Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Vetle Birkeland Huglen

Cardiac MRI-Image Segmentation by Parametric Curve Learning

A novel deep-learning based approach

Master's thesis in Industrial Mathematics
Supervisor: Markus Grasmair
July 2020

NTNU
Norwegian University of Science and Technology

Vetle Birkeland Huglen

Cardiac MRI-Image Segmentation by Parametric Curve Learning

A novel deep-learning based approach

Master's thesis in Industrial Mathematics
Supervisor: Markus Grasmair
July 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Abstract

Data-driven methods are flourishing in medicine, improving our understanding of diseases, simulating the effect of treatment, and assisting medical professionals in surgery. At the base of these methods lies image segmentation, a vital task for extracting information from medical images. While achieving high segmentation scores, existing methods for image segmentation have flaws in their ability to represent curved shapes accurately. Several data transformation steps may be necessary before the segmentation output can be put to good use. This thesis proposes a novel learning setup, combining the shape-representation superiority of parametric curves with the well-investigated task of Cardiac MRI-image segmentation.

The proposed setup consists of a parametric curve model for the heart shape, a custom network architecture, and custom loss and gradient formulations. We validate the method on the benchmark ACDC-dataset [1] and compare the performance to a state-of-the-art network for medical image segmentation. The results obtained indicate that the method does not work in current setup, but that it should be further developed and tested. Some obvious improvements could likely significantly enhance the method's performance, but these are left for future studies.

Preface

This thesis concludes my master's degree and my studies at NTNU. It has been a long journey with a strange ending, due to the bizarre circumstances under COVID19. Closing this chapter of my life evokes both great excitement for the future and a strong sense of nostalgia. I will always yearn the freedom I've felt and friendships I've made as a student.

I want to thank all my fellow students for five unforgettable years in Trondheim. You make the student experience so much larger than just studying. Thanks to my flatmates in PKV8, W6th, and HLG2 for all the adventures. I thank my mom and dad for financial support throughout my studies, but most of all, for their persistent encouragement and unconditional love. A huge thanks to my supervisor Markus for great conversations and for always being available for consultation. Lastly, a big thank you to Elias for all the moments of joy and laughter while struggling to finish our theses. These last months would have been much less enjoyable without your company.

Table of Contents

List of Figures

Abbreviations

MRI	=	Magnetic resonance imaging
ACDC	=	Automated Cardiac Diagnosis Challenge
CVD	=	Cardiovascular disease
CAD	=	Computer-aided Diagnosis
CAS	=	Computer-aided Surgery
CT-scan	=	Computer tomography scan
RV	=	Right ventricle
LV	=	Left ventricle
MYO	=	Myocardium
CAD	=	Computer-aided Design
CAM	=	Computer-aided Manufacturing
NN	=	Neural network
CNN	=	Convolutional neural network
ReLU	=	Rectified Linear Unit

Chapter 1

Introduction

For as long as humans have practiced medicine, vision has been an integral part of diagnosing, treating, and preventing disease. Still, medical professionals use their sight to look for bodily indications of diseases when evaluating a patient. Modern medical imaging technology lets us look inside the human body without resorting to invasive surgery, providing means to detect diseases that are not visible on the exterior of the body. The rise of computer vision has unleashed computers' powers on diagnostics, utilizing their unique ability to discover patterns and outliers in images, and discovering symptoms invisible to the human eye. At the fundaments of medical computer vision lies the image segmentation task, letting computers extract information and detect objects in medical images. While well-investigated, medical image segmentation is far from solved, and improvements has the potential to significantly enhance the utility of computer-based medical diagnostics, surgery-aid, and treatment.

1.1 Curve Segmentation through Parameter Learning

In this thesis, we present, investigate and implement a novel deep-learning-based approach to image segmentation. While traditional image segmentation looks to assign each pixel of an image to a class, we seek instead a parametric curve representing the shape of the object to be segmented. This alternative approach is motivated by multiple factors. First of all, we assume that a parametric curve representation can provide a more realistic shape of an object than a pixel-wise discrete representation. Secondly, in the specific application of image segmentation on cardiac MRI-images, the desired end-result is typically a geometric model of the heart. Our proposed approach generates a segmentation output much closer to the desired end-product.

1.2 Thesis Outline

This thesis mainly focuses on the presentation and implementation of the novel method for cardiac MRI-image segmentation introduced in the previous section. Developing the code and managing the unexpected difficulties one encounters when developing a new method has undoubtedly been the most challenging and time-consuming part of the thesis work. There have been significant uncertainties concerning the expected results, or even if the method would work at all. The results should thus be interpreted as an initial investigation of the method - further testing is necessary to draw any definite conclusions.

The reader will be presented with the thesis work in an orderly manner. We start by introducing some necessary supporting theory before discussing technical aspects of the proposed method's implementation, and the results of the numerical experiment. In Chapter 2, we examine the task of image segmentation in general, before narrowing down the scope to the specific task of Cardiac MRI-image segmentation. We further look at some technical aspects of MRI-images necessary for understanding the task at hand. In Chapter 3, we introduce spline curves and B-splines, and the theory of which we build our parametric curve heart-shape model. The model is explicitly defined near the end of the chapter. The next chapters account for the various aspects of the learning setup. In Chapter 4, we apply the parametric heart model to a supervised learning framework and define our specific image segmentation task. We discuss the proposed network architecture at the end of the chapter. Chapter 5, presents the loss function, its gradient and their implementation. In the experimental part of the thesis, the proposed method is implemented and validated on the ACDC-dataset [1]. The results obtained are presented in Chapter 6, and the performance is compared to the performance of a traditional segmentation method validated on the same dataset. Finally, in Chapter 7, we discuss the obtained results and explore improvements of our proposed method.

Chapter 2

Cardiac MRI-Image Segmentation

Image segmentation is the division of a digital image into separate meaningful subparts. The task has traditionally been executed manually by humans, but automatic image segmentation methods are rapidly replacing the need for manual work. Despite its analogy being a natural and trivial part of human vision [2], the task of automatic image segmentation has been an object of significant scientific interest for decades. Deep learning has dramatically enhanced the performance of image segmentation methods in the last decade [1]. Currently, image segmentation is powering development within the fields of vision for autonomous vehicles [3], face recognition [4], and computer-aided diagnosis based on MRI-images [5]. This chapter provides a short but necessary introduction to image segmentation and especially its applications on Cardiac MRI-images.

2.1 A Brief Introduction to Image Segmentation

Digital images are composed of image elements called pixels, associated with a discrete value representing some color information [6]. We will only consider grey-scale images, in which case each pixel holds a value representing the brightness of that pixel on a scale with white at one endpoint and black at the other endpoint. The resolution of an image is usually represented by two numbers: the number of pixel columns (the width of the image) and the number of pixel rows (the height of the image). We define the domain of an image of resolution $h \times w$ as $\Omega = \{x, y \in \mathbb{R} \mid 0 \leq x \leq w, 0 \leq y \leq h\}$. We assign to Ω a square grid of elements with side lengths 1. These elements correspond to the pixels of the digital image. Grey-scale images can thus be represented by a matrix, where each element of the matrix is a pixel, and the value of the matrix element is the grey-scale value. The resolution of the image defines the size of the matrix. Given a grey-scale image with h pixel rows and w pixel columns, we represent it by the matrix $M \in \mathbb{R}^{h \times w}$. We will make no distinction between a digital image and its matrix representation in this thesis, and both terms may be used interchangeably depending on the context. We refer to the representation of a digital image by a matrix as *raster graphics*.

Image segmentation is partitioning the pixels of digital images into regions, which

Figure 2.1: The first image shows two climbers on a cliff by the sea. The second image shows an image segmentation output based on the first image. In this case, the classes are people (white) and background (green).

are usually required to be non-overlapping. Typically, some classes of interest should be correctly identified and separated from the background of the image. In the binary case, there are only two classes: the object of interest, which we label 1, and the background, which we label 0. The output of binary segmentation can be represented by a prediction matrix of the same size as the input image, for which each element holds a probability, a value in $[0, 1]$, that the corresponding pixel belongs to the object of interest. Alternatively, a threshold can be applied such that each element of the prediction matrix is either 0 or 1. Figure 2.1 shows a binary image segmentation output for the task of segmenting humans from the background. The image to the left is the input image, and the image to the right shows a prediction map for which the humans (white) in the image are separated from the background (green). If the image segmentation task concerns multiple classes, the output can be represented by a prediction matrix for each class. The segmentation task in this thesis is a binary one, and we will not further discuss multi-class image segmentation.

We make a distinction between semantic image segmentation and instance image segmentation. In semantic segmentation, we are only concerned with labeling each pixel with the correct class. The middle image of Figure 2.2 visualizes the output of semantic image segmentation. Note that while the pixels of the people are correctly identified (labeled with blue), no distinction is made between the individuals. Instance segmentation is a more complicated procedure, because, in addition to identifying each pixel class, we want to separate different instances of a class. The rightmost image of Figure 2.2 visualizes the

Figure 2.2: This figure illustrates the difference between semantic segmentation and instance segmentation. The leftmost image shows the original image. The middle image visualizes the result of a successful semantic segmentation. The people are all labeled with the correct class, but no distinction is made between the individuals. The rightmost image visualizes the result of a successful instance segmentation. All the people are correctly identified as individuals.

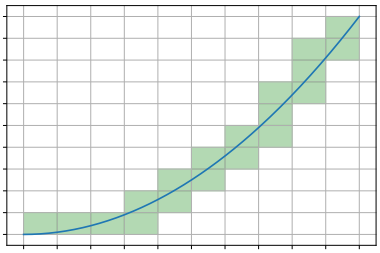


Figure 2.3: Representing a curve, in blue, by pixels, in green.

output of instance image segmentation. Here, the individuals are all labeled with different colors. While an important task in many segmentation applications, instance segmentation is not relevant in our task of cardiac image segmentation.

2.1.1 Representation of Segmentation by Enclosing Curve

Representing object boundaries by raster graphics is practical and straightforward. Evaluating the representation corresponds to looking up values in a fixed matrix. Furthermore, resolution is easily adjusted by changing the size of the representation matrix, though the process will cause a loss of information. However, many objects will have a shape which is poorly represented by raster graphics. Curved primitives such as circles or ellipses can only be *approximated* by pixels, typically resulting in a ”staircase” formation, as seen in Figure 2.3. Parametric curves better represent curved object boundaries. Furthermore, a curve representation does not have a fixed resolution, so scaling does not result in loss of resolution. Shifts and rotations of objects are trivial in a curve representation but complicated in a rastered representation. This thesis is based on the assumption that parametric

curves better represent the shape of a human heart than raster graphics. To apply a parametric curve representation of the heart shape to image segmentation, we make a small modification to the image segmentation task described in Section 2.1. Instead of assigning a class to each pixel, we will assign to each class an enclosing curve such that every pixel inside the enclosing curve belongs to that class. The input to the segmentation task will still be raster graphic cardiac MRI-images, but the output will be a closed parametric curve enclosing the heart pixels. Applied to a learning framework, as we will do later in this thesis, it may seem slightly counterintuitive to try to learn a parametric curve representation from pixel-wise ground-truth images, but we believe that this setup could lead to a more accurate learning of the true heart shape. Furthermore, many of the applications of cardiac MRI-image segmentation, revolve around performing mathematical calculations and simulations on the derived heart shape. Such processes will be significantly simplified by directly obtaining a parametric curve representation from the image segmentation.

2.2 Cardiac MRI and its Applications

Magnetic resonance imaging (MRI) is a non-invasive imaging technique leveraging magnetic fields and gradients, as well as radio waves, to generate images of the insides of a body. Cardiac MRI is used to create pictures of the heart, useful for detecting heart diseases and planning surgery. Though MRI-imaging is an expensive and elaborate procedure, the technique is widely used due to its strong performance in differentiating between different types of biological tissue [7], as well as not exposing the subject to health risks caused by radiation [8]. Cardiac MRI-images have become the gold standard for detecting *ischemia* [9], the process where lack of oxygen necessary for cell metabolism causes tissue in the heart to die. Ischemia is strongly linked to cardiovascular diseases (CVD), a group of diseases affecting the heart and vessels. CVD's are marked by the World Health Organization as the number one cause of death worldwide, claiming an estimated 17.9 million lives annually [10]. As more and more diseases are linked to physical features observable through MRI, the use of MRI in diagnosis context steadily grows, according to Lee et al. [11].

2.2.1 Basic Heart Anatomy

The heart is an essential organ in the human body, responsible for pumping blood containing oxygen and nutrients to body tissues [12]. There are four chambers in the heart, namely the left and right atria and the left and right ventricles. These chambers are shown in Figure 2.4. The atria receive blood returning from the body, and the ventricles pump the blood back out to the body again. A cardiac cycle is one heartbeat, and can be divided into a *systole* and *diastole* phase [13]. The systole is the phase in which blood pumps out from the heart. It begins with the left and right atria contracting, pushing blood from the atria into the left and right ventricles. The ventricles then contract, pushing the blood out into the body through arteries. Specifically, the right ventricle sends blood to the lungs to refill with oxygen, while the left ventricle supplies the body with oxygen holding blood through the aorta, the largest artery in the body. In the diastole phase, the heart muscle relaxes, and blood returns through veins from the lungs and body to the atria. The different valves seen

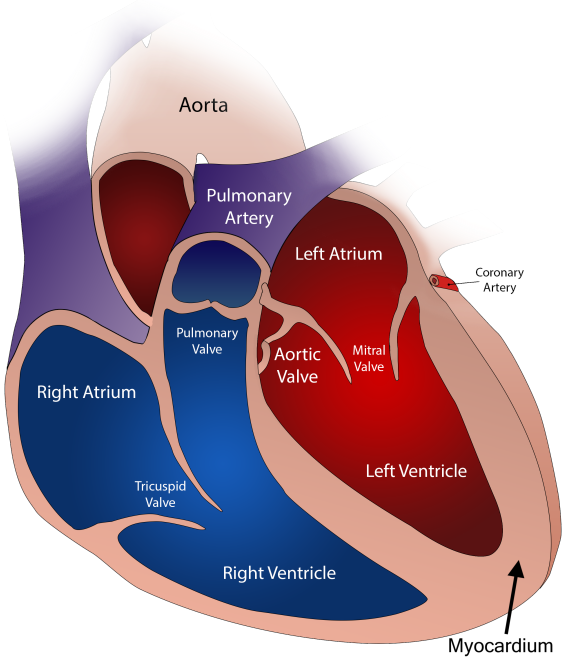


Figure 2.4: Overview of heart anatomy. The picture is edited from the original version from Pixabay.com.

in Figure 2.4 restrict the blood to only flow in the intended direction, avoiding backflow. The myocardium or heart muscle (labeled at the bottom of the figure), is the muscle responsible for the contraction in the systole phase, and constitutes the main part of the heart walls. To describe the heart's spatial properties, the *apex* and *base* of the heart are often used as references. The apex is located at the tip of the left ventricle and is close to the point of maximum movement during the cardiac cycle. Figure 2.5 shows the location of the apex in relation to the ventricles. The base of the heart lies opposite the apex, mainly in the left atrium.

