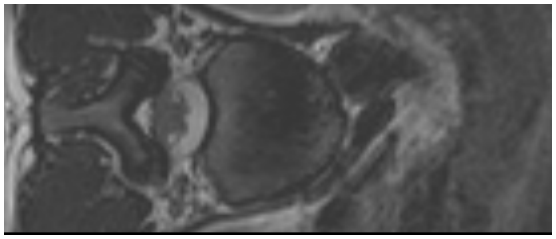

FIGURE 1.7: MRI axial slice in the lumbosacral spinal cord

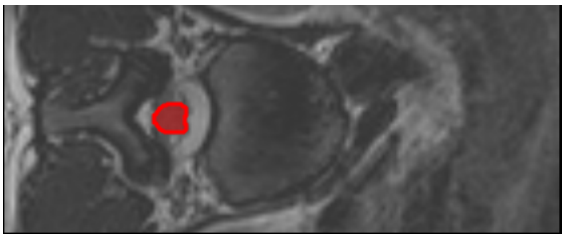

FIGURE 1.8: Manual segmentation of the lumbosacral spinal cord

1.5 Thesis structure

Chapter 2: Prerequisite knowledge

Chapter 2 presents some key concepts mandatory to the good understanding of the thesis. The chapter starts with a brief introduction about image segmentation and presents two generic segmentation techniques that will be used through this work: active contour and deep learning with U-net structure. It ends with a presentation of the classical MR images format files, DICOM and NIFTI.

Chapter 3: Spinal Cord Toolbox

Chapter 3 provides a detailed description of the Python toolbox dedicated to the study of the spinal cord: the Spinal Cord Toolbox. This toolbox has been heavily used in this work through two spinal cord segmentation algorithms. Both algorithms are presented in depth in this chapter. The first one, *propseg*, uses active contour techniques and the the second one, *deepseg*, is based on deep learning with U-net architecture.

Chapter 4: Fine-tuning of existing U-net model

Chapter 4 describes what has been done for the creation of an algorithm dedicated to the segmentation of the lumbosacral spinal cord. It starts with an overview on fine-tuning techniques. It continues with explanations about how we have applied fine-tuning on the existing deep learning model of *deepseg* in order to obtain a model corresponding to our needs. After that, our code is presented in details. It continues with the creation of the training and validation sets and a presentation of the metric used. Then it provides a complete analysis of each parameter impacting the quality of the fine-tuning. It finishes with an overview of the results we obtain and a discussion about it.

Chapter 5: Conclusion

The thesis finishes with a conclusion divided in three parts. It starts with a general overview of the thesis. It is followed by a presentation of the different improvement possibilities and further works that could be done in a later phase. Finally it ends with the contributions of this thesis in the area of spinal cord study.

Chapter 2

Prerequisite knowledge

2.1 Segmentation in medical imaging

During the last decades, medical imaging has become essential in diagnosis and treatments. Different techniques of medical imaging exist like X-ray image, computed tomography and the technique that we use in this thesis: Medical Resonance Imaging (MRI). This technique provides 3D images divided in parallel slices with high contrasts between tissues. Segmentation is one of the most important field of interest in image processing. The objective in image segmentation is to define a Region Of Interest (ROI) on the image via semi-automatic or automatic algorithms (Norouzi et al., 2014). In our case the ROI is the spinal cord.

Different segmentation techniques have been developed through the years. In this thesis two different techniques are used: active contour method and deep learning.

2.2 Active contour method

The idea behind active contour is to use a deformable spline called *snake* which minimizes an energy function to fit the contours of the ROI. The *snake* is defined as a set of points which forms a polygon in 2D and a polyhedron in 3D. The energy function to minimize is composed of three types of energies: the internal, image and constraint energies. The internal energy reflects the smoothness and continuity of the contour (the *snake*). The image energy is based on the detailed characteristics of the image. Its role is to guide the contour to the edges of the ROI by minimizing the function. The constraint energy allows user interactions to guide the *snake*. It will not be used in our case as we focus on automatic segmentation. Minimizing the sum of all energies leads to a smooth contour which fits with the edges of the ROI. The minimization of the energy function is done through an iterative process using gradient descent (Wikipedia, 2020). In this work we will apply a set of active contour techniques for the segmentation of the spinal cord.

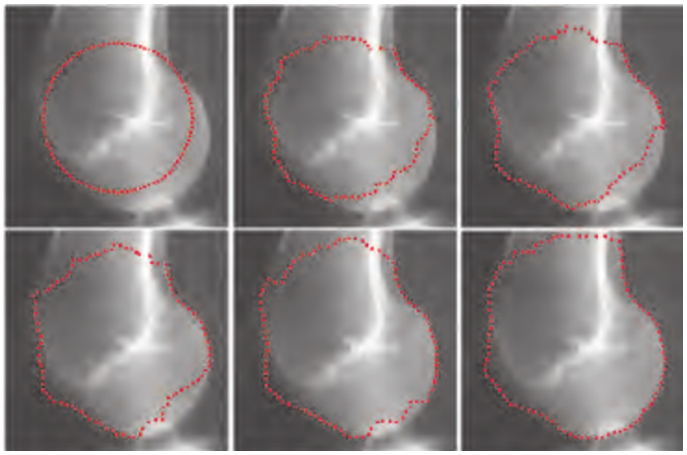

FIGURE 2.1: Iterative process of an active contour segmentation (Olivier and Paulhac, 2011)

2.3 Deep learning

2.3.1 The basics

Deep learning algorithms have been more and more used during the past years. Most of the new segmentation algorithms have been designed using deep learning techniques. The main idea behind this concept is to use a set of input data combined with their output result in order to learn to a generic model how to predict the desired outputs and become a specific model dedicated to the trained task. In our case the input data will be MR images and the output will be the spinal cord masks. In practice the algorithm will try to minimize the difference between the real output and its predicted output for the available set. We call this set the *training set*.

With this training the model learns the recurrent patterns from the input to the output. Once trained, the model can predict with a certain degree of precision the output for new input data. This technique is called deep learning because the model is composed of several layers: the input layer, the output layer and the hidden layers. The input layer is the layer where we feed the model with our data. The output layer is the final layer of the model where it gives its prediction. The hidden layers are all the layers between the input and the output. All the layers are composed of nodes. Information is passed through the layers via a network of links between the nodes. In a classical fully connected network, each node of a layer is linked to each node of the next layer as on figure 2.2. An adjustable weight is associated to each link. Similarly, an adjustable bias is associated to each node. The values of these weights evolve during the training in order to obtain a better prediction. An activation function is also associated to the nodes. This function is applied to the data that goes through the node and provides the output value transmitted by the links. The activation function is generally non-linear and allows the model to learn non-linear relationships which is a key element of deep learning. The hidden layers will learn more and more complex patterns as we go deep in the network.

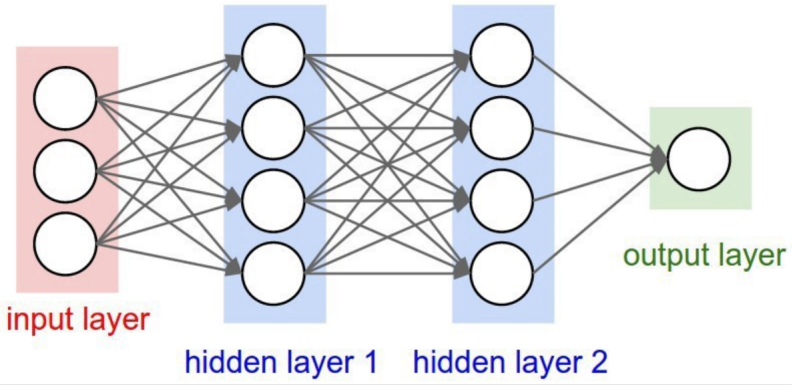

FIGURE 2.2: Classical fully connected network (Liang, 2018)

The optimization of the parameters uses *back propagation*. First the input data of the *training set* is fed into the model and flows through all the network to the output layer. The difference between the expected output and the calculated output is computed using a *loss function*. The goal is to minimize this function by adapting the weights and bias. By using gradient descent from the end to the beginning of the network and adapt the parameters through the way, the algorithm will upgrade its prediction. Ultimately the model can predict very complex function. Since deep learning can be used in almost all situations it is a very powerful tool. Increasing the number of layers allows to predict very complex function with very simple activation functions (Liang, 2018).

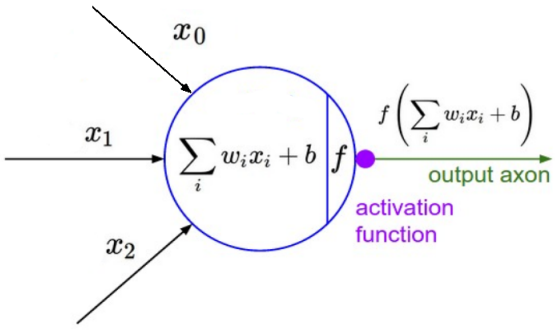

FIGURE 2.3: Structure of a node in the network (Liang, 2018)

2.3.2 Convolutional Neural Network

When dealing with images as input data such as in our work we can not use the classical fully connected network as just presented. Indeed the amount of input nodes (pixel in case of an image) is very high and the amount of adjustable parameters would be way too large. Classical networks are also not translation invariant. Imagine one wants to detect a cat on images and you have two set of images: one with

cats in the top left corner and one with cats in the bottom right corner. If the network is trained with the first set it will never detect cats in the second set since only the weights for to the top left pixels will be relevant. In case of image processing we will use a Convolutional Neural Network (CNN). In a CNN, all the weights are replaced by filters. The filter is an array of values which is moved all across the image to perform a convolution operation with the image part by part. The filter aggregates the influence of nearby pixels. In a CNN, the values of the filter are optimized. It allows to have way less values to optimize since they are now common for the whole image. It is also translation invariant since the filter moves all across the image. More than one filter can be used on a single image.

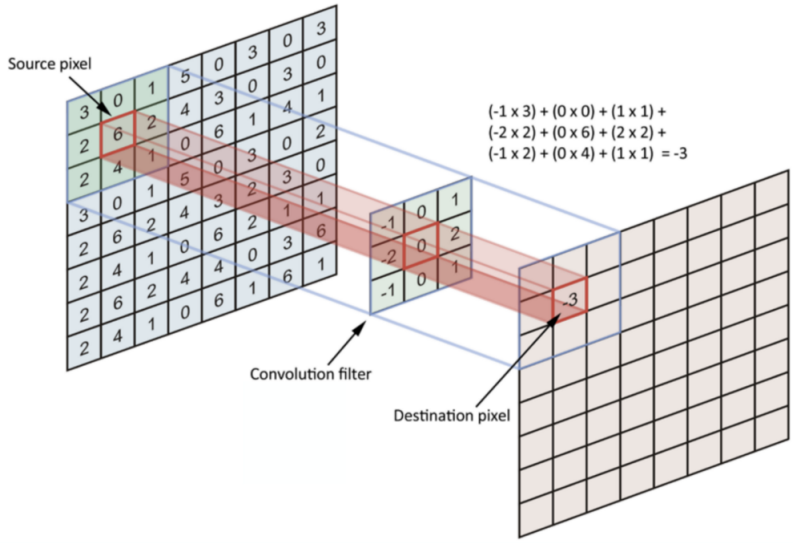

FIGURE 2.4: Convolutional layer with the input map, the filter and the output map (Stewart, 2019)

Each filter produces an array of the same dimension as the input called feature map. A set of feature maps constitutes the output of a convolutional layer. These maps are taken through an activation function like in a classical network. Convolutional layers are most of the time combined with pooling layers which reduce the dimension of the feature maps (Stewart, 2019). The most common pooling layer is *max pooling* which keeps the maximum value in a square of pixels and discards other values. An example of a classical CNN is provided in figure 2.5.

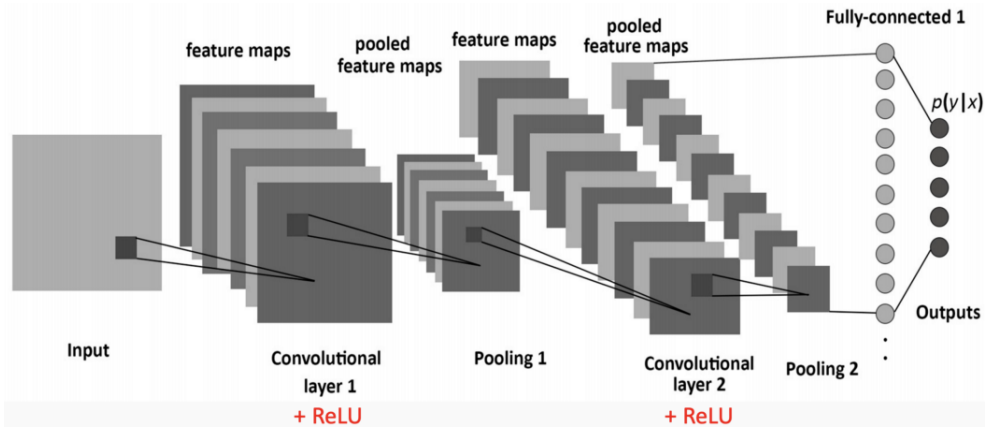

FIGURE 2.5: A classical CNN for classification task (Stewart, 2019)

At the end of the process, the outputs are single values. Such CNN would be used for classification. In segmentation tasks the output of the CNN will be an image with the same dimensions as the input. Specific CNN architecture used for segmentation task is detailed in next section.

2.3.3 The U-net architecture

The U-net architecture is a deep learning structure designed especially for image segmentation tasks. Its main advantage is the reduced time needed for the training. Some parts of the training are reused multiple times and good performances are obtained with small training sets. This structure has been developed in the context of the International Symposium on Biomedical Imaging (ISBI) challenge for segmentation of neuronal structures in electron microscopic stacks in 2015 (Ronneberger, Fischer, and Brox, 2015).

The main difficulty in deep learning with image segmentation is reconstructing the segmented image from the vector derived from the feature maps (represented at the Fully-connected 1 layer on figure 2.5). In order to obtain an image as output from a CNN we need to reproduce the CNN described in section 2.3.2 in reverse way which is more difficult since we need to increase the dimensions instead of reducing them. U-net architecture uses the feature maps learned form the conversion of the image to a vector in order to construct the segmented image from the vector.

