A Machine Learning Approach To Improve Consistency In User-Driven Medical Image Analysis

Edvarda Regine Winlund Eriksen

Thesis submitted for the degree of
Master in Programming and Networks
60 credits

Department of Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2019

A Machine Learning Approach To Improve Consistency In User-Driven Medical Image Analysis

Edvarda Regine Winlund Eriksen

Abstract

Medical imaging is an increasingly important core component of the clinical cardiovascular analysis, supporting and sometimes leading the clinical decision-making process with respect to a patients diagnosis and follow-up. In order to diagnose patients with the the help of medical imaging, trained experts have to analyse and contour the medical images. The analysis of medical images requires a large degree of agreement across the world, and standardisation of procedures to interpret the images and the measures that are obtained from them. This standardisation covers all aspects of the clinical workflow, and in recent times, it has been expanded to include the training of image analysts.

The work we present in this thesis stems out from the need of standardisation of training image analysts. We will particularly focus on the training of image analysis using a growingly popular medical imaging technique, T1 mapping MRI. Its usefulness relies on the ability to detect abnormalities in the cardiac tissue myocardial structure due to a range of pathologies in a non invasive and mostly contrast-agent free manner. T1 mapping is not yet a routinely used clinical imaging modality, but as more evidence of its potential is published, it is foreseen by experts to soon become a fundamental clinical tool.

We present the work carried out to produce the software Fabulinus. It supports the training of image analysts through automatically generated feedback, aimed at maximising the agreement in training of analysts and fostering repeated verification of such agreement over time. In Fabulinus, we deploy a deep convolutional neural networks as a core component to automatically generate an "expert" contour. The generated contour is used to enable quick and consistent feedback to the trainees using the system. The work presented has shown strong, promising results in regards to assisting the standardisation of the training process. It is also highly interdisciplinary in its nature and presents methodological and technical novelties in the software prototype design. It includes features which we believe have great potential to increase awareness of the importance of image analysis standardisation. In addition, the training process as a whole can be improved by a more consistent flow of feedback. Lastly, its strengthening the increasingly important role machine learning has in the medical field.

Acknowledgements

What a long strange trip it has been. First and foremost, I want to thank my irreplaceable supervisors, Valentina Carapella, Michael Riegler and Pål Halvorsen. I cannot thank you enough for pushing me to be the best version of myself, and for encouraging and enabling me to do such an exciting project. I wish you all the best in your professional careers, and hope we get the opportunity to work together again. Also, for any aspiring master student that might have ended up reading my thesis, I can unequivocally recommend all of them as supervisors for any thesis. They truly walk the extra mile for their students.
Additionally, I want to thank my informal supervisor Steven Hicks, a PhD student at Simula. The work conducted in this thesis would not have been the same without your patience, hard work and feedback.

Secondly, I want to thank the Corelab at the Oxford Centre for Clinical Magnetic Resonance Research at the University of Oxford. Thank you so much for your cooperation, for having trust and confidence in this project, and for giving me invaluable feedback along the way. Also, I want to thank you for fulfilling my lifelong dream of being able to work with one of the greatest academic institutions in the world. I will hold it as a token of pride for the rest of my days.

Additionally, I want to thank Simula Research Laboratory for letting me write my thesis as part of your organisation, and for providing such amazing conditions for the students affiliated with you. At the same token, I want to thank everyone at Simula who made the entire master project so much more enjoyable. Thank you Rune, Nicolay, Sharu, Mathias, Asad, TS, Marius, Pia, Vajira and Debesh, for making the everyday life of the thesis a joy. Here is to collecting more pant to buy more PS4 games, and to *Simulas Allmektige* staying connected in the future.

Not to forget, I want to thank all my (other) friends for keeping me somewhat social and sane during this thesis work. Especially, I want to thank Duy and Anmer, who always listened patiently, took part in several late night discussions, and perhaps most importantly, for believing in me when I could not believe in myself. I firmly believe I would have succumbed to the pressure without you.

And of course, I want to thank my family for your unconditional love and support throughout my life. I also want to thank you all, however especially my Dad, for giving me such a privileged life, and making it possible for me to pursue higher education. Particularly, I want to thank my late grandfather, Endre Lund, who passed away during this thesis work. Your unshakeable positivity and reassurance still lingers in me and keeps me moving forward, however I wish you could have been here to see how far it went.

And lastly, just to be safe, I in general want to thank *everyone* who someway, somehow pushed me in the direction that got me to this point. As Illidan Stormrage, one of my favourite video game characters, put it; Now I am prepared.

Contents

List of Figures

List of Tables

List of Code Excerpts

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation and Background

The World Health Organisation (WHO) estimated that around 31% of all global deaths in 2016 were due to cardiovascular diseases (CVDs), naming it the worlds number one killer [1]. Therefore, it is crucial and urgent to improve and accelerate all parts of clinical cardiovascular diagnosis, with the goal of widespread and early diagnosis of cardiovascular abnormalities across the population. It is also a goal to make all diagnostic methods as little invasive as possible. Medical imaging has a key role in this framework, because, in order to uncover CVDs, medical practice makes heavy use of imaging scans to quantify cardiac function through segmentation.

There are many medical imaging modalities, such as magnetic resonance imaging (MRI), computed tomography (CT), ultrasound (UT) and late gadolinium enhancement (LGE). LGE is the current standard for fibrosis detection, however the diagnosis depends on gadolinium-based contrast which comes at a risk. This is because patients suffering from CVDs often have other diseases, such as severe renal dysfunction, that impair other organs. These dysfunctions can lead to potentially lethal adverse reactions to gadolinium-based contrast agent. [2]

It follows, that (native) T1 mapping in cardiovascular MRI has become a rapidly emerging modality which is increasingly popular as an alternative to LGE, due to it being contrast agent-free and considered low-risk. However, post-processing of T1 mapping images is a complex multi-step process, affected by a range of sources of variability that directly affect the outcome of the predicted T1 value. This is because the estimation of T1 values is based on the delineation, or segmentation, of regions of interest (ROIs), and so far, most segmentations of T1 maps have been manual or semi-automatic.

There are efforts in the field of deep learning directed towards making the segmentation step fully automatic for such imaging scans. They aim to develop accurate, automatic segmentation methods, in order to improve consistency and reproducibility in medical image analysis. Meanwhile, assessing the accuracy of the automatic segmentation methods already created can be difficult given the lack of a gold standard or ground truth to how the segmentation should be done [3], [4]. However, this does not mean that there is no consensus in the field. There are several guidelines [5] to ensure as accurate and similar segmentation as possible worldwide. This consensus is the base of all teaching and clinical practice.

On the contrary, when it comes to the training of medical analysts trainees, training is carried out at the level of a single centre or medical unit, with little agreement across centres. To put it differently, this means that there is some form of consistency across people undergoing training by the same expert; meanwhile, there is little agreement on the training process among experts. Furthermore, the training process can also be a tedious and repetitive process where the instructor may have to sit with each trainee individually, repeating steps and correcting errors as they go along. Due to this, the standardisation and digitisation of the training process becomes an equally important issue as the automatic segmentation.

1.2 Problem Statement

Only recently, more organised efforts have been made to improve the quality and standardisation of the training process, and training standardisation protocols are being discussed and formalised. As part of this process, the field welcomes new ideas and tools to help build this standard. Finally, this brings us to the aim of this thesis; we aim to improve consistency and reproducibility of the training process across medical centres and -units, and we hope to answer the following research questions by the end of this thesis:

1. ***How can we design a tool to support training in T1 mapping analysis based on a digitised approach?***
2. ***Can Convolutional Neural Networks (CNNs) create automatic feedback based on non-expert annotations?***

We will explore if a digitised, automatic feedback software can be used to support the expert in training process, by making a software tool that can provide the trainees with instant feedback where they can analyse and interpret the differences between their own and expert contours, as well as investigating deep learning models as another tool to supply automatic feedback to trainee segmentations.

To ensure what we make is useful to the field, we have been fortunate to cooperate with the Oxford Centre for Clinical Magnetic Resonance Research (OCMR) at the University of Oxford. OCMR is a clinical research unit that "aims to advance science and clinical medicine through the development, utilisation and promotion of magnetic resonance imaging (MRI) and spectroscopy (MRS) techniques." [6] At OCMR they carry out medical image analysis daily, and they also have a training program for analysing the T1 mapping in cardiac MRIs, which has been growing alongside research carried out in the field. We have been in communication with them throughout the project, discussing use cases and tasks that needed addressing in their day to day training of trainees. They have also supplied us with a full data set to T1 mapping MRI images, that we will use for both the implementation of the software and the training of the CNN.

1.3 Scope and Limitations

Based on our research questions, the scope of this thesis is to make a prototype of a system that can be used as assistance in the aim of training a trainee image analyst. Here, it is important to emphasise that web-based systems are fairly uncommon in the medical field, and it follows that this system is experimentative. As mentioned in our motivation, we will mainly carry out research on how a web-based system can be designed in order to assist the standardisation of the training process. As part of said system, we will also investigate several pre-trained deep learning models in order to generate consensus contours based on the trainee contours, in order to see if this can be useful in the training process. The main use case we will address is a way for the trainee to get feedback on their progress throughout the training, by uploading their contours into our system, and get both visual and numeric feedback.

While we acknowledge that there are many useful file formats to use in our system that trainees are exposed to throughout their training, however we have limited it to only using the DICOM image format, as this is the data we have available. In addition, we will only be investigating deep convolutional neural networks, even though other models such as recurrent neural networks (RNNs) have proven to be successful as discussed in Section 2.7.6. This is because CNNs are the currently most popular models applied on image analysis problems.

1.4 Research Method

In 1989, the Association for Computing Machinery (ACM) Task Force on the Core of Computer Science published a report [7] presenting, amongst several other things, a new guideline for how to approach the *discipline of computing*, a phrase used in the report to embrace all of computer science and -engineering. Here, they also discuss and disprove the misleading notion that "computer science equals programming [7, p. 11]", making a point that this notion is not only false but also "limits our ability to speak about the discipline [of computing] in terms that reveal its full breadth and richness [7, p. 9]". In contrast, they subsequently present a new and richer definition, placing computer science at the intersection of applied mathematics, science and engineering, where all the processes are equally important. In its essence, the definition comes down to the underlying question of "what can be (efficiently) automated? [7, p. 12]"

In an interdisciplinary project like ours it is crucial to correct the mislead notion and to provide an accurate definition. This is not only because the misleading notion still lingers today - especially in other fields, but also because not doing so could impair the value of the research contribution from the computer science component of this thesis.

For the research carried out in this thesis, we will follow the three major paradigms described in the report; (i) theory, (ii) abstraction and (iii) design [7, pp. 10–11]. While it is important to realise that these paradigms are very closely connected, in that instances of any of the paradigms can occur at every stage of the other two (for example theory can be a part of every stage of both abstraction and design), they are all also distinct based on their area of competence. These areas are as follows, accompanied by how our thesis relates to each process:

- **Theory:** The theory process is the basis of applied mathematics within computer science, and the process iterates over the four steps of (i) characterising the objects of a study by defining them, (ii) hypothesising the possible relationships found between the objects into a theorem, (iii) asserting the correctness of the relationships, and (iv) interpreting the results, and restarting the process if flaws or inconsistencies are found.

  For the theoretical part, we applied several different deep learning methods in order to find possible relationships between the neural network hyperparameters and the models predicted output. After each iteration, we assessed how well the hyperparameters performed and by analysing the correctness of the ouput as compared to the ground truths.

- **Abstraction**[1]**:** The abstraction process is the basis for the (natural) sciences within computer science and is grounded in the experimental research method. This process revolves around investigation, through the four stages of (i) establishing a hypothesis, (ii) creating a model and predicting it's outcome based on what is made, (iii) d esigning and carrying out experiments while collecting data, and (iv) analysing the collected data.

  We followed this paradigm by running several experiments for many different models, hypothesising whether neural networks can carry out the goals of our project. We attempted to predict how the different models would do, and were continuously observing how the different models compared to each other. As a final step we analysed why the models achieved different levels of success.

[1]"Modelling" and "experimentation" was discussed in the report [7, p. 10] as possible substitutes for the name of this paradigm

- **Design:** The design process is the basis of engineering within computer science, and the processes is carried out by (i) stating the requirements for the construction of a system, (ii) stating the specifications for the construction of a system, (iii) designing and implementing the system in order to solve a problem, and (iv) testing and evaluating the system.

  This process has been the biggest one in the thesis as all of the theories have been implemented into both the software- and the machine learning system, but especially the software. Here, it is worth to mention that step (iii) and (iv) was carried out in a cycle, where we would continuously test and implement new additions to the system in order to solve problems.

1.5 Main Contributions

As part of this thesis work, we have built the web-application *Fabulinus*, which is a tool meant to assist experts in the training process of trainee image analysts. Our tool can both visually and numerically display discrepancies between the trainees analysis, and the expert analysis. As part of the work with this web-application, we have also built a parser for MRI DICOM images and their contours, that translates them from their original format to a web-usable format, where the contour is displayed ontop of the DICOM image. In addition, we have developed the functionality of overlaying contours from two separate analysts on the same DICOM image, in order to visually show the differences in the contours in our web-application.

To compliment our tool, we have made a deep learning architecture that aims to translate student contours and their respective DICOM image into expert annotations, by training the network on already existing expert annotations. The trained model derived from this architecture can be used to provide feedback to new student annotations without needing a lot of expert annotations to be made. While the trained model presented in this thesis is not at the stage yet of producing expert segmentations that can be used in a training setting, the results are very promising.

Finally, to demonstrate the work carried out in this thesis, we submitted a research paper for publication at the International Conference on Content-Based Multimedia Indexing (CBMI) 2019, that will take place in Dublin in September. The date for notification of acceptance regrettably falls after the deadline of this thesis, however the paper can be found in its entirety in Appendix C.

1.6 Thesis Outline

This thesis is separated into four parts: I Introduction, II Method, III Experiments and IV Conclusion and Future Works. Within these parts there are 7 chapters, where the first and second chapter cover the introduction and background knowledge needed to understand the topics of this thesis. The third and fourth chapter cover the methodology, that is, the different steps of our work and how we developed the system as we went along. The fifth chapter covers the experiments carried out during the machine learning training, and the sixth and seventh chapter cover the conclusion for our thesis, as well as suggestions for future work.