2.2.2 Computational Anatomy and Cardiac Segmentation

Medical imaging techniques allow the computational powers of computers to be unleashed on medical tasks that previously have required time-consuming manual work or not been possible at all. Driven by development within medical imaging, computer vision, statistical shape analysis and machine learning, Computer-aided Diagnosis and Computer-aided Surgery are becoming essential tools in clinical practice [15], assisting medical professionals in both discovering and treating diseases. Fully automated diagnosis systems are

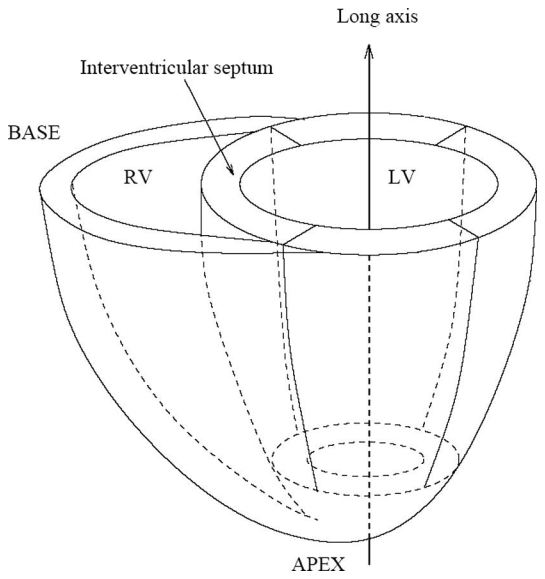


Figure 2.5: Orientation of the right and left ventricles in relation to the vertical axis. Figure provided by Petitjean and Dacher [14].

on the horizon. In research, computer-based methods let scientists uncover differences in biological structures between healthy and diseased people, improving our understanding of diseases [15, 16]. The availability of data and data-driven methods makes quantitative analysis a powerful tool for delineating the human body structures and the biological processes affecting them [17].

Common for many of the data-driven methods is their reliance on the data provided by the medical images and the information extracted by the image segmentation process. Sengur et al. [5] claims that image segmentation is widely accepted as an essential subprocess of computational medicine, and the problem is far from solved. Even though the big international competition MICCAI'17 reports that state-of-the-art methods for Cardiac MRI-Image segmentation achieve expert accuracy [1], effort is constantly made to get methods that are faster, more generalizable, more reliable or that need less human input or smaller datasets. Better segmentation can power improvements in nearly all aspects of medical computing, and image segmentation research does not seem to be declining anytime soon.

In the context of cardiac MRI-images, image segmentation is partitioning the pixels of an MRI-image containing the heart into some relevant regions. Typically, the regions of interest are some subparts of the heart, but we will only discuss the segmentation of the heart as a whole for this thesis. Our setup is thus a binary segmentation of the heart. Figure 2.6 shows a Cardiac MRI-image with its corresponding segmentation. The red pixels in the image belong to the heart class, while the others are background.

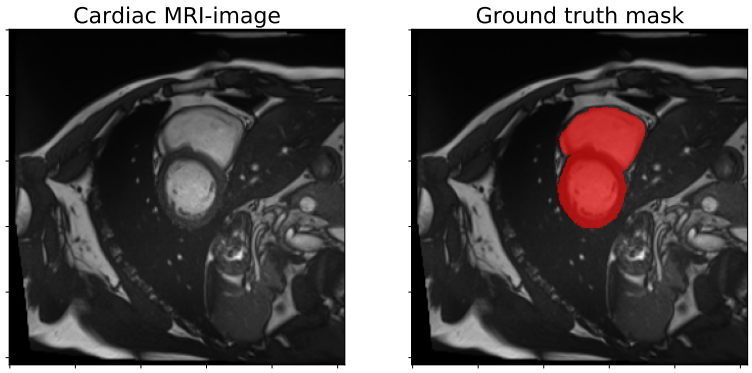


Figure 2.6: The left image shows the original MRI-image from the ACDC-dataset [1], and the right image shows the corresponding ground-truth heart pixels in red.

2.2.3 Cardiac MRI-Image Format

MRI-images are two-dimensional images trying to capture a three-dimensional anatomy. Images are taken sequentially along some axis in order to build a 3D-volume of the region of interest. To better describe this process, it is necessary to introduce some anatomical terms of location, created to describe a location in a body unambiguously [18]. Figure 2.7 show different planes used in medical context. In cardiac MRI-imaging, the anatomical terms of location are defined in relation to the long-axis of the left ventricle, shown in Figure 2.5. The long-axis of the left ventricle lies in the plane called the *coronal plane*, shown in blue in Figure 2.7. In coronary context, the short-axis view show planes perpendicular to the long-axis. While long-axis images provide useful supplementary insights to the heart anatomy, improving the accuracy of heart models [19], short-axis imaging is the most common format for diagnostics [20]. Long-axis images will not be discussed further in this thesis. Figure 2.6 show a short-axis MRI image of a heart with its corresponding labels.

To capture the whole heart volume, parallel slices of short-axis MRI-images are taken along the long axis from the apex to the base capturing the entire ventricles. Stacking the short axis slices along the long axis creates a complete 3D-image of the heart. Technical issues put a lower limit on the slice thickness, limiting the resolution along the long axis for the 3D-image. This resolution varies heavily depending on the MRI-machine. The ACDC-dataset [1], which will be used as validation dataset for our proposed method in Chapter 6, contain images with a resolution of 5 to 10 mm per pixel along the long axis, and a resolution of 1.34 to 1.68 mm^2 per pixel in the horizontal plane.

Modeling blood flow and heart movement during a cardiac cycle is useful for diagnostics and evaluation of many heart diseases [21]. To achieve this, spatial 3D-images are captured at different times during the cardiac cycle. Thus, the cardiac MRI-images can also be represented as 4D images with a temporal dimension in addition to the spatial

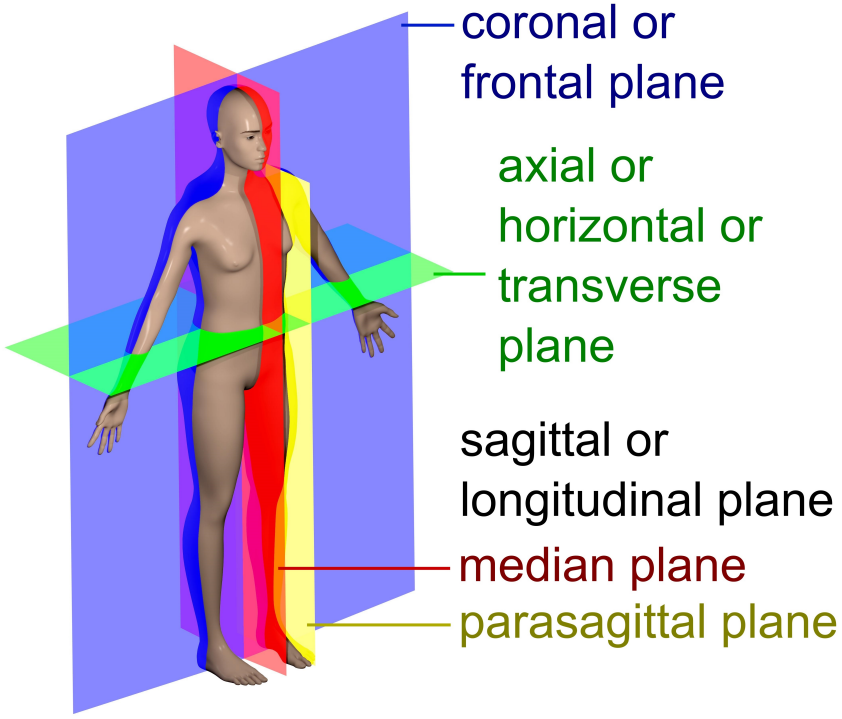


Figure 2.7: Different planes used in medical context in relation to a human body. Created by David Richfield and Mikael Häggström, M.D., and used with permission.

dimensions. In the ACDC-dataset [1] the temporal resolution varies from 28 to 40 images per cycle. Variation along the temporal dimension is mainly of interest for diagnostics, and not relevant for this thesis. Instead, we consider the images in a temporal sequence as separate images. In the ACDC-dataset, two temporal snapshots are labeled per person.

2.2.4 Datasets for Cardiac MRI-image Segmentation

Modern automated image segmentation methods are usually constructed as supervised learning tasks. In supervised learning, the objective is to learn some function that maps an input to an output given some training dataset of corresponding input and output pairs. When applying machine learning to automate a task, the result depends heavily on the collection of data used in training. In general, both sufficient size of the dataset and sufficient quality of the data is necessary to achieve valuable results. In the field of semantic image segmentation, the response variable used in training is typically a *ground truth image*, where each pixel is labeled with its correct class.

Labeling datasets for image segmentation is, in general, a very time-consuming procedure. Remez et al. [22] note that it took 40 person-years of labeling effort to label 80 object categories of the COCO dataset [23]. Labeled images are created by humans iterating through every pixel in an image and recording to which object or class the pixel belongs. For medical MRI-images, this labeling strategy is exponentially more complicated compared to most other applications. Firstly, the labeling must be performed by a medical professional. While it is trivial to determine the motive of images in datasets containing numbers or everyday objects, medical images can often be the subject of substantial interpretation, slowing down the process and making the result less sure. Thus, the process relies on finding personnel with sufficient skills, willingness, and capacity to label the images. Secondly, creating a dataset of *unlabelled* medical images has its complications. MRI-scanners are expensive to use, and while CT-scans are significantly cheaper, the subject is exposed to ionizing radiation [9]. The consequence in both cases is that image-taking is, to a large degree, restricted to patients. Regardless of the subjects, an anonymization procedure has to be performed prior to access, and hospitals may still be reluctant to share the images. The combination of these factors makes it altogether challenging to get the large datasets necessary for many traditional deep learning approaches. Consequently, it was not until 2015 that Ronneberger et al. [24] proposed a specialized network and training strategy for deep learning on cardiac image segmentation, which is described in the next section.

2.3 U-Net

In this section, we take a look at the U-Net, the most commonly used network for cardiac MRI-image segmentation in the last years. The U-Net architecture was strongly considered for the method proposed in this thesis, and the choice to abandon the U-Net is discussed in Chapter 4. In Chapter 6, we compare the performance of our proposed method to the performance of a U-Net applied to the same dataset. For these reasons, we provide a short introduction to the network to complete this chapter.

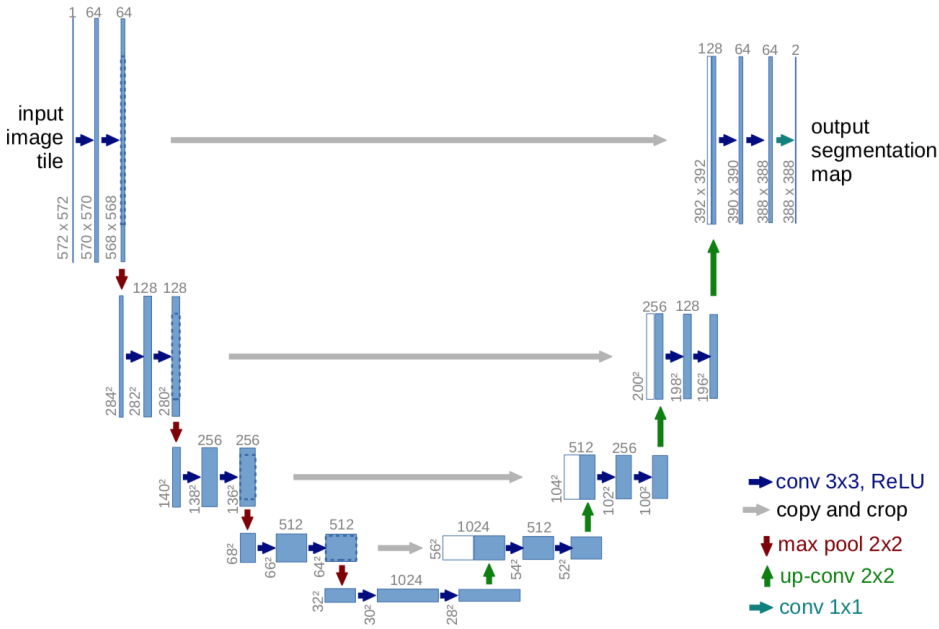


Figure 2.8: U-Net architecture, created by Ronneberger et al. [24].

The U-Net is a deep neural network, first proposed in 2015 by Ronneberger et al. [24] as a deep learning approach to biomedical image segmentation. While classification algorithms were achieving good results at the time the U-Net was proposed, visual tasks like biomedical image segmentation demanded a network that preserved spatial information and performed on a limited sized training set. The solution was a network building on the successful recipe of Convolutional Neural Networks [25] (CNN), but with an expanding path recovering spatial information following the quite traditional contracting path, yielding an almost symmetric architecture. A training strategy leveraging data augmentation strategies was proposed to ensure sufficient training on relatively small datasets.

U-Net has its name from the U-shaped architecture of the network, which can be seen in Figure 2.8. The contracting path is built of repeated groups of two 3x3 convolution layers, followed by a ReLU, and a 2x2 max pooling layer, designed to capture the context in the image. The convolution-filters are ”unpadded,” and the stride for the max pooling operation is 2 for downsampling, effectively halving each of the image’s dimensions and doubling the number of feature channels at each step. The trailing expansive path recovers spatial information. Each step consists of upsampling of the feature map and a 2x2 up-convolution followed by a concatenation with the corresponding feature map from the contracting path, as well as two 3x3 convolutions with a ReLU. The necessity of cropping is caused by the unpadded convolutions at each step loosing edges. The final layer of the network maps to the desired number of classes.

The data augmentation scheme proposed primarily focuses on building robustness to-

wards rotation, shifting, deformations, and gray-scale variations. Data augmentation is necessary due to the large variations found in MRI-images. Especially random elastic deformations are emphasized as a critical ingredient in doing training with small datasets possible. In recent years many state-of-art methods for cardiac image segmentation build upon the original U-Net architecture, and over one-half of the entries in the MICCAI 17 segmentation competition were U-Nets.