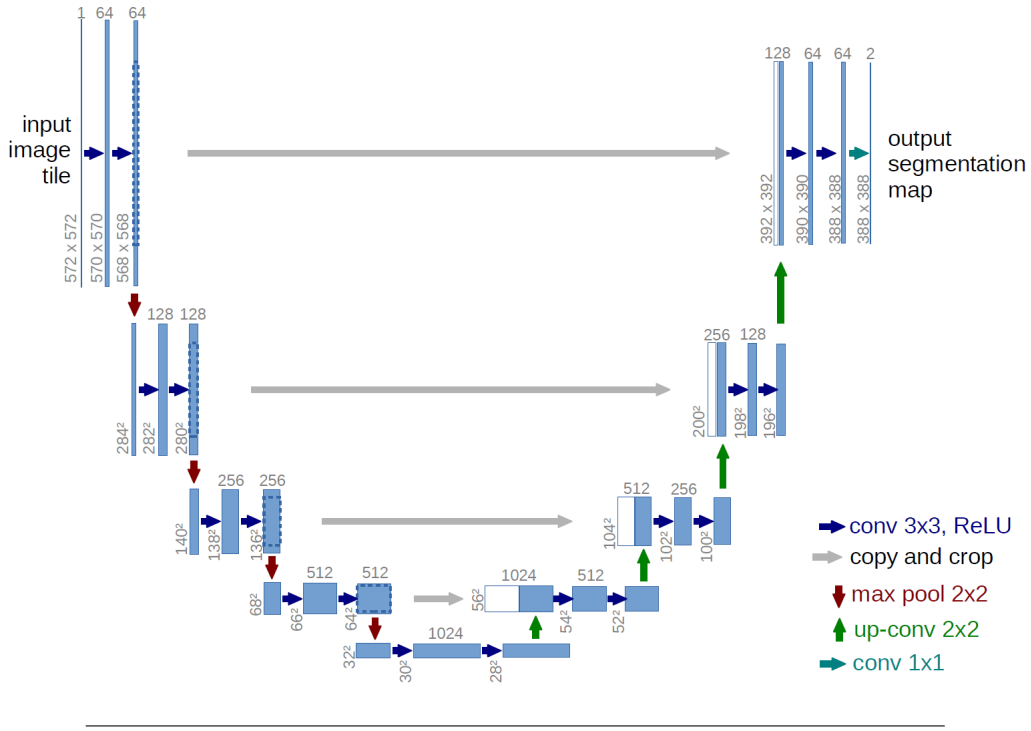

FIGURE 2.6: The U-net architecture (Sankesara, 2019)

The U-net architecture is composed of three sections: The contraction, the bottleneck, and the expansion section.

The contraction section consists of the repeated application of two 3x3 convolutions. Each convolution is followed by a Rectified Linear Unit (ReLU) as activation function. After the two convolutions, a *max pooling* operation on squares of size 2x2 is used for downsampling. At each downsampling step the number of filters is doubled creating twice as much feature maps. It allows to increase the complexity of the model and thus learn more complex structures. There are 4 downsampling steps in the contraction section.

The bottleneck is composed of two 3x3 convolutions followed by a ReLU activation.

The main interest of this architecture lies in the expansion section. The expansion section can be seen as a mirror of the contraction section. A up-convolution operation, also called transpose convolution, is an up-sampling operator. It increases the dimensions of the input (Shibuya, 2017). With a 2x2 up-convolution the dimensions of the image are doubled. Each up-sampling step (4 in total) is composed of a 2x2 up-convolution followed by two 3x3 convolutions and its ReLU activation. In opposition to the contraction section the number of filters is divided by 2 at each step. The architecture finishes with 1x1 convolution used to produce the output. At the beginning of each up-sampling step of the expansion section the feature maps of the corresponding downsampling step is appended to the input. This action ensures that the features that are learned while contracting the image will be reused to reconstruct it (Sankesara, 2019).

The U-net architecture is now widely used for segmentation in medical imaging

since it is quite fast due to its reuse of feature maps. With this reuse less images than with classical architectures are needed for the training.

2.4 Medical image formats

In 3D medical imaging two main format of image are used: DICOM and NIFTI. DICOM is the format used for the image delivered by the scanner. A complete DICOM folder consist of one header which contains information about the patient (name, gender, number of frames, modality, etc.) and a set of files with the image data. For a single acquisition there will be one folder containing multiple DICOM image files containing the header and the frames of the image. In opposition to the DICOM format, NIFTI allows to stock the medical image in a single file under the form of a 3D array. It also contain a affine matrix which position each voxel index in the spatial location (Roy, 2017).

Since the NIFTI format provides the complete image data in a single file, it is the preferred format in a Python environment: the image can be processed as a simple 3D array. Fortunately, there are multiple tools available to convert DICOM files (which is the format used in most medical databases) to NIFTI file. We have the program *dicom2nifti.exe* for the conversion. Nicolas Delinte, a researcher from UCLouvain, provided us this tool as part of course *Digital processing of medical images (WSBIM2243)*. At some point in our work we will need to make the reverse conversion which is less common. We will detail the process in the corresponding section.

Chapter 3

Spinal Cord Toolbox

I have decided to use the Spinal Cord Toolbox as basis for my work. The Spinal Cord Toolbox (SCT) is a free open-source software dedicated to the processing of spinal cord MR images. SCT contains a lot of functions for working on the spinal cord in several different fields (De Leener et al., 2017). In our case we will obviously use the functions dedicated to the segmentation of the spinal cord. Most of the functions in the SCT are the state-of-the-art in their field. A lot of recent works in the spinal cord domain have used the SCT in their research (SCT, 2021). SCT works on Unix environment. The complete Application Programming Interface (API) of SCT used for this thesis is available at https://spinalcordtoolbox.com/en/stable/dev_section/api.html.

Multiple papers describe spinal cord segmentation tools. We have had to choose the algorithms we will enhance for the lumbosacral segmentation purpose. It has been decided to use SCT for two main reasons.

First, the SCT proposes two algorithms for the spinal cord segmentation and both of them are quite recent. The large amount of studies using the SCT to achieve their results is a good hint about the quality of the proposed algorithms. Moreover one of those algorithms is based on deep learning techniques. As said earlier deep learning is a very powerful tool and is also very flexible! Existing deep learning models can be used to create new models dedicated to similar tasks. Having access to a deep learning algorithm for the spinal cord segmentation is a key advantage for the creation of an algorithm specialised in the lumbosacral segmentation.

Second, the SCT and its algorithms are open-source and very easy to use. Moreover it has an active community of developers which make sure that the SCT stays at the top of the current techniques.

3.1 Function *propseg*

This algorithm has been developed in 2014 by researchers from Polytechnique Montréal and the University of Monréal (De Leener, Kadoury, and Cohen-Adad, 2014). It is totally automatic and is designed to segment the entire spinal cord. Unfortunately the results of the segmentation in the lumbosacral part are not precise enough for taking accurate cross-sectional measurements.

The algorithm consists in the iterated propagation of a deformable 3D mesh which ends up corresponding to the spinal cord. It is divided into two modules: the detection and the propagation module.

The detection module consist in finding spinal cord position and orientation. The module is divided in three phases followed by a validation step. It starts by the automatic selection of an axial slice (e.g. the middle one). Using the symmetry of the body, the medial antero-posterior line that passes through the spinal cord is detected. The medial antero-posterior line position is computed by maximizing the mutual information between the two parts of the image separated by the line (the line defines the middle of the symmetry). A restrained image is then created by cropping a region of 5 cm length in the left-right direction of the symmetrical line. Due to the circular shape of the spinal cord, the second phase consist in performing a circular Hough transformation (Ballard, 1981) on the cropped image. Since other circular shape structure can be present on the image only circles embedded in other circles are kept (the spinal cord lies in the spinal canal). Note also that a stretching factor is applied on the Hough transformation in order to detect elliptical shapes which correspond more to the spinal cord shape in certain parts. Finally those two steps are repeated on 10 axial slices, 5 rostral and 5 caudal to the starting plane (each separated by several millimeters) in order to improve detection rate. The neighbouring points (between different slices) of the detected structures are then connected. It is assumed the longest connected chain is the spinal cord. In order to ensure the spinal cord was detected, a validation step is executed. This validation uses a classification method based on easily computable metrics.

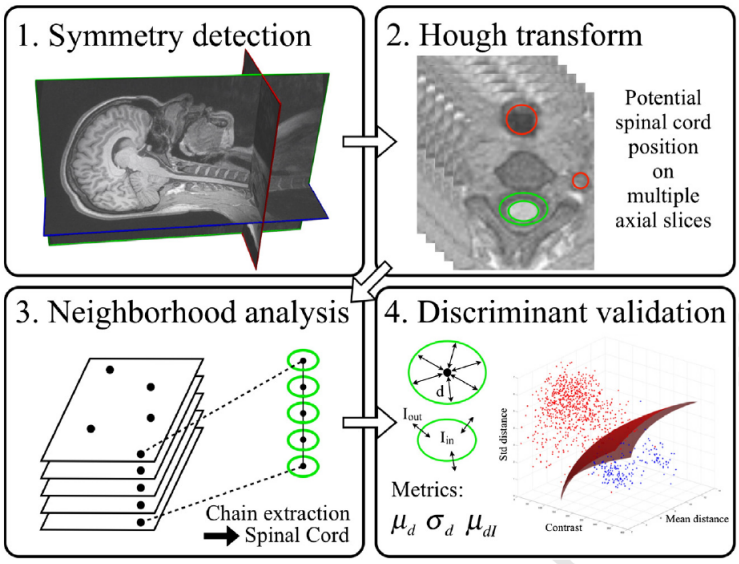

FIGURE 3.1: Workflow of the detection module (De Leener, Kadoury, and Cohen-Adad, 2014)

Using the information collected in the detection module, a triangular tubular mesh is created. During the propagation module this mesh will be transformed to correspond in the best way possible to the spinal cord. This iterative process is divided in two phases.

The first phase consists in selecting the most promising points where the mesh has to be deformed. The second phase consists in the use of active contour techniques

where each selected point is displaced to minimize an energy function. Once the mesh is deformed one section M_i of it is duplicated and translated in order to propagate the mesh. A different energy term is maximized in order to determine the orientation of the mesh at the new points. The process is mainly inspired from the work of Kaus et al. (Kaus et al., 2003). For computational conveniences the iterative process is first computed on a low-resolution mesh and second on a refined mesh interpolated from the first mesh once it has been deformed and propagated. This provides a complete automated algorithm for the segmentation of the spinal cord. In section 3.3 we will show the entire spinal cord is detected well but the segmentation remains quite coarse in the lumbosacral section and it propagates too far compared to reality.

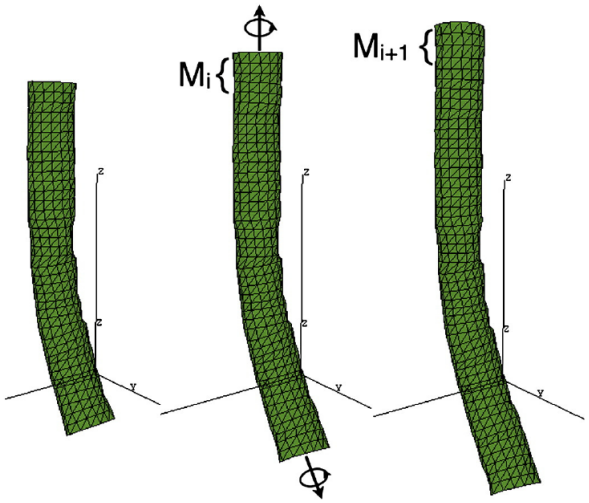

FIGURE 3.2: Propagation of the deformable mesh (De Leener, Kadoury, and Cohen-Adad, 2014)

3.2 Function *deepseg*

Researchers of the University of Montreal have developed in 2019 a new segmentation algorithm for the spinal cord using deep learning and U-net architecture (Gros et al., 2019). It contains also a model designed for the detection of spinal cord sclerosis. Since sclerosis detection is not part of this thesis, we did not use and we will not describe this model.

The new segmentation algorithm is divided in two different convolutional structures which are applied one after the other to the image.

On classical MR images, spinal cord voxels represent less than 1% of the total amount of voxels. The role of the first CNN is to find the center of the spinal cord such that the image is then cropped around it in order to remove the useless information. This CNN uses 2D convolution and is a slightly derivation of the classical U-net architecture. The number of downsampling steps is reduced from 4 to 2. This is possible because the conventional convolution are replaced by dilated convolution in the contraction section. A dilated convolution is a convolution which uses a sparse filter which provides an exponential expansion of the receptive view (the

"window" of pixels that is scanned through the filter for each convolution). Due to the larger receptive view, this process captures more contextual information compared to a classical convolution with the same amount of parameters in the filter. As the dimensions of the feature maps will decrease quicker after each convolutional layer, it allows to use less downsampling steps . The CNN creates a prediction mask which indicates the degree of confidence each voxel is part of the spinal cord. The centerline of the spinal cord is then computed using *OptiC*. This is a fast global-curve optimisation algorithm, which regularises the centerline continuity along the Superior-to-Inferior axis (Gros et al., 2018) .

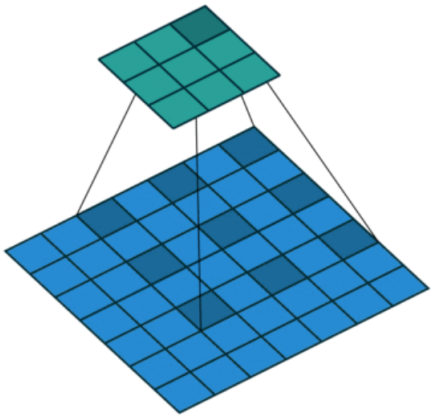

FIGURE 3.3: A 2D dilated convolution with a receptive view of size 5x5 and a filter of 9 parameters (output image in green) (Pröve, 2017)

The second CNN uses 3D convolutions and is computed in a volume around the spinal cord centerline. From the volume, 3D patches of size 64x64x48 are extracted. Intensity normalization is applied on the patches to homogenise the intensity distribution on standardised intensity range (Nyúl and Udupa, 1999). We also normalise the patch intensities a second time by centering the mean and normalising the standard deviation. This CNN is a 3D derivation of the 2D U-net architecture, with 3D convolution (Çiçek et al., 2016). The role of this CNN is to segment the spinal cord. Its output is a binary mask with same dimensions as the input image. The full *deepseg* framework can be applied through a SCT function on NIFTI image and allows different options choices such as the contrast of the image. The exact structure of the second CNN is shown on figure 3.4 to clearly identify the differences with a classical 2D U-net architecture.