Below we present Figure 1.1 displaying a road map of the suggested reading order. All chapters at the same level can be read in an arbitrary order, meanwhile as the reader proceeds downwards in the map, all chapters adjacent or above should be read in order to fully comprehend the material. The figure is followed by a short summary of each chapter.

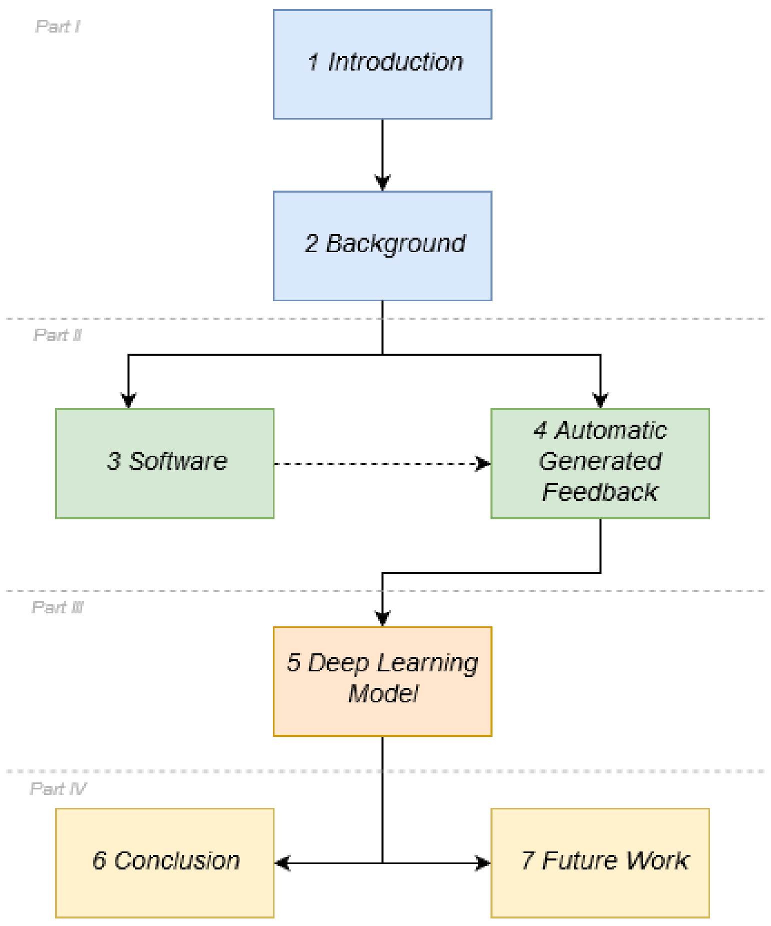


Figure 1.1: Suggested reading order for the chapters in the thesis

Chapter 2: Background In this chapter we go over all the topics needed as background knowledge for this thesis in greater detail, both with respect to the medical side and the technical side. This includes the medical background of cardiac anatomy, the image analysis pipeline and the key modalities used to carry out image analysis. In addition, we explain what machine learning, neural networks, and deep learning is, in addition to several key concepts required to understand how machine learning algorithms learn.

Chapter 3: Software Name Here, we present our web application, describing the process of how we ended up at the software. Then, we cover the technical aspects, including the libraries, frameworks and tools used to make the software. Finally, we present the system, accompanied with screenshots, and proposed usecases where the system could produce value.

Chapter 4: Automatic Generated Feedback Using CNN We cover the methodology used for the neural network. We introduce the dataset used in training, the deep learning environment we set up all our experiments in, as well as the decisions made along the way for choice of deep learning models, alterations to hyperparameters and activation features and a brief teaser for the results we had.

Chapter 5: Deep Learning Model Here, we sequentially go through each experiment carried out during the machine learning. First, we give a brief description of the metrics and evaluation methods that are used for our results. Moreover, we discuss why the different metrics and evaluation methods were chosen. Finally, we present the configuration for each trained model, as well was the results both with respects to metrics, and where applicable also visually.

Chapter 6: Summary and Conclusion We answer the research questions and conclude the work carried out in this thesis. We draw comparisons to similar systems, and summarise.

Section 6.1: Future Work Finally, we offer suggestions as to how the work carried out in this thesis can be continued in the future. We both point towards factors in the existing system that can be improved, and new functionality and approaches that we believe will be useful.

Chapter 2

Background

This chapter will cover the background knowledge needed for the reader. As this is an interdisciplinary project, the knowledge needed includes the fundamental understanding of cardiac anatomy, segmentation of different anatomical features in a magnetic resonance image (MRI), machine learning algorithms and how they are applied.

There will be an assumption that the reader has a basic understanding of both computer science and calculus, in adherence to the requirements for applying to a master degree at the Department of Informatics at the University of Oslo.

2.1 Physiology and Anatomy of the Heart

2.1.1 Cardiac Anatomy

Humans have a four-chambered heart, consisting of the (upper) left and right atrium, and the (lower) left and right ventricle. It is common for observers to refer to these chambers by their abbreviations; LA, RA, LV, and RV, respectively. The atria are the chambers of the heart that receive the blood from the veins carrying blood into the heart, and then pump the blood into the ventricles; the two large chambers of the heart that pumps blood into the arteries, that consequently carry blood away from the heart.[8], [9]

The heart wall surrounds the chambers and consists of three layers: epicardium, myocardium, and endocardium. The epicardium is the out-most layer of the heart that protects the heart and lubricates it, to prevent friction between the heart and the surrounding organs. Below is the thickest, muscular layer of the heart wall: the myocardium, that contains the cardiac muscle tissue. Myocardium, being the muscular layer, functions as the pump that pumps blood out to the arteries. Thus, it also makes up for the majority of the mass of the heart wall and is also the thickest of the three layers. Lastly, the endocardium is a thin layer between the myocar-

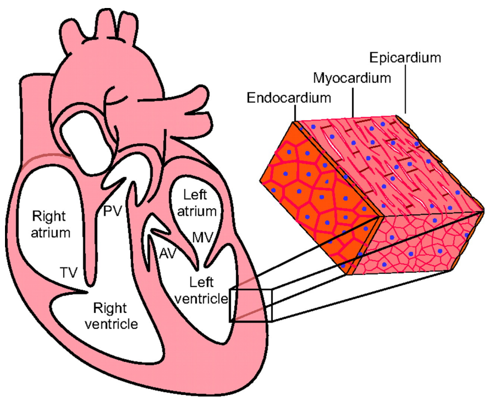


Figure 2.1: The structure of a mammalian heart. [10]

dium and the inside of the chamber, which prevents blood from sticking to the heart wall and potentially forming blood clots. [9]

2.1.2 Cardiac Cycle

During a heartbeat, the heart and its chambers move between two states - systole and diastole. Systole means that the heart is contracting to push blood out of the chambers, and on the contrary, diastole is when the cardiac muscles relax to allow the chambers to fill with blood. More formally this is called the cardiac cycle. At the end of each state, we get the *end systole* (ES) and *end diastole* (ED).

As the names suggest, end systole is when the chamber space is fully contracted and contains the least amount of blood, and end diastole is when the chamber space is fully dilated and contains the most amount of blood. It is important to mention that the atrium and the ventricles alternate between systole and diastole, that is, when the atrium is entering end systole, the ventricles are entering end diastole.

While the RV supplies the lungs with blood, the LV supplies the rest of the body, so the LV is what cardiologists focus on for the quantification of the cardiac function during the cardiac cycle. Thus we will only be talking about LV from here. To be able to quantify the cardiac function based on LV, one needs to derive segmentations from medical images, generated by different medical imaging techniques, such as Magnetic Resonance Imaging (MRI).

2.2 Magnetic Resonance Imaging (MRI)

Magnetic Resonance Imaging (MRI) is an imaging modality, that is, a medical imaging technique. MRI uses strong magnetic fields, radio waves, and a computer to generate detailed black and white images of virtually any internal body structure [11], [12]. As MRI imaging only exposes the patient to radiation in the radio frequency range[1], which does not damage tissue [11], it follows that it is considered less harmful to the patient than other imaging modalities.

Other significant benefits of an MRI scan is its ability to examine the cardiac morphology, function, perfusion, and viability in a single imaging session, as well as its ability to extract three-dimensional reconstructions of anatomical shapes of clinical measures. These benefits have led to MRI becoming an essential reference examination [13], [14], and with this in mind, we will focus on cardiac MRI examinations in this thesis.

MRI scans can be acquired using different *sequences*, in which the resulting images will have different information content, using modalities such as cine-MRI, late gadolinium enhancement (LGE) and T1 mapping. In particular, for the aim of this project, we employ data T1 mapping MRI. T1 mapping MRI is acquired at a single time in the cardiac cycle, usually either end diastole or end systole, and it captures the cardiac tissue structure health. It is, as a result, helpful to identify local or diffuse disruption of cardiac tissue, for example, abnormal growth of connective tissue in the myocardium, such as fibrosis or tissue scars, or the presence of inflammation. We will discuss this modality in closer detail in Section 2.3.1.

2.2.1 Cardiac Imaging Planes

The three standard imaging planes for cardiac MRI include the short axis view, the horizontal long axis view (also known as the four-chamber view) and the vertical long axis view (also known as the two-chamber view). These views or planes are arranged based on a line spanning from the cardiac apex, at the very bottom of the heart, to the centre of the mitral valve, the valve between LA and LV. This line is also known as the true long axis of the heart. [15] The short axis view, shown in Figure 2.2a is the plane most

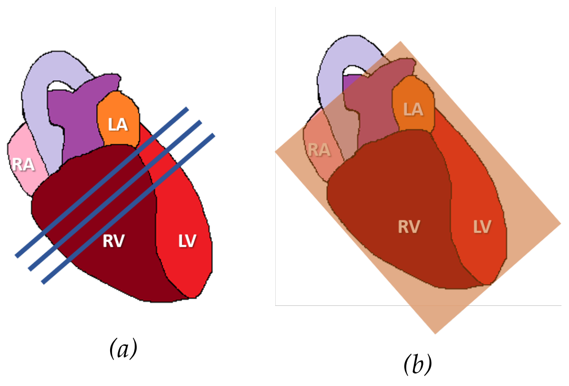


Figure 2.2: Figure displaying (a) the short-axis view-, (b) the horizontal long-axis view-, and (c) the vertical long axis view of the heart

[1]Radio frequency is the frequency used for phone communication and broadcasting, for example, and thus we are exposed to it daily.

commonly used to assess cardiac function [13], and extends perpendicular to the true long axis, about midway into LV. Evidently, from their chamber names, the horizontal long axis view, shown in Figure 2.2b shows a slice of all four chambers of the heart (LA, RA, LV, RV), and is perpendicular to the short axis again. Meanwhile, the vertical long axis, shown in Figure 2.2c, goes through the heart from top to bottom, first showing LA and RA, the showing LV and RV, and is the vertical plane positioned orthogonally on the short axis plane. [15]

In the short axis view, a stack of images is acquired moving through the heart, while in the long axis a single, stationary slice is followed through time. These stacks provide MRI with its 3D nature. However, it is important to realise that MRI is not truly 3D; observers typically construct the 3D effect in a later step of the software pipeline.

2.3 Clinical Imaging Workflow

When carrying out a medical examination depending on clinical images, all facilities follow a clinical imaging workflow. While these workflows can vary, the general concept is that a patient is typically scanned for a prolonged amount of time, according to different scanning protocols. Although the reader does not need to understand these scanning protocols as part of this thesis, there is good literature going through examples of scanning protcols. [16] These protocols depend on the type of pathology that the patient is being imaged for, however, even the simplest scanning protocol will have a range of imaging planes (both in short axis and long axis), and make use of different types of imaging modalities, for example *cine-MRI* and *T1 mapping*. As adapted from Ferreira *et al.* [17], a good visualisation of different imaging modalities taken for a single patient during one imaging session can be seen in Figure 2.3. While the

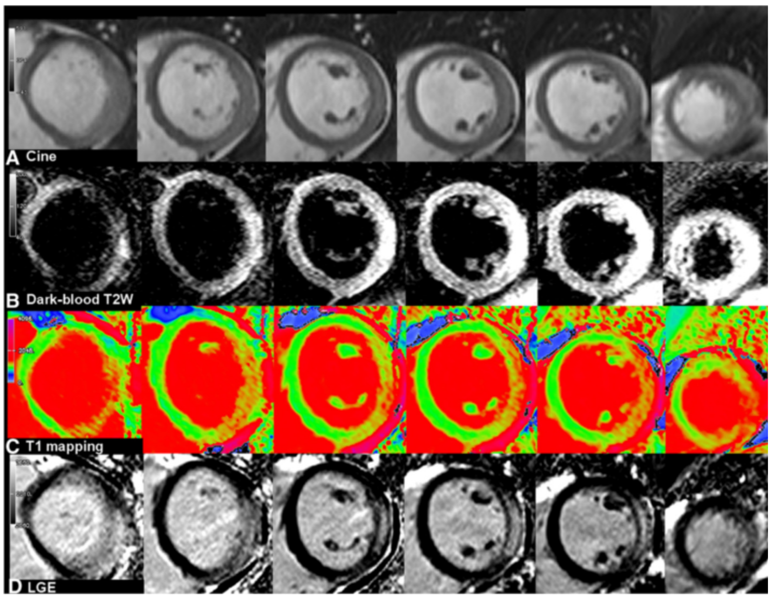


Figure 2.3: Figure showing an example of "cardiovascular magnetic resonance (CMR) tissue characterisation in acute myocarditis. (Left to right) Short-axis slices covering the left ventricle from base to apex" - see row A and C for Cine and T1 mapping respectively. Source: Adapted from [17]

different imaging modalities have different benefits when assessing the cardiac function, we will only focus on T1 mapping from here on in this thesis, as this is the data we have available.

2.3.1 T1 Mapping

In this section we will cover what T1 mapping is, and why it is used in clinical imaging. First, we will introduce an elementary explanation of what T1 mapping is as well as why we look at it. Then, we will briefly cover the base line of the physics behind the T1 mapping. However, it is important to realise that T1 mapping is a very technical and complex modality. A complete understanding of T1 mapping, let alone the physics behind it, far exceeds both the needed understanding for the reader and the scope of this thesis. The focus should be to grasp *why* we look at the T1 value.