Chapter 3

A Spline Based Heart Model

Based on the discussion about curve segmentation in Section 2.1.1, we propose a parametric curve representation of the heart shape, which we will eventually apply to a cardiac MRI-image segmentation task in a supervised learning setup. In this chapter, we introduce the necessary theory and define our curve based heart model. While the intention is that the method is generalizable to multiple dimensions, this thesis focuses on the shape of a heart in two dimensions.

In general, we will seek a parametric planar curve $S(t) = (x(t), y(t))$ defined on some interval $t \in [a, b]$ to represent the outline of the heart shape. The curve must be closed and sufficiently smooth. We will also require a certain degree of flexibility in order to achieve sufficient accuracy in the representation. Spline curves, constructed by piecewise polynomials with some prescribed smoothness, satisfy these conditions and turn out to be suitable for our model. Splines are a cornerstone in mathematics, and their applications include curve and surface representation in computer graphics and manufacturing (CAD-CAM), methods for solving differential equations, and function approximation of data [26]. We will first take a closer look at the theory of splines and the basis for splines called B-splines before applying the theory to our specific application. The theory is mainly based on Floater [27].

3.1 B-Splines - Basis Functions for Spline Curves

Spline curves are piecewise polynomials defined on an interval $[t_0, t_m]$ subdivided into smaller intervals $[t_j, t_{j+1}]$. The breakpoints t_j are called knots. We will refer to the vector $\mathbf{t} = [t_0, t_1, \ldots, t_m]$ containing the breakpoints as the knot vector and the closed interval $[t_j, t_{j+1}]$ as the j'th knot span. In general, the only restriction on the knot sequence is that it is non-decreasing. It follows that duplicate knots are allowed. Duplication of knots can be used to alter smoothness properties of the spline curves - a useful property in some applications. The heart model we present in this chapter does not require alterations of the smoothness of a curve. Furthermore, we will control the curve's path using weights

called control points, presented in Section 3.2. We will thus not dwell on the theory of knot duplication, and enforce strictly increasing knot sequences.

We now introduce the B-splines, or *basis splines*. There are many different ways to define B-splines [26], but we here present one of the more explicit formulations, namely Cox-de Boor's recursion formula [28]:

Definition 3.1.1. A degree zero B-spline is the step function

$$B_{j,0}(t) = \begin{cases} 1 & \text{if } t_j \leq t < t_{j+1}, \\ 0 & \text{else,} \end{cases} \tag{3.1}$$

and B-splines of degree d can be recursively defined as

$$B_{j,d}(t) = \frac{t - t_j}{t_{j+d} - t_j} B_{j,d-1}(t) + \frac{t_{j+d+1} - t}{t_{j+d+1} - t_{j+1}} B_{j+1,d-1}(t). \tag{3.2}$$

Here, a degree zero B-spline $B_{j,0}$ is a step function evaluating to 1 in the knot span $[t_j, t_{j+1}]$ and zero elsewhere. Higher degree B-splines are piecewise polynomials created by calculating a weighted average over two B-splines of degree one less. This is visualized in Figure 3.1 for B-splines up to degree $d = 3$ on the knot vector $\mathbf{t} = [0, 1, 2, 3, 4]$. Figure 3.1b shows three B-splines of degree $d = 1$ created by pairwise weighted averages of the degree $d = 0$ B-splines in Figure 3.1a. Similarly, Figure 3.1c shows two B-splines of degree $d = 2$ created by pairwise weighted averages of the degree $d = 1$ B-splines in 3.1b and Figure 3.1d shows a degree $d = 3$ B-spline which is the weighted average of the two degree $d = 2$ B-splines in 3.1c.

There is a lot to be said concerning B-spline theory and its different applications. For this thesis, it will suffice to discuss only the relevant theory. In the following section, we look at some properties of B-splines necessary to build our heart model.

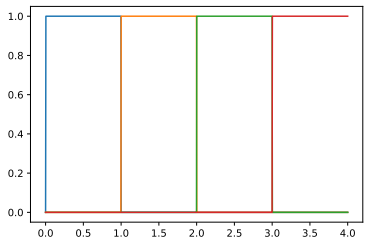
3.1.1 Some Properties of B-splines

Local Support

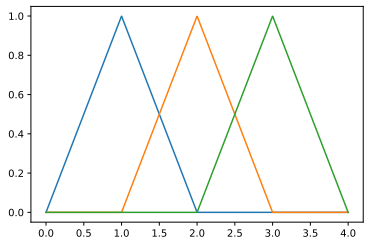
The right term in Equation (3.2) shifts one B-spline to the right in every recursion step. A degree d B-spline requires d recursion steps before a degree $d = 0$ B-spline is reached. Consequently, the j'th B-spline of degree d depends on $d+1$ degree zero B-splines, namely $B_{j,0}, B_{j+1,0}, \ldots, B_{j+d,0}$. It follows that $B_{j,d}$ has support on the interval $[t_j, t_{j+d+1}]$. We call this property the local support property of B-splines. Reversely, we can also conclude that on a given knot span $[t_j, t_{j+1}]$, at most $d+1$ B-splines of degree d are nonzero. These are the B-splines $B_{j-d}, B_{j-d+1}, \ldots, B_j$.

Uniform B-splines

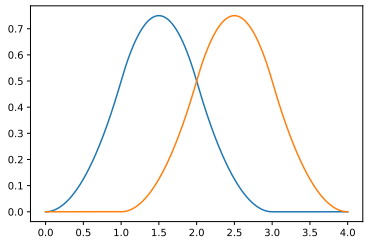
In the example in Figure 3.1, we used equidistant knots. B-splines for which there is a fixed distance h between the knots, are called uniform B-splines. We will refer to the distance h as the knot spacing. We denote by $\mathbf{t}^h$ a knot vector with a fixed knot spacing h.



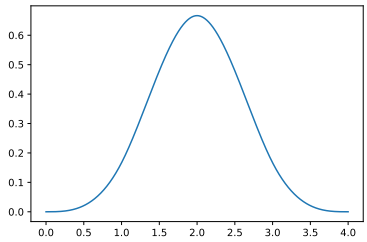
(a) B-splines of degree 0



(b) B-splines of degree 1



(c) B-splines of degree 2



(d) B-spline of degree 3

Figure 3.1: The illustration shows B-splines defined on the knot vector $\mathbf{t} = [0, 1, 2, 3, 4]$ of increasing degree. The recursive formulation in Equation (3.2) shows that B-splines of degree d are a combination of two B-splines of degree $d-1$. The degree $d = 3$ B-spline in the bottom right corner is constructed recursively by the four degree $d = 0$ B-splines in the upper left corner.

In these cases Cox-de Boor's recursion formula (Definition 3.1.1) simplifies to

$$B_{j,0}(t) = \begin{cases} 1 & \text{if } t_j \leq t < t_{j+1} \\ 0 & \text{else} \end{cases}. \tag{3.3}$$

and

$$B_{j,d}(t) = \frac{t - t_j}{dh} B_{j,d-1}(t) + \frac{t_{j+d+1} - t}{dh} B_{j+1,d-1}(t) \tag{3.4}$$

$$= \frac{(t - t_j)B_{j,d-1}(t) + (t_{j+d+1} - t)B_{j+1,d-1}(t)}{dh}. \tag{3.5}$$

Uniform B-splines are convenient because they simplify some of the theory as well as being straight forward to generate.

Derivatives

Differentiating the recursive formula in Equation (3.2) yields

$$\frac{\mathrm{d}}{\mathrm{dt}} B_{j,d}(t) = d\left(\frac{B_{j,d-1}(t)}{t_{j+d} - t_j} - \frac{B_{j+1,d-1}(t)}{t_{j+d+1} - t_{j+1}}.\right) \tag{3.6}$$

We see that the derivative of a B-spline has a simple relationship with B-splines of one degree less. The proof of this result [29] is a bit technical and is not included in this thesis. We also note that B-splines of degree d are d times differentiable at the knots. Between the knots, B-splines are polynomials and consequently smooth.

Positivity of B-splines and Partition of Unity

The recursive formulation of Equation (3.2) implies that B-splines of any degree are weighted averages of degree zero B-splines. The degree zero B-splines are either 0 or 1, and thus

$$0 \leq B_{j,d}(t) \leq 1. \tag{3.7}$$

Furthermore, the construction by weighted averages leads to the partition of unity property [30]

$$\sum_j B_{j,d}(t) = 1. \tag{3.8}$$

3.1.2 Spline Functions

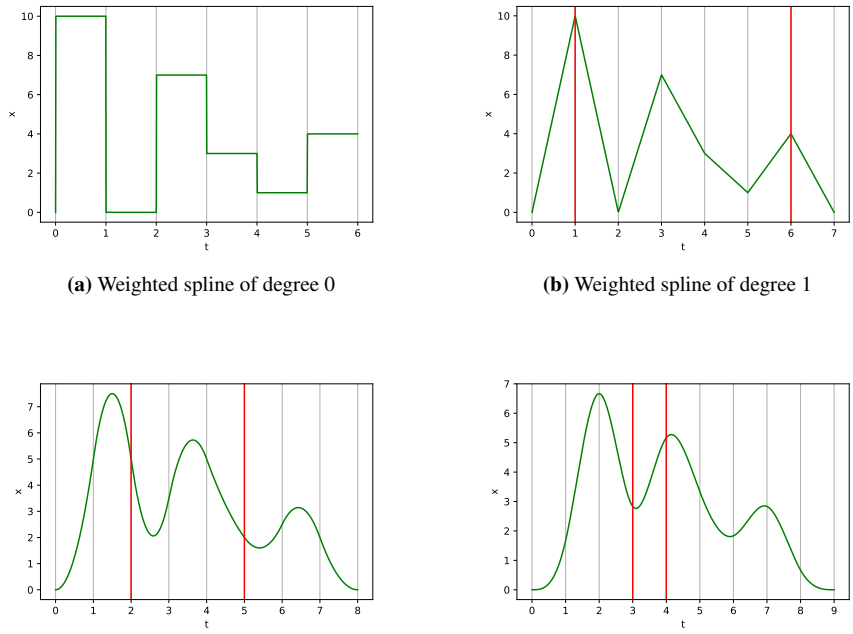
We recall that splines are piecewise polynomials glued together on breakpoints called knots, with some prescribed smoothness. B-splines can be used to approximate functions by assigning weights to them and evaluating the sum

$$S_d(t) = \sum_{j=0}^{n} w_j B_{j,d}(t). \tag{3.9}$$

The elegance of B-splines lies in the property that weighted sums like this generates functions $S_d(t)$ which automatically have continuous derivatives up to degree $d-1$ if the knot vector $\mathbf{t}$ is strictly increasing [30]. We call $S_d(t)$ a spline of degree d.

Support and Domain

The spline in Equation (3.9) is a sum over $n+1$ pairs of weights and B-splines. In Section 3.1.1 we saw that $B_{j,d}$ has support on the knot span $[t_j, t_{j+d+1}]$. Thus the support of the spline $S_d(t)$ is the combined support of the $n+1$ B-splines of degree d, specifically the interval $[t_0, t_{n+d+1}]$. It follows that we need $m+1$ knots where $m = n+d+1$ to define the $n+1$ B-splines necessary to evaluate $S_d(t)$. Figure 3.2 shows weighted B-splines of increasing degree with weights $\mathbf{w} = [10, 0, 5, 3, 1]$. The number of knots increases with



(a) Weighted spline of degree 0

(b) Weighted spline of degree 1

(c) Weighted spline of degree 2

(d) Weighted spline of degree 3

Figure 3.2: The illustration shows splines of increasing degree with weights $\mathbf{c} = [10, 0, 5, 3, 1]$. Notice that the number of knots increases with the degree to give support to the B-splines. The red lines indicate the start and end of the interval of the spline with support from only non-zero B-splines.

the degree of the spline to define the underlying B-splines. We henceforth denote the knot vector necessary to define $n + 1$ B-splines by $\mathbf{t} = (t_j)_{j=0}^{m}$, where $m = n + d + 1$.

We note that for $d > 0$, the splines evaluate to zero at both endpoints. This is because the knot spans $[t_0, t_1]$ and $[t_{m-1}, t_m]$ each only have one B-spline which is non-zero at the interval. We remember from Section 3.1.1 that at most $d + 1$ B-splines of degree d are non-zero at any given knot span. On $[t_0, t_1]$ only $B_{0,d}$ is non-zero, and at $[t_{m-1}, t_m]$ only $B_{n,d}$ is non-zero. In fact, only the interval $[t_d, t_{m-d}]$ has $d + 1$ non-zero B-splines. For this reason we choose to only define splines of the form (3.9) on the domain $[t_d, t_{m-d}]$. We refer to the weights $(w_j)_{j=d}^{m-d}$ as *inner weights* and the remaining weights as *outer weights*. This definition does not remove the influence on the spline by the outer weights, but it does imply that they have less influence than the inner weights. The red lines in Figure 3.2 indicate the start and end of the interval $[t_d, t_{m-d}]$ which has $d + 1$ non-zero B-splines. A degree $d = 0$ spline $S_0(t)$ is a piecewise constant function with the constant values being equal to its weights, as seen in Figure 3.2a. A degree $d = 1$ spline $S_1(t)$, as seen in Figure 3.2b, is a linear interpolation of its weights. Higher degree splines will in general not pass through the weights. We discuss the interpretation of weights more in detail in Section 3.2.2.

3.2 B-spline Curves

A spline *curve* is a vector-valued parametric curve in $\mathbb{R}^q, q \geq 2$ defined by piecewise polynomials in its argument t. The most important property of B-splines is that they are basis functions for general spline curves. Any spline curve can be expressed as a linear combination of B-splines and weights $\mathbf{c_i}$ called control points. This is the exact formulation we investigated in Section 3.1.2, with the additional modification that the weights must be vectors. We will refer to the degree d of the spline curve as the degree of the piecewise polynomials on t. Based on the discussion in Section 3.1.2, we will only define spline curves on the interval for which there are $d+1$ non-zero B-splines, namely the interval $[t_d, t_{m-d}]$.