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 1, 64, 64, 4	0	
conv3d_1 (Conv3D)	(None, 24, 64, 64, 4	672	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 24, 64, 64, 4	96	conv3d_1[0][0]
activation_1 (Activation)	(None, 24, 64, 64, 4	0	batch_normalization_1[0][0]
conv3d_2 (Conv3D)	(None, 48, 64, 64, 4	31152	activation_1[0][0]
batch_normalization_2 (BatchNor	(None, 48, 64, 64, 4	192	conv3d_2[0][0]
activation_2 (Activation)	(None, 48, 64, 64, 4	0	batch_normalization_2[0][0]
max_pooling3d_1 (MaxPooling3D)	(None, 48, 32, 32, 2	0	activation_2[0][0]
conv3d_3 (Conv3D)	(None, 48, 32, 32, 2	62256	max_pooling3d_1[0][0]
batch_normalization_3 (BatchNor	(None, 48, 32, 32, 2	192	conv3d_3[0][0]
activation_3 (Activation)	(None, 48, 32, 32, 2	0	batch_normalization_3[0][0]
conv3d_4 (Conv3D)	(None, 96, 32, 32, 2	124512	activation_3[0][0]
batch_normalization_4 (BatchNor	(None, 96, 32, 32, 2	384	conv3d_4[0][0]
activation_4 (Activation)	(None, 96, 32, 32, 2	0	batch_normalization_4[0][0]
up_sampling3d_1 (UpSampling3D)	(None, 96, 64, 64, 4	0	activation_4[0][0]
concatenate_1 (Concatenate)	(None, 144, 64, 64,	0	up_sampling3d_1[0][0] activation_2[0][0]
conv3d_5 (Conv3D)	(None, 48, 64, 64, 4	186672	concatenate_1[0][0]
batch_normalization_5 (BatchNor	(None, 48, 64, 64, 4	192	conv3d_5[0][0]
activation_5 (Activation)	(None, 48, 64, 64, 4	0	batch_normalization_5[0][0]
conv3d_6 (Conv3D)	(None, 48, 64, 64, 4	62256	activation_5[0][0]
batch_normalization_6 (BatchNor	(None, 48, 64, 64, 4	192	conv3d_6[0][0]
activation_6 (Activation)	(None, 48, 64, 64, 4	0	batch_normalization_6[0][0]
conv3d_7 (Conv3D)	(None, 1, 64, 64, 48	49	activation_6[0][0]
activation_7 (Activation)	(None, 1, 64, 64, 48	0	conv3d_7[0][0]

Total params: 468,817
Trainable params: 468,193
Non-trainable params: 624

FIGURE 3.4: Complete structure of the segmentation model

The model of the second CNN is composed of 24 layers. The 2D convolutions are replaced by 3D ones. The contraction section goes from layers *input_1* to *max_poolind3d_1*. The bottleneck section from *conv3d_3* to *activation_4* and the the expansion section from *up_samplind3d_1* to *activation_7*. Compared to the classical 2D U-net structure there is less pooling and up-sampling (only one of each). The feature maps from the contraction section are appended only one single time to the inputs of the expansion section (layer *concatenate_1*). This U-net structure is made of only one downsampling step and one up-sampling step. In total the model has 468 193 trainable parameters.

The training has been divided by MRI contrast (T1,T2 and T2*) in order to make trained models available for each contrast. A dataset containing 1943 images (151 for T1, 904 for T2 and 888 for T2*) and coming from 30 different centers has been used for the training of the models. The training set contains very heterogeneous images in resolution and orientation which allows the algorithms to work on almost every MR image. One major drawback of the training set is that almost all data do not propose a vertebral coverage that contains the whole spinal cord. Only 3 out of the 17 centers that contributed with more than 20 MR images proposed mainly data that covers the whole spinal cord. For the 14 others, the lower part of the spinal cord which we are especially interested in for this thesis was missing most of the time. We will demonstrate later the segmentation of the lumbosacral spinal cord is not accurate enough with the resulting models. We expect the framework to work for lumbosacral segmentation with an appropriate training set. We will reuse the proposed framework and adapt the training of the second CNN in order to obtain a model able to segment automatically the lumbosacral spinal cord. One should note that despite the weak lumbosacral training *deepseg* is able to segment entire spinal cord on certain MR images (see fig. 4 (Gros et al., 2019)). However we will demonstrate the model fails with some of them.

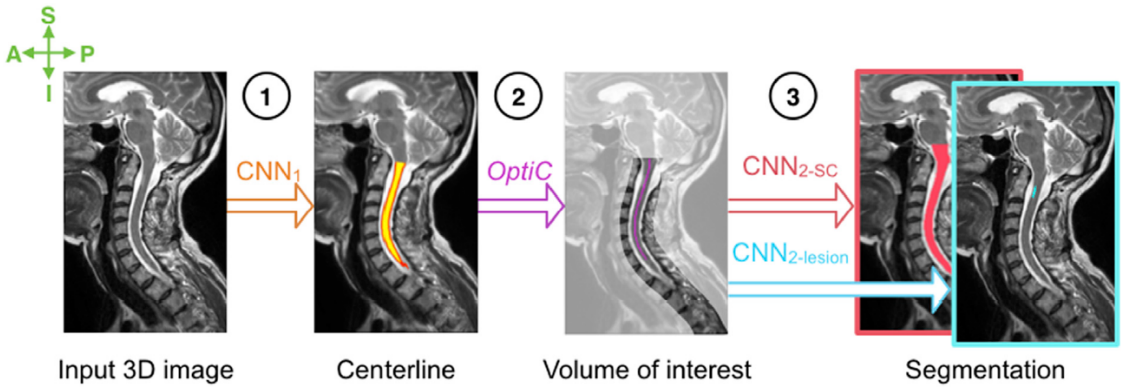

FIGURE 3.5: The journey of an image through *deepseg* (Gros et al., 2019)

3.3 Results of both segmentation function

The segmentation results obtained on complete MRI spinal cord with T2 contrast for both algorithms can be found on figures 3.6 and 3.7 . We received this MR image from Dr. Aleksandar Jankovski. On both figures, the result of the segmentation appears in green while the physical spinal cord end is marked in red. The *deepseg* segmentation stops in the lumbosacral part of the spinal cord. A small portion of the spinal cord was not segmented. The missing part may seems tiny but as highlighted in the introduction, this part is of utmost importance for SCI treatments. Therefore we are looking for the most precise segmentation possible and improvements are clearly possible. The lack of segmentation was expected since there was not enough training on that specific part of the spinal cord in the models used. On the other side, *propseg* detects the entire spinal cord but also detects spinal cord after its physical end. This is obviously not correct. We could imagine to measure the morphometric characteristics, as described in the introduction, on this segmentation until we reach the physical end of the spinal cord. However, when taking a closer look on an axial slice in the lumbosacral area (figures 3.8 and 3.9) one can see the segmentation is quite coarse and is not precise enough for the measurements we need.

Those results prove us the need to develop a specific algorithm for the segmentation of the lumbosacral spinal cord. Those two state-of-the-art algorithms can not segment correctly the end of the spinal cord. To construct our model dedicated to the lumbosacral segmentation we will use the *deepseg* framework and train the existing segmentation model with new data focusing on the lumbosacral part of the spinal cord. Our model will shift from a model dedicated to the segmentation of the complete spinal cord to a new model specialised in the lumbosacral segmentation.

FIGURE 3.6: Segmentation with *deepseg*

FIGURE 3.7: Segmentation with *propseg*

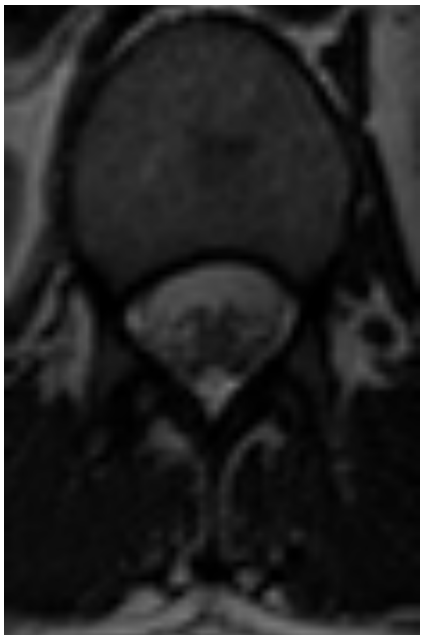

FIGURE 3.8: Lumbosacral axial slice

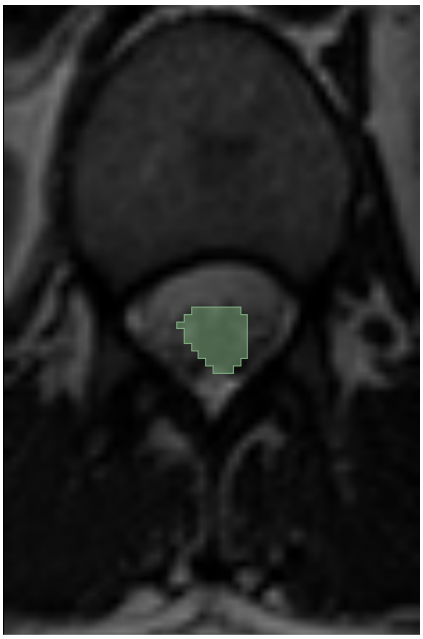

FIGURE 3.9: *Propseg* segmentation

Chapter 4

Fine-tuning of an existing U-net model

4.1 Fine-tuning

4.1.1 The general idea

Fine-tuning an existing model is to reuse an existing trained deep learning model and modify it in a such way it fulfills a different task than the original one. In our case we will modify the trained model of *deepseg* to specifically segment lumbosacral spinal cord. Fine-tuning process can be divided in two phase: the *network surgery* phase and the *unfreezing phase*.

The first phase is called *network surgery* because we replace some layers of the model. A replica of the original model is created without the "tail" of the model. The tail consists of the last layers of the model. The size of the tail depends on the model and the new task. We replace the cut layers with new similar non-trained layers. Before training the model, we need to freeze the layers already trained. It means those layers will not be trained and their weights will not change during the training. This allows to take benefit of the existing learning without completely removing it. We can now train the new model to the new desired task with our specific training set. *Network surgery* allows to use less data in our new set since a big part of the model is already trained. This point will be a key advantage in our case since we do not have a huge set of segmented lumbosacral images.

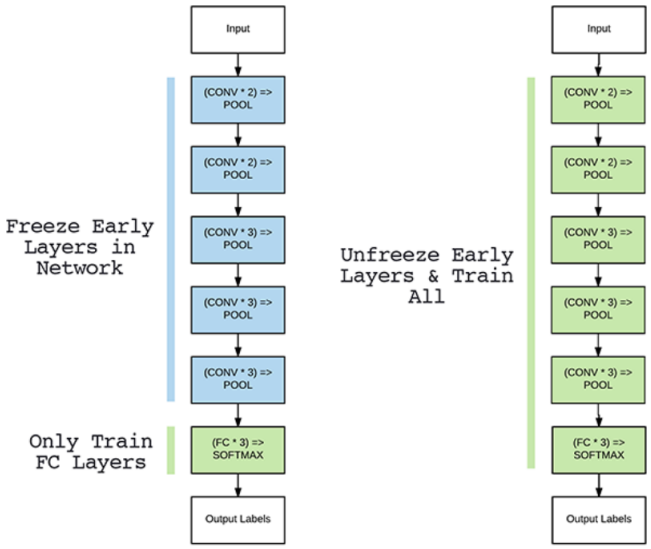

FIGURE 4.1: Evolution of the layers during the *network surgery* and *unfreezing phase* (Rosebrock, 2019)

The *unfreezing phase* takes place during the training of the model. Once the new layers have started to learn patterns we pause the training and unfreeze the all or a part of the frozen layers. We continue the training with all the unfrozen layers with a very low learning rate . The learning rate manages the intensity of the weight updates during the training. A low learning rate will avoid the already trained weights to change drastically. Such *unfreezing phase* allows our model to be more precise than only with *network surgery* (Rosebrock, 2019). We stop the training when the obtained precision reaches our expectations.

4.1.2 Our contribution

Several researchers have worked on the advanced training of Spinal Cord Toolbox models. We have used an existing work as inspiration for our own pipeline or sequence of fine-tuning scripts. The skeleton of our pipeline is directly taken from an existing GitHub repository (Kyathanahally, 2018) where the model for sclerosis segmentation is retrained to increase its efficiency for a particular purpose. As the GitHub repository is 3 years old, we have also adapted the outdated Python code in order to ensure the usage of the SCT functions are aligned with the current version of the SCT.

Observations made on lumbosacral MR images have shown centerline detection and the cropping around it keeps the spinal cord in the cropped image even if the spinal cord is not perfectly centered. An example of automated centerline detection in lumbosacral area is shown on figure 4.2. Therefore we have decided to concentrate on fine-tuning the segmentation model. Our fine-tuning pipeline is divided in two phases: preprocessing and fine-tuning. We have also created a prediction script to use our fine-tuned model instead of the original one in the existing *deepseg* framework.

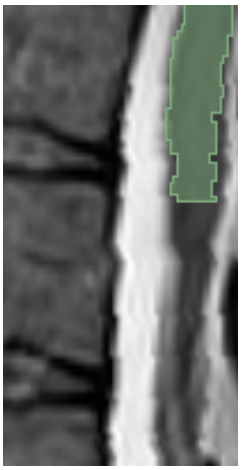

FIGURE 4.2: Cropped image around the spinal cord centerline following automatic centerline detection in lumbosacral area

During the preprocessing we adapt our input NIFTI files such that we can feed them in the 3D CNN of *deepseg*. We start with reorientation and resampling of the images. Then we detect the centerline of the spinal cord. Cropping around the centerline is then applied.

The second phase is the fine-tuning of the 3D model. In order to feed the model we divide our input data in patches of size 64x64x48 which is the dimension expected by the model. Once this is done, we apply a classical fine-tuning framework on the model. The exact parameters of the fine-tuning are determined through a series of optimization tests.

For the creation of the prediction script, two possibilities exists: creating a new script from scratch or adapt the source code of the SCT in order to have *deepseg* using our fine-tuned model in place of the basic segmentation model. After testing both possibilities we have preferred the second one which is way more convenient since almost all the work is already done in the SCT. It takes only very few changes in order to use our model in the SCT framework.

For compatibility purposes with the computer used, we have not worked with the last official release of the SCT but a modified version of this release. SCT still works on Python 3.6 and we are already working on Python 3.7. Fortunately, adapting the SCT to Python 3.7 is a current concern of the SCT developer community. A branch not yet pushed exists for this adaptation (Newton, 2021). We have enhanced this branch to also include more recent versions of Tensorflow and Keras, two deep learning Python libraries, before adapting our local SCT. We have focused our tests for this enhanced branch to the functions of the SCT relevant to our usage which is limited to *deepseg* and some other functions. Therefore, it is important to note our pipeline is **not** designed to work on the current version of SCT which is available online. Our pipeline is designed for a custom version of the SCT which works fine for *deepseg* but has not guaranties to work for the complete set of SCT functions.

4.2 Detailed presentation of our pipeline

In this section, the content of each file of our pipeline is described. The entire code presented and our modified version of SCT can be found on the following GitHub repository:

https://github.com/nidebroux/lumbosacral_segmentation

4.2.1 Preprocessing_script.ipynb

In this script we process our raw training data (NIFTI files of no particular dimension) with their corresponding segmentation mask to obtain NIFTI files that correspond to the expected format after centerline detection in the *deepseg* framework. First, the data is reoriented to match the RPI orientation (Right-to-left, Posterior-to-anterior, Inferior-to-superior) and we resample the data to 0.5mm isotropic images. Performing those two changes is a mandatory prerequisite to use the first CNN of the *deepseg* framework.