The T1 value, or T1 relaxation time, is in its essence, is a way to quantify the proton density and freedom of motion in tissue, as the T1 value will vary over different types of tissue. The values intensity are then mapped across each pixel in an MRI image, a step referred to as *T1 weighting*. [18] To illustrate, take the example of row C in Figure 2.3. Here, the different colours depict different types of tissue, meaning that (a) red is blood (b) green is muscle tissue and (c) blue is fat. Observe how the bright green muscle tissue of the green circle, being the heart wall, starts to disperse on the right side of the circle as the slices go from left to right. This is a classic example of damaged or scarred tissue, as healthy tissue would remain bright green in the entire circle throughout.

Following this, T1 mapping can make it easier to detect pathologies based on tissue, and is especially an important measurement to successfully identify, for example, edema or inflammation in the heart, where there is an excess of water content, cardiac scar tissue, as scar tissue is typically denser than healthy tissue ot mycardial infarction.

2.3.2 Physics behind MRI and T1 Mapping

This section aims to cover the fundamental understanding of the physics behind MRI and T1 mapping. As a disclaimer, this level of comprehension is not needed for most readers.

MRI relies on the physical phenomenon of nuclear magnetic resonance, that is, the characteristic of some atoms to align with the direction of an applied external magnetic field. This is due to the fact that these atoms nuclei have a non-zero magnetic moment, that they have the property of spin. When an external magnetic field $\mathbf{B_0}$ is applied, some atoms will align parallel to it, while other will have an antiparallel orientation, as shown in the Figure 2.4. Typically, the group of parallel spins is larger than the antiparallel one, producing a net magnetisation $\mathbf{M}$ in the material.

When an electromagnetic pulse (that is a short time signal) is applied to such system, it causes all the nuclei spins to change at once their position becoming misaligned with respect to the external magnetic field by a certain angle, which varies with the strength and duration of the pulse applied. Upon termination of the pulse, the spins will slowly recover their (parallel or antiparallel) position by emitting energy. The magnetic resonance (MR) signal is produced by the small difference in energy between parallel and antiparallel spin groups as they return to their original state. Over time the MR signal decays due to the release of energy and the loss of coherent orientation among the spins, a phenomenon called relaxation [19]. Under the same magnetic field and pulse conditions, different biolo-

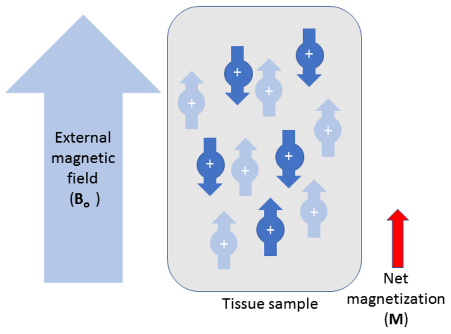


Figure 2.4: Figure visualising the basics of the physics behind MRI

gical tissues will show different relaxation times, a property that depends on the tissue structure and water content.

There are many techniques that exploit the magnetic resonance properties of atoms. In the particular case of MRI, the atom of interest is hydrogen, which is present in abundance in biological tissue water content, and as part of other constitutive chemical compounds. The temporal and spatial information about the different relaxation times of the hydrogen atoms contained in different forms in a tissue sample is hence transformed into an 2D image.

Upon relaxation the net magnetization vector M slowly returns to align with the external magnetic field B_0. The vector M is typically decomposed in its longitudinal and transverse components, Mz and Mxy, respectively. Mz, the component of M along the direction of the external field, is the longitudinal component, and T1 is a measure of relaxation time along this direction. The relaxation time in the transverse orientation is measured instead by means of T2. Magnetic Resonance Imaging can exploit either T1 or T2 to produce 2D images. Their information content is different so is their purpose in terms of differentiation of tissues types. In the following section we focus on imaging of T1 relaxation times.

2.4 Medical Image File Formats

In order to store and process an MRI, there are some different standardised imaging file formats that can be used, that bring different benefits and drawbacks. In this section we will cover the general concepts that are shared by all imaging formats, then we will present some of the image file formats that are typically used in the medical field and how they relate to each other, and finally we will go into detail on the Digital Imaging and Communications in Medicine (DICOM) [20] standard, which is the image format used in this thesis.

2.4.1 General Concepts

Recall from earlier in this chapter, that for an MRI as it also is for any medical image modality, the aim is to reconstruct and represent virtually any internal body structure or anatomic function. The image that is produced is a result of the procedure that maps numerical values to positions in space, and also sometimes time, into an array of pixels or *voxels*, the term used for graphical units in the three-dimensional space created by magnetisation transfer (MT) or computed tomography (CT). A point often overlooked by those who are not in the medical field is that the numerical values stored per pixel, are not necessarily values in the normal colour range as we often expect based on normal JPG/JPEG/PNG images, but rather these values greatly depend on the imaging modality and the acquisition protocol being used when taking the image, as well as how the image is reconstructed after the fact and lastly, the post-processing at the end of the pipeline. [21] The concept of the numerical values store in an image are called the *pixel depth* and the *photometric interpretation*.

Pixel Depth is the number of bits used to store the information in a pixel, or in the case of images the number of bits used for a given colour in a pixel, for example, a greyscale image with a pixel depth of 8-bit, would store the grey values as numbers ranging from 0 to 256, whereas a greyscale image with a pixel depth of 12-bit would store the grey values as numbers ranging from 0 to 4095. Thus, this concept declares that the more information, or the amount of tint and detail, we want to store in an image per pixel, the more computer storage we will need for it.

Photometric Interpretation is the deciding concept of how the pixel data should be displayed in regards to whether it is a monochrome or colour image. In this concept we talk about the *number of channels* needed to display greyscale or colour images, in that greyscale images does not need a third channel for colour values, see Table 2.1. As a result, greyscale images has a smaller pixel depth and requires less storage space.

Image Type	Coordinates	Example
2D grayscale	(width, height)	(64, 64)
2D multichannel	(width, height, colour_channel)	(64, 64, 3)

Table 2.1: Examples of greyscale and coloured (multichannel) image shapes

When talking about photometric interpretation, we also want to introduce the term *pseudo-colour*, a variant of *false colour*. A pseudo-coloured image is an image where the colours does not represent the true colours that an eye or a camera can see, but rather the intensity or degree of features that cannot normally be seen. A common example of a pseudo-coloured image is thermal imaging, where the colour intensity conveys the temperature differences of the image. In order to make pseudo-coloured images, you use the ascribed look up table (such as a colour palette) or function assigned to the greyscale image, that decides how to map the intensity of the values in the greyscale image into colours in the pseudo-colour image.

MRI images are another example of pseudo-coloured images. That is to say, they are interpreted as greyscale, however they also have a colourmap that enables the images to be colour-coded based on the T1 value, in such a way that low values depict fat and turn out blue, medium range values depict the myocardium and turn out green, and high values depict blood and turns out red, as displayed in Figure 2.3

Metadata is the final concept we will cover, and is all information that is stored in describing the file beyond its pixel data. In the case of medical images, the images have more metadata, such as information on how the image was taken and the settings for the capturing device, personal information about the patient such as age, sex and weight, and specifically for MRI images we typically have parameters concerning the pulse sequence. [21]

2.4.2 Medical Image File Formats: DICOM and Nifti

What makes medical image file formats different from regular image formats is that they typically need to store a lot more information, such as information about the image scan device and pathology, patient information and several image slices over space or time. This information is kept in both the pixel values, and in the meta data of the image (either stored at the beginning of the file or as a separate meta data file). Generally speaking, all medical image formats can be separated into two main categories; (a) the image formats aiming to standardise the way images are generated by medical modalities, and (b) formats aiming to improve the post processing analysis. [21] There are a few different formats in each category, whereas we will look into DICOM, which is in the first category, and Nifti [22] , which is in the second.

While NIFTI is considered more agile, and hence perhaps better suited for experimental applications, DICOM remains the backbone format of most medical practice. This is because the DICOM format aims to be self-descriptive, in that all the metadata stored in the header should be sufficient to gain *all* knowledge needed for both the image, the imaging modality and the patient. Although, as a trade-off, the DICOM has also been criticised for its complexity, with a problematic documentation only being available in a HTML format, especially by researchers and computer scientists aiming to get a deeper understanding of the standard. [21] Regardless of this, DICOM remains the preferred medical image format, and it is likely that it will remain this way for years to come. This is the main factor as to why we will only use the DICOM image format in our work.

2.5 Segmentation of the Heart

The assessment of cardiac function based on MRI or other imaging techniques is primarily based on segmentation (or contouring). Segmentation is the process of outlining an anatomical feature of interest, on any image within the whole imaging dataset of a subject, to quantify one or more variables of interest.

2.5.1 Clinically Relevant Measurements

In particular, in the case of the evaluation of global cardiac function in cine-MRI scans, the LV provides the most common variables of interest, namely the mass and volume of the heart during end systole and end diastole, to calculate the *stroke volume* (SV) and subsequently the *ejection fraction* (EF).

SV is a measurement of the total amount of blood pumped by the heart, in our case with respects to LV, in one contraction. Mathematically this is calculated by finding the end-systolic volume ($\mathrm{ES}_{\mathrm{vol}}$) and end-diastolic volume ($\mathrm{ED}_{\mathrm{vol}}$), *volume* meaning how much blood the heart contains at ES and ED, and subtracting $\mathrm{ES}_{\mathrm{vol}}$ from $\mathrm{ED}_{\mathrm{vol}}$ [23] as follows:

$$SV = ED_{vol} - ES_{vol} \tag{2.1}$$

Which we again use to calculate EF, namely the fraction of blood pumped out by the ventricle, as follows:

$$EF = \frac{ED_{vol} - ES_{vol}}{ED_{vol}} = \frac{SV}{ED_{vol}} \tag{2.2}$$

These measurements are crucial for assessing the efficiency of the cardiac function, that again helps to diagnose cardiovascular diseases. Finding these measurements is mostly done by either manual segmentation by observers or semi-automatic segmentation done by various software tools.

In T1 mapping MRI, the measurement obtained is the T1 value, a value we described in closer detail in Section 2.3.1 and Section 2.3.2. For our approach, we will focus on T1 mapping, as well as the wall thickness (WT), as this is the data we have available. The WT value is extracted from a segmentation by measuring the average distance between endocardium and epicardium. Typically, a number of radii are drawn from the centre of the left ventricular blood pool towards the outside, and the length of each segment between the two curves is measured and averaged. Image analysts manually draw four segments at opposite quadrants of the left ventricular ring, meanwhile algorithms draw many more, in the range of a hundred approximately. An example of how can be seen in Figure 2.5 While we did not carry out this step, we were given datafiles containing the needed values per segmentation.

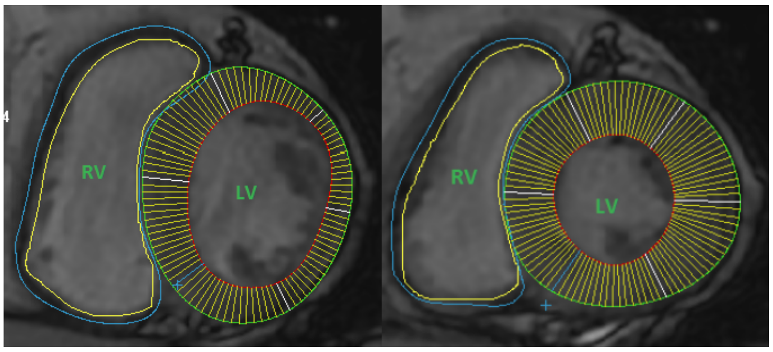


Figure 2.5: An example MRI image, displaying how different radii are measured between endocardium and epicardium in order to extract WT.
Adapted from: Source [24]

2.5.2 Challenges and Sources of Variability

It is well known that the manual segmentation of cardiac MRI is a tedious and time-consuming process. While it depends on both experience and the software tool in use, an observer can require anywhere between ten minutes up to an hour doing a single segmentation, where the estimated time per ventricle is roughly twenty minutes [13], [14].

Moreover, the manual segmentation process is prone to many sources of variability. To illustrate, consider the biological variability across the population; that is, each heart is different in shape and function. The differences are due to both the natural- and genetic variances, as well as the variances caused by health and lifestyle. Then, when a single heart is imaged through MRI and analysed multiple times by multiple observers, we get inter- and intraobserver variability [13], [14]. This variability can include under- and overestimation of the volumetric measurements of the heart - based of the positioning of the image slices in the cardiac imaging planes [14], unrealistic performance, and overfitting of the validation of the segmentation data during post-processing [3], as well as overconfidence in segmentation where the observer makes the segmentation too detailed.

The overconfidence can, however, also be caused by the level of detail provided by the segmentation software tool. If the software tool allows the observer to zoom too far in, and to make too detailed adjustments to the segmentation, this can almost encourage said overconfidence in segmentation.

Because of these variabilities, there is a lot of research and funding aiming to find and develop accurate, automatic segmentation algorithms and software tools that can do the segmentation for the observers, for instance by using machine learning. It is also important to realise that the accuracy of these automatic segmentations are critical, as any technique with questionable validity can have fatal consequences at worst.

2.6 Software for CMR Image Analysis

There are a multitude of software and annotation software already available in the field as of today, and while we are strictly not making an annotation or contouring tool, we want to bring the attention to some of the leading softwares used to analyse the images from the clinical workflow. While it is important to mention that one is typically not interested in commercial tools in research, the majority of medical practice use commerical tools in their analysis. The following tools are vastly used in the field and have been good sources of inspiration at an abstract level for the software developed in the thesis.

cvi$^{42®}$ [25] is one of the biggest Cardiac MRI softwares on the market and has a wide range of modules to analyse different modalities for cardiovascular MRI, including analysing the T1-value, the flow of the heart and so forth. It is also fully integrated with the picture archiving and communication system (PACS) that allows medical images and reports to be stored securely online.

QMass [26] is another leading cardiac MRI software, that supports the analysis of quantification of the cardiac function, anatomy and tissue segmentation, and MR parametric maps such as T1 mapping and T2*, to mention some. It is intended to be used in clinical practice by medical experts.