We summarize this discussion in the following lemma:

Lemma 3.2.1. *A degree d spline curve $S_d(t)$ in $\mathbb{R}^q$ is uniquely defined by a knot vector $\mathbf{t} = (t_j)_{j=0}^m$ and a set of control points $\mathbf{c}_j \in \mathbb{R}^q, q \geq 2$ such that*

$$S_d(t) = \sum_{j=0}^{n} \mathbf{c}_j B_{j,d}(t), \;\; t \in [t_d, t_{m-d}]$$

Furthermore, the space of all spline curves in $\mathbb{R}^q$ of degree d and knots $\mathbf{t}$ is

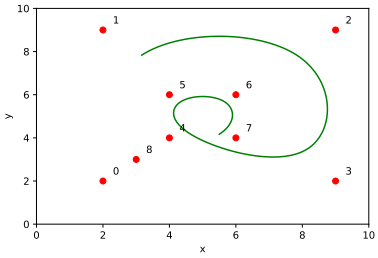
$$\mathbb{S}^q_{d,\mathbf{t}} = \{\sum_{j=0}^{n} \mathbf{c}_j B_{j,d} \mid \mathbf{c}_j \in \mathbb{R}^q \text{ for } 0 \leq j \leq n\}.$$

While we will mostly be interested in the notion of closed periodic B-spline curves, it is useful to firstly define an open B-spline curve. The next two sections discuss open and closed curves.

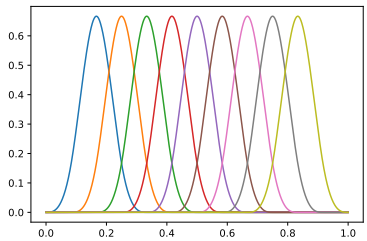
3.2.1 Open, Closed and Periodic Curves

An open spline curve is simply a spline curve where the start and end of the curve are disconnected. The left image in Figure 3.3 shows an open curve of degree $d = 3$ defined on the 8 control points shown in red. The numbers beside the control points in the figure indicate this order. The right image in the figure shows the corresponding B-splines of degree $d = 3$ used to create the spline curve. The B-splines are defined on a knot vector with $m+1$ knots on the interval $[0, 1]$. The spline curve is defined on a subinterval of $[0, 1]$, namely the interval $[t_d, t_{m-d}]$.

A closed curve is a curve where the start and the end of the curve are connected. We are mainly interested in the special case of closed periodic spline curves. A closed periodic spline curve has the additional property that the join between the beginning and end of the curve has the same smoothness properties as the other knots. As we saw in Section 3.1.2, this property is $(d-1)$-times continuous differentiability. Periodicity is achieved by forcing the end of the curve to mimic the trajectory of the beginning of the curve. Uniform B-splines of the same degree are identical, so it will suffice to apply some periodicity to the control points. The local support property of B-splines presented



(a) Example of an open spline curve of degree $d = 3$ in $\mathbb{R}^2$. The 8 control points are shown as red dots, and the number indicate their ordering.



(b) B-splines of degree $d = 3$.

Figure 3.3: The left image shows an open spline curve of degree 3 and the right image shows a series of B-splines of degree $d = 3$ from which the spline curve in left image is created. The B-splines are defined on a knot vector with uniform spacing $h = 1/12$ on the interval $[0, 1]$, and the spline curve is defined on the interval $[0.25, 0.75]$.

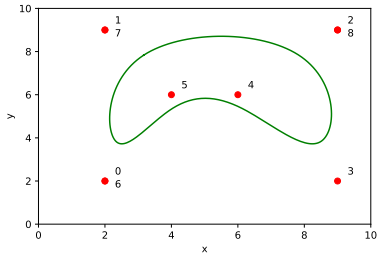
in Section 3.1.1 states that on a given knot span $[t_j, t_{j+1}]$ only the degree d B-splines $B_{j-d}, B_{j-d+1}, \ldots, B_j$ can be non-zero. It follows that the knot span only is affected by the control points $c_{j-d}, c_{j-d+1}, \ldots, c_j$. The simplest procedure to enforce periodicity of a closed spline curve is thus to wrap the d first and d last control points. That is, we let a spline curve be defined by a knot vector as defined earlier and a sequence of control points $\mathbf{c} = (c_i)_{i=1}^n$ for which $c_0 = c_{n-d}, c_1 = c_{n-d+1}, \ldots, c_{d-1} = c_n$. This spline curve will be periodic on the interval $[t_d, t_{m-d}]$ and will have the desired smoothness properties.

The left image of Figure 3.4 shows a closed periodic spline curve of degree $d = 3$, and the right image shows the underlying B-splines of degree $d = 3$. Notice that the closed curve in Figure 3.4a is constructed from the same B-splines as the open curve in Figure 3.3a. The control points are visualized as red dots and numbered by their ordering. In the closed periodic curve we see that $c_0 = c_6$, $c_1 = c_7$ and $c_2 = c_8$. If the numbering of the control points were removed, there would be no way of telling the curve's start point or direction.

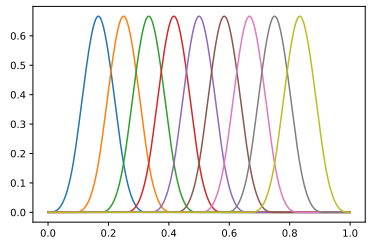
3.2.2 The Role of Control Points and the Control Polygon

A B-spline curve with a given degree d and a prespecified number of control points $n + 1$ can be modified by modifying the control points' location or changing the knots. Modifying the control points is arguably a simpler and more predictable procedure, and is the method we will use.

A spline curve with strictly increasing knots is guaranteed to pass through its control points for degree $d < 2$, as discussed in the previous section. Higher degree spline curves will "bend" towards the control points, but it is difficult to predict the exact trajectory. From Lemma 3.2.1 it is clear that a control point $\mathbf{c}_j$ affects the part of the spline curve $S_d(t)$ for which $B_{j,d}(t)$ is non-zero. From Section 3.1.1 we know that $B_{j,d}(t)$ has support



(a) Example of a closed spline curve of degree $d = 3$ in $\mathbb{R}^2$. The 8 control points are shown as red dots, and the number indicate their ordering. Notice that the $d = 3$ first and last control points are wrapped.



(b) B-splines of degree $d = 3$.

Figure 3.4: The left image shows a closed spline curve of degree $d = 3$ and the right image shows a series of B-splines of degree $d = 3$ from which the spline curve in left image is created. The B-splines are defined on a knot vector with uniform spacing $h = 1/12$ on the interval $[0, 1]$, and the spline curve is defined on the interval $[0.25, 0.75]$.

on the knot span $[t_j, t_{j+d+1}]$. Thus, $\mathbf{c}_j$ only affects the spline curve on $[t_j, t_{j+d+1}]$.

The *control polygon* of a spline curve is the polygon created by connecting all the control points sequentially with straight lines. It can be shown, due to the convex combinations in the construction in Equation (3.2), that a spline curve will always lie inside the convex hull of the control polygon [30]. However, the spline curve formulation in Lemma 3.2.1 can not distinguish between a B-spline evaluating to zero and the corresponding control point being zero. We, therefore, need to include 0 as a control point in the convex hull property.

We established in Section 3.1.1 that on some given interval $[t_j, t_{j+d+1})$ only the B-splines $B_{j-d}(t), B_{j-d+1}(t), \dots B_j(t)$ may be non-zero. It follows that only control points $\mathbf{c}_{j-d}, \mathbf{c}_{j-d+1}, \dots \mathbf{c}_j$ affect the given interval. Once again we have to include zero as a control point in the general formulation. However, on the interval $[t_d, t_{m-d}]$ we have shown that there are $d + 1$ non-zero B-splines, and consequently zero does not need to be included on this interval. We can now strengthen the convex hull property to the following:

Lemma 3.2.2. *For $t \in [t_j, t_{j+d}]$ a spline curve $S_d(t)$ will lie inside the control polygon spanned by the control points $0, \mathbf{c}_{j-d}, \mathbf{c}_{j-d+1}, \dots \mathbf{c}_j$. Furthermore, for $t \in [t_j, t_{j+d}] \subset [t_d, t_{m-d}]$ a spline curve $S_d(t)$ will lie inside the control polygon spanned by the control points $\mathbf{c}_{j-d}, \mathbf{c}_{j-d+1}, \dots \mathbf{c}_j$.*
Consequently, a closed periodic spline curve always lies in the precise control polygon spanned by its control points.

Figure 3.5 shows a closed spline curve and convex hulls of four consecutive control points.

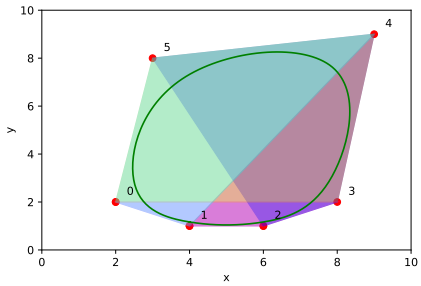


Figure 3.5: A closed spline curve of degree $d = 3$ defined by six independent control points and three wrapped control points. The coloured polygons visualize the convex hull of four consecutive control points.

3.2.3 Advantages of B-spline Based Curves

Before we dive into defining our spline-based heart model, we take a moment to look at why B-spline curves were chosen over equivalent representations. Among spline-based methods, the closely related Bezier-splines and B-splines are frequently used, especially in computer graphics. B-splines are a generalization of Bezier-splines, requiring more information in its creation and having a more complex theory [30]. B-splines make up for the higher complexity by several advantages. Firstly, the number of control points is independent of the degree of the splines. This property makes it possible to create low-degree spline curves with the flexibility and customizability that a high number of control points allow. In Bezier-splines, the control points and the degree are directly related. Secondly, changing any control point of a Bezier-spline affects the whole curve, making it difficult to modify the curve in only a specific section of its domain. This property is undesirable in our learning setup, which will be discussed in more detail later. The local support property of B-splines makes it much easier to modify a section of the spline curve. Thirdly, the strong convex hull property of B-splines grants higher and more intuitive control over the curve's trajectory. Bezier-splines do not share this property.

In general, choosing a framework for the heart shape model is a trade-off between the complexity of the theory, practicality, and accuracy of representation. In this thesis, we chose B-splines as the best compromise between the two. A generalization of B-splines called NURBS [31] provides excellent flexibility and precision, but was deemed to complex given the time restrictions on the thesis.

3.3 A Spline-based Model of the Heart

We are now equipped to develop our parametric heart shape model. We want a B-spline curve of degree $d = 3$ to represent the outline of the heart shape. The relatively low degree is chosen to avoid accuracy issues associated with high-degree polynomials [32]. Instead, the flexibility of the model comes from the number of control points. The number of control points used in our heart model reflects a trade-off between flexibility and bias.

A large number of control points allow the representation of complex shapes, but is more prone to overfitting in the learning process. Oppositely, a low number of control points is less prone to overfitting but will carry a strong bias in the heart shape and will not be able to represent complex shapes. Additionally, many control points will increase the risk of curve intersections difficult for the learning setup to escape. This aspect is more closely examined in 3.3.1. In this thesis we chose to use 8 independent control points $\mathbf{c}_i \in \mathbb{R}^2$. This relatively low number affects the resulting B-spline curve's ability to represent particularly complex shapes. The reasoning behind the choice is that a higher number of control points make learning more difficult. Since we are investigating a novel approach, we are primarily interested in whether the method works, and we are less concerned with optimizing the performance.

The B-spline curve should be closed and periodic to best represent the heart shape. To satisfy the choice of 8 independent control points, we, therefore, need a total of 11 control points, for which the first and last 3 are wrapped, as discussed in Section 3.2.1. From Lemma 3.2.1 we need 15 knots on some interval $[a, b]$ to provide support for the B-splines. Note that while the values a and b do not matter for our definition, we normalize the interval to $[0, 1]$ due to numerical accuracy issues concerning floating-point arithmetic computations [33]. We will thus use the uniform knot vector $\mathbf{t}$ obtained by partitioning $[0, 1]$ into 15 equally spaced knots where the first and last are 0 and 1, respectively.

Using the definition of a spline space in Lemma 3.2.1, we specify our heart model:

Definition 3.3.1. The outline of the heart shape is represented by a degree $d = 3$ closed periodic spline curve $S_3(t)$ with 11 control points $\mathbf{c}_i \in \mathbf{R}^2$ and knot vector $\mathbf{t}^h, h = 1/14$ consisting of 15 equally spaced knots on $[0, 1]$. This spline curve lies in the spline space

$$\mathbb{S}^2_{3,\mathbf{t}} = \{\sum_{j=0}^{10} \mathbf{c}_j B_{j,d} | \mathbf{c}_j \in \mathbb{R}^2 \text{ for } 0 \leq j \leq n-1\},$$
$$\text{where } \mathbf{c}_0 = \mathbf{c}_8, \mathbf{c}_1 = \mathbf{c}_9, \mathbf{c}_2 = \mathbf{c}_{10}$$

3.3.1 Considerations with the Proposed Model

In restricting the number of control points in our heart model, we are also restricting the flexibility of the spline curve. Using 8 independent control points allows for shapes with some complexity, but can not accurately represent shapes with many complex regions. The idea is that our proposed heart model will be able to provide a good representation for the general shape of the heart, even though it will probably miss areas where the path of the curve changes direction rapidly.

Some sequences of control points cause the closed curve to intersect itself, as shown in Figure 3.6. This behavior creates shapes that are impossible in a real heart. We will see that this poses a problem for our supervised learning image segmentation method presented in the next chapters. Unfortunately, there is not any straight forward way of guaranteeing that the curve does not intersect itself. Some possible solutions revolve around detecting intersections and modifying the control points so that the curve no longer intersects itself.

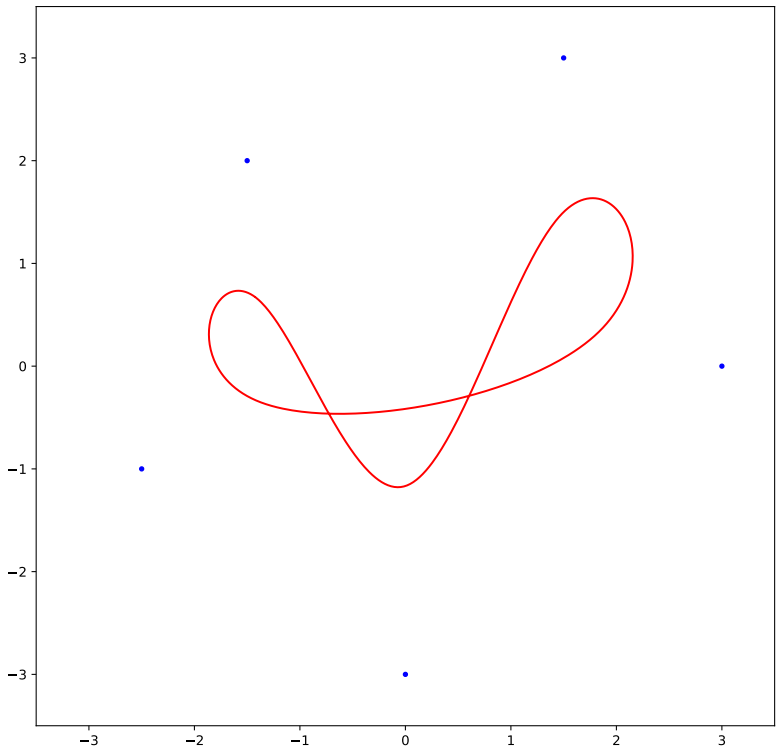


Figure 3.6: A closed spline curve of degree $d = 3$ defined by the five control points in blue. The curve contains twists which we ideally should avoid, as there can be no such twists in the true heart shape.