Second, We detect the centerline of the spinal cord using the integrated SCT command *sct_get_centerline* which compute the center of mass of the provided segmentation mask on each axial slice to find the centerline. The images are then cropped around the spinal cord and finally intensity normalization on the cropped images is applied. This normalization is required before starting the 3D segmentation CNN.

4.2.2 Main_file.ipynb

This script contains the fine-tuning of the existing *deepseg* segmentation model to obtain our own model dedicated to the lumbosacral spinal cord segmentation. It also contains the tests performed on the different parameters to optimize the fine-tuning.

First the data is divided randomly into two separate sets: the training set which will train the model and the validation set which will be used as benchmark for the quality of the prediction. The validation set is not used for the training to avoid any bias. The training set contains 80 % of the data. The data of both sets of NIFTI files are divided in patches. Those patches are arrays of size 64x64x48 which is the dimension expected by the 3D segmentation CNN. The patch intensities are normalized by centering the mean and normalising the standard deviation. It follows the same treatment as the patches used for the original training. The mean and standard deviation used here are the one from the *deepseg* training set. We then create generators with our training and validation set. Generators are Keras objects which allows to use the GPU memory more efficiently and avoid memory issues with a very large dataset.

Second, the fine-tuning composed of *network surgery* and *unfreezing phase* is performed on the original *deepseg* model. Several tests have been made on the training parameters to ensure the best fine-tuning possible. A detailed review of those tests and their results can be found in section 4.5.

4.2.3 Custom_deepseg.ipynb

The role of this script is to segment data using our fine-tuned model in the *deepseg* framework. With the changes we made in the SCT source code this script is pretty

straightforward. We only need to use the integrated SCT functions to segment our data.

Our fine-tuned model is designed to segment only the lumbosacral part of the spinal cord. We did not perform any tests with complete spinal cord images as input. Therefore, to perform a complete segmentation of the spinal cord, it is mandatory to combine the original model and our fine-tuned mode.

4.2.4 Support files

Different support files are used in the three scripts described above.

- **config_file.py**: Contains a dictionary which has an entry for most of the training parameters and files direction pointing towards the training data and the saving paths of the fine-tuned models.
- **generator.py**: Contains functions to transform the set of data patches into generator objects. It also contains functions for the data augmentation. More details about data augmentation are given in section 4.5.
- **utils.py**: Out of the support files, this is the most important one. It contains the function to create the patches from the NIFTI files and the function to train the model which is an enhanced version of the basic *fit* function available in Keras. During the training, the model is saved each time it achieves better results than the one saved currently. The learning rate drops while training is progressing. Other small functions are also present for data normalization and visualisation purposes.

4.2.5 Changes in the SCT

We can divide the changes made in two distinct categories: the one for custom segmentation purpose and the one for compatibility purpose. The first category is pretty simple. We have added a boolean parameter *custom* to the *sct_deepseg_sc* command. This parameter indicates if the fine-tuned model needs to be used during the segmentation instead of the basic segmentation model or not. The functions related to the command have been adapted accordingly. For the second category we have adapted a non yet pushed branch (Newton, 2021) of SCT GitHub to support recent versions of Tensorflow (2.2) and Keras (2.4.3).

4.3 Training and validation set

In order to be able to use our pipeline and fine-tune the existing deep learning model we need to have a training set. We use thus MR images of lumbosacral spinal cord and the corresponding segmentation mask. The segmentation of the training set needs to be done manually by a specialist. Dr. Aleksandar Jankovski has spent a lot of time in creating these segmentation masks which are a key element to the success of this thesis.

4.3.1 Size of the dataset

The MR Image database is composed of 9 "in-vivo" and 16 "post-mortem" complete spinal cords. Luckily for us all of those MR images have a T2 contrast which allows

us to only fine-tune a single model (which will be designed only for T2 MR images). Since those MR images contain the whole spinal cord we cropped them with the help of Dr. Aleksandar Jankovski to focus only on the lower part of the spinal cord.

Since we need to keep a part of the dataset for the validation set, our training set is even smaller. Having a larger set would obviously be an important improvement and allow a better fine-tuning of the model. Despite this small training set, we will observe segmentation improvements when comparing the results with the basic model.

4.3.2 CHARP platform

For the manual segmentation it is essential to have a platform to upload the MR images and segment them. Dr. Aleksandar Jankovski will create the segmentation masks. I will upload the MR images and download manually the segmentation masks. For this purpose we have used a BETA version of an online platform developed by UCLouvain called CHARP. CHARP allows to host medical images and their masks in DICOM format and proposes a series of tools to create and modify segmentation masks.

Working with Dr. Jankovski and Sylvain Deffet who is involved in the CHARP project, we have adapted the platform to fit as much as possible the expectations of Dr. Jankovski for a segmentation tool. The CHARP platform is still in development at UCLouvain. Our work was one of the first using it for manual segmentation purposes. Since CHARP hosts DICOM files a script was needed to transform our NIFTI files into DICOM. Thanks to the previous work of Sylvain Deffet a Python API for the platform was already existing. It is possible to use this API to convert the NIFTI files into DICOM format and transfer the DICOM files on the server.

In order to have a precise segmentation mask it is important to have voxels as small as possible on the MR image. To do so we have decided to increase the number of voxels per axial slice by a factor of 9 on each image we upload on CHARP. We have used linear interpolation between the voxels to estimate the value of the new voxels.

4.3.3 *Propseg* as approximation of manual segmentation

As manual segmentation is a time-consuming task, it was not possible to obtain a manually segmented dataset early enough to optimize the parameters of the fine-tuning. As workaround, we have decided to simulate manual segmentation with an automatic algorithm for our series of tests presented in section 4.5.

Training our model with data segmented via *deepseg* would have had no sense since the output would have been predicted using the model we want to fine-tune. Therefore, the *propseg* algorithm has been chosen to support the workaround. Choosing *propseg* as segmentation tool might seem weird since we have mentioned in section 3.3 this algorithm may segment outside of the spinal cord. However, it allows us to test our fine-tuning pipeline without waiting for perfect manual segmentation masks.

The obtained segmentation using *propseg* is quite coarse and the quality varies a lot from one MR image to the other. Obviously a model trained with those segmentation masks will not predict good segmentation. However, it can still be used to optimize the parameters of the training since our automatically segmented dataset and the manually segmented one share the same input data.

For 6 of our 25 MR images the automatic segmentation is too bad to be used. We will thus be limited with 19 MR images in that case. 4 of those 19 MR images will be chosen randomly for the validation set.

4.4 Metric and benchmarks of existing algorithms

In image segmentation, to evaluate the accuracy of the model, the similarity between the exact segmentation mask and the predicted one is measured. The most common coefficient for validation of image segmentation is the DICE coefficient. For two set X and Y (binary mask in our case) the DICE coefficient is defined as:

$$DICE \equiv 2 \cdot \frac{|X \cap Y|}{|X| + |Y|}$$

The symbol $\cap$ represent the intersection between two sets. The intersection of two binary sets, is the set where the corresponding element has a non zero value in both sets. The symbol $||$ stands for the number of elements in the set with a non zero value. The DICE coefficient always provides a measure between 0 and 1 (Dr Daniel J Bell, 2019). The greater the value of the indicator when comparing sets, the more the sets X and Y are similar.

For images with a T2 contrast, the respective DICE coefficients of *propseg* and *deepseg* are 0.832 and 0.924. However, those results are obtained on complete spinal cord and do not reflect the quality of the algorithms in the lumbosacral area. Since both algorithms provide high quality results in the upper part of the spinal cord, we will uses those values as benchmarks for our work.

In order to measure the quality of the algorithms in the lumbosacral spinal cord we have calculated the DICE coefficient for our data set. In the example hereunder, we have segmented one of the cropped MR images of our training set with both algorithms. We have used the DICE coefficient to compare the results with a manual segmentation. For lumbosacral segmentation, the DICE coefficients are 0.534 for *propseg* and 0.520 for *deepseg* . These results are very low compared to the ones obtained on the complete spinal cord. The resulting segmentation and the manual one are available on figures 4.3, 4.4 and 4.5. As already mentioned in section 3.3, the *propseg* algorithm continues to segment after the end of the spinal cord where *deepseg* stops the segmentation before reaching the end. Full results for *deepseg* segmentation on our complete dataset are available in table 4.1. They confirm the need of a dedicated algorithm.

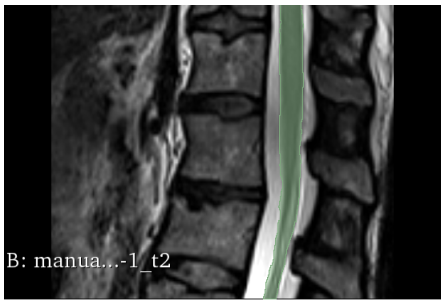

FIGURE 4.3: *Propseg* segmentation (DICE = 0.534)

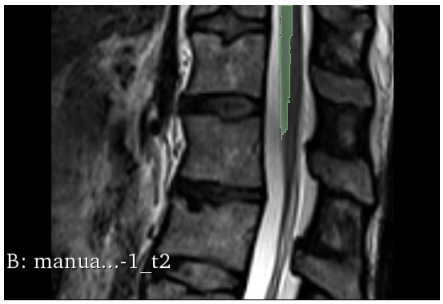

FIGURE 4.4: *Deepseg* segmentation (DICE = 0.520)

FIGURE 4.5: Manual segmentation

MR images	DICE coefficient for *deepseg* segmentation
in-vivo 1	0.534
in-vivo 2	0.738
in-vivo 3	0.71
in-vivo 4	0.702
in-vivo 5	0.474
in-vivo 6	0.701
in-vivo 7	0.498
in-vivo 8	0.623
in-vivo 9	0.646
post-mortem 1	0.542
post-mortem 2	0.702
post-mortem 3	0.721
post-mortem 6	0.610
post-mortem 9	0.037
post-mortem 10	0.089
post-mortem 11	0.497
post-mortem 12	0.640
post-mortem 13	0.737
post-mortem 14	0.190

TABLE 4.1: DICE coefficient of *deepseg* segmentation for each MR image of our dataset

4.5 Parameter analysis

In order to obtain the best possible fine-tuning, we will optimize 6 different parameters impacting the training. To optimize the different parameters, for each of them, we have run the training for different values of the parameter keeping constant values for the other ones. This section will cover the different tests we have made and their results.

As explained in section 4.3.3 we will use a training set segmented with *propseg* for those tests. The assessment of the quality of the model will be done using the loss function value (i.e. the DICE coefficient) of the validation set. In this section our aim is to identify the value for each parameter which provides the best results. At the beginning of the test for a specific parameter, the system splits randomly the data set (19 images) into a training set (15 images) and a validation set (4 images). Due to the small size of the validation set, the overall value of the validation loss will change from one validation set to the other. For this reason we will rather focus on how the model converges to a stable high value of the validation loss to perform our parameter optimization. Therefore, we will not only take into account the value of the parameter which provides the highest validation loss at a specific epoch but the value of the parameter where the evolution of the validation loss remains stable at a high level over the different epochs of the training process.

Our analyze will be visualized on graphs where the evolution of the validation loss

is displayed for different epochs. Due to the small validation set, the validation loss can vary back and forth between consecutive epochs. In order to improve the visualization of the results, we have decided to display the mean of the validation loss for the 5 last epochs for every 5 epochs. The validation loss at the very beginning of the training process is not relevant, therefore the displayed values only starts after at least 20 epochs.

4.5.1 Selected Parameters for optimization

The following parameters have been investigated to evaluate their influence on the quality of the fine-tuning. The role of the tests will be to determine which of the parameters have the most impact and what the optimized values are.

- **The network surgery**: Changing the layers that are replaced by new ones in the model clearly influences the fine-tuning. We will determine the best layers to replace in order to obtain the best possible fine-tuning.
- **Unfreezing phase**: Similarly to the network surgery, the training results are impacted by changing the unfrozen layers during the unfreezing phase.
- **The batch size**: The batch size corresponds to the number of data fed into the model before updating the weights. It can vary between 1 and the size of the training set. If the batch size is equal to the size of the training set, the number of iterations (one iteration = one update of the weights) is equal to the number of epochs (one epoch is done when the complete training set has been processed through the network). If not, the number of iterations will be greater. Generally a small batch size leads to a longer training and a more precise model with a better validation loss. Too small batch sizes can also present the risk of converging to unreliable models (Transfer, 2019). Therefore, the key is to find a good compromise between accuracy and fast training.
- **The overlap**: When dividing the training data into patches of size 64x64x48 an overlap can be added. Without specifying any overlap system will generate the patches using a step of 48 and each voxel will only be included in a single patch. When an overlap value is provided, the system will define the step size as equal to the provided overlap value. In such a scenario, voxels will be included in two or more different patches. This is a way to artificially increase the training set size.
- **The learning rate**: The learning rate controls how quickly the model is adapted to the problem. The value of the learning rate lies between 0 and 1, it has a major influence on the training. A learning rate close to 1 may converge too quickly to a sub-optimal model and a too small learning rate can lead the training to stop progressing (Brownlee, 2019). We have chosen to add a learning rate drop to our training. It avoids our training to oscillate around a sub-optimal solution.
- **Data augmentation**: Data augmentation consists in applying some operations on the training data before the training. It is used to avoid overfitting. Overfitting happens when the model learns to well how to predict the results for the training set and produces poor results for other data. Data augmentation artificially creates new training data. Our overlap is a form of data augmentation by increasing the training set. In our case, the data augmentation consists

in flipping each of the patches randomly along their axis. By default we will perform data augmentation.

4.5.2 Test strategy

Our series of tests begins with an analyze of different possible network surgeries. Once the best network surgery is identified, we will optimize successively the batch size, the overlap and the learning rate during the network surgery.

Next, we focus on the unfreezing phase to decide which layers should be kept frozen during the whole fine-tuning process. A different learning rate is optimized for this phase since the trained layers have changed compared to the network surgery. Overlap and batch size do not need to be optimized for this particular phase. Indeed, overlap can not be changed between the two phases since it changes the training set. The optimal batch size depends mostly on the training set size which remains constant in both phase.

We will also analyze the potential under-performance data augmentation could bring. Once we will have demonstrated data augmentation does not harm our results we will investigate very small overlap values.

We will finish by testing the efficiency of including the optimal network surgery in our fine-tuning framework.

4.5.3 The network surgery

As described in section 4.1.1, in a sequential model, the network surgery normally starts at the last layers of the network. However, we are not using a sequential model where the sequence of the layers is clearly defined. We use a U-net architecture where layers from the contraction section are linked to the layers expansion section. Multiple definition of the last layers could be used. One can argue the layers from the expansion section are the most relevant ones since they are the closest from the output. Another option is to choose the layers from the bottleneck section since they are the further away from the input. A study conducted on the fine-tuning of U-net architecture with ultra sound images shows best training results were achieved by fine-tuning only the contraction section (Amiri, Brooks, and Rivaz, 2020). Although this study does not perform network surgery but only unfreezing, it demonstrates the necessity to test non-classical possibilities for fine-tuning when using U-net architecture.