The main goal of these softwares is to provide an extensive array of functionalities to analyse cine, T1, LGE and so on. While there are free alternative software, such as ImageJ [27], these typically does not come close in the level of complexity and amount of functionalities. In any case, these softwares have served as good inspiration for our software, especially with regards to design ideas. In addition, they aided our understanding of what functionalities are already available, especially as we originally set out to make a variant of a segmentation tool. What they all are missing however, is ways to give instant feedback to trainees with respects to some expert analysis, which is the ultimate goal of our software.

2.7 Machine Learning

Machine learning (ML) is a subcategory of artificial intelligence (AI) and is the science of allowing computer systems to learn without being explicitly programmed [28], [29]. Instead, the computer systems use algorithms that provide a representation of tasks, as well as methods producing performance evaluation of the learning, and how to optimise the performance in the next iteration, allowing the computer system to improve its learning autonomously over time [28].

In other words, Tom M. Mitchell provides a commonly used and more formal definition; "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."[30, p. 2]. This definition emphasises the foundation of how computer systems learn through machine learning, namely that they learn from experience by repeating a task several times while improving its performance.

2.7.1 Learning Methods

In the field of machine learning, we typically speak about two main problems or categories, namely supervised- and unsupervised learning. Simply put, supervised learning is learning with a teacher or reference values, namely that we provide the algorithm with "correct answers" that it can improve its accuracy from. These correct answers are more formally called labelled data. Typical supervised learning applications are image analysis, where we teach the algorithm to classify what it sees in different images, or spam-filters in emails.

Evidently, unsupervised learning is the opposite, where the algorithms learn without any reference values. Thus, as there is no ground truth, there is no sense of accuracy, as compared to the supervised approach. It follows that unsupervised learning applications are used in a more speculatively. Classic examples are recommendation systems, such as Spotify trying to recommend music to its users based on what they listen to, or search engines such as Google.

While there are several other learning categories, such as reinforcement- and representation learning, these are often talked about as subcategories or in comparison to supervised- and unsupervised learning.

2.7.2 Artificial Neural Networks (ANN)

Artificial neural networks (ANN), often also just referred to as neural networks, are computational machine learning models. The *neural* part of the term neural network is because these models are loosely inspired by neuroscience and the model of a brain. The inspiration stems from our understanding of how a brain works, and in particular how a brain learns; namely by example. While it is important to stress that the complexity of the brain and a biological neuron far exceeds the examples we are about to give, the essence between the concepts remain relevant. The computational unit of a brain is called a *(biological) neuron*. Put simply, each neuron receives impulses through their dendrites from many other neurons, and then it produces a new output signal and sends this signal through an axon to other, different neurons. [31] Similarly, as seen in Figure 2.6, a regular neural network consists of an *input layer*, one or more *hidden layers*, and finally an *output layer*. Each layer consists of one or more *(artificial) neurons*, also called units or perceptrons. The lines that connect the neurons are called *weights*, which are trainable values the represents the magnitude of how much influence the first neuron has on the next. In addition, you can

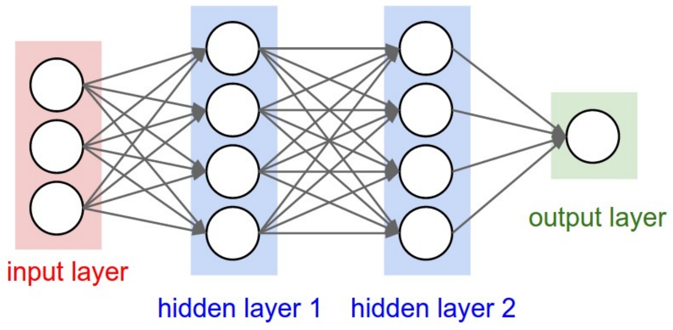


Figure 2.6: A neural network consisting of three layers, with three inputs, two hidden layers of four neurons each, and one output layer
Adapted from: Source [31]

have some *bias* for a node, making it either more less likely to end up as the predicted output Y.

$$Y = \sum(weight * input) + bias \tag{2.3}$$

More specficially, the input layer neurons are the input data of the model, and typically consist of examples of the data to be trained on, the hidden layer neurons each have an activation function a and are trainable, so they also transform the input data based on the weights and bias, and the output layer neuron is the estimated value from the training, often written $\hat{y}$.

$$\hat{y} = a_k^{[l]} = g\left(\sum_{j=i}^{n^{[l-1]}} w_{jk}^{[l]} a_j^{[l-1]} + b_k^{[l]}\right) \tag{2.4}$$

In Equation 2.4, k is an arbitrary node in layer l, and j is an arbitrary node in the previous layer, $l - 1$. Seeing that, each (artificial) neuron in the hidden layers of a neural network is playing a role that is similar to that of its biological counterpart, as each neuron of the layer receives data from many other neurons and computes its own value that it sends on to the following layers. With this being said, it is important to realise that we say loosely inspired because modern neural networks no longer have the goal of modelling biological neural systems, but rather have diverged to be more driven by mathematical- and engineering disciplines, to achieve statistical generalisation and good predictions. [32] Besides, artificial neural neurons typically need need many thousand examples to learn sufficiently, as opposed to the biological ones.

The *network* part of the term neural network stems from the fact that neural networks are often represented as a combination of many functions, typically in a chain structure. Consider this example taken from Goodfellow et al. [32, p. 164]:

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))). \tag{2.5}$$

Here, each f is its own function, and the value of n in $f^{(n)}$ decides what layer each function is, so for instance, $f^{(1)}$ is the first layer of the network.

Finally, it is important to emphasise that typical neural networks are much larger than our given example, typically consisting of several hundreds and thousands of neurons and layers in one model.

2.7.3 Deep Learning

Deep learning is a specific type of machine learning, and is essentially any neural network with many hidden layers. The term *deep learning* originated from discussing the depth of a neural network by the number of its hidden layers. [32] Deep learning was initially motivated by and developed to tackle some central challenges in AI, such as speech recognition or image analysis, after some of the more traditional machine learning methods did not succeed in solving these problems. Time would show that deep learning was infact a significantly superior approach, as in recent year deep learning methods have significantly improved the state of the art in speech recognition, image analysis in regards to object detection and many other domains. [33]

As such, deep learning methods have developed into representation learning methods, where the network can be fed with raw data and automatically discover in its own what features and representations are essential for detection and classification of classes.

2.7.4 How Supervised Neural Networks Learn

Neural networks are often referred to as *black boxes*, because the internal process of exactly how the network decides on what features that are most important in order to decide the predicted output of the model, is both difficult to understand and interpret. While this is true, we typically speak of certain methods that a supervised network uses to learn and train; *loss functions* and *gradicent descent*.

As with any type of learning, the neural network needs a way to quantify how correct its prediction was. In order to do so, it uses the designated loss function, also known as the error function. This function measures the error, or the distance between the predicted value and the ground truth value. It follows, that ideally the error should be as close to 0 as possible, as that would mean the distance between the prediction and the actual class is insignificant, if not 0. Thereafter, in order to learn, the network needs a way to reduce the error.

In order to improve its prediction and reduce the error, the network will adjust the internal adjustable parameters, for example weight and bias, and it utilises gradient descent to do so. More specifically, the model computes a gradient descent vector that indicates how much the error would increase or decrease for each weight, based on a small adjustment of the weight. [33] Based on this, the model will adjust its weight in the opposite direction of the gradient descent. This can be metaphorically described as an elevated, hill-like landscape, where we want to find the fastest way down. Here, for each step, the negative gradient descent value will always point in the direction of the steepest descent, bringing us one step closer to the goal value of 0.

Backpropagation

Another therm that often comes up when training neural networks is *backpropagation*. Although it sometimes is misinterpreted as a something used to train a network, for example "we trained the network with backpropagation", this is not entirely correct. In its simplest form, backpropagation is a specific technique used to compute gradients on multilayer networks, so rather than a function used to compute the loss/error, it is used in unison with such functions as a way to compute the gradients throughout the network.

In technical terms, what backpropagation does it to start at the predicted value (the output node), compute the gradient descent for a function on the weights of a multilayer stack, all the way back to the input data. It does so by simply applying the chain rule of derivatives. This is beneficial, because it can compute partial derivatives in linear time, whereas naïve gradient computations increase exponentianlly as the depth of the network increases. As pointed out by Lecun, Bengio and Hinton [33], the key insight is that the computation happens *backwards* from the gradient with respect to the output of the model, to the gradient with respect to the input.

2.7.5 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN), also known as ConvNets, are deep neural networks that are designed to process data in the form of multiple arrays or matrices, such as images with three colour channels. This is because of its grid-like topology. As pointed out in Goodfellow *et al.* [32], the characteristcs of a CNN is any neural network where, for any layer in the network, it carries out convolution instead of general matrix multiplication.

The step of convolution, in its essence, is a special type of linear operation that combines two functions and produces a third function, that we typically refer to as a feature filter. In CNNs specifically, the combination is between the input function and the *kernel function*. The kernal, also known as the convolutional filters, is a spatial grid typically equalling the input size in width and height. As the filter is passed along in the network, the filters get "slided" across another filter, computing the dot product between the entire filter and the input at any position. This results in what is usually referred to as a feature map.

CNNs were not popular or widely used at for a long time following its discovery, due to the substantial amount of time it took to train them. However, after the remarkable success of ImageNet competition in 2012, following the efficient use of GPUs, certain deep learning methods and generation of more training examples by deforming the input data during training, its popularity escalated immensely to the point where it became the dominant approach. Since then, CNNs are considered the superior approach to nearly all recognition and detection tasks, and is the quintessential method for image analysis [33]. At certain tasks, it can even approach human accuracy.

2.7.6 Related Work

Over the years, multiple approaches for automatically segmention the left ventricle have been proposed. In 2014, Hu et al. [34] proposed a method for automatically segmenting the left ventricle using local binary fitting models and dynamic programming techniques. Overall, their method shows good results, yet they struggle to segment the overlap between intensity distributions within the cardiac regions. Abdelfadeel et al. [35] use maximally stable extremal regions to segment the left ventricle of cardiac MRI. Their model achieves a DICE metric of 0.88 on the Medical Image Computing and Computer Assisted Intervention (MICCAI) 2009 challenge database [36]. Similar to our method, Zreik et al. [37] propose a method based on deep CNN, where they try to segment the left ventricle in cardiac CT images. Their method uses a combination of three different CNN, each detecting the presence of the left ventricle in the axial, coronal, and sagittal independently. Our work, however, differs from these approaches as we try to measure the quality of a student contour by generating an expert consensus based on the student contour and MRI image using a deep CNN.

Part II

Method

Chapter 3

Fabulinus: A Web-Based Software for Quality Assessment in the Image Analysis Workflow

Following our first research question and the motivation of our thesis, with the goal of making a software tool that can be used to assist experts in the training of new image analysts, we present the system *Fabulinus*. With this application, we attempt to produce ways for trainee image analysts to get feedback throughout the training, by giving them both visual and numerical feedback automatically.

In this chapter we will cover how the software took shape throughout the thesis work. We will start by giving a brief introduction to what the software is, and its purposes and goals. Then we will cover how we went through several ideas, use cases and what concerns we should address as part of this project, touching on the scope challenges and the field of the unknown. Then we will go through the technical aspects of the system we made in the following order: Backend architecture, then the custom API that was made as part of this thesis, then the frontend architecture, and lastly a summary.

Again, we want to stress that this application is not complete and neither ready to be put into production. That is, it is still in the prototype stage where what we will present in the following sections count as a valuable starting point.

3.1 Fabulinus

Fabulinus, sharing its name with the god of education and children from Greek mythology, is a client-server web application. As demonstrated in Figure 3.1, the five main components are the client, the local, frontend server running on Node.js [38], the Django API [39], the backend local server based on Django [40], and lastly the database. Of which, we will come back to each in the respective sections.

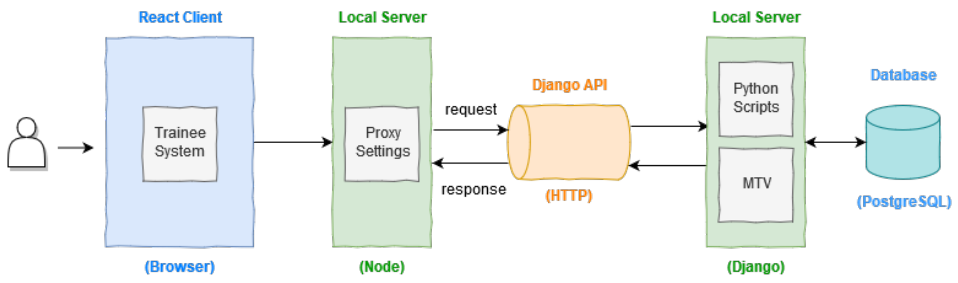


Figure 3.1: Diagram giving an overview of the Fabulinus web-application

The software was built with the following goals:

Main Problem As many clinical facilities typically only have one expert teacher to train trainees at a time, the trainees can typically do several repeats of segmentations before getting the feedback they need from a teacher. This can lead to patterns and misconceptions being repeated and internalised in the trainee.

Main Objective With the goal of addressing the Main Problem, we set out to make an application that could catch the most noticeable mistakes that the trainees could do, which directly affect the WT and T1 values. With both visual and numerical feedback from a program to the student, it would function as a supplement to the teacher more than a replacement. If the student could both visually and numerically compare their own segmentation to a segmentation done by either an expert or a deep learning model, chances are the trainee could correct their own faults as they go along in the training, before getting the proper feedback from the teacher one-on-one.

In addition to the previous goal, a software tool could also help standardise the training, by ensuring that the students got the "same" feedback initially, and albeit it being outside the scope of this thesis, it could even help gather statistics over time and show general trends in what typically goes wrong or where there typically is the most discrepancies between trainees and experts in segmentation. In return, this could help shape the future training sessions, where the teacher could emphasise the points trainees typically misunderstand.

In the following sections, we will cover the initial ideas and discussions that shaped Fabulinus into what it is today. Then, we will briefly cover the technical aspects of Fabulinus in the order of backend, API modifications, frontend and then a visual presentation of the software joined with the most central use cases. At last, we will summarise the main take away from this chapter.