Chapter 4

Curve Segmentation as Supervised Learning

In the previous chapter, we defined a parametric curve representation of the heart shape. In this chapter, we put the proposed heart model in a deep learning framework to construct a complete image segmentation method as a supervised learning task.

4.1 Curve Segmentation using Deep Learning

Definition 3.3.1 in Section 3.3 describes the spline space in which we are looking for curves to represent the heart shape. Since the knot vector, the degree of the spline curve, and the number of control points are all fixed, modifying the control points is the only means by which we can adjust the spline curve. Our curve segmentation task can thus be defined as follows:

Task 4.1.1. Given some input MRI-image, the segmentation output is a sequence of 8 control points $\mathbf{c}_i \in \mathbb{R}^2$ defining a spline curve in $\mathbb{S}^2_{3,\mathbf{t}^h}$ with $h = 1/14$ on $[0, 1]$, which encloses the heart pixels in the input image.

We propose the Curve Segmentation task be solved as a supervised learning task, using a dataset of Cardiac MRI-Images denoted X_i and corresponding ground-truth labels denoted Y_i as discussed in Section 2.2.4. To achieve this, we define a Convolutional Neural Network [25] (CNN), which takes MRI-images as input and yields predicted control points as output. Specifically, the output of the network from an input image X_i is a vector containing the x and y coordinates of the eight predicted control points. The transformation between the output vector and the control point sequence is trivial so that we will keep the notation used in Chapter 3. Namely, we denote by $\hat{\mathbf{c}}^i = (\hat{c}^i_j)_{j=0}^7$ the sequence of predicted control points based on an input image X_i. Furthermore, the j'th control point of $\hat{\mathbf{c}}^i$ is denoted by $\hat{c}^i_j$.

The goal is that, given a training dataset of cardiac MRI-images, the network will learn to accurately perform the curve segmentation task defined above. This learning setup is nontraditional in that the output of the network is in a different format than the ground truth images in the training dataset. To handle this mismatch, we need a custom loss function. These topics are thoroughly discussed in the next chapter. In the next section, we discuss the architecture of our network.

4.2 Choice of Network Architecture

Image segmentation on medical images has, for the last years, been dominated by approaches and architectures building on the U-Net, introduced in Section 2.3. It was natural to try the U-Net architecture also for the curve segmentation method proposed in this thesis. The symmetric architecture of the U-Net generates an output matrix with the same dimensions as the input image. To get the output in the format discussed in the last section, a fully connected layer of 16 nodes was added to the end of the network. However, we achieved no learning when using the U-Net. It is difficult to say precisely why the U-Net was a bad fit. As discussed in Section 2.3, the decoder part of the U-Net uses skip-connections to retrieve spatial information from the high-resolution images at the beginning of the network. It seems that this spatial information not easily translates to locating good control points for our curve segmentation.

Instead, we used a simpler CNN architecture, specifically a modified version of the ConvNet proposed by Krizhevsky et al. [34]. This ConvNet is typically used for image classification. It is not evident that an image classification network should do well on this task. However, unlike the U-net, the ConvNet is built to generate a low dimensional output.

4.2.1 Curve CNN

CNN's used for image recognition leverage convolutional layers and max-pooling layers to sequentially downsample the input image while increasing the number of feature maps. In this way, the network can identify increasingly complex features in the input image. A convolutional layer has several learnable filters which convolve across the input image and yields a feature map. In the proposed network architecture, the size of the filter is 3×3, and the number of filters is doubled at each convolutional layer, starting at 32. A padding preserves the image size through the convolutions. The max-pooling layers downsample the image size by only sending the maximum value across some sliding filter to the next layer. The proposed network uses max-pooling filters of size 2×2. The most common activation function for CNN's is the Rectified Linear Unit [35] (ReLU), which is defined as $f(x) = \max(0, x)$. ReLU has many advantages, yielding sparse activations, efficient computations and tackling the vanishing gradient problem [36]. Leaky ReLU [37] is an improvement of the traditional ReLU [38], created to avoid the phenomenon of ReLU death [39] in which the signal from ReLU's get stuck at zero. This is achieved by letting the function have a small, positive gradient even when it is not activated. The function for

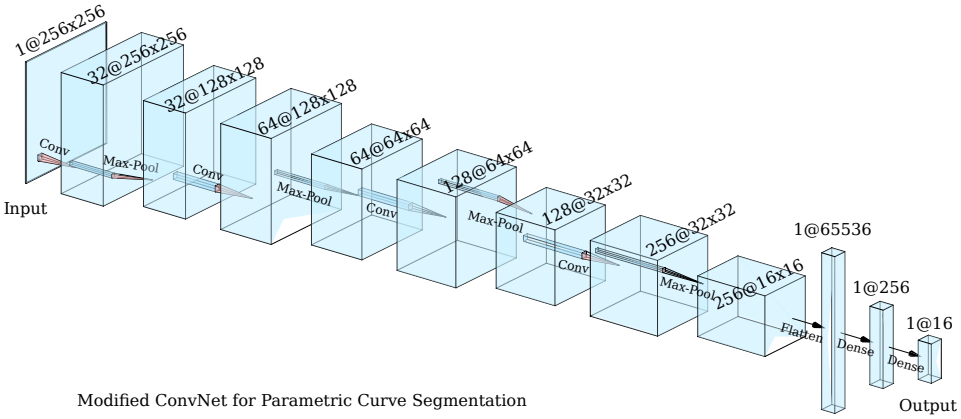


Figure 4.1: The Curve CNN used in the experiment in Chapter 6.

Leaky ReLU is presented in Equation (4.1).

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{else} \end{cases}. \tag{4.1}$$

Each convolutional layer in the proposed architecture is followed by a Leaky ReLU activation function.

The proposed network architecture Curve CNN is presented in Figure 4.1. The numbers at the top indicate the dimension of each operation's output on the format *number of channels @ image height* $\times$ *image width*. *Conv* indicates a convolutional layer, *Max-Pool* indicates a max-pooling layer, *Flatten* refers to the transformation from any tensor to a single vector holding all the values of the tensor and *Dense* indicate a fully connected layer. The input to the network is a 256×256 image. A series of convolutions and max-pooling operations are then applied to the image. After the last max-pooling operation, the output is 256 channels of 16×16 images. A flattening operation transforms the output to a vector of size 65536. The last two layers are fully connected, yielding an output of the desired size. Lastly, a Sigmoid activation function is applied, forcing the output values to lie in the range of the input image's height and width. In this way, the control points can be forced to lie in the input image domain. The network was implemented using the Keras API [40] in TensorFlow 2.0 [41].

Chapter 5

Loss Function for Curve Segmentation

We established in Chapter 4 that the output of our proposed learning method is a vector of x and y coordinates for control points denoted by $\hat{\mathbf{c}}^i$ defining a parametric curve representation of the heart shape. We do not, however, have access to ground-truth parametric curves. As far as we know, no such dataset exists, and it would be very time consuming to generate one. Thus, the task of validating the predicted curves is not a trivial process. Instead, our ground truth images are pixel-wise segmentation maps in the form of matrices denoted by Y_i, as discussed in Section 2.2.4. We, therefore, require a loss function that translates between these two representations. There is no immediate metric to compare a parametric curve to a pixel-wise image, but multiple approaches are possible. The method presented in this thesis uses the most straightforward approach: drawing an image of the predicted curve and comparing the curve image P_i with the ground truth image Y_i. The most significant advantage of this setup is that it allows us to utilize traditional image segmentation procedures for evaluating the similarity between two pixel-wise classifications. This chapter contains a thorough description of the loss function and its gradient.

5.1 Outline of Loss Function

The construction of the loss function presented in this chapter can be divided into two steps. The first step is the process discussed in Section 5.2, which aims to transform the predicted curve representation of the heart shape to a curve image represented by a matrix. We refer to this process as *curve pixelation*. We define a prediction matrix P_i with the same dimensions as the input image X_i and initialize it to zero. We then trace the predicted spline curve and fill in values, representing each pixel area intersected by the curve, in the matrix P_i. Finally, we fill in every element of P_i lying completely inside the curve. We will refer to this process as *curve filling*. The generation of the curve image only relies on the predicted control points $\hat{\mathbf{c}}^i$. We define a function $f(\hat{\mathbf{c}}^i) = P_i$ representing the

transformation from control points to a curve image. The second step is to compare the curve image P_i with the corresponding ground truth image Y_i. For this part, we can use existing loss functions for image segmentation. This step is discussed in Section 5.3.

5.2 Transforming a Spline Curve to an Curve Image

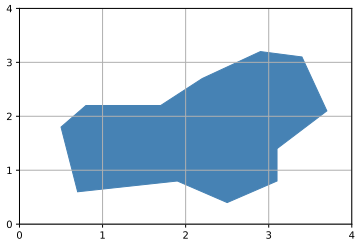
We remember from Chapter 3 that given some knot vector $\mathbf{t} = (t_j)_{j=0}^{n+d+1}$, any spline curve in $\mathbb{R}^2$ of degree d defined on that knot vector can be expressed as a linear combination of B-splines and control points $\mathbf{c}_j \in \mathbb{R}^2$:

$$S_d(t) = \sum_{j=0}^{n} \mathbf{c}_j B_{j,d}(t). \tag{5.1}$$

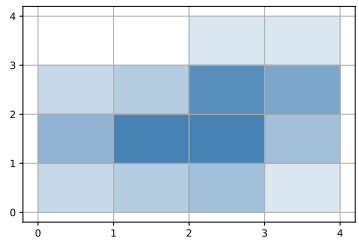
In Section 3.2.2 we established that a spline curve lies completely inside the control polygon spanned by its control points. Furthermore, we saw in Chapter 4 that the control points $\hat{\mathbf{c}}^i$ produced by the network lie in the domain of the input image. It follows that the spline curve $S_d(t)_i$ is contained in the same image domain. Recall from Section 2.1 that this domain has assigned to it a square grid with elements called pixels. We define a matrix P_i representing this square grid. P_i we be the matrix representation of the image of the spline curve defined by $\hat{\mathbf{c}}^i$ using Equation (5.1). P_i should have some value in $(0, 1)$ for every element representing a pixel intersected by the curve, value 1 for every element representing a pixel completely inside the curve and value 0 for any element representing a pixel completely outside the curve, to match the format of the ground truth image Y_i.

We find the elements intersected by the curve by applying a discretization to t and evaluating Equation (5.1) at the resulting steps. We will use a uniform discretization with some stepsize Δt_i. We index the stepsize, because the stepsize might not be equal for every training sample. To ensure that all elements intersected by the curve are found, we would need $\Delta t_i \to 0$. However, as we will discuss in Section 5.2.1 below, we will only demand that Δt_i is small enough to guarantee that the curve pixelation generates an image of a closed curve. The curve filling procedure is described in Section 5.2.2.

When transforming a spline curve to an image we are going from a continous to a discrete representation. In the process we are losing information of the exact trajectory of the spline curve. To restrict the information loss, the elements of P_i intersected by the spline curve should reflect the fraction of the corresponding pixel lying inside the curve. While information about the exact trajectory is still lost, it allows the loss function to make distinctions between predicted spline curves intersecting the same pixels. It follows that the learned segmentation can be more precise. Additionally, as will be discussed in Section 5.4, this procedure makes the gradient of the loss function at least piecewise continuous. Figure 5.1 shows an artificial example of the curve pixelation process. The left image shows a polygon approximating the spline curve, created by interpolating the points of the spline curve evaluated in the pixelation algorithm. The right image shows a possible matrix representation for the pixelation. We can not represent the trajectory of the curve within the pixel, so we represent instead the area of the generated shape within each pixel. Higher color-intensity corresponds to larger area.



(a) Constructed example of curve pixelation. The curve is evaluated once in every pixel, and a polygon is constructed by connecting the evaluation points.



(b) A possible matrix representation for the pixelated curve in (a). The area of the curve in each pixel is represented by color-intensity, where higher color intensity represents a larger area.

Figure 5.1: The left image shows an artificial example of the result of pixelating a curve. The right image shows how the area intersected by the curve in each pixel can be represented in a matrix. Higher color-intensity represents a larger value.

The example in Figure 5.1 has only one spline evaluation per pixel. In practice, the spline curve is evaluated multiple times per pixel. The obtained values allow us to approximate the curve-and-pixel-intersection area by integrating over the area between the spline curve and the edge of the pixel. Specifically, we use the trapezoid rule [42] for numerical integration to approximate the area. The trapezoid integration function `trapezoid` used in the implementation is provided by the Scipy library [43], and takes an array of x-coordinates and an array of y-coordinates as input. The function then calculates the approximated area between the curve defined by the y-values and the x-axis. Depending on the orientation of the curve within the pixel, some adjustments are made to get the correct area, as seen in Algorithm 1.

Algorithm 1 shows the procedure for calculating the curve-and-pixel intersection area. The variable `array` contains the x and y coordinates of spline curve evaluations within a single pixel. Depending on whether the pixel in question contains a north, east, west, or south edge of the spline curve, the numerical integrals are slightly different.