We will test 5 different network surgeries on our model:

- replacement of the contraction section,
- replacement of the expansion section,
- replacement of both contraction and expansion section,
- replacement of the bottleneck section,
- replacement of the entire model.

All of the 5 tested models have been trained with the same parameters: 100 epochs, learning rate of 0.001, overlap of 24 and batch size of 1. The results are available on figure 4.6.

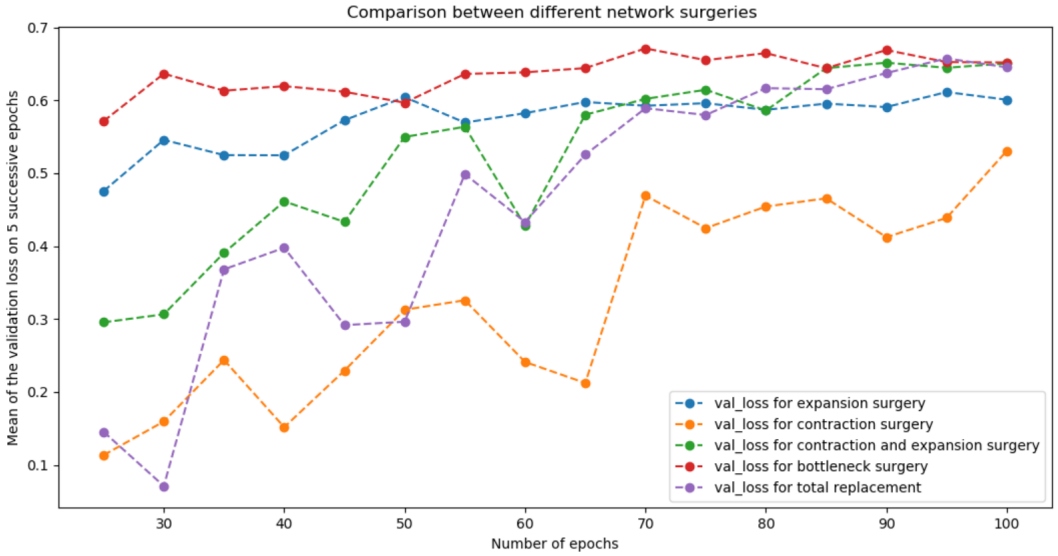

FIGURE 4.6: Comparison between the different possible surgeries

The models retrained with total replacement and the one where both contraction and expansion section have been replaced are the ones where the largest part of the original training are removed. The training of these models is way more unstable than the other models reaching similar good loss values. This clearly indicates the interest of keeping a sufficient part of the original training. Therefore we will directly discard those two options. It is a rather encouraging result since it gives credit to our primary hypothesis which was to reuse the existing very large cervical training as basis for our small lumbosacral training.

Between the 3 remaining models the choice is easy. The best model is clearly the one where the bottleneck section has been replaced: validation loss does not fluctuate a lot and it provides the best stabilized value. By replacing the layers of the bottleneck section, the deepest layers in term of distance from the input are being replaced. If replacing the expansion section shows relatively good results as well, the results by replacing the contraction section are awful. We assume this can be explained by the following statement. The contraction section includes the basic patterns of the learning which do not need to change between initial training and our lumbosacral specific training. Removing the contraction section destroys a solid basis which can not be reconstructed with our small amount of training data.

Our results differ from the ones obtained the study presented above (Amiri, Brooks, and Rivaz, 2020) which recommends to unfreeze the contraction section. The difference in the results may come from the different origin of the training set: MRI in our case, ultrasound in theirs. The difference in architecture might also impact the results. We do not use the classical U-net architecture. Our version is quite 'flat' with only one downsampling and upsampling and uses 3D convolutions. Moreover the study in reference uses natural images for the pretraining and medical ones for the fine-tuning. The interest of fine-tuning the contraction section could be explained

by the necessity to learn some specific low-level medical patterns at the beginning of the network. Since our pretraining is composed of medical images very similar to the one used for fine-tuning we do not have the same needs.

4.5.4 Batch size

For this test, 5 different values of batch size were chosen: 1, 5, 10, 20 and 50. Knowing our training set is consists of 271 patches, it would have been possible to test greater values but those leads to an Out Of Memory error. The models have been trained with 50 epochs for reduced computational time. The results are displayed on figure 4.7.

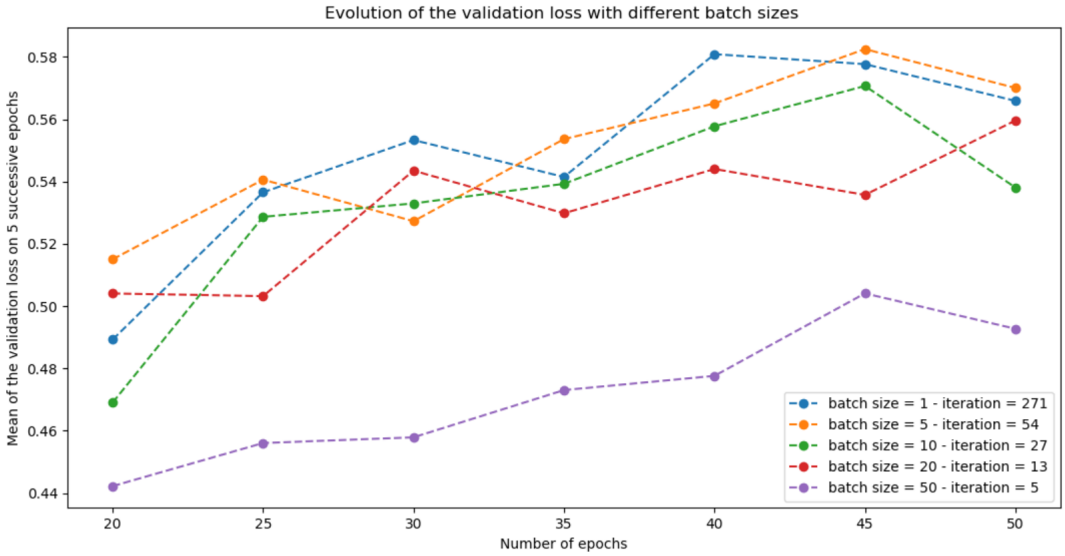

FIGURE 4.7: Comparison between the different batch sizes

The expected tendency is well observed: accuracy of the model decreases when batch size increases. With a batch size of 5, one can observe results are remaining quite similar to the ones with a batch size of 1. The amount of iterations per epoch drops, which means the training is clearly accelerated. In consequence, it has been decided to use a batch size of 5 for the rest of the analysis since it does not affect the quality of the model and allows a faster training.

The attentive observer will notice the loss values obtained here are lower compared to the previous section. If the lower number of epochs is a part of the explanation, the main reason is the validation set which varies from one test to the other. For each test, the system select 4 images randomly out of the 19 available images to define the validation set. Since the training set and the validation sets are very small, the selection of the validation set has a major impact on the results. Additional tests have shown the values of the loss function are impacted, however, the ranking of the results for the different scenarios are not affected. Therefore, it is important to remember the results of the tests should only be used to compare different scenarios tested with the same training set and same validation set but should not be used to compare the results between the different tests.

4.5.5 The overlap

In this section, the training set itself will be impacted by testing different overlap values. The lower the overlap value, the greater the size of the training set. The values tested for the overlap are: No overlap (=48), 42, 36, 24 and 12. We will use a batch size of 5 for all those values. Results are shown in figure 4.8.

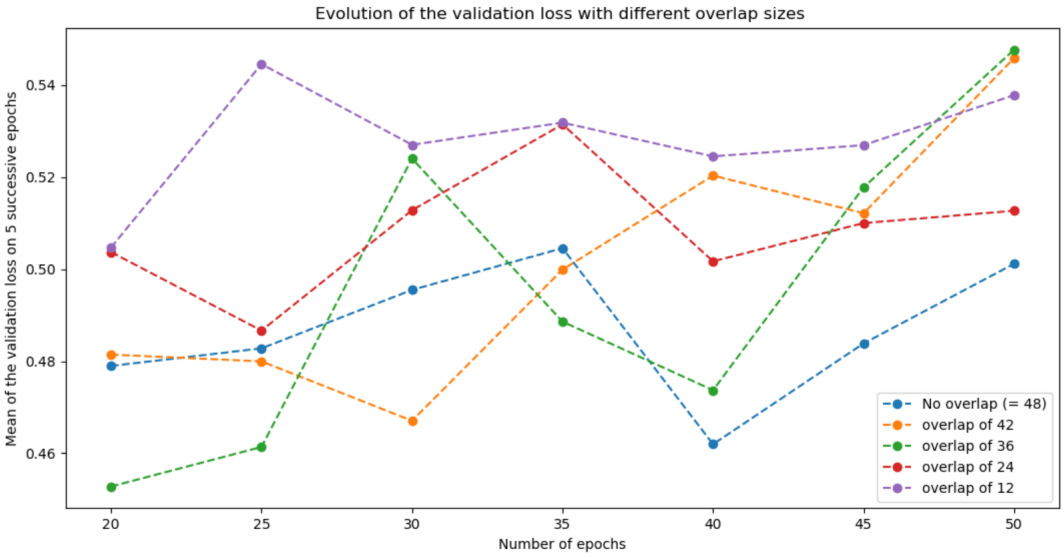

FIGURE 4.8: Comparison between the different overlap values

On the Y axis, the used scale is pretty tight. The observed results only vary slightly for the different scenarios. The overlap of 12 gives clearly the most stable and however still pretty high overall results. This test demonstrates the interest of the overlap which artificially increases the size of the training set. One may ask why we did not test smaller values than 12 which potentially might have provided better results. We did not want to do it at this moment to avoid excessive computational time for the rest of our tests. However, we will test smaller overlap values after the data augmentation section. The results are available in section 4.5.10.

4.5.6 The learning rate during network surgery

The tested learning rates are: 0.1, 0.01, 0.001 and 0.0001. The system also includes a learning rate drop mechanism which divides the learning rate by 2 when the training is on a *plateau* for 10 epochs. We have decided to train for 100 epochs in order to make sure the learning drop mechanism is activated several times.

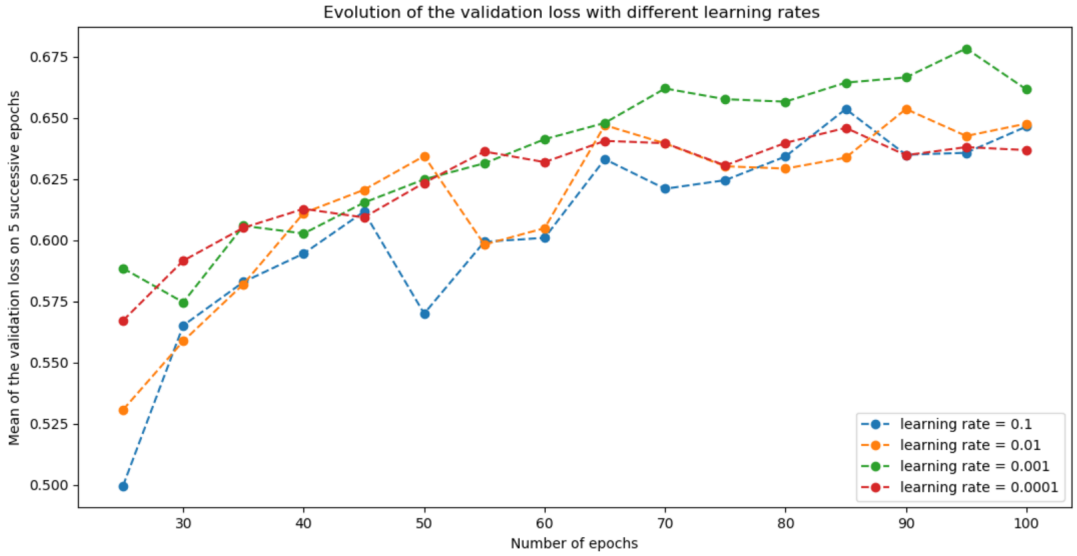

FIGURE 4.9: Comparison between the different learning rates

As we see on figure 4.9 the final loss values are quite similar for all learning rates. However, the stability of the training increases as the learning rate decreases. Even if the learning rate of 0.0001 generates the most stable results, we observe it can not reach a as high loss values as with a learning rate of 0.001. Therefore, the learning rate of 0.001 will be our optimal choice since it combines a stable training with a high loss values.

4.5.7 Unfreezing phase

Next test is dedicated to the unfreezing phase. We test here which layers of the model the system should unfreeze during the fine-tuning. In section 4.5.3 we have decided to use the bottleneck section as the section to be replaced for the network surgery. Therefore, the bottleneck section is always unfrozen during this phase. We will test 3 different unfreezing possibilities : unfreeze the entire model, keep only the contraction section or the expansion section frozen. This second training has been made with the optimal parameters found in the previous tests except for the learning rate. We will not use the optimal learning rate determined for network surgery. For network surgery, layers need to be trained from scratch. When unfreezing layers, since the layers are already pretrained, a lower learning rate has to be used. In this section we will use a learning rate which has been reduced to 10^{-5}. It ensures that only minor changes will be made in the pretrained weights. The results are displayed on figure 4.10. In the next section, we will identify the optimal learning rate to be used when unfreezing layers.

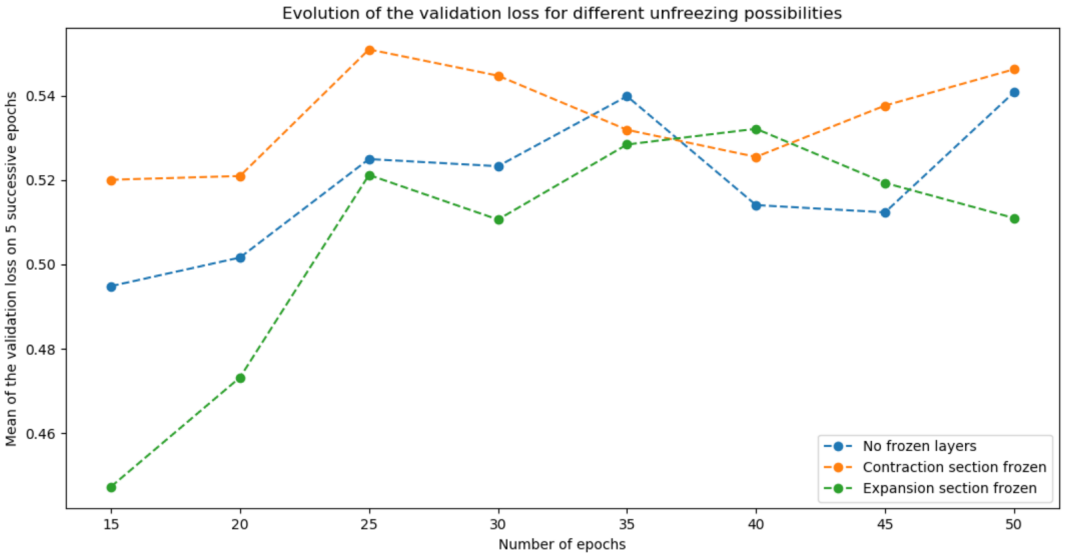

FIGURE 4.10: Comparison between the different unfreezing possibilities

The best loss values are obtained by the model where the contraction section remains frozen. These results are totally aligned with the results of section 4.5.3 where surgery on the contraction section was providing the less optimal results. As already mentioned, this could be explained by the fact that the contraction section learns low-level patterns. As our training data is quite similar to the one used in the pretraining (spinal cord MR images in both case), unfreezing the contraction path can not really improve the fine-tuning and only leads to a less stable training.