3.1.1 Planning Phase and Desired User Stories

At the very beginning of the thesis work the original idea in the thesis proposal was to make a Java program, where the trainees could contour directly into the program and get instant feedback from a machine learning method as they were contouring. Java was suggested in order to utilise add-ons already created in the field. However, after airing the idea to OCMR, they emphasised that there were already too many segmentation tools available in the market, so they did not see the value in producing yet another tool. In addition, after closer discussion, we decided that the complexity of such a program would far exceed the domain knowledge and time frames of a master thesis.

Another important factor that came up during these debates, was that a program that needs to be installed on any medical systems, would have to meet very strict regulations in regards to for example security, robustness, and reliability. Naturally, in an environment that handles very sensitive, private data and that needs to be fail-secure, any component that has to be installed into said system has to go through extensive quality assessment and security checks to be accepted.

As a result, we started investigating the possibility of making it a web-based system instead. To clarify, web-based here is not meant to say that it will be an application built to be hosted online, but rather a local area network (LAN) application that would run through a browser. Given the geographical proximity of a classroom environment, where the trainees and experts will mostly be at the same clinic or centre, a LAN application would be very well suited, without having to pass the extensive installation requirements. It can simply be run at essentially any device that supports a relatively modern web browser, while the backend can hosted at a local server, without ever having to interact with the internal systems.

Following this, we wanted to build the web application on technology and frameworks that are frequently used in modern web development, to ensure that if the web application would ever see a production state, it would be easily maintainable. In addition, we decided to separate the backend and frontend logic onto two separate servers, so that they could be interchangeable with little modification if need be. We will come back to the technologies chosen in the following respective chapters.

While the technologies was being decided, we also spent a lot of time in meetings with experts from OCMR to figure out what user stories to explore first. As the time restriction of a master thesis is a inescapable factor that needs to be taking into consideration for any master, we wanted to pick the user stories that would be the most rewarding within our time frame.

My supervisor had a meeting with a few experts at OCMR and they came up with the user stories shown in Table 3.1. They also provided us with

As a Supervisor	**As a Trainee**
- I want to upload one full training session with all the contour repeats	- I should not be able to compare my contours or values to experts unless I have done at least two repeats.
- Upload or access consensus contours and T1/WT values for the same dataset	- I want to overlay experts contours on my contours, when possible
- Get report showing basic statistics of training progress	- I want to keep track of progress.
- Upload or access the current status of a team of trainees	- I want to be flagged the cases where my contours or my T1/WT values are too far off the consensus

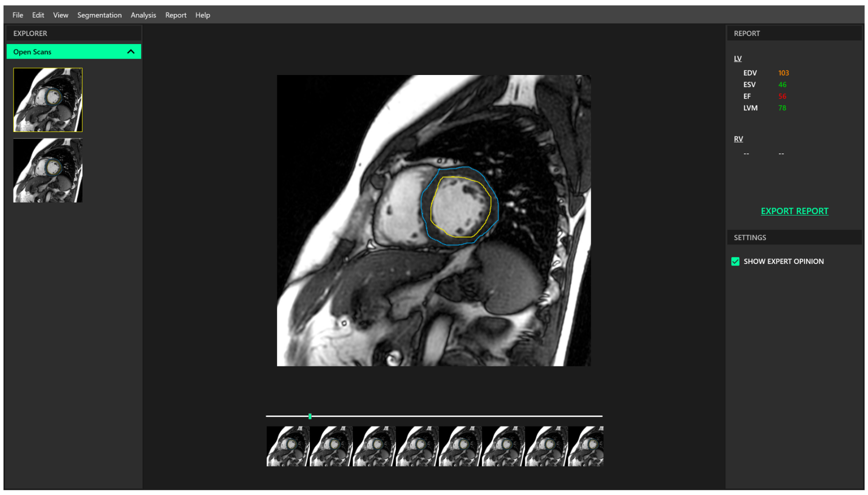
Table 3.1: Table showing initial desired user stories for the software system

some general comments concerning the requirement to system as a whole before it could be production ready, in that

- It would be good to have a different set of functionalities available for the expert and trainee, so user handling was needed
- It is important to avoid bias/cheating for the trainee, so access to the comparison of results to experts should only be made available after the trainee has uploaded at least two repeats of the same contours
- Clear definition/distinction of what type of consensus curve (or data) was available, in that it should be clearly visible what was an expert consensus (uploaded by a human expert) and the automatic consensus (that was computed by a CNN)

While all these features were tactful and useful, we decided to focus on only the trainee initially, as they are the main target of this program. The first use cases we would address was the overlaying of contours of expert and trainee, as we believe the visual difference of two graphs are indisputable from a learning perspective. We also wanted to implement and show the numerical difference in the key values WT and T1, as this teaches the gravity the relative "small" changes on a segmentation can have on the result of a medical examination.

With this established, we made began making sketches for a visual profile for the software system in Adobe XD [41], a program enabling people to make interactive web design without any code. The finalised sketch before the development started can be seen in Figure 3.2. The design is based on certain design principles within medical image analysis tools, such as the



LV

EDV	103
ESV	46
EF	56
LVM	78

RV

--	--

[✓] SHOW EXPERT OPINION

Figure 3.2: Initial design made of the software in Adobe XD

fact that the background should be a dark colour, the image should be centrally placed in the dashboard, and settings and other information should either be placed in the navigation header or in side menus. A design principle we have decided not to follow is that, in medical programs, menus and settings are typically displayed as an icon, rather than text. For example, instead of having a drop-down menu with the text "Folders", one would typically have an icon for a folder instead. We chose not to follow this standard due to the prototypic nature of the software. To summarise, we made an effort to make Fabulinus resemble a typical segmentation program like those discussed in Section 2.6.

3.1.2 Backend Architecture, Libraries and Tools

For the backend architecture, we have built a server using Django [40], which is a high-level Python web framework. It encourages and aims to provide a framework that can be used to build web software from scratch rapidly. While this aim suited our web application well, we also chose Django because we are dependent on our own Python scripts in order to process the DICOM files containing MRI images and the IDL files containing the contours. This is a process we will come back to in Section 4.1. We considered the microframework Flask as an alternative to Django, but as Django comes with many out-of-the-box features, such as an administration dashboard and built-in object relational mapping (ORM) for database abstraction and modelling. At large, Django is simply more expandable and has more features than Flask.

Although, one of the major challenges with Django is that while it comes with a lot of features, it is also fairly difficult to learn at first. You use a command-line interface (CLI) to set up Django, which in return produces the folder structure and main files needed to run the Django server. From this point, only a few lines of code are needed to have a web application running on ones browser, however as a trade off, a lot of stuff is happening behind the scenes. We were no exception to this, so getting the Django server to act as we expected it to took a lot of trial and error.

Django is based on the the Model-View-Controller (MVC) design pattern, however with their own variant. Normally, when we operate with the MVC design pattern on web applications, the *model* is typically a representational class that stores the different fields and functions related to the data, the *view* is how the data should be presented in some interface, and lastly controller is the logic of the design, where all modifications and preparations of the data is made. Meanwhile, Django operates with models, templates and views (MTV), so when a Django application refers to a Template, that translates to a View in MVC, and when it refers to a View, that translates to the Controller in MVC. While this is confusing at first, the Django creators argue that the (Django) view should decide *which* data gets presented and the business logic, meanwhile the template handles the presentational logic and decide *how* the data is presented.

As for the database, we chose to use PostgreSQL [42], which is a powerful, open-source object-rational database that runs on its own database server. We chose PostgreSQL because Django and PostgreSQL work seamlessly together, as well as the fact that PostgreSQL is highly extensible, supports special filetypes in its database - such as binary fields, multi-dimensional array fields, file fields and so on, which left us with several different options when deciding how to store our data in the database, and lastly because of its proven reliability, security and performance.

3.1.3 Implementing REST using Django REST Framework

Following this, we decided to modify our Django server structure to follow the Representational State Transfer (REST) architectural style. This was achieved by using the library and toolkit Django REST Framework [39]. The REST architecture comes with the benefit of enabling the client and server to be fully separated, because any software built after the REST architecture is *stateless*. That is, the client side does not need to know what state the server side is in, and vice vera. This means that they can be implemented independently, as long as they know what format to communicate with.

We communicate with the REST API and the database by sending HTTP requests over the network, where the request contains a HTTP verb (get, post, put or delete) deciding what operation to carry out, a header containing information about the request being sent, a path to a resource, and optionally a body that contains more data. Figure 3.3 gives a general overview of how the HTTP requests flow through the Django REST API. While we will cover the client side in greater detail in Section 3.1.4, the client sends a request through the `Node.js` server, that redirects it into the Django server, running at `localhost:8000`. Here, the request is met by the Web Server

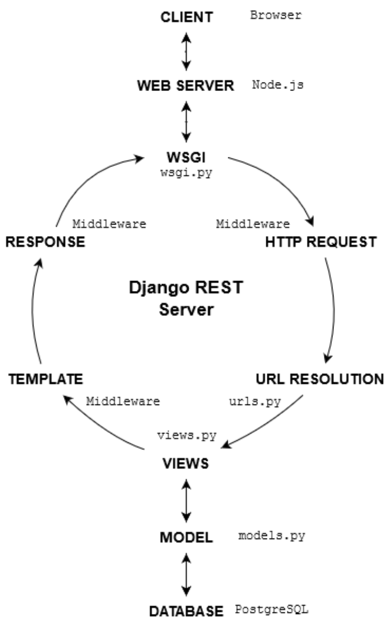


Figure 3.3: Flow Diagram of the Django Server

Gateway Interface (WSGI), that parses and routes the HTTP request, based on the resolved URL from the request. For the URL parsing, we have made endpoint URLs that support different HTTP verbs for users and analyses, where analyses means all information regarding the medical image analysis process.

The most challenging step here was to incorporate the Django MTV structure with the REST framework serializers, while running our own scripts in order to extract the data, before producing a useful output. As the business logic is obscured between the Django view and the Django framework itself, it was challenging to find a place where the data modification scripts felt rightfully placed.

Another challenge that rose from the benefit of having a separable client and server, was making servers running on two different ports communicate without facing Cross-Origin Resource Sharing (CORS) errors. That is, there is a security feature incorporated in any server, that if a local request is made to a local server on a different port, it is blocked for security reasons. In order to handle this, without simply white listing the entire port, something that is considered bad practice, we had to make some adjustments to the client, which will be covered in the following section.

3.1.4 Frontend Architecture, Libraries and Tools

We built the frontend of our system using HTML5, the latest release of Hyper Text Markup Language which is the quintessential standard for any web application, Sass, which is a commonly used Cascading Stylesheet (CSS) framework, and the JavaScript library React [43], which is maintained by Facebook. More specifically, we based our software on the tool Create React App (CRA) [44], which is a tool built by Facebook to quickly set up a single page application (SPA), with the building scripts out-of-the-box, as well as a development- and production server. While some argue that CRA is too bloated and contains too many dependencies for smaller projects like ours, we decided to use it as it is great for building prototypes. Besides, as we built the backend following the REST architecture, it is fairly effortless to replace a CRA-based application with one that is not based on CRA.

CRA is built using webpack [45] and Babel [46]. To clarify, webpack is used to statically bundle all the assets, such as images, CSS files, and fonts, and internally make a dependency graph. Meanwhile, Babel is used to compile next-generation JavaScript, such as the JavaScript written in React, into browser-compatible JavaScript. In order to run the frontend code, we used the development server provided by CRA, that runs on Node.js [38].

It was here, when trying to make the Node.js server and the Django server communicate, that we first encountered the CORS errors. In order to resolve the CORS error without white listing ports, we had to eject CRA in order to access the configuration files of webpack and the Node server. Put in simple terms, CRA is a tool that abstracts away all of the dependencies and configuration files, so that the developer only has to deal with one dependency; the CRA tool itself. In return, the CRA tool maintains all the configurations and dependency updates. *Ejecting* means that you move away from the tool, and receive all the dependencies and configuration files for the web application in a `package.json` file. However, this also means that ejecting is a irreversibly action, and from this point the you have to maintain dependencies manually. Because of this, it is worth to mention that ejecting CRA is a highly controversial topic.

Nevertheless, after ejecting and gaining access to the configuration files, we figured out a way to avoid the CORS error, by adjusting the proxy configuration, that enabled us to redirect the HTTP requests.
While we acknowledge that the following information is on the verge of being too technical, it was a critical breakthrough in the application work. In Listing 1, we modified the `start.js` script, which is the configuration script that handles the development server. Here, we specify that whenever the client sends a request to `'/api'`, for example `fetch('/api/users')`, the Node server will redirect this request to its target. As shown in line 101 the target is the url Django is running at, namely `http://localhost:8000/`. In addition, we rewrite the path and remove the `'/api'` part in line 105-

```
 99 const proxyConfig = {
100   '/api': {
101       target: 'http://localhost:8000/',
102       changeOrigin: true,
103       xfwd: true,
104       ws: true,
105       pathRewrite: {
106           '^/api': '',
107       },
108    },
109 };
```

Listing 1: Proxy configuration from start.js that enabled communication between the Node and React Server

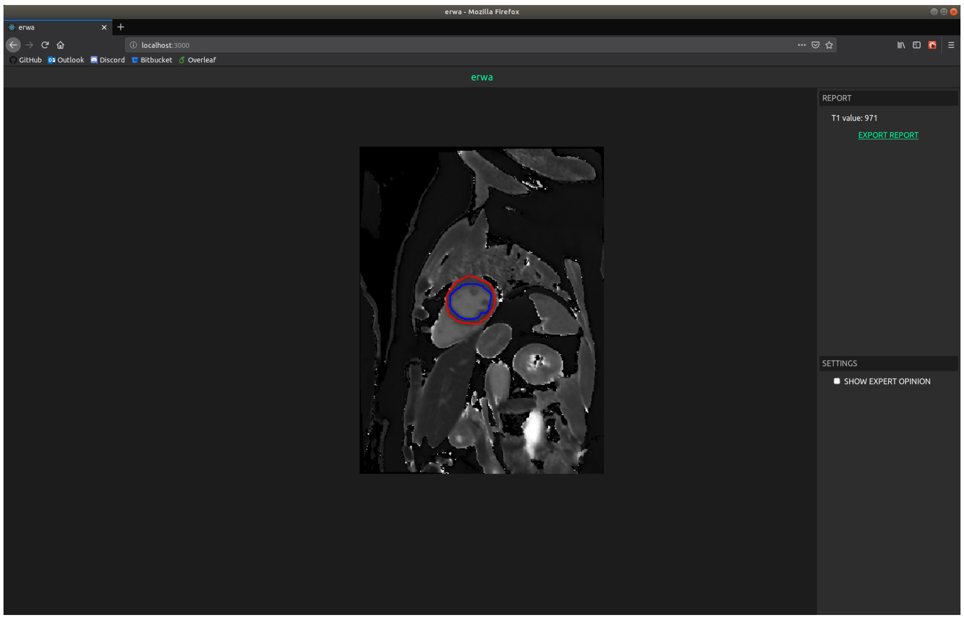
106, so our fetch example would result in a request towards the endpoint `http://localhost:8000/users`. This is because from a client perspective, a fetch that specifies it is going towards the API improves the readability, however on the API side, setting up URL endpoints where every endpoint begins with `'/api'` simply becomes clutter, as we already know we are dealing with the API.