5.2.1 Stepsize Δt

To ensure that the spline curve's image is a closed curve, we need to find some upper bound on the stepsize Δt used to discretize the spline curve argument t. However, the curve pixelation algorithm needs to evaluate Equation (5.1) $1/\Delta t$ times for each pass. It turns out that the size of Δt has a significant impact on the time complexity of our learning setup, so it is crucial that Δt is not unnecessary small. We achieve an image of a closed curve by choosing a stepsize Δt, such that

$$\|S_d(t + \Delta t) - S_d(t)\| \leq 1, \text{ for all } t \text{ in } [0, 1]. \tag{5.2}$$

Algorithm 1: calcIntersection Algorithm. This algorithm takes an array containing spline curve values for a single pixel as input and estimates the fraction of the pixel area laying inside the curve by numerical integration.

input : A vector array of the x and y coordinates for curve evaluations within a pixel.

output: A number representing the approximated area of the curve-and-pixel intersection.

if *north edge* **then**
  area = `trapezoid` (array (y), array (x))
if *south edge* **then**
  area = 1 - `trapezoid` (array (y), array (x))
if *east edge* **then**
  area = `trapezoid` (array (x), array (y))
if *west edge* **then**
  area = 1 - `trapezoid` (array (x), array (y))
return area

The left hand side also has the following bounds

$$\|S_d(t+\Delta t) - S_d(t)\|_2 = \left\| \int_t^{t+\Delta t} S'_d(s)ds \right\|_2 \tag{5.3}$$

$$\leq \int_t^{t+\Delta t} \left\| S'_d(s) \right\|_2 ds \tag{5.4}$$

$$\leq \Delta t \cdot \max_s \left\| S'_d(s) \right\|_2, \tag{5.5}$$

where the first inequality follows from the triangle inequality and limit preservation of non-strict inequalities. Setting this result into Equation 5.2, we get the following upper bound on Δt

$$\Delta t \cdot \max_s \left\| S'_d(s) \right\|_2 \leq 1. \tag{5.6}$$

In Section 3.1.1 we saw that the derivative of a B-spline is a weighted sum of B-splines of degree one less, yielding

$$\max_s \left\| S'_d(s) \right\|_2 = \max_s \left\| \sum_{j=0}^{n+d-1} \mathbf{c}_j \frac{\mathrm{d}}{\mathrm{ds}} B_{j,d}(s) \right\|_2 \tag{5.7}$$

$$\leq \max_s \sum_{j=0}^{n+d-1} \left\| \mathbf{c}_j \right\|_2 \cdot \left| d \left(\frac{B_{j,d-1}(s)}{t_{j+d} - t_j} - \frac{B_{j+1,d-1}(s)}{t_{j+d+1} - t_{j+1}} \right) \right|. \tag{5.8}$$

From Section 3.1.1 we know that the range of B-Splines is the closed set $[0, 1]$. Thus, the upper bound for the maximum in Equation 5.8 is when $B_{j,d-1}(s) = 1$ and $B_{j+1,d-1}(s) =$

0. We get

$$\max_s \left\| S_d'(s) \right\|_2 \leq d \sum_{j=0}^{n+d-1} \frac{\|\mathbf{c}_j\|_2}{t_{j+d} - t_j}. \tag{5.9}$$

For Uniform B-Splines as discussed in Section 3.1.1, we have $t_{j+d} - t_j = dh$, yielding

$$\max_s \left\| S_d'(s) \right\|_2 \leq d \sum_{j=0}^{n+d-1} \frac{\|\mathbf{c}_j\|_2}{dh} \tag{5.10}$$

$$= \frac{1}{h} \sum_{j=0}^{n+d-1} \|\mathbf{c}_j\|_2. \tag{5.11}$$

We can now put this into Equation 5.6, resulting in the bound

$$\Delta t \leq \frac{h}{\sum_{j=0}^{n+d-1} \|\mathbf{c}_j\|_2}. \tag{5.12}$$

Stepsizes satisfying this bound are guaranteed to generate an image of a closed curve in the pixelation algorithm.

5.2.2 Flood Fill

The labels available at training time are matrices $Y_i \in \mathbb{R}^{h \times w}$ for which an element $y_{kl} = 1$ if the corresponding pixel belongs to the heart and $y_{kl} = 0$ if the corresponding pixel does not belong to the heart. The method outlined in the previous sections create a curve image P_i for which only a closed curve has elements with value not equal to zero. In order to compare P_i to a label matrix Y_i, it is necessary to fill the elements of the interior of the closed curve in P_i with a value 1.

To obtain the desired format of P_i, we need some algorithm that identifies connected elements of the same value and changes their value to some target value. The Flood Fill Algorithm [44] takes an image, a start element, a target value, and a replacement value as inputs. The algorithm iteratively uses either a stack or a queue to find all elements of the target value that can be reached by a path from the starting node and replaces that value with the replacement value. Based on the heart usually being smaller than the background in the MRI-images, it makes sense from a time perspective that the start node is in the heart's interior. However, as discussed in Section 3.3.1 in Chapter 3, the spline curve might intersect itself, creating multiple closed curves that should be classified as belonging to the heart. There is no simple way to identify these intersections and determine the number of closed curves. As follows, the curve pixelation algorithm would have no way to identify when all closed curves are filled by the Flood Fill Algorithm, and would necessarily have to check every row of P_i to see if it contains any interior elements.

A better approach is to instead apply the Flood Fill to the area outside of the curve. In the case where boundary pixels of the curve image P_i are classified as heart pixels, this area may also be disconnected. We avoid this problem by padding P_i with one element in both dimensions, and initialize the new elements with value 0. The background will now

be a connected set of value zero elements, and the starting cell for Flood Fill can be set to any of the padded elements to guarantee that the algorithm behaves as intended. The replacement value is set to 1, and the result is a predicted matrix P_i, which has elements of value 0 in the interior of the pixelated curve and elements of value 1 outside the curve. Our desired format is achieved by changing all elements that are either 0 or 1 to the other value, such that the interior gets elements of value 1, and the outside of the curve gets elements of value 0. Finally, we remove the padding.

5.2.3 Implementation of Pixelation Algorithm

To increase the speed of the algorithm, pre-written Python packages are used whenever possible. Specifically, we use the `BSpline` function from the SciPy library [43] and the `floodFill` function from the OpenCV library [45]. The `BSpline` function is constructed by feeding in a knot vector $\mathbf{t}$, a degree d and a set of control points $\mathbf{c}_i$ to a construction function we will call `constructBSpline`. Once constructed, `BSpline` takes in a variable t and returns the x and y coordinates of the B-spline at t. The `floodFill` function takes the arguments described in Section 5.2.2 and returns an image for which the pixelized curve has been filled in. We also use the function `calcIntersection`, described in Section 5.2.

We are now equipped to formulate our complete pixelation algorithm. The function `calcStepsize` calculates the maximum steplength defined in Section 5.2.1. The function `pad` takes a matrix and a number k as argument, and adds k elements of value zero to the beginning and end of each row and column as described in Section 5.2.2. `binaryInvert` makes zeros one and ones zero. The function `floor` is the floor function, returning the greatest integer smaller or equal to its argument. Algorithm 2 shows the complete pixelation process.

5.3 Loss Function

We now introduce the loss function for our supervised learning setup. We recall that the output of the network is a sequence of control points $\hat{\mathbf{c}}^i$ as defined in Chapter 4. P_i is the curve image generated from $\hat{\mathbf{c}}^i$ by the curve pixelation algorithm. The corresponding ground truth image is denoted by Y_i. The image loss is

$$l(P_i, Y_i) = \mathrm{BCE}(P_i, Y_i), \tag{5.13}$$

where the function $\mathrm{BCE}()$ is the Binary Cross Entropy, defined as follows:

Definition 5.3.1. Given a prediction matrix $P = (p)_{ij}$ and a ground truth matrix $Y = (y)_{ij}$, Binary Cross Entropy loss is given by

$$H_p(q) = -\frac{1}{N}\sum_i \sum_j y_{ij} \cdot \log(p_{ij}) + (1 - y_{ij}) \cdot \log(1 - p_{ij}), \tag{5.14}$$

where N is the number of elements $h \times w$ in the matrices.

Algorithm 2: curvePixelation algorithm. This algorithm takes a vector of control points **c** as input at returns a matrix P representing a curve image of the spline curve defined by **c**.

input : A vector **c** of 8 control points in $\mathbb{R}^2$.
output: A matrix P of size $m \times n$ representing the pixelized spline curve.

```
BSpline ← constructBSpline (c, t, d);
Δt ← calcStepsize (BSpline);
while t < 1 do
    x, y ← BSpline (t) ;
    element = floor (x), floor (y);
    if element is same as previous element then
        append x, y to array;
    else
        P [previous element ] ← calcIntersection (array);
        previous element = element;
    t ← t + Δt;
P ← pad (P, 1) startPixel ← findStartFloodFill (P);
P ← floodFill (P, startPixel);
P ← binaryInvert (P) P ← pad (P, -1) return P
```

5.3.1 Regularizing the Control Points

To avoid a training problem where the control points move towards the edge of the image domain at the beginning of the training, we use a small penalty term. The idea is to encourage the control points to stay near the middle of the image domain until some stable learning strategy is found. For a point O representing the middle of the image domain, the penalty term is a simple squared error on the form

$$E(\hat{\mathbf{c}}^i) = \lambda \|\hat{\mathbf{c}}^i - O\|_2 \tag{5.15}$$

which is added to the loss function. To account for the fact that this problem only occurs at the beginning of training, λ decreases during the learning process.

5.4 Gradient

The training of a neural network requires a gradient for the loss function, which is used to update the network's weights through backpropagation [46]. Specifically, the gradient will contain the partial derivatives of the loss function $l()$ with respect to each of the predicted control points. Since each control point $\hat{c}_j$ is a vector in $\mathbb{R}^2$, the partial derivatives will also be vectors in $\mathbb{R}^2$. We here present a numerical approximation to the gradient of the loss function defined in Section 5.3, by using finite differences. A numerical approximation is chosen over the analytical gradient due to continuity issues. Specifically, the analytical gradient is discontinuous at the points where a change in a control point coordinate causes

the curve to enter a new pixel. A finite difference approximation to the gradient alleviates this problem.

5.4.1 Finite Difference Approximation

For a multivariate function $g(x, y) \in C^2$, a central difference approximation to its partial derivative is given by

$$\frac{\partial g(x,y)}{\partial x} = \frac{g(x + \Delta x, y) - g(x - \Delta x, y)}{2\Delta x} + \mathcal{O}(\Delta x^2), \tag{5.16}$$

We call $\delta g = g(x + \Delta x, y) - g(x - \Delta x, y)$ a central difference of f.

Our loss function is not twice differentiable, so we can not say anything about the central difference approximation error. However, due to its symmetry, we believe central differences still provide a better approximation for the gradient than a single-sided difference. We now recall that our curve images can be expressed as $P = f(\hat{\mathbf{c}}) = f(\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_7))$. Denote by $\hat{\mathbf{c}} + \Delta c_j^x$ that a small step Δc is added to the x component of the j'th control point. The same applies to the y component. The central difference for l with respect to the x component of the j'th control point is

$$\delta_j^x l = l(f(\hat{\mathbf{c}} + \Delta c_j^x), Y_i) - l(f(\hat{\mathbf{c}} - \Delta c_j^x), Y_i). \tag{5.17}$$

For the y component the expression becomes

$$\delta_j^y l = l(f(\hat{\mathbf{c}} + \Delta c_j^y), Y_i) - l(f(\hat{\mathbf{c}} - \Delta c_j^y), Y_i). \tag{5.18}$$

Using the notation just introduced, a partial derivative of the loss function with respect to a control point $\hat{c}_j$ has central difference approximation

$$\frac{\partial l(P_i, Y_i)}{\partial \hat{c}_j} = \begin{bmatrix} \partial l(P_i, Y_i)/\partial c_j^x \\ \partial l(P_i, Y_i)/\partial c_j^y \end{bmatrix} = \begin{bmatrix} \delta_j^x l / 2\Delta c \\ \delta_j^y l / 2\Delta c \end{bmatrix} + \mathcal{O}(\Delta c^2). \tag{5.19}$$

The gradient is created by calculating Equation (5.19) for each control point. It follows that the curve pixelation algorithm is called 32 times for each time the gradient is created, making this algorithm a bottleneck concerning computational efficiency.

When training the network, we will use a variable control point step Δc. Specifically, Δc will decrease every 10 epochs. This strategy intends that it will allow the network to find its way out of the local bad minima created by self-intersections of the curve (as discussed in Section 3.3.1) in the beginning of the training phase, while still getting the necessary precision of the gradient towards the end of the training phase.

5.4.2 Implementation of Gradient

Algorithm 3 shows the implementation of the numerical gradient computation. The function `curvePixelation` is presented in Algorithm 2. The function `append` appends an element at the end of a vector.

Algorithm 3: Numerical Gradient Algorithm. This algorithm takes a sequence of predicted control points $\hat{\mathbf{c}}$ as input and returns a numerical approximation of the gradient of the loss function l with respect to each of the control points $\hat{c}_j$.

input : A sequence $\mathbf{c}$ of 8 control points in $\mathbb{R}^2$.
output: A vector of size 16 holding numerical approximation of the gradient of the loss function with respect to each component of every control point.

initialize the empty vector grad;
for c_j **in c do**
  forwardStepX $\leftarrow$ `curvePixelation` $(\mathbf{c} + \Delta c_j^x)$;
  backwardStepX $\leftarrow$ `curvePixelation` $(\mathbf{c} - \Delta c_j^x)$;
  grad $\leftarrow$ `append` $((\text{forwardStepX} - \text{backwardStepX})/\ 2\Delta c)$;
  forwardStepY $\leftarrow$ `curvePixelation` $(\mathbf{c} + \Delta c_j^y)$;
  backwardStepY $\leftarrow$ `curvePixelation` $(\mathbf{c} - \Delta c_j^y)$;
  grad $\leftarrow$ `append` $((\text{forwardStepY} - \text{backwardStepY})/\ 2\Delta c)$;
return grad

Chapter 6

Curve Segmentation on ACDC Cardiac MRI-Images

We now validate the proposed method on a real-world clinical data set of Cardiac MRI-images. The dataset is described in detail in Section 6.1. Specifically, we train a Curve CNN neural network with architecture as described in Chapter 4 to perform the proposed supervised learning Cardiac MRI-image segmentation task. The training scheme utilizes the loss function and gradient described in Chapter 5. We present the results towards the end of this chapter, and the results are discussed in the next chapter. The performance of the proposed method is compared to the performance of a traditional pixel-wise image segmentation setup using a U-Net [24] and trained on the same dataset. The implementation of the Curve CNN can be found at `https://github.com/vetlebh/spline_learn`.

6.1 The ACDC dataset

The ACDC-dataset consists of data from real clinical exams provided by the University Hospital of Dijon, and was created for the MICCAI 2017 challenge. The competition, as described by [1], had the goal of comparing state-of-the-art automatic methods for segmentation and classification based on the MRI-images.