4.5.8 The learning rate during unfreezing phase

In the previous section we have selected which layers to unfreeze. We can now explore different possible values for the learning rate for the unfreezing phase. As already explained, in order to achieve a good stability of the training during the unfreezing phase, we will use a lower learning rate compared to the network surgery. The learning rates which have been explored are: 10^{-4}, 10^{-5} and 10^{-6}. We decided not to test smaller learning rates since we think such low learning rates will barely have no effect on the weights. Results will confirm this assumption. In order to observe the real effects of the unfreezing phase on the quality of the results, we have also decided to include in the graph the validation loss evolution during network surgery phase where we used the optimal a learning rate of 0.001. Results are aggregated in figure 4.11.

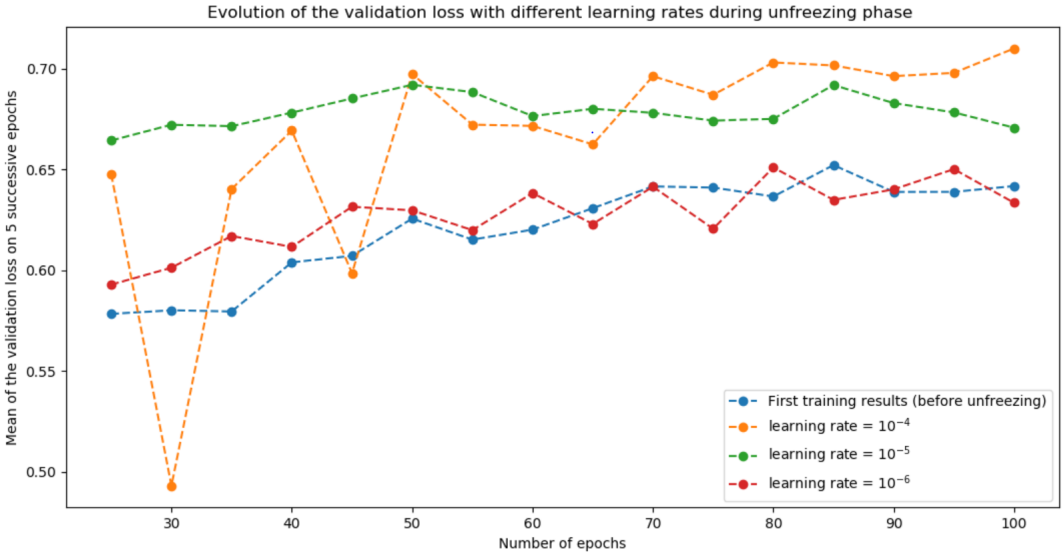

FIGURE 4.11: Comparison between the different learning rates during the unfreezing phase

One should not forget the training during the unfreezing phase is conducted once the training during network surgery is complete. The epochs need to be read as the epochs in the respective cycles. The final validation loss before unfreezing is the starting validation loss of all unfreezing scenarios. The first thing we observe on figure 4.11 is the unfreezing phase improves the overall results. The scenario with a 10^{-4} learning rate is totally unstable for the first 50 epochs. It starts to stabilize after some huge learning rate drops. Even if this scenario ends reaching the highest loss values, we can not consider such training as reliable. This is not surprising since the learning rate used is only 10 times smaller than the one used during network surgery. At the other end, the 10^{-6} learning rate is not optimal. Although the stability of the learning is quite good, the weight updates are to tight to improve the quality of the results. The final loss values does not increase compared to the first training. Finally, the best results are obtained with the 10^{-5} learning rate which is a good compromise between the stability of the training and the improved final validation loss.

4.5.9 Data augmentation

In this test we will compare the fine-tuning results with and without data augmentation. The comparison has been made on the network surgery training with 70 epochs. In the graph we will also show the evolution of the loss on the training set to observe potential overfitting. Overfitting will be detected when training loss increases while at the same moment validation loss decreases. See figure 4.12 for results.

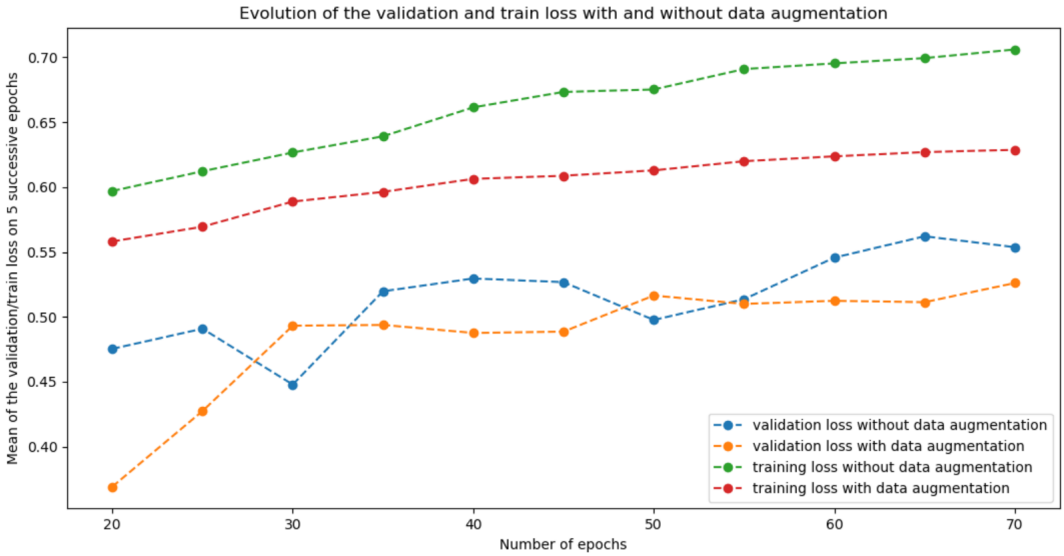

FIGURE 4.12: Data augmentation test

First, since no particular decrease of the validation loss values can be observed, no overfitting seems to happen neither with nor without data augmentation. The train loss function shows better results for the training set without data augmentation. Looking at the train loss values is only interesting to detect potential overfitting. Train loss values alone are of no interest when validating a model. Second, since the validation losses are quite similar for both cases, we have decided to keep the data augmentation for future tests as this technique is known for reducing the risk of overfitting. The use of overlap is also a major point which has guided our choice: without data augmentation the overlap generates very similar training patches. By flipping the patches, data augmentation creates differences between the successive patches.

4.5.10 The overlap: second round

In section 4.5.5 we have done a first set of tests with different overlap values. We did limit our tests with overlap values equal to 12 or higher. We initially decided not to test overlap values lower than 12 since selecting such values would have been very time consuming for the next tests. In this section we want to investigate if lower values for the overlap can increase the final results. We will test two new overlap values smaller than 12: 6 and 3.

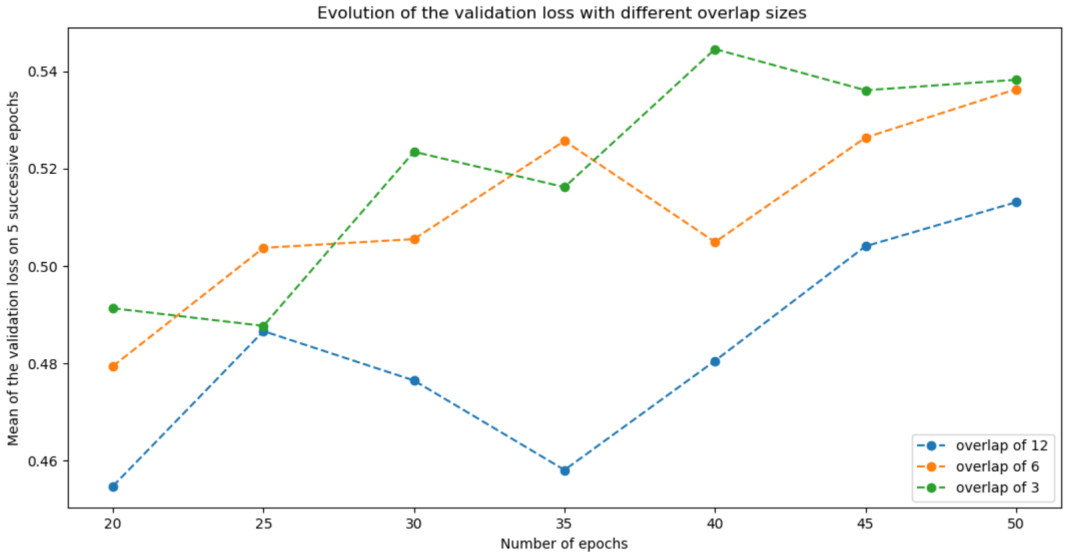

FIGURE 4.13: Comparison between different small overlaps

Figure 4.13 shows the two smaller overlaps provide better results compared to the overlap of 12 we used before. There is no sign of quality collapse which could indicate an overfitting. The results obtained with overlap 6 and 3 are quite similar with a very slight advantage for the overlap of 3. This indicates reducing the overlap further would not bring significant improvements. Since reducing the overlap to 3 will not provide a significant advantage, we will use the overlap of 6 for the next test in order to reduce the computational time. The overlap of 3 will be kept as the optimal value for the optimized fine-tuning with the manually segmented training set in section 4.6.

4.5.11 Is network surgery needed ?

In section 4.5.3 we have demonstrated the best surgery to use is the one where the bottleneck layers are replaced. In this section, we will evaluate if surgery is beneficial when we look at the complete training process with the unfreezing phase. We have compared a model trained using a classical fine-tuning framework where network surgery is followed by an unfreezing phase to a model without any network surgery. This second model was trained on the bottleneck and expansion section with the same learning rate of 10^{-5} as in the unfreezing phase. Both model have been trained for 150 epochs (75 of network surgery and 75 of unfreezing phase for the first case) with the optimal parameters found in the previous tests. Results are displayed on figure 4.14.

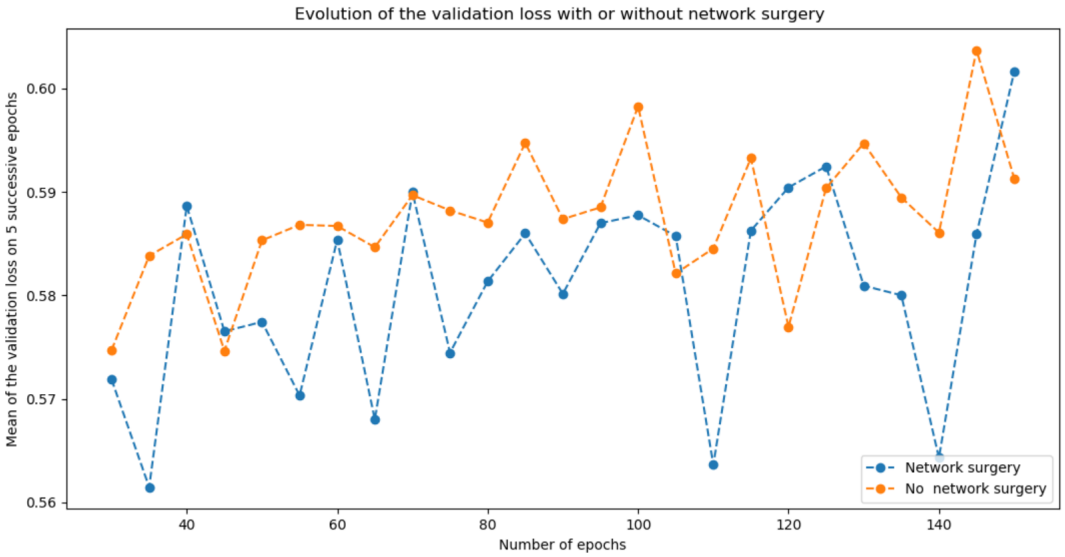

FIGURE 4.14: Comparison between the two models

The model without network surgery shows a more stable training and slightly better results. It shows network surgery needs to be used carefully and is not ideal for small training sets such as ours. Indeed, our training set is not large enough to retrain layers from scratch. Under these circumstances, it is preferable to fine-tune the pretrained weights. Moreover since the pretraining is very close to our training the pretrained weights should not be changed that much in order to be optimal for our task.

4.5.12 Analysis overview

Our series of tests have allowed us to identify values to optimize the different parameters for the fine-tuning with our data set. The optimized parameters are the following:

- No network surgery.
- The contraction section is frozen during the whole fine-tuning.
- Overlap of 3.
- Batch size of 5.
- Learning rate of 10^{-5}.
- Flipping of all the training patches along their axis to avoid overfitting.

In general, the differences in results for the tested values of the parameters were not very important. The stability of the training was more a concern. Since larger training sets would increase the complexity of the training and lead to a better validation loss, it is very likely those differences will increase with larger training sets. For a larger data set, this series of test can be reused and might lead to other optimized parameters.

Out of all tested parameters, the most important ones relate to the architecture of

the fine-tuning itself: the network surgery and the unfreezing phase. The tests concerning the parameters of the architecture should therefore absolutely be reproduced when a larger training set is used.

Our results tend to stabilize after less than 50 epochs. This is very fast. It demonstrates the size of our training set is too small and needs to be increased to reach good validation loss values. Similarly, the variations of the validation loss between consecutive epochs should not be as important. This results from the small size of the validation set. Finally the major drawback is the influence of the chosen validation set on the validation results. For our tests, due to time constraints, we have had to use a data set where the segmentation was done using *propseg*. This algorithm does not provide stable quality segmentation in the lumbosacral area. Combined with the small data set this instability generates validation sets of inconsistent quality. This issue should be less significant with a manually segmented data set created by an expert.

One may ask if our results are relevant when using the manually segmented training set. As mentioned the most important parameters are clearly the ones influencing the architecture of the fine-tuning. Observing our conclusion in sections 4.5.3, 4.5.7 and 4.5.11, those are more dependent on the nature and the size of the training set than its quality. The optimized values for the other parameters are close to the values commonly accepted . Indeed, a small overlap associated with flipping allows to increase the training set size. The batch size is quite small and generates no risk. The optimal learning rate for fine-tuning is lower than the one used during the training of the original model ($5 \cdot 10^{-5}$) (Gros et al., 2019). This is the expected result while reusing a pretrained model. Overall, the optimal parameter values resulting from our tests are also suitable for our manually segmented training set.