Put in simpler terms, this means that as far as the client is concerned, it is only communicating with the Node server. It is the Node server that handles the redirecting and reception of requests and responses from the backend.

3.2 The Trainee Dashboard

After making some adjustments to the design presented in Figure 3.2,. Although, we decided to adjust the focus from a full MRI segmentation stack to a single segmentation. The user is first met with a form where the user is asked to upload a DICOM file, a SAV file, and an HTM file containing the key values, that is also a part of the dataset. Once the user submits this form, after it is processed and stored in the backend, the user interface (UI) will update and the DICOM image with the contour is displayed, as well as the key values in the right sidemenu. An example of the dashboard can be seen in Figure 3.4

While the UI might seem a bit scarce, it is important to stress again that web-based systems are very rare in this type of clinical practice. Therefore we had to make a lot of functionality from scratch, and there were few sources to take inspiration and guidelines from. An example of this is that we made an MRI DICOM image parser with python scripts in the backend, that converted the DICOM images and the SAV contours into a web-friendly format, namely a PNG, where the contour is put over the DICOM, as shown in Figure 3.4. Additionally, we also extended these



T1 value: 971

Figure 3.4: An example of the Fabulinus trainee dashboard. Note that here the web-application still had its old, placeholder name erwa.

scripts to support two sets of contours to be overlayed on the same image, in order to compare the trainee contour *and* the expert contour over one DICOM. Lastly, we also made scripts to extract the key values from the HTM files.

Another important aspect we spent a long time discussing and considering was where in the web-application the waiting time should end up. Typically, when developing web-applications, everything should ideally be swift. Users might be more patient to wait for processes such as image uploads, however anymore than a few seconds would leave most users frustrated. Yet when dealing with medical image processing, especially when used in a new environment, the conversion and storage might take more than a few seconds.

Following this, our initial idea was the cascade the waiting time throughout the application. For example, when the trainee uploads their analysis, the backend would only process the uploaded analysis and then respond. Then, when the trainee would ask to compare their results to the expert, the backend would only now compute the difference and then produce the output. However, after confiding with the experts at OCMR, we found out that what they are used to in the programs they use on the regular, is to upload their data and then wait up to several minutes before they actually could start working. As they put it, they preferred this approach as they could grab a coffee in the meantime. As result, despite it being somewhat against web-application standards, we decided to adjust our approach to our target user base and carry out all needed computations at the beginning (after the form submit), at the expense of a slightly longer waiting time. However, as the web-application format of the images, as it stands today, does not need anywhere near the same amount of data as fully fledged image analysis tools, the waiting time does not come anywhere close to several minutes, but perhaps a few seconds longer than normal applications.

3.3 Summary

In this chapter we have presented our software tool, Fabulinus, which aims to aid the training of new observers. Our software is built on modern technologies, and aids the training process by allowing trainees to visually and numerically evaluate their work by comparing against both expert contours and an automatically generated "expert" consensus using deep CNN. In addition, we have made several different converters in order to convert the original medical file formats into web-application friendly formats. While it is still in the prototype stage, we have already received positive feedback that this tool could very well one day be used in a true classroom setting, after the finalising of the development is carried out.

Chapter 4

Automatic Generated Feedback Using Convolutional Neural Networks

In this chapter, we will portray the various steps of how our deep learning architecture and model unfolded. Here, we explore the supervised machine learning problem of generating "expert" contours from different types of raw input data, using deep convolutional neural networks. Our aim is to provide a different way to assess the segmentation performed by trainees during training, in order to possibly reduce the volume of human expert contours needed for evaluation and comparison of analyses.

First, we will describe the dataset we used, and how we prepared it in order to apply it as raw data input for our deep neural network model. Then, we present the environment that was used to perform the training of the model, and the most important libraries and frameworks used during training. Subsequently, we cover all the pre-trained architectures that was used, and finally, we cover how the model training was carried out step by step, including some brief reflections on why the models performed the way they did.

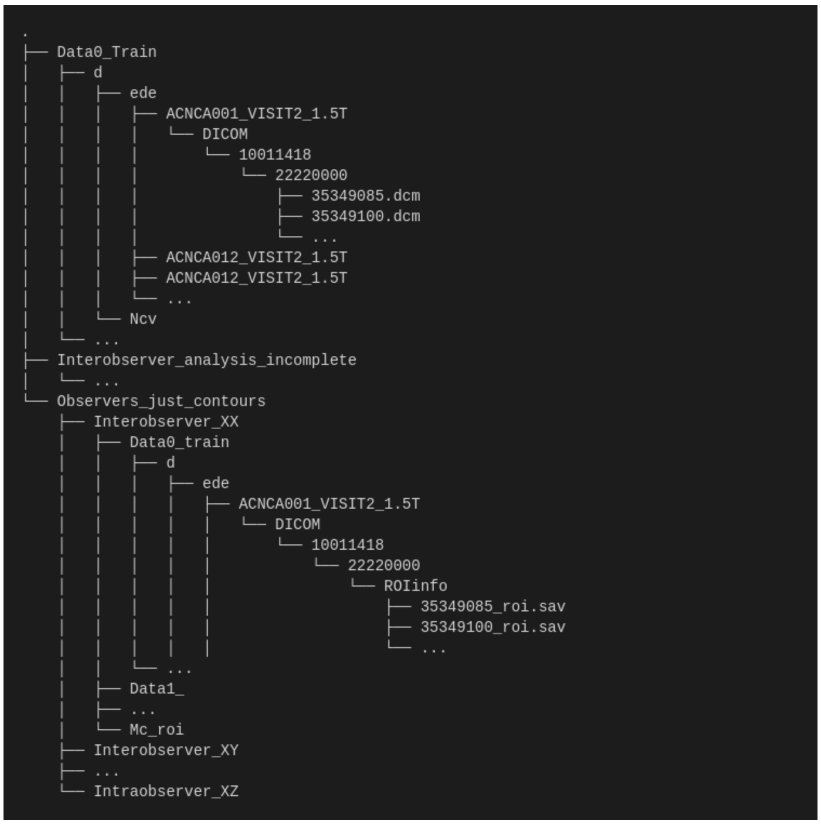
4.1 Dataset Details and Preparation

As part of our collaboration with OCMR Corelab, they provided us with a dataset they employ for training, that consists of single mid-ventricular 2D T1 mapping images of the heart for 42 cases. These cases are equally divided between healthy volunteers and patients with a range of cardiovascular pathologies, and the MRI images are stored in the DICOM format. Additionally, the dataset contains several repetitions of contours from three expert- and 22 trainee image analysts, that outline the epicardium and endocardium of the heart, which are stored in the SAV format. Further details about the MRI acquisition setting and patient characteristics can be found in Piechnik *et al*. [47] The dataset does not have a name and is under a non-disclosure agreement (NDA), so it is not available to the general public.

The dataset has a complex nested structure, as it was obtained by sampling a number of pre-clinical and clinical studies at OCMR and structured to be processed by software tools available at the centre. Therefore, the dataset had to be transformed into a different folder structure in order to provide input for our deep learning pipeline, in particular for the deep learning preprocessing, as covered in the following sections.

4.1.1 Generating A New Folder Structure For The Dataset

The dataset we were provided was created by an in-house software at OCMR called MyoCardial Regions Of Interest (MC_ROI). MC_ROI was developed for semi-automatic delineation of regions of interest for T1- and WT quantification, and image quality assessment, and is specifically designed to facilitate the analysis process of a complete set of patients at once. It is also routinely employed to carry out the T1 mapping analysis at OCMR. In addition, MC_ROI is fairly flexible and insensitive to the folder structure of the dataset, due to a powerful DICOM indexing functionality that automatically enlists all the patients set of scans present in the folder, and finds the corresponding contour. It achieves that by exploring the metadata in the DICOM. This is how it can cope with any arbitrary folder tree structure coming from any clinical study. Therefore, there are not many requirements towards the folder structure, and in this dataset case it is obscure from both a human-readable perspective and a programmatical perspective (with regards to path walking, relations between files and so forth), as well as its not easily applicable to other use cases such as machine learning. To add some context, in Figure 4.1, the first `Data0_Train` folder contains all the DICOM images, and no contours. Then, in the `Observers_just_contours` folder, each observer has their own folder with their initials at the end (anonymised in the figure as `Interobserver_XX`) that contains the contours carried out by the observer. Here, the contours are first separated into folders following the pattern `Data0_Train`, `Data1_`, `Data2_` and so on, these folders describing how many repetitions of contours were done for each patient, and then each repetition is separated into folders following the pattern `ACNCA###_VISIT#_1.5T` (# being a number de-

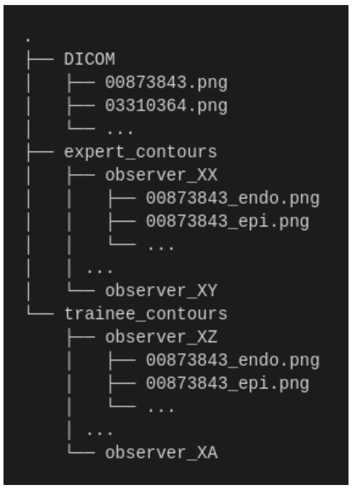


```
.
├── Data0_Train
│   ├── d
│   │   ├── ede
│   │   │   ├── ACNCA001_VISIT2_1.5T
│   │   │   │   └── DICOM
│   │   │   │       └── 10011418
│   │   │   │           └── 22220000
│   │   │   │               ├── 35349085.dcm
│   │   │   │               ├── 35349100.dcm
│   │   │   │               └── ...
│   │   │   ├── ACNCA012_VISIT2_1.5T
│   │   │   ├── ACNCA012_VISIT2_1.5T
│   │   │   └── ...
│   │   └── Ncv
│   └── ...
├── Interobserver_analysis_incomplete
│   └── ...
└── Observers_just_contours
    ├── Interobserver_XX
    │   ├── Data0_train
    │   │   ├── d
    │   │   │   ├── ede
    │   │   │   │   ├── ACNCA001_VISIT2_1.5T
    │   │   │   │   │   └── DICOM
    │   │   │   │   │       └── 10011418
    │   │   │   │   │           └── 22220000
    │   │   │   │   │               └── ROIinfo
    │   │   │   │   │                   ├── 35349085_roi.sav
    │   │   │   │   │                   ├── 35349100_roi.sav
    │   │   │   │   │                   └── ...
    │   │   └── ...
    │   ├── Data1_
    │   ├── ...
    │   └── Mc_roi
    ├── Interobserver_XY
    ├── ...
    └── Intraobserver_XZ
```

Figure 4.1: The original folder structure of the dataset.

scribing the patient or what visit it was). Finally, the contours are placed in a folder structure of `DICOM/########/########/ROIinfo/`.

Hence, with a folder depth of seven to reach the DICOM images and ten to reach the the related SAV files, we decided to make scripts that placed the files in a more streamlined folder structure, that simplifies the processing for the deep learning pipeline. As shown in Figure 4.2, the new folder structure puts the DICOM- and contour folders at the top level, and separates the image analysts and their contours based on their level of expertise. In addition, as the original SAV files included both the endocardium- and the epicardium contour, we separated them into two files with a trailing `endo` or `epi` suffix, in order to be able to analyse them separately.



```
.
├── DICOM
│   ├── 00873843.png
│   ├── 03310364.png
│   └── ...
├── expert_contours
│   ├── observer_XX
│   │   ├── 00873843_endo.png
│   │   ├── 00873843_epi.png
│   │   └── ...
│   │ ...
│   └── observer_XY
└── trainee_contours
    ├── observer_XZ
    │   ├── 00873843_endo.png
    │   ├── 00873843_epi.png
    │   └── ...
    │ ...
    └── observer_XA
```

Figure 4.2: The new folder structure of the dataset, note that here we have regular .png suffixes

With the new folder structure, we could then start extracting information and converting the dataset files into formats that are typically used in machine learning for image analysis.

4.1.2 Handling DICOM and SAV files

In order to begin visualising the images in the web application, or running a CNN model on the images, the image and contour information had to be extracted from the filetypes. As the filetypes are not suitable for image analysis in machine learning, we had to make specialised scripts to extract the data from the files. Extracting the DICOM files turned out to be fairly painless, as the python library `pydicom` [48] provides excellent modules to extract the image data. However, the image data extracted results in an instance of a two-dimensional array, that needs to be displayed. Initially we used `matplotlib` to plot the values stored in the DICOM, however the ensuing step of having to strip away all the axes, labels and so forth generated by `matplotlib` to make it a plain image made the modules both lengthy, slow and difficult to read. The lengthy code in question can be seen in Appendix Section B.1. Therefore we improved the scripts and extracted the DICOM files using `cv2` instead, as follows:

```
1  import pydicom
2  import numpy as np
3  import cv2
4
5  def store_dicom_image(input_dicom):
6      # read_file() returns an instance of FileDataset
7      dicom_dataset = pydicom.read_file(input_dicom)
8      # pixel_array is an array of the image
9      dicom = np.array(dicom_dataset.pixel_array, dtype=np.float32)
10     dicom *= (255/np.max(dicom))
11     dicom = np.around(dicom, decimals=5)
12     cv2.imwrite("example_dicom.png", dicom)
```

Listing 2: Extraction of image from DICOM file

The extra steps in line 10-11 of normalising the values and rounding them off by decimals are due to the fact that `pixel_array` does not return a numpy array of RBG-values, but rather the byte information of the image. Thus we normalised the values in order to display the image as values between 0 and 255. Additionally, as we carry out division on the numpy array in line 10 with rather large numbers, we had to override the `pixel_array` dtype in line 9 (as the default is `int16`).