The dataset contains 150 medical exams, each from a different patient. In addition to the MRI-images, each patient's data includes height, weight, and diastolic and systolic phase instants. The dataset is divided into a training set consisting of 100 medical exams and a test set consisting of the remaining 50. The patients are distributed into the following five groups, each consisting of 30 patients' medical exams: healthy subjects, patients with previous myocardial infarction, patients with dilated cardiomyopathy, patients with hypertrophic cardiomyopathy, and patients with abnormal right ventricle. In total, there are 25351 images among the 100 medical exams for training. Out of these images, 1902 contain labels. There are 6-18 slices per patient, and two frames per slice have labels. The

data is stored in NIfTI format, a format developed for neuroimaging.

6.2 Training of the Curve CNN Network

Training was conducted using Google CoLab Pro on a NVIDIa Tesla P100 GPU (graphical processing unit). The optimizer used in training the network was the Keras implementation of the Adam-algorithm [47]. The algorithm is a modification of stochastic gradient descent specifically created for training deep neural networks with large datasets and high-dimensional parameter space. The network was trained on a dataset of 68 individual slices, with a validation set of 8 slices. The training set is just a fraction of the available data, but a larger training dataset's computational cost was too big for the GPU's to handle. The size of the training set strongly influences the learning of Neural Network, and should be taken into consideration when evaluating the model. The U-Net used for comparison was trained on 764 individual slices, more than ten times as many.

The ACDC-dataset comes in a standardized, preprocessed format. The only further processing necessary for training was resizing the images to pixel-resolution $255x255$. Most of the samples had a higher initial resolution and had to be downsized. In order to preserve features in the images, anti-aliasing was used. Also, edge-padding was used in cases were images needed to be upsized, meaning that pixels added in the resize procedure were appointed the same value as their closest neighbor inside the original image. While other preprocessing steps could have improved performance, experimentation with preprocessing steps is outside this thesis's scope. The network was trained in batches of 128.

6.2.1 Evaluation Metrics

In this experiment, accuracy alone does not yield a meaningful measure of the performance of the network. The MRI-images are mostly background, yielding a classification problem where one of the classes is vastly over-represented. In cases where the network can predict the general shape and location of the heart, the accuracy will be nearly 100%, giving the impression that it is an almost perfect segmentation. This is not the case. A weak algorithm classifying all pixels as background could get an accuracy over 90% (if there are few heart-pixels in the relevant image).

Specifically, we look at mean IoU (mean intersection over union), also called Jaccard Index [48]. As the name suggests, IoU is a measure of similarity between two sets measured by dividing the size of the intersection of the sets by the size of the union of the sets. In our case, the IoU measure compares the MRI ground truth mask with the predicted mask. If there are no overlapping pixels in the two masks, the IoU score will be 0, regardless of how many pixels are correctly identified as background. If the two masks are completely overlapping the IoU score will be 1.

We also include precision and recall in our results. Precision is the fraction of correctly classified heart pixels by the total number of predicted heart pixels. Recall is the fraction of correctly classified heart pixels by the total number of true heart pixels.

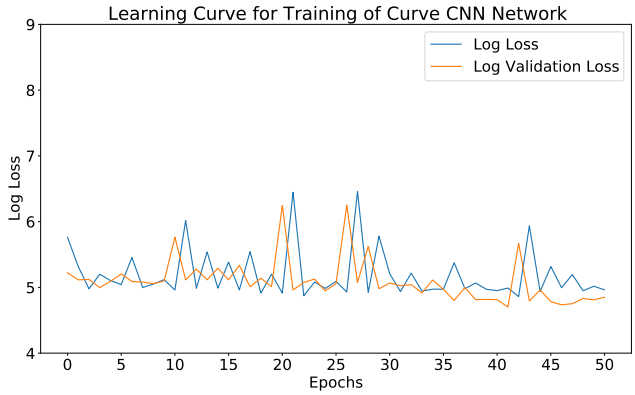


Figure 6.1: Learning curve of a training run of the Curve CNN Network. The y-axis shows log training loss and log validation loss.

Model	Epochs	IoU	Precision	Recall	Accuracy
Curve CNN	60	0.5623	69.07%	19.52%	72.94%
U-Net	60	0.7030	95.52%	80.62%	98.92%
U-Net	180	0.8434	95.61%	94.36%	99.53%

Table 6.1: Metrics on a test set after 60 epochs of the two different models.

6.3 Results

We here present the results of the numerical experiment on the ACDC-dataset. Both the curve CNN and U-net were trained for 60 epochs on data from the same dataset. However, the U-Net was allowed to utilize its much larger capacity for dataset size. Furthermore, metrics for the U-Net after running for 180 epochs are included, to show its full potential. Figure 6.1 shows the learning curve for a training run of 50 epochs for the Curve CNN. There is a small improvement over time, but the learning is extremely slow. Figure 6.2 shows the learning curves of 5 runs of the U-Net for comparison. All but two of the runs achieve significant learning within the first 20 epochs.

Table 6.1 shows performance metrics of the Curve CNN after 60 epochs and the U-Net after 60 and 120 epochs respectively. The average IoU shows that the Curve CNN generally outputs bad predictions. The higher precision than recall indicates that the Curve CNN usually under predicts the size of the heart. In comparison the U-Net outperforms the Curve CNN in all included metrics.

Figure 6.3 and 6.4 show some predictions by the Curve CNN. They are both poor in their own way, and reflects what most of the predictions looked like. In the first, Curve CNN predicts a self-intersecting curve. In the second, the predicted curve overlaps none of the true heart pixels.

Figure 6.5 shows a prediction by the U-Net. The prediction almost perfectly overlaps the true heart pixels.

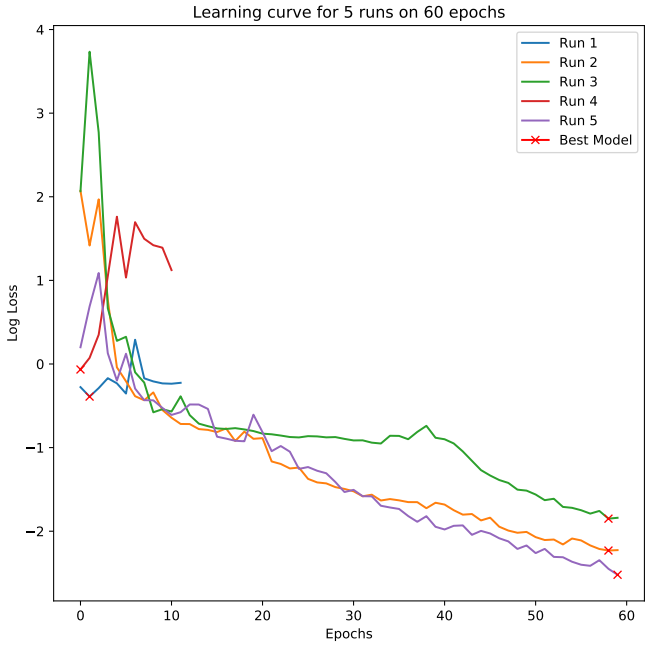


Figure 6.2: Learning curve of five training run of the U-Net Network. The y-axis shows log validation loss.

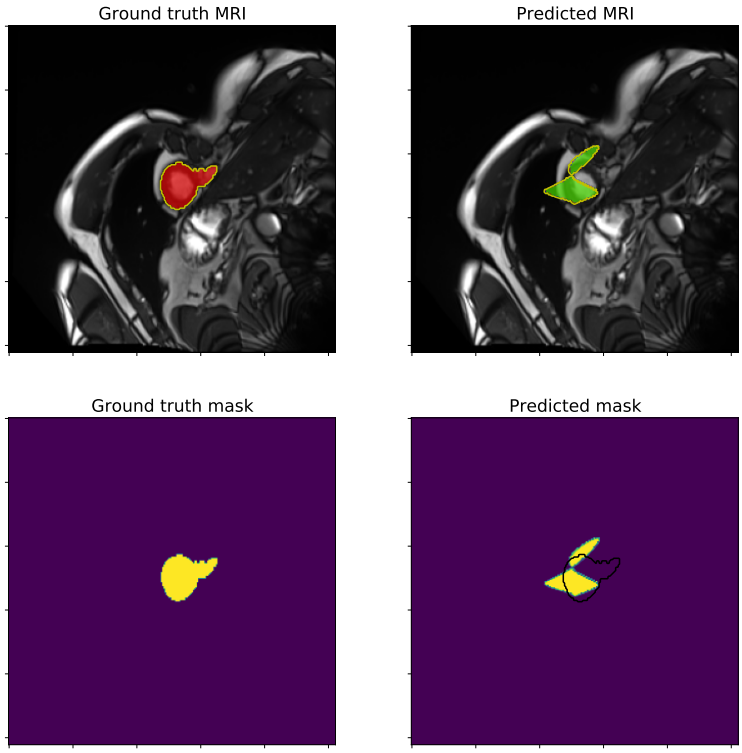


Figure 6.3: Prediction by Curve CNN after 60 epochs. While in the right area of the image, the prediction totally misses the shape of the heart. The predicted curve self-intersects - a problem which happens quite often in the predictions.

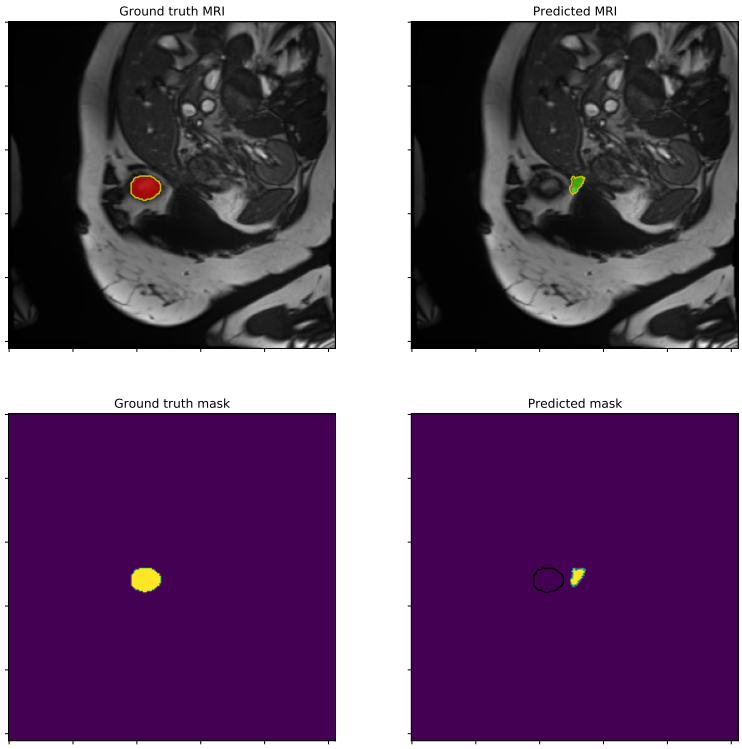


Figure 6.4: Prediction by Curve CNN after 60 epochs. The prediction is not too far off in size, but misses the shape of the heart. Even worse, the prediction also misses the location of the heart and has no overlapping pixels with the ground truth segmentation.

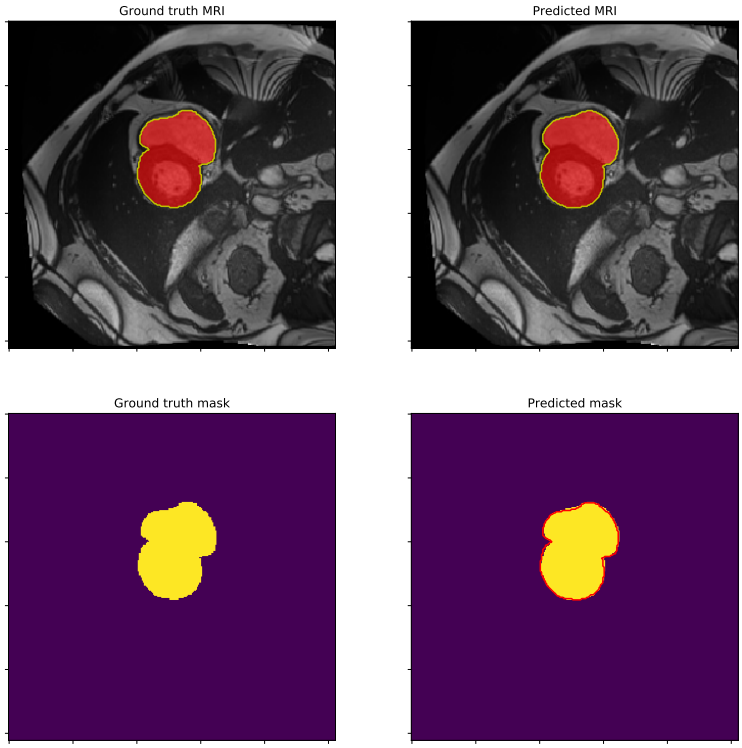


Figure 6.5: Prediction by U-Net after 60 epochs. The U-Net is able to accurately predict the heart shapes.

Chapter 7

Discussion

The results clearly show that the method presented in this thesis does not work in the intended way. While the implemented U-Net achieves high performance after 60 epochs, the Curve CNN produces mostly worthless segmentations. There is some learning, but it is prolonged, and no satisfactory results were achieved within a reasonable time. Part of the problem is that the computational cost of training the proposed network puts strict restrictions on the size of the training set and makes it difficult to experiment with hyperparameters.

The problem might lie in the presented learning strategy as a whole. It could be that the network is simply unable to learn the connection between predicted control point coordinates and spline curves, and between the spline curves and the ground truth segmentation. We believe, however, that it is more likely a matter of improving the model. As a novel method, many aspects of the procedure could be improved, and some might be vital for the network's learning.

The most significant problem with the proposed method is that we do not resolve the issue with self-intersections on the curve, as discussed in Section 3.3.1. Self-intersecting predicted curves are present all through the learning process. Random initialization of the network weights makes it likely that self-intersecting curves are present from the start of the learning process. It is difficult for the network to untangle the self-intersections, because the current loss function has bad local minima on some control point predictions corresponding to self-intersecting curves. The best way to handle this would be to add a loss term, which explicitly penalizes self-intersections. It is possible to implement an algorithm that detects self-intersections, but it would be computationally costly in the current setup, putting an even higher toll on the run time.

In our learning setup, the control points are only rewarded for representing the heart shape accurately. The heart model does not distinguish between representations where the control points have different ordering as long as the spline curve is the same. This may be a weakness. Instead, it could be advantageous to give incentive to the control points to "specialize" in different heart features. The consequences could be a more stable performance and possibly resolving the issue with self-intersections, because the control

points would learn an internal ordering. We saw in Section 2.2.3 that cardiac MRI-images are taken in the same spatial view of a heart at different times during the cardiac cycle. Our learning strategy could take advantage of the temporal dimension of the images by penalizing the movement of a single control point between different images in a sequence. In this way, the network would be encouraged to place a control point on the same place in each image.