4.6 Results with manually segmented training set

Out of the 25 MR images in our possession, 19 were manually segmented when we have run these tests. We have performed 5 different fine-tuning where we selected the validation set manually. Doing this has allowed to have each MR image one time in a validation set. Therefore we can compute our fine-tuned segmentation on each MR image of the dataset without bias.

To evaluate the quality of our fine-tuned model we have performed the complete *deepseg* framework on our MR images with both the original segmentation model and our fine-tuned model. We then compare the DICE coefficient obtained by the two predicted binary masks.

4.6.1 The centerline detection issue

During the fine-tuning of the model, the training gives feedback about the validation loss evolution. Normally we should observe similar values between the validation loss given during the training and the DICE coefficient resulting from our predicted binary mask after passing the validation MR images trough the *deepseg* framework using the fine-tuned model.

However, we have quickly observed the values do not correspond. Where we observe validation losses around 0.8 during our fine-tuning, the obtained binary masks after the complete *deepseg* framework have a DICE coefficient around 0.4 when comparing them with the manually segmented masks used as references.

To understand the origin of the differences, we first need to compare the path followed by a MR image in the *deepseg* framework and the one followed in our fine-tuning pipeline. Second, the way the DICE coefficient is computed in both cases also impacts the results.

As described in section 3.2, the *deepseg* framework consists in the successive application of two deep learning models. Following the preprocessing phase, the spinal cord centerline is detected using the first deep learning model. After cropping the image around the centerline and dividing it into patches, the second model is applied to segment the spinal cord. Based on our preliminary observations, we have decided to concentrate in fine-tuning the segmentation model. Finally, post-processing is used to reconstruct the full binary mask from the cropped patches in order to fit with the original image. Once the binary mask is obtained, we can evaluate its accuracy by comparing it with the manually segmented mask through the DICE coefficient.

In our fine-tuning pipeline, we applied on our dataset the same preprocessing as in the *deepseg* framework. In *deepseg*, the first deep learning model detects the spinal cord centerline by analysing the MR image. In the fine-tuning, we have used a more simple algorithm which uses the segmentation masks to detect the spinal cord centerline. Experience has shown this provides an almost perfect centerline detection. We then crop the images around the centerline and divide them into patches as in *deepseg*. The model is then fine-tuned with the optimized parameters found in previous sections. There is no post-processing included in the fine tuning pipeline at the end of each epoch. Therefore the DICE coefficient can not be computed on a single full image. During the fine-tuning, the validation losses correspond to the average DICE coefficient measured on the multiple validation patches. Figure 4.15 shows the differences between both paths.

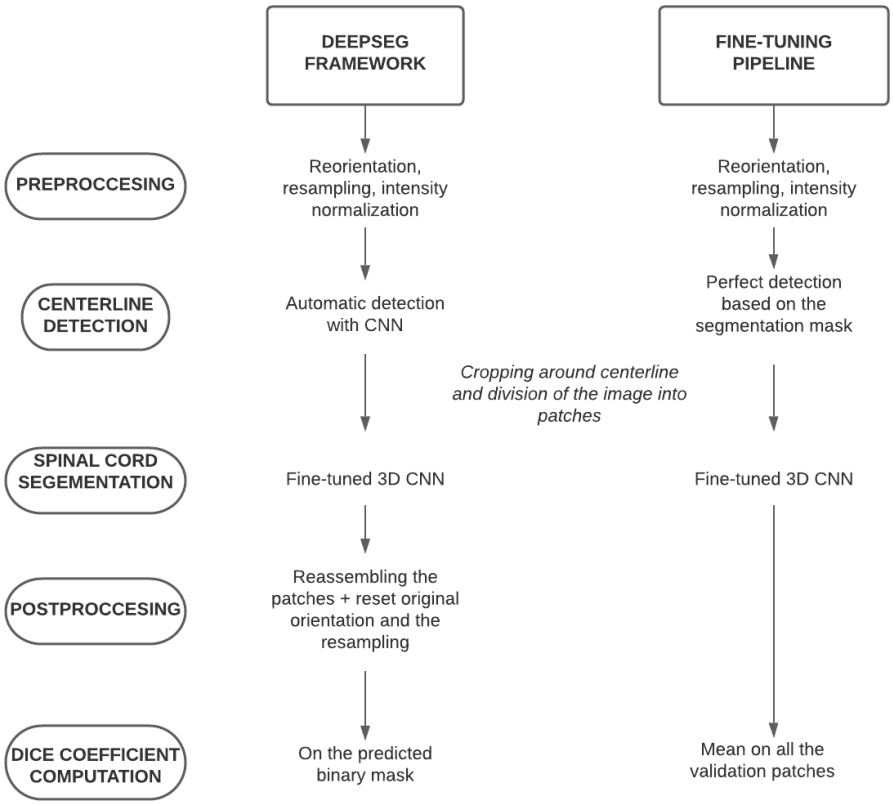

FIGURE 4.15: Comparison between the *deepseg* framework and the fine-tuning pipeline

The approaches for two main steps in the two paths can explain the difference in validation loss: the centerline detection and the absence of post-processing during the fine-tuning. Without post-processing, the validation loss is a mean calculated over a high number of validation patches. The absence of post-processing could explain a small differences between the validation losses due to the resampling but certainly not as large differences as we observe. The main difference must be related to the centerline detection.

One may ask why we have chosen to detect the centerline using the segmentation mask in our fine-tuning pipeline. As we have seen on figures 3.6 and 4.4, the *deepseg* algorithm tends to segment less than what is expected. Therefore, it is more important to learn to the model to detect accurately spinal cord in the lumbosacral area rather than learning not to detect spinal cord in zones where there is no spinal cord. Detecting the start, the end and the position of the centerline with high precision allows to only fine-tune the model with patches where the spinal cord is really present. When the image is cropped using our perfect centerline, the axial slices of the MR images which are positioned after the end of the spinal cord are discarded and replaced by slices composed only of zero values. In case of automatically detected centerline, the axial slices after the end of the detected centerline will not be discarded but cropped using the last known direction of the centerline. In consequence, our centerline detection allows to concentrate the learning on the detection in the lumbosacral area rather than on the detection on slices without any spinal cord.

FIGURE 4.16: MR image after cropping around the perfectly detected centerline. The slices under the spinal cord have been discarded.

Before we can conclude the loss validation difference comes from the centerline detection, we need to identify the underlying mechanisms generating the problem. Three causes have been identified. First, the automatic centerline detection does not detect accurately the spinal cord centerline. Consequently, the cropped images do not always include the necessary data to allow the segmentation model to segment correctly the spinal cord. Second, with the *deepseg* framework, the fine-tuned model may continue to segment after the end of the spinal cord. During the fine-tuning the problem is not detected since those parts have been discarded. Third, in the fine-tuning pipeline, perfect centerline detection can generate patches composed only of zero value in the validation and training sets. Those "black" patches do not disturb the training but the corresponding DICE coefficient is always 1 independently of the training as a full zero patch will always predict another full zero patch. These perfect DICE coefficients are increasing artificially the validation loss.

To have a better understanding on what happened we have decided to perform a new segmentation with the fine-tuned model. We have performed a bespoke version of the complete *deepseg* framework. We have replaced the automatic centerline detection with our exact centerline. The resulting segmentation masks have a much better DICE coefficient compared to the ones using the automatic centerline detection. The results are close from the validation losses obtained during the fine-tuning. It confirms the automatic centerline detection is problematic and needs to be fine-tuned in the same way as we did for the segmentation model.

4.6.2 Result table

We have decided to compare in depth the results with three different segmentation frameworks. The first segmentation framework is the basic *deepseg* one with the automatic centerline detection model and the original segmentation model. The second one is the bespoke version of the complete *deepseg* framework which uses the

exact centerline with the original segmentation model. The third one uses the exact centerline with the fine-tuned segmentation model. Comparing the results of the first and second segmentation allows to highlight the impact of a perfect centerline detection on the segmentation quality. A comparison between the second and third segmentation shows the impact of our fine-tuning on the segmentation quality. DICE coefficients for the three segmentation are available for each MR images of our dataset in table 4.2. In the next section we will discuss those results and our work in general.

We have observed two images of the dataset (*post-mortem* 8 & 9) generated very poor segmentation results with the original model. Therefore, we have decided to also perform the segmentation with a model fine-tuned without those two outlier images. It allows to observe the impact of those outliers on the fine-tuning quality.

The numbers in red in the table refer to outlier data. They have not been considered in the computation of the statistical data (maximum, minimum, mean, median) of their respective column.

MR images	DICE original model with automatic centerline	DICE original model with exact centerline	DICE fine-tuned model with exact centerline	DICE fine-tuned model with exact centerline without outlier images
in-vivo 1	0.534	0.818	0.762	0.828
in-vivo 2	0.738	0.894	0.804	0.831
in-vivo 3	0.710	0.820	0.757	0.872
in-vivo 4	0.702	0.863	0.734	0.861
in-vivo 5	0.474	0.823	0.742	0.817
in-vivo 6	0.701	0.874	0.848	0.906
in-vivo 7	0.498	0.805	0.581	0.834
in-vivo 8	0.623	0.850	0.802	0.875
in-vivo 9	0.646	0.846	0.734	0.872
post-mortem 1	0.542	0.845	0.116	0.252
post-mortem 2	0.702	0.825	0.001	0.001
post-mortem 3	0.721	0.817	0.798	0.838
post-mortem 6	0.610	0.767	0.497	0.701
post-mortem 9	0.037	0.012	0.375	/
post-mortem 10	0.089	0.019	0.248	/
post-mortem 11	0.497	0.801	0.818	0.693
post-mortem 12	0.640	0.788	0.246	0.232
post-mortem 13	0.737	0.860	0.772	0.732
post-mortem 14	0.190	0.720	0.605	0.765
MINIMUM	0.190	0.720	0.116	0.232
MAXIMUM	0.738	0.894	0.848	0.906
MEAN	0.544	0.822	0.463	0.626
MEDIAN	0.640	0.823	0.738	0.830

TABLE 4.2: DICE coefficient of each segmentation framework for all the MR images

4.7 Discussion

Comparing the results from the complete *deepseg* framework and the ones of the original segmentation model using the exact centerline, highlights a big gap in terms of performance. Giving the exact centerline to the segmentation model clearly leads to a significantly better segmentation. Except for the two outliers MR images which are really poorly segmented, we observe a quite good segmentation for the rest of the set once the exact centerline is given.The DICE coefficient mean raises from to 0.544 to 0.822 with the exact centerline. Those results are close to what we would expect

form a good algorithm dedicated to the lumbosacral segmentation. The difference between the results with and without automatic centerline detection clearly shows the centerline detection model needs to be enhanced for the lumbosacral area similarly as we did for the segmentation model. Furthermore, an automatic centerline detection will always provide a lesser quality compared to the perfect centerline as provided here. Therefore the segmentation model needs to become more robust to support an imperfect centerline detection.

When we compare the results of our fine-tuning with and without the outlier images, segmentation quality clearly improves when the outliers images are not included in the dataset. It shows the importance of the training set quality in order to obtain a correct fine-tuning. It proves also our training set is too small. Indeed with a larger training set, such outlier images would only have had a a minor impact on the training. A larger training set leads to a more robust fine-tuning providing better results on a larger variety of MR images.

Even without the outliers in the training set, the fine-tuning model gives mixed results. Compared to the original model, the mean result is lower but the median result is better. Out of the 17 MR images, 8 provide a better segmentation with the fine-tuned model. It proves our fine-tuned model is capable of giving great results, better than with the original one, for large number of MR images. It also adapts with more difficulties to some other MR images. This demonstrates once more the need for a larger training set in order to have more diversity and improve the potential of the model. The original model has been trained on 904 images. With only 17 images our fine-tuning can not compete. We need more data to ensure a better lumbosacral segmentation without completely disturbing the existing training.

The best example of MR image badly segmented by the fine-tuning is image *post-mortem 2*. With the original model, the segmentation is just above average in the lumbosacral area. However, with the fine-tuned model, the segmentation becomes completely wrong. This image contains a spinal cord with a particularity. Half of the spinal canal seems to be missing: part of the white zone around the spinal cord which can generally be observed is black on the image. While the original model is robust enough to detect this anomaly, our fine-tuned model has turned mad and has segmented the missing part of the spinal canal instead of the spinal cord. An axial slice of this particular spinal cord is available on figure 4.17 with the manually segmented mask and on figure 4.18 with the segmentation mask generated by the fine-tuned model.

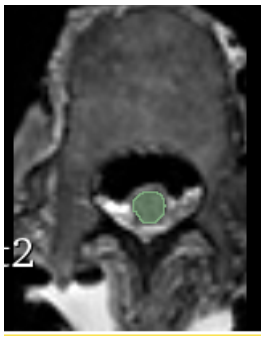

FIGURE 4.17: Manual segmentation of the spinal cord

FIGURE 4.18: Segmentation with the fine-tuned model

In a later stage, once a better training set is available, we would first need to fine-tune the centerline detection model in order to allow it to adapt to the lumbosacral area. Second we would need to fine-tune the detection model again. The same training set can be used for both tasks. During the prepossessing of the second fine-tuning we should use the fine-tuned version of the first model for centerline detection. Indeed, using the automatic centerline detection for the fine-tuning of the segmentation model allows it to learn how to process the spinal cord on images which are not perfectly centered.

Now all results are available, our initial choice to focus on fine-tuning the segmentation model and use the exact centerline in the preprocessing can be questioned. Without a correctly segmented dataset until the very end of our work, it was not possible to detect accurately all the problems in the original *deepseg* models. At the moment when the choice had to be done, observations on the few data that we had in our possession, have shown us that even if the centerline was not perfectly detected, the spinal cord was well contained in the cropped image as on figure 4.2. As a consequence it has been decided to focus on the fine-tuning of the segmentation model which was not providing the expected results in the lumbosacral region. With the latest dataset with manually segmented masks and our fine-tuned model, we have understood the end to end mechanisms which lead to a poor segmentation of the lumbosacral spinal cord by the original *deepseg* algorithm .

Our results are highlighting what should be improved in the *deepseg* models to obtain a better algorithm dedicated to the lumbosacral segmentation. At first, the centerline detection did not appear to be an issue for lumbosacral segmentation. Our manually segmented dataset has enabled us to discover it is as important to fine-tune the centerline detection model as it is for the segmentation model. The fine-tuning pipeline we created only needs small adjustments to support the fine-tuning of the centerline detection model. The results have also shown us the need of a larger training set. The training data should ideally originate from different scanners and hospitals in order to ensure the fine-tuned models are adapted to a versatile range of MR images and avoid overfitting. The series of test we have performed on the different fine-tuning parameters should be reused with the new training set in order to achieve the best possible fine-tuning for both models.