On the other hand, the SAV files added another level of complexity. First of all, the SAV files were written in the proprietary programming language Interactive Data Language (IDL) [49], which is specifically designed for making visualisations of numerical data and is currently being maintained by Harris Geospatial Solutions, Inc. SAV files in general are largely used in astronomy and physics.

```python
1  import matplotlib.pyplot as plt
2  from scipy import io as sp_io
3  
4  def extract_contour(input_sav):
5      # Reads a single IDL file (extension is .sav)
6      idl_dict = sp_io.readsav(input_sav, verbose=True)
7      # Extracts the binary mask from the dictonary
8      bmask = idl_dict.savedroi.MXMASK.squeeze().tolist()
9      # Extracts the different contours from the binary mask
10     # by finding the different colour levels
11     contours_set = plt.contour(bmask)
12 
13     # Multiple levels of erosion are stored both for epi and endo
14     # Thus we only take one contour for endo and one for epi in
15     # the middle of the erosion range
16     endocardium_contours = contours_set.allsegs[2][0]
17     epicardium_contours = contours_set.allsegs[5][0]
18 
19     return endocardium_contours, epicardium_contours
```

Listing 3: Condensed code sample of extraction of contours from IDL files

In order to extract the numerical data, we used the the input and output library of SciPy, `scipy.io`, to read the IDL file with the module `readsav()`. This module returns a python dictionary of the data. Thereafter, the conversion steps needed to achieve the format we need to employ deep learning algorithm turns quite complex.

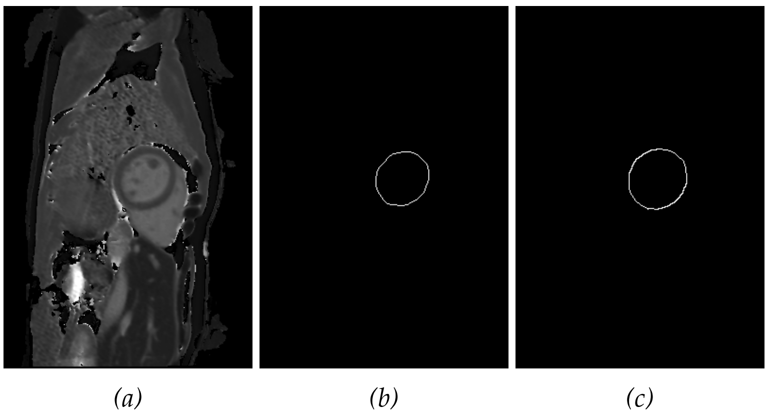


Figure 4.3: Figure displaying (a) an example MRI image from the dataset, as well as (b) the endocardium contour and (c) the epicardium contour

In line 8 in Code Excerpt 3, we extract a binary mask from the saved region of interest dictionary, trailing into the dictionary under `savedroi`, finding the binary mask and compressing it to a list. We then use `matplotlibs contour` module to find the contours in the binary mask, whereas the `contour` returns a `QuadContourSet`. This is a set containing all the contour. It also has a parameter called `allsegs[]`, which are all the contours at all levels. That is, given a contour, it is stored in several variations, where it is both expanded or eroded by one pixel in order to reduce bias. Therefore, we pick a single contour that is in the middle of the erosion range, return these in the module and later store them using `cv2`

Following this, the dataset was now ready for the "conventional" preprocessing, a process that we will get back to in Section 4.4.1. An example of the modified data can be seen in Figure 4.3

4.2 Deep Learning Environment Setup

Level	Category	Name	Version
Hardware	GPU	NVIDIA GTX Titan X	*N/A*
	CPU	Intel i7	*N/A*
	Memory	Corsair 16GB DDR4	*N/A*
Software	Operating System	Ubuntu Bionic Beaver	18.04.2 LTS
	Conda VirtualEnv	Anaconda	4.6.8
		Python	3.6.8
		Tensorflow-GPU	1.12.0
		Keras	2.2.4
		CUDA 9	9.2
		cuDNN 7.2	7.2.1

Table 4.1: System specifications for the computer used for the training of the machine learning models

In this section we will cover the system environment specifications used for the deep learning model, both for training and evaluation. First, we will cover the hardware specifications, then we will present the software and libraries used in the source code, and finally what deep learning pre-trained model used to carry out the training. See Table 4.1 for a general overview. All the training and evaluation of the deep learning model was carried out on a stationary computer, with the specifications presented in Table 4.1.

4.2.1 Software and Libraries Used

The deep learning model we made relies heavily on software libraries and pre-trained models. In this section we will cover the most important software and libraries that made this model possible, but it is important to realise that this is not an extensive list.

Anaconda and Conda Virtual Environment

Anaconda [50] is a free and open-source software (FOSS), that distributes the programming language Python and R, as well as being a infrastructure for scientific computing and offering a package repository. In this thesis we have largely used Anaconda for all package management and version control of these packages. In addition, we have used Anaconda to handle the virtual Python environment with all the libraries needed to run the deep learning model, as this is beneficial when handling several libraries that depend on each other. That is to say, as shown in Table 4.1, all the libraries that we will shortly introduce, are installed and set up within this Anaconda virtual environment.

TensorFlow

TensorFlow [51] is a open source library and state of the art machine learning system, that focuses on training of deep neural networks, and is created and officially being maintained by Google internally, while an implementation of TensorFlow is also available on GitHub as a open-source project. We decided to use TensorFlow in this thesis for several reasons, that include (a) it is a machine learning system that is scaleable and maintainable, as well as flexible in regards to its availability for operating systems such as Windows, iOS, Linux, and Android. These factors have led it to be (b) a well suited framework for systems that has the potential to proceed from research to production, as compared to for example PyTorch and Chainer, which are machine learning frameworks that are flexible enough for research, but less scaleable for production. Perhaps most important, it is (c) a widely popular framework that is used by several serious parties such as Google, OpenAI, NVIDIA, and Intel to mention a few, which also means it is updated regularly, has a very active community and is used in many serious research projects. Lastly, (d) it provides powerful functionalities such as the ability to save and restore models after the training is done, and supports powerful tools such as TensorBoard, that allows visualisation of the training during execution.

As for what TensorFlow does, it utilises numerical computation in data flow graphs, where the definition of the computational operations are stored in the graph nodes, and the executed numerical values are stored in the graph edges as n-dimensional tensors (where 0-dimensional tensors are scalars, 1-dimensional tensors are vectors, 2-dimensional tensors are matrices, and so on). To put it differently, tensors can be viewed as data flowing through the graph. A simple using scalar tensors can be seen in Figure 4.4. While the benefits of using this computational system is

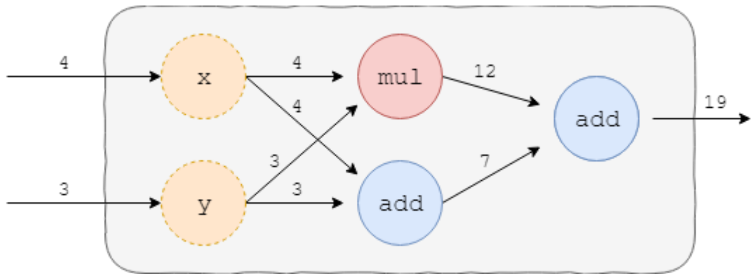


Figure 4.4: A simple example of what a data flow graph used in TensorFlow could look like

not apparent in a elementary example like the one above, TensorFlow's strength is that it stores each execution in a session, or in other words every graph edge value is stored. Imagine a more typical deep learning example, where the network consists of hundreds of thousands of nodes and edges, if every single node and edge had to be recalculated for each update or backpropagation the computational power and time spent would be immense. And lastly, TensorFlow can facilitate distributed computation, where several servers, CPUs or GPUs can be orchastrated to compute different parts of the data flow graph, reducing the work load on one unit.

Keras

Keras [52] is a high-level neural networks API, that is written in Python and built upon the principles of user friendliness, modularity, and easy extensibility, with a focus on fast prototyping and experimentation. While Keras can be ran on top of both the Microsoft Computational Network Toolkit (CNTK) [53] and Theano [54], we will run it ontop of TensorFlow, as this is the default for Keras and it suits our needs better.

We chose Keras because for its simplicity, and because it provides several pre-trained architectures, which enables us to quickly set up experiments and run many iterations. This feature in particular became invaluable during our experimentation of different applications of the models on our dataset, with minimal setup time and configuration. Simultaneously as being simple, Keras also provides its own API called the *Keras backend API*. This API reinforces the Keras principle of being extensible, in that it enables advanced configurations to be carried out on the pretrained models. For example, the backend API allows the user to override the modules and operations on the Keras backend, in order to customise the model to their own needs, and improve accuracy and results.

Here, it is also worth to mention that both TensorFlow and Keras are the two most popular deep learning frameworks, based on factors such as job listings, publications, Google search activity, and GitHub activity, as found in this Medium article written by Jeff Hale in 2018 [55]. The results between the selected [1] deep learning frameworks can be seen in Figure 4.5.

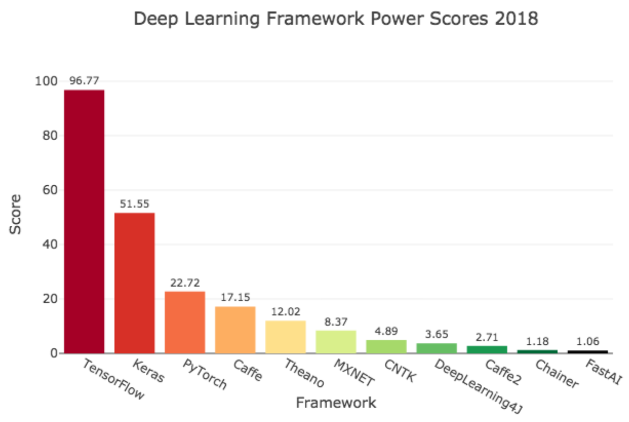


Figure 4.5: Results from the Deep Learning Power Scores, based on popularity metrics. Adapted from: [55]

[1]Any deep learning framework with more than 1% of reported usage on KDNuggets usage survey

4.2.2 Architectures Used for Training and Evaluation

When choosing the different pre-trained architectures from Keras to apply on our dataset, we decided on architectures that have proven to be very accurate in image analysis on well known datasets. An overview of the

Model	Size	Top-1 Acc.	Top-5 Acc.	Parameters	Depth
VGG16	528MB	0.713	0.901	~138 million	23
VGG19	549MB	0.713	0.900	~143 million	26
ResNet50	98MB	0.749	0.921	~28 million	-
InceptionResNetV2	215MB	0.803	0.953	~55 million	572

Table 4.2: Information about the different Keras architectures used, including size (of the pre-trained model), accuracy, number of trainable parameters and topological depth of the network.
Adapted from: [56]

architectures we chose can be seen in Table 4.2. Here, **Top-1 Accuracy** and **Top-5 Accuracy** refers to the accuracy achieved when the models were trained on the ImageNet validation dataset (ILSVRC). [57] That is, ImageNet is a dataset containing over 14 million images with over 21 thousand categories (when including synsets or subcategories, for example 'high-level' category being bird and subcategories being swan, eagle, and so on). To clarify, for every image out of the 14 million available the model can receive as input data, there are 21 thousand possible output labels. Out of these 21 thousand labels, Top-1 Accuracy means that the class label that got predicted as the highest probability for the respective image by the model, in fact was the ground truth label, and in the same way Top-5 Accuracy means that the ground truth label was in the top 5 predicted classes. Coming back to Table 4.2, we see that all the models have an accuracy of at least 90% for the Top-5 Accuracy. In the following sections, we will cover each model more in detail.

VGG Architecture

The Visual Geometry Group (VGG) neural network is a network built on convolutional blocks, introduced in a paper from 2014 by Simonyan and Zisserman; *Very Deep Convolutional Networks for Large Scale Image Recognition*. [58] In this paper, they proved that by considerably increasing the depth of a CNN, the previous state of the art performance in terms of classification accuracy on the ImageNet dataset could be surpassed. This achievement led to the neural network placing both first and second in the ImageNet Challenge for localisation and classification in 2014. [58]

The characteristic of a standard VGG configuration is that it takes 224 x 224 coloured images as input, and is built using five blocks of convolutional layers, where each convolutional filter is typically very small, the default being 3 x 3. Following each block, there is a max pooling layer resizing the next block. The depth of network is typically either 16 or 19 hidden layers, whereas these two configuration depths are typically referred to as VGG16 and VGG19, respectively. After the fifth convolution block, the network carries out feature extraction by using two fully connected layers of the size 1 x 4096, and lastly one softmax layer is applied for classification, that leaves one layer by the size 1 x 1000, where all values are between 0 and 1 and the sum of the values become 1.

ResNet Architecture

The residual neural networks (ResNet) was introduced by a group of researchers at Microsoft in their paper from 2015; *Deep Residual Learning for Image Recognition.* [59] Here, they proposed a new solution to one of the biggest issues of training very deep networks, the vanishing gradients problem[2]. Although there were different ways to somewhat address the problem, such as adding an auxiliary loss [60], these methods did not make the issue go away entirely.

He *et al.* solves this issue with ResNet by utilising *identity shortcut connection* for residual learning. Shortcut connections essentially skip one or more layers in the network by performing *identity mapping*, where each output is added to the output of the stacked layer, by learning the difference between the input and the output and adding them together, as shown in Figure X. In theory, the network ends up not producing a training error that is higher than its shallower layers.

While similar concepts utilising shortcut connections were already present in other proposed networks, such as "highway networks" [61], He *et al.* showed their approach was superior both due to the fact that ResNet does not add parameters and as a result requires less computation and storage power, and also that ResNet demonstrated higher accuracy at depths over 100, as opposed to highway networks. Therefore, after proposing layer depths with increased accuracy of 50, 101 and 152, ResNet ended up winning the 2015 ILSVRC as well as the Microsoft Common Objects In Context (COCO) Competition, and secured first place in all of the main tracks (classificaiton, detection and localisation for ILSVRC, detection and segmentation in COCO).