A noticeable improvement would be to formulate the problem to allow for a continuous analytical gradient. In the current setup, the loss function is severely complex, and it is possible that the gradient approximation not adequately represents the analytical gradient. To achieve this, one would need to formulate an entirely new loss function.

7.1 Conclusion

We conclude that the presented method does not work in the current setup. However, we firmly believe that given more computational resources, improvements could be made that would significantly enhance the performance of the method. This project was carried out as an initial investigation, and it is therefore too early to conclude on the validity of the method. We hope that further studies explore the mentioned improvements, and a more decisive conclusion can be drawn.

Bibliography

[1] O. Bernard, A. Lalande, C. Zotti, F. Cervenansky, X. Yang, P. Heng, I. Cetin, K. Lekadir, O. Camara, M. A. Gonzalez Ballester, G. Sanroma, S. Napel, S. Petersen, G. Tziritas, E. Grinias, M. Khened, V. A. Kollerathu, G. Krishnamurthi, M. Rohé, X. Pennec, M. Sermesant, F. Isensee, P. Jäger, K. H. Maier-Hein, P. M. Full, I. Wolf, S. Engelhardt, C. F. Baumgartner, L. M. Koch, J. M. Wolterink, I. Išgum, Y. Jang, Y. Hong, J. Patravali, S. Jain, O. Humbert, and P. Jodoin. Deep Learning Techniques for Automatic MRI Cardiac Multi-Structures Segmentation and Diagnosis: Is the Problem Solved? *IEEE Transactions on Medical Imaging*, 37(11):2514–2525, Nov 2018. ISSN 1558-254X. doi: 10.1109/TMI.2018.2837502.

[2] James Dicarlo, Davide Zoccolan, and Nicole Rust. How Does the Brain Solve Visual Object Recognition? *Neuron*, 73:415–34, 02 2012. doi: 10.1016/j.neuron.2012.01.010.

[3] Michael Treml, Jose Arjona-Medina, Thomas Unterthiner, Rupesh Durgesh, Felix Friedmann, Peter Schuberth, Andreas Mayr, Martin Heusel, Markus Hofmarcher, Michael Widrich, Bernhard Nessler, and Sepp Hochreiter. Speeding up Semantice Segmentation for Autonomous Driving. 12 2016.

[4] Patrik Kamencay, Martina Radilova, Robert Hudec, Roman Jarina, Miroslav Benco, and Jan Hlubik. A Novel Approach to Face Recognition using Image Segmentation Based on SPCA-KNN Method. *Radioengineering*, 22:92–99, 04 2013.

[5] Abdulkadir Sengur, Umit Budak, Yaman Akbulut, Murat Karabatak, and Erkan Tanyildizi. 7 - A Survey on Neutrosophic Medical Image Segmentation. In Yanhui Guo and Amira S. Ashour, editors, *Neutrosophic Set in Medical Image Analysis*, pages 145 – 165. Academic Press, 2019. ISBN 978-0-12-818148-5. doi: https://doi.org/10.1016/B978-0-12-818148-5.00007-2.

[6] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., USA, 2006. ISBN 013168728X.

[7] Donald W. McRobbie, Elizabeth A. Moore, Martin J. Graves, and Martin R. Prince. *MRI from Picture to Proton*. Cambridge University Press, 2 edition, 2006. doi: 10.1017/CBO9780511545405.

[8] R. Smith-Bindman, J. Lipson, and R. Marcus. Radiation Dose Associated With Common Computed Tomography Examinations and the Associated Lifetime Attributable Risk of Cancer. *Journal of Vascular Surgery*, 51:783, 03 2010. doi: 10.1016/j.jvs.2010.01.042.

[9] Giles Wesley Vick. The Gold Standard for Noninvasive Imaging in Coronary Heart Disease: Magnetic Resonance Imaging. *Current opinion in cardiology.*, 24(6): 567–579, November 2009. ISSN 0268-4705.

[10] World Health Organization. Cardiovascular Diseases (CVDs), 2017. URL `https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)`.

[11] Daniel Lee, Michael Markl, Erica Dall'Armellina, Yuchi Han, Sebastian Kozerke, Titus Kuehne, Sonia Nielles-Vallespin, Daniel Messroghli, Amit Patel, Tobias Schaeffter, Orlando Simonetti, Anne Valente, Jonathan Weinsaft, Graham Wright, Stefan Zimmerman, and Jeanette Schulz-Menger. The Growth and Evolution of Cardiovascular Magnetic Resonance: A 20-year History of the Society for Cardiovascular Magnetic Resonance (scmr) Annual Scientific Sessions. *Journal of Cardiovascular Magnetic Resonance*, 20, 12 2018. doi: 10.1186/s12968-018-0429-z.

[12] Marianne Fraser and Steven Kang. Basic Anatomy of the Heart. URL `https://www.urmc.rochester.edu/encyclopedia/content.aspx?contenttypeid=85&contentid=P00192`.

[13] B.J. Gersh. *Mayo Clinic Heart Book, Second Edition: Completely Revised and Updated*. HarperCollins, 2000. ISBN 9780688176426.

[14] Caroline Petitjean and Jean-Nicolas Dacher. A Review of Segmentation Methods in Short Axis Cardiac MR Images. *Medical Image Analysis*, 15(2):169–184, 2011. ISSN 1361-8415. doi: 10.1016/j.media.2010.12.004.

[15] Hidefumi Kobatake and Yoshitaka Masutani. *Computational Anatomy Based on Whole Body Imaging: Basic Principles of Computer-Assisted Diagnosis and Therapy*. Springer Publishing Company, Incorporated, 06 2017. doi: 10.1007/978-4-431-55976-4.

[16] Nicholas Ayache. Computational Anatomy and Computational Physiology for Medical Image Analysis. pages 1–2, 10 2005. doi: 10.1007/11569541_1.

[17] Raimond L. Winslow, Natalia Trayanova, Donald Geman, and Michael I. Miller. Computational Medicine: Translating Models to Clinical Care. *Science Translational Medicine*, 4(158):158rv11–158rv11, 2012. ISSN 1946-6234. doi: 10.1126/scitranslmed.3003528.

[18] William Alexander Newman Dorland et al. *Dorland's Illustrated Medical Dictionary*, volume 31. Saunders Philadelphia, 1994.

[19] Sharon Kirschbaum, Timo Baks, Ed Gronenschild, Jean-Paul Aben, Annick Weustink, Piotr Wielopolski, Gabriel Krestin, Pim Feyter, and Robert-Jan Geuns. Addition of the Long-Axis Information to Short-Axis Contours Reduces Interstudy Variability of Left-Ventricular Analysis in Cardiac Magnetic Resonance Studies. *Investigative radiology*, 43:1–6, 02 2008. doi: 10.1097/RLI.0b013e318154b1dc.

[20] Michael Salerno and Christopher Kramer. Advances in Cardiovascular MRI for Diagnostics: Applications in Coronary Artery Disease and Cardiomyopathies. *Expert opinion on medical diagnostics*, 3:673–687, 11 2009. doi: 10.1517/17530050903140514.

[21] Selim Bozkurt. Mathematical Modeling of Cardiac Function to Evaluate Clinical Cases in Adults and Children. *PLOS ONE*, 14(10):1–20, 10 2019. doi: 10.1371/journal.pone.0224663.

[22] Tal Remez, Jonathan Huang, and Matthew Brown. Learning to Segment via Cut-and-Paste. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 39–54, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01234-2.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

[25] K Fukushima. Neocognitron: a Self Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological cybernetics*, 36(4):193—202, 1980. ISSN 0340-1200. doi: 10.1007/bf00344251.

[26] Ewald Quak. About B-splines. Twenty Answers to One Question: What is the Cubic B-spline for the Knots -2,-1,0,1,2? *J. Numer. Anal. Approx. Theory*, 45(1):37–83, Sep. 2016.

[27] Michael S. Floater. Lecture Notes in INF-MAT5340 - Spline Methods. URL `https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF-MAT5340/v07/undervisningsmateriale/`.

[28] Carl de Boor. On Calculating with B-splines. *Journal of Approximation Theory*, 6(1):50 – 62, 1972. ISSN 0021-9045. doi: https://doi.org/10.1016/0021-9045(72)90080-9.

[29] Jana Prochazkova. Derivative of B-Spline function. In *25th Conference on Geometry and Graphics, Czech*, 2005.

[30] C.-K. Shene. CS3621 Introduction to Computing with Geometry, 2014. URL `https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/`.

[31] Munira Mohd Ali, Nur Najmiyah Jaafar, Fazilah Abdul Aziz, and Z. Nooraizedfiza. Review on Non Uniform Rational B-spline (NURBS): Concept and Optimization, Materials Research, pages 338–343, 02 2014.

[32] Andrew Gelman and Guido Imbens. Why High-Order Polynomials Should Not Be Used in Regression Discontinuity Designs. *Journal of Business & Economic Statistics*, 37(3):447–456, 2019. doi: 10.1080/07350015.2017.1366909.

[33] Nicholas M. Patrikalakis and Takashi Maekawa. Chapter 25 - Intersection Problems. In Gerald Farin, Josef Hoschek, and Myung-Soo Kim, editors, *Handbook of Computer Aided Geometric Design*, pages 623 – 649. North-Holland, Amsterdam, 2002. ISBN 978-0-444-51104-1.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, 25, 01 2012. doi: 10.1145/3065386.

[35] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. volume 15, 01 2010.

[36] Sepp Hochreiter. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998. doi: 10.1142/S0218488598000094.

[37] Andrew L. Maas. Rectifier Nonlinearities Improve Neural Network Acoustic Models. 2013.

[38] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR*, abs/1511.07289, 2015.

[39] S. C. Douglas and J. Yu. Why RELU Units Sometimes Die: Analysis of Single-Unit Error Backpropagation in Neural Networks. In *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pages 864–868, Oct 2018. doi: 10.1109/ACSSC.2018.8645556.

[40] François Chollet et al. Keras. `https://keras.io`, 2015.

[41] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker,

Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[42] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003. doi: 10.1017/CBO9780511801181.

[43] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.

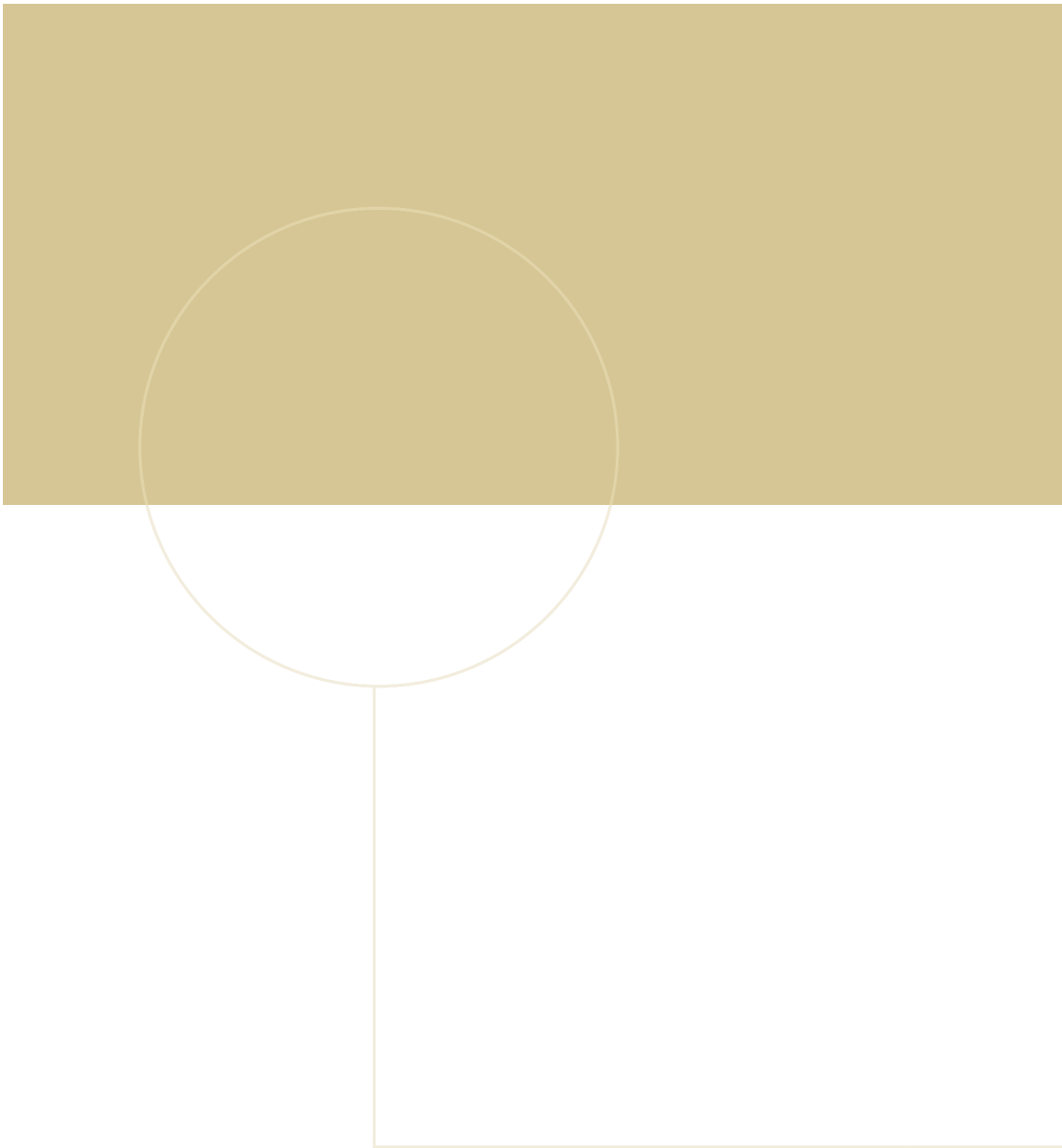
[44] Shane Torbert. *Applied Computer Science*. Springer Publishing Company, Incorporated, 2nd edition, 2016. ISBN 3319308645.

[45] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[46] Henry J Kelley. Gradient Theory of Optimal Flight Paths. *Ars Journal*, 30(10): 947–954, 1960.

[47] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[48] Sven Kosub. A Note on the Triangle Inequality for the Jaccard Distance. *Pattern Recognition Letters*, 120:36 – 38, 2019. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2018.12.007.



NTNU

Norwegian University of
Science and Technology