Working together with Dr Aleksandar Jankovski and Sylvain Deffet on this thesis

has demonstrated the CHARP platform now delivers a suitable tool for online manual segmentation of MR images. CHARP supports the need for the expansion of the initial training set which was highlighted. Our work also provides a complete overview of what should be done to arrive to an algorithm dedicated to the lumbosacral segmentation. It is as well a strong basis to complete the necessary fine-tuning. The amount of work and research necessary to reach the initial goal has been considerably reduced.

Chapter 5

Conclusion

To conclude this thesis, we will start by general overview of the performed work. It will be followed by a presentation of the different improvement possibilities and further works that could be done in a later phase. Finally we will discuss the contribution of this thesis to the area of spinal cord study.

5.1 General overview

The main goal of this thesis was to create an algorithm capable of segmenting the lumbosacral spinal cord correctly and automatically on MR images.

Of course, segmentation algorithms for the spinal cord already exist but they all have the same drawback: they can not ensure a precise segmentation of the lumbosacral part. Despite showing good results in their publication for the entire spinal cord, our own test have made clear weaknesses visible for the segmentation . Therefore, we need to develop an algorithm dedicated to the lumbosacral segmentation.

Deep Learning is one of the most powerful and actual technique in image segmentation. Therefore, we have chosen to use an existing deep learning algorithm (Gros et al., 2019) as basis for our work. This algorithm is available in the open-source Spinal Cord Toolbox. The algorithm uses U-net models. The first one is dedicated to the detection of the spinal cord centerline. The second one performs the spinal cord segmentation on MR images cropped around the detected centerline. Using fine-tuning techniques it is possible to specialise the existing U-net models to the segmentation of the lumbosacral spinal cord. We have chosen to fine-tune the segmentation model rather than create a complete new one. This approach is motivated by two mains reasons. First deep learning models support fine-tuning to accommodate specific requirements. Second, it allows us to improve an existing good segmentation algorithm in place of creating a new one from scratch.

In order to complete our fine-tuning we have created a dedicated Python pipeline inspired by an existing GitHub repository (Kyathanahally, 2018). This pipeline contains two scripts. The preprocessing script prepares the dataset to make it compliant to the standards used by the segmentation model of *deepseg*. The main script is in charge of the fine-tuning of the segmentation model and contains a series of optimization tests made on the fine-tuning parameters.

Finally, once we have fine-tuned the segmentation model, the prediction script allows to use the *deepseg* algorithm with our optimized model instead of the original one.

To execute our fine-tuning, we need a training set. Manual segmentation is mandatory to obtain a good training set. To support the manual segmentation, we have worked with the CHARP platform. CHARP has been developed at UCLouvain and supports hosting medical images with their respective segmentation masks. The platform includes the necessary tools to segment the hosted images. Our work is one of the very first using CHARP for manual segmentation. During our thesis, the feedback provided by Dr. Aleksandar Jankovski, who used his expertise to create the manual segmentation masks, to Sylvain Deffet, who is actively developing the platform, has allowed to improve the quality of the tools available on the platform.

As manual segmentation is a long and intensive process, the final dataset was not available when we started our developments: we have had to use a workaround to optimize the parameters of our fine-tuning. To create a complete dataset for the optimization tests, we have processed our MR images with an automatic segmentation algorithm to replace the missing manual segmentation. To do this, we have used the *propseg* algorithm (De Leener, Kadoury, and Cohen-Adad, 2014). It is an automatic algorithm based on active contour techniques. As the size and the nature of the training set do not change with this workaround, we can fairly assume the optimized parameters found while fine-tuning with the workaround are similar for the manually segmented training set. The tests used to optimize the parameters have demonstrated the particularities of the U-net model. Classical rules of network surgery were not applicable on such non-sequential model. We also discovered the most relevant parameters relate to the architecture of the fine-tuning itself: the network surgery and the unfreezing phase. In our case network surgery appeared not to be a good option due to the small training set size. The contraction section was frozen during the entire training as well.

Our final results with the manually segmented dataset are indicating potential future developments which could be applied to our work. Similarly to the segmentation model, the centerline detection model of *deepseg* needs to be fine-tuned. Indeed, we have discovered the exact centerline detection provides major improvements to the original segmentation model. We also need a larger training set, coming from different sources to include more variety. The fine-tuned segmentation model shows good results on a part of our dataset but some of the images are poorly segmented. It demonstrates the model is not yet robust enough for the variety of images it may encounter. More training data will lead to a more complete and robust fine-tuning. The original *deepseg* model for T2 contrast was trained using 904 images. With only 17 images in our dataset, it was predictable our fine-tuned model would show some weaknesses. Our results are encouraging and opening new perspectives in the lumbosacral segmentation.

With our work, the improvements needed on the *deepseg* models in order to reach an accurate lumbosacral spinal cord segmentation have been clearly identified. We have developed a functional fine-tuning pipeline for those improvements and have contributed to the improvement of the CHARP platform. This platform can be used for the development of training sets.

5.2 Improvements and further works

The obvious improvement that has to be done is creating of a better training set with more data coming from different hospitals. Ideally we would need three different training sets, one for each of the MRI contrast (T1, T2 and T2*). The CHARP platform can be used for the manual segmentation of the dataset and to share the data between medical experts and IT developers. Once sufficiently large and diverse training sets will be available, the training of the models will be straightforward with the support of the pipeline developed during this thesis.

A fine-tuning pipeline for the centerline detection model needs to be created. Fortunately, only few changes are necessary on the pipeline we developed to include the fine-tuning of the centerline detection model.

After having trained the models correctly, the next step would be to create a function that segments the entire spinal cord to use both the original *deepseg* models and the fine-tuned ones where they are the most relevant. Such function needs to be able to split the MR images of the entire spinal cord into two different parts: the cervical-thoracic part and the lumbosacral part. Once those two parts created, the algorithm can use the most appropriate model for each part. In the introduction we have presented an automatic algorithm for the vertebrae labelling and localisation (Ullmann et al., 2014). This algorithm is available on the SCT and could be used to split the MR images. Using one of the last thoracic vertebrae could potentially separate the complete spinal cord into the two parts we are looking for. Further investigations of course need to be confirm the optimal vertebra to use.

Finally the last step would be to contact the SCT developer community in order to integrate our derived algorithm to the SCT. A scientific paper needs also to be written for that purpose. A lot of quality tests are mandatory before expecting sharing our algorithm with the entire scientific community. If the resulting algorithm would be included in the SCT, this would be a fantastic achievement after a long journey for our work. Indeed we would have created an algorithm able to segment correctly the entire spinal cord. This new algorithm would be much more precise and robust in the lumbosacral part compared the current existing one. The algorithm would also be available on the reference tool for the spinal cord processing and participate to future medical progress.

5.3 Contribution to the area of spinal cord study

As described in the introduction of the thesis, for decades Spinal Cord Injuries (SCI) were considered as incurable. These last years, SCI treatment have become a new subject of studies. Those treatments are based on the stimulation of the lumbosacral spinal cord (Minassian and Hofstoetter, 2016). Being able to precisely position on MR images the SCI and subsequently the zone to stimulate could greatly improve those treatments.

The most natural and precise way to refer to position in the spinal cord is to use the spinal segments. Unfortunately, we are not yet able to position them correctly on MR images. A recent study concerning the morphometric characterisation of dissected spinal cord (Aleksandar Jankovski, 2021) could lead to a mathematical model

able to predict the segment position with automatic measurements based on MR images. Such measurements can not be taken without a complete mask of the spinal cord. Current automatic algorithms for spinal cord segmentation have trouble to segment the lumbosacral area. Therefore, we have set the first milestones of an algorithm specialised in the segmentation of the lumbosacral spinal cord.

A tool able to split the entire spinal cord and combining this algorithm with an existing algorithm will provide a robust and accurate segmentation of MR images for an entire spinal cord. Our work is contributing to a solution which will lead to a better positioning of the spinal cord injuries, help to improve treatments in the spinal cord and ultimately improve the life of injured people.

Bibliography

Aleksandar Jankovski, Catherine Behets et al. (2021). "Post-mortem characterization of human spinal cord morphometric landmarks and individual segments".

Amiri, Mina, Rupert Brooks, and Hassan Rivaz (2020). "Fine-Tuning U-Net for Ultrasound Image Segmentation: Different Layers, Different Outcomes". In: *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* 67.12, pp. 2510–2518. DOI: 10.1109/TUFFC.2020.3015081.

Ballard, D.H. (1981). "Generalizing the Hough transform to detect arbitrary shapes". In: *Pattern Recognition* 13.2, pp. 111–122. ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(81)90009-1. URL: https://www.sciencedirect.com/science/article/pii/0031320381900091.

Brownlee, Jason (2019). URL: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (visited on 05/18/2021).

Çiçek, Özgün et al. (2016). "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*. Ed. by Sebastien Ourselin et al. Cham: Springer International Publishing, pp. 424–432. ISBN: 978-3-319-46723-8.

De Leener, Benjamin, Samuel Kadoury, and Julien Cohen-Adad (2014). "Robust, accurate and fast automatic segmentation of the spinal cord". In: *NeuroImage* 98, pp. 528–536. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2014.04.051. URL: https://www.sciencedirect.com/science/article/pii/S1053811914003310.

De Leener, Benjamin et al. (2017). "SCT: Spinal Cord Toolbox, an open-source software for processing spinal cord MRI data". In: *NeuroImage* 145, pp. 24–43. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2016.10.009. URL: https://www.sciencedirect.com/science/article/pii/S1053811916305560.

Dr Daniel J Bell, Dr Candace Makeda Moore et al (2019). *Dice similarity coefficient*. URL: https://prod-assets.static.radiopaedia.org/articles/dice-similarity-coefficient?lang=us (visited on 05/17/2021).

GBS (2017). *Spinal Cord Segments – Cross-sectional Anatomy*. URL: https://www.getbodysmart.com/spinal-cord/cross-sectional-anatomy (visited on 05/15/2021).

Gros, Charley et al. (2018). "Automatic spinal cord localization, robust to MRI contrasts using global curve optimization". In: *Medical Image Analysis* 44, pp. 215–227. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2017.12.001. URL: https://www.sciencedirect.com/science/article/pii/S136184151730186X.

Gros, Charley et al. (2019). "Automatic segmentation of the spinal cord and intramedullary multiple sclerosis lesions with convolutional neural networks". In: *NeuroImage* 184, pp. 901–915. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2018.09.081. URL: https://www.sciencedirect.com/science/article/pii/S1053811918319578.

Harrison, Megan et al. (2013). "Vertebral landmarks for the identification of spinal cord segments in the mouse". In: *NeuroImage* 68, pp. 22–29. ISSN: 1053-8119. DOI:

https://doi.org/10.1016/j.neuroimage.2012.11.048. URL: https://www.sciencedirect.com/science/article/pii/S1053811912011603.

Kaus, M.R. et al. (2003). "Automated 3-D PDM construction from segmented images using deformable models". In: *IEEE Transactions on Medical Imaging* 22.8, pp. 1005–1013. DOI: 10.1109/TMI.2003.815864.

Kyathanahally, Julien Cohen-Adad & Sreenath P (2018). *deepseg-training*. https://github.com/sct-pipeline/deepseg-training.

Liang, James (2018). *An introduction to deep learning*. URL: https://towardsdatascience.com/an-introduction-to-deep-learning-af63448c122c (visited on 05/03/2021).

Minassian, Karen and Ursula S. Hofstoetter (2016). "Spinal Cord Stimulation and Augmentative Control Strategies for Leg Movement after Spinal Paralysis in Humans". In: *CNS Neuroscience & Therapeutics* 22.4, pp. 262–270. DOI: https://doi.org/10.1111/cns.12530. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cns.12530. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cns.12530.

Newton, Joshua (2021). *Use Python 3.7 instead of Python 3.6 for SCT's conda environment*. https://github.com/spinalcordtoolbox/spinalcordtoolbox/pull/3361/files.

Norouzi, Alireza et al. (2014). "Medical Image Segmentation Methods, Algorithms, and Applications". In: *IETE Technical Review* 31.3, pp. 199–213. DOI: 10.1080/02564602.2014.906861. eprint: https://doi.org/10.1080/02564602.2014.906861. URL: https://doi.org/10.1080/02564602.2014.906861.

Nyúl, László G. and Jayaram K. Udupa (1999). "On standardizing the MR image intensity scale". In: *Magnetic Resonance in Medicine* 42.6, pp. 1072–1081. DOI: https://doi.org/10.1002/(SICI)1522-2594(199912)42:6<1072::AID-MRM11>3.0.CO;2-M.

Olivier, Julien and Ludovic Paulhac (Dec. 2011). "3D Medical Image Segmentation: Interactive Texture-Based Approaches". In: ISBN: 978-953-307-774-1.

Pröve, Paul-Louis (2017). *An Introduction to different Types of Convolutions in Deep Learning*. URL: https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d (visited on 05/30/2021).

Rejc E., Angeli C.A. Atkinson D. et al (2017). "Motor recovery after activity-based training with spinal cord epidural stimulation in a chronic motor complete paraplegic." In: *Sci Rep 7* 13476. DOI: https://doi.org/10.1038/s41598-017-14003-w.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv: 1505.04597 [cs.CV].

Rosebrock, Adrian (2019). *Fine-tuning with Keras and Deep Learning*. URL: https://www.pyimagesearch.com/2019/06/03/fine-tuning-with-keras-and-deep-learning/ (visited on 04/10/2021).

Roy, Taposh (2017). *Medical Image Analysis with Deep Learning, Part 4*. URL: https://www.kdnuggets.com/2017/07/medical-image-analysis-deep-learning-part-4.html (visited on 05/13/2021).

Sankesara, Heet (2019). *Unet*. URL: https://towardsdatascience.com/u-net-b229b32b4a71 (visited on 05/03/2021).

SCT (2021). *Studies using SCT*. URL: https://spinalcordtoolbox.com/en/stable/overview/studies.html (visited on 03/15/2021).

Shibuya, Naoki (2017). *Up-sampling with Transposed Convolution*. URL: https://naokishibuya.medium.com/up-sampling-with-transposed-convolution-9ae4f2df52d0 (visited on 05/03/2021).

Stewart, Matthew (2019). *Simple Introduction to Convolutional Neural Networks*. URL: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac (visited on 02/27/2019).

Transfer, knowledge (2019). URL: https://androidkt.com/batch-size-step-iteration-epoch-neural-network/ (visited on 05/18/2021).

Ullmann, Eugénie et al. (2014). "Automatic Labeling of Vertebral Levels Using a Robust Template-Based Approach". In: *International Journal of Biomedical Imaging* 2014, p. 9. DOI: https://doi.org/10.1155/2014/719520.

Wikipedia (2020). *Active contour model*. URL: https://en.wikipedia.org/wiki/Active_contour_model (visited on 05/17/2021).

— (2021). *Spinal cord*. URL: https://en.wikipedia.org/wiki/Spinal_cord (visited on 05/15/2021).

Young, Wise (2019). *Spinal cord injury levels & classification*. URL: https://www.sci-info-pages.com/levels-and-classification/ (visited on 05/15/2021).

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**