Inception

Inception is a convolutional network, presented by Szegedy *et al.* in their paper *Going Deeper with Convolutions* [60], in order to reduce the computational power needed for training of deep networks. Put simply, the straightforward way of increasing the performance and accuracy of a deep network is to increase its size (or depth). However, increasing the size

[2]Recall how this occurs when a backpropagated gradient, over the depth of the layers, gets repeatedly multiplied to the point where it turns infinitely small, and the network learning rate either stagnates or even starts dropping

is also prone to introducing one or both of the following drawbacks; (a) making a network deeper also increases the number of parameters, that as a result increases the storage needed and the chance of overfitting, and (b) by increasing the size, one also computational power needed. As resources will always be finite, in order to address these two issues, Inception was made.

4.3 Deciding On Appropriate Cases

When we first considered appropriate cases where machine learning can be applied, the first one that arose was a system that is meant to provide feedback to the trainee observers by generating an expert contour out of the original trainee's contour. To generate this feedback a convolutional neural network was going to be used. The idea was a network that consists of an input layer which gets an image as input. The image consists of four channels; the red, green, and blue colour channels plus the trainee observer contour in the fourth channel. We discussed using a hidden layer architecture that is designed like AlexNet [62] with the modification of supporting the 4-channel images. Then, the output layer is designed as a per-pixel classification layer, that is each pixel is a binary class leading to a number of pixels times classification problem. Each pixel is compared with the expert contour for error calculation. The advantage of using pixel-wise classification is that each pixel can be seen as a training example and therefore the training is more efficient with the smaller amount of images in the data set. It is important to point out that the *hyperparameters*, meaning the parameters that configure the machine learning model before the training starts that are deciding for how the model will behave and how successful it will be, can be adjusted.

However, we wondered if the system could learn to make the expert contours without being provided with a reference contour in the input data. Provided that this was possible, it would eliminate the need of providing a contour in the input data, as well as potentially making the system more robust to different cases. With this in mind, we decided to go for this system instead. We also adjusted the approach and architectures used, something we will cover in the next section.

4.4 Predict Expert Contour From DICOM

In this section we will first describe the preprocessing that was carried out on the data, then present the machine learning model we came up with, what other models were considered along the way, as well as the iterations we went through and briefly touch upon the results that drove the methodology forwards. For a more detailed walkthrough of results and a discussion of them, please refer to Part III Experiments, and its chapters.

4.4.1 Preprocessing of the Data

As covered in Section 4.1.2, we converted the DICOM and SAV files to plain images in the PNG format. In doing so, we discovered that the image resolution of the DICOM images ranged from 384 x 232 pixels to 384 x 344 pixels, with a lot of small increments along the y-axis along the way. In order to carry out the machine learning, the images had to be rescaled into smaller, quadratic shapes. We decided on 224x224 in order to save as much image detail as possible, without making the input data too large.

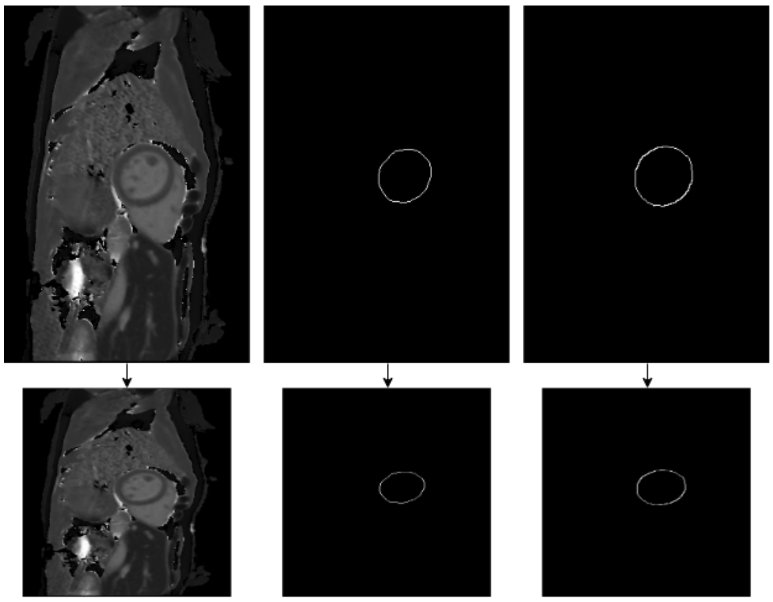


Figure 4.6: The dataset images before and after rescaling

4.4.2 First Iteration

Pre-trained Model	VGG16
Loss Function	Mean Square Error (mse)
Metrics	Mean Absolute Error (mae)
Optimizer	Nadam
Batch Size	2
Epochs	50

Table 4.3: Specifications for the First Machine Learning Model

During the first iteration our main goal was to see if the model could learn something valuable from our data set and approach. As briefly touched upon earlier in Section 4.3, we decided to approach this as a *multiclass classification problem*, where we would carry out *per-pixel classification*. That is, for each pixel in the MRI image, the deep learning model attempts to predict whether the pixel is a contour pixel or not, something we will come back to in greater detail the preceeding sections.

Preprocessing

Per-pixel classification is done by turning pixels that were in a contour on (by changing their value to 1), and pixels that were not in the contour off (by changing their value to 0), where 0 and 1 will be the two classes. In other words, for each expert contour, we changed all values greater than 0 (0 meaning the colour black), into 1 as part of the preprocessing. In order to seamlessly compare pixel values, we also had to implement another preprocessing step, by turning the test data (the expert contours) into vectors. This is done by taking each pixel row of the contour image, and concatenating them after each other turning, in our case, a 224 x 224 image into a 1 x 50176 vector.

As for creating the training data- and test data matrices, we had several repeats of contours per MRI image. In order to give the model as much training data as possible, we decided to train on both endocardium and epicardium in the same model. Following this, we had in total 948 expert contours, including all repetitions of both endocardium and epicardium per expert per image. In contrast, we only had 84[3] MRI images, so to handle this we generated a input training set where we repeated the MRI image for each contour, so that the indices would line up, as shown here:

```
            X_DATA         Y_DATA
index 0 [MRI_XX, ] [XX_EXP_CONTOUR_1,]
index 1 [MRI_XX, ] [XX_EXP_CONTOUR_2,]
index 2 [MRI_XY, ] [XY_EXP_CONTOUR_2,]
...
```

In `X_DATA`, we repeat `MRI_XX` for every contour that is done on `MRI_XX`. And this continues for every MRI image and contour. As for the separation of training and test data, we decided to split it roughly 80% to 20% - which leaves us with training and test data shapes as follows:

X_train	(758, 224, 224, 3)
X_test	(190, 224, 224, 3)
Y_train	(758, 50176)
Y_test	(190, 50176)

Table 4.4: Data shapes for the first model

Training Configuration

Initially we tried the VGG16 and ResNet50 pretrained models from Keras. We used mean absolute error as the loss function, as well as Nadam for optimisation. For the first attempt we ran it with a batch size of 2, due to the substantial size of training parameters, and let the model train for about 50 epochs. We also shuffled the data in order to reduce bias, as the images were added to the training- and test data sets chronologically.

Custom Layers

For our model we also made two custom layers, to address our way of carrying out classification. As shown in the code excerpt in Listing 4,

[3]42 multiplied by two due to repetitions in the DICOMs

we made one layer for normalisation, and one layer for rounding. The `__init__()`, `build()` and `compute_output_shape()` are standard for a custom Keras layer, which is briefly covered in its own chapter in the Keras documentation [63], however the `call()` module is where the specialised logic we created lies. In short, we carry out the same steps on the output

```python
from keras.layers import Layer
import keras.backend as K

class Normalize(Layer):
    def __init__(self, **kwargs):
        super(Normalize, self).__init__(**kwargs)

    def build(self, input_shape):
        super(Normalize, self).build(input_shape)

    def call(self, x):
        return x / K.max(x)

    def compute_output_shape(self, input_shape):
        return input_shape

class Round(Layer):
    def __init__(self, **kwargs):
        super(Round, self).__init__(**kwargs)

    def build(self, input_shape):
        super(Round, self).build(input_shape)

    def call(self, x):
        return K.abs(K.round(x))

    def compute_output_shape(self, input_shape):
        return input_shape
```

Listing 4: Custom Layers for the Deep Learning Model

data of the deep learning model as we did when preparing the DICOM images, something we covered in Section 4.1.2. That is, we use the Keras backend API modules `K.max(x)` and `K.round(x)`, where K is the Keras backend, to element-wise normalise all the values so all the values are between 0 and 1, and the we round them off to the closest integer, where the rounding mode used is "half to even".

What We Learned

The very first iteration we did was with ResNet50, MSE loss function and MAE metrics. We let it train for 50 epochs, and the network quickly

worked its way to a MAE value of 0.0046 - a result that in itself looked promising. However, after the model was done training and we visualised the predictions done by the model, the result was discouraging; the model had only turned on one pixel, something we will show and discuss in Part III Experiments. To make matters worse, the pixel that was turned on was not inside the perimeter of the contour or region of interest.
To rule out the possibility that ResNet provided too many trainable parameters, we ran the same experiment with the VGG16 pretrained model, which has less layers and less trainable parameters, instead, keeping all other hyperparameters the same. Unfortunately, the result was no different. In a final attempt to make this approach work, we generated our own loss function that would punish the network much harder for guessing a contour pixel where there is not one, as follows in Listing 5. Essentially, the idea was that the network would get a much smaller

```
def custom_loss(y_true, y_pred):
    return K.abs(y_true - (y_pred * 100))
```

Listing 5: Custom Loss Function Only Used In First Iteration

loss if it guessed correctly. Nonetheless, after training this model with otherwise the same hyperparameters the result was even worse, where the model did not turn on a single pixel. Now, questioning whether our approach was even plausible, we decided to reevaluate the earlier steps and consider where the main issues of our method was. A more in-depth discussion regarding what potentially went wrong can be found in Section 5.4, but in short we concluded that the essence of the problem was that segmentations were hollow circles, and that the local minima where the model was turning all pixels off resulted in an acceptable loss value, whereas finding the segmentation would take a lot of time and computations. So, for our second approach we decided to update the input data into filled segmentations.

4.4.3 Second Iteration

Having learned from the first iteration that our initial idea did not work, the main component we wanted to alter was the testing data. As mentioned in the previous section, we came up with the idea of modifying the segmentations to be filled circles rather than hollow ones, so that the model could have more leeway. In order to that we had to update the preprocessing step.

Preprocessing

We decided to approach the problem of filling the segmentations by finding the edges of the segmentation and making all values in between a colour rather than black. To achieve this we made the script, where the essence is showed in Listing 6. In brief, for each row we verify that is has a max

```
# ... Loading the contour
contour = np.load(contour_path)
img = np.reshape(contour, (224, 224))
img[ img == 1] = 255
for row in img:
    if np.max(row) > 0:
        max_value_indices = np.where(row==255)
        if len(max_value_indices[0]) == 1: pass
        else:
            start_i = np.min(max_value_indices[0])
            end_i = np.max(max_value_indices[0])
            col[start_i:end_i] = [255 for max_val in row[start_i:end_i]]
# Storing the contour ...
```

Listing 6: Custom Layers for the Deep Learning Model

value greater than 0 (to skip rows that are all black). Then, we use the `np.where()` module to extract all indices in the row that has a value of 255. Lastly, we slice the row between the first and the last index index with a value of 255, and make all values in between 255. After this, we store the images in the same folder structure as the one described in Section 4.1, whereas the corresponding contour image change can be seen in Figure 4.7 Here it is worth to mention that we later in the thesis work found out

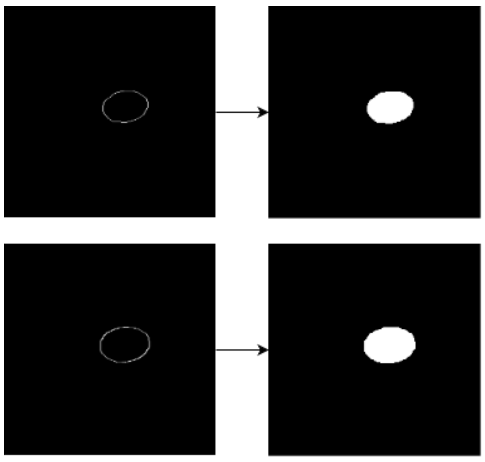


Figure 4.7: Figure showing the result of filling the contours in the preprocessing, where the endocardium is in the top row and epicardium in the bottom row

`matplotlib.contour()` has an equivalent called `matplotlib.contourf()` that produces filled contours, however this was discovered after several iterations were run. While we did not evaluate the difference, if any, this is probably the superior approach to filling the contours.

Changes to Training Configuration and Custom Layers

While we made minimal changes to the model, as we wanted to verify that the input data was the problem, we let the model run for 500 epochs in order to reduce the chances of the model getting stuck in a local minima. In addition, we ended up not using the `Round` custom layer that we described in the section above.

What We Learned

After the training was done the MAE was already looking a lot more promising. After visualising the predicted results after the fact, we were pleasantly surprised to see that the results were a lot better. Again, for more in-depth discussion and result evaluation, please refer to Section 5.4. That being said, as briefly mentioned in the first iteration chapter, we trained on both the endocardium and epicardium segmentations at the same time. As a result, we were presumably confusing the model as epicardium is always larger than endocardium by quite a substantial bit. This seemed to be causing the predicted contours to have blurry edges. Therefore, we wanted to run a few more experiments, separating the epicardium and endocardium segmentations into two data sets and running experiments on both in order to improve the edge accuracy, and accuracy overall. We also wanted to test other popular CNN architectures, such as rerunning the ResNet and trying the Inception architecture, to investigate the differences the architectures would produce. We will come back to how this went in Part III Experiments.

Part III

Experiments

Chapter 5

Deep Learning Model

In Part II, chapter 1 we covered the architectures and process of how these experiments took place, going over the decisions made and how the process developed throughout the thesis work. In this chapter, we will go into the more technical details covering the metrics and hyperparameters chosen for the models, and introduce the evaluation models used to evaluate the output of our model. We will present each experiment with all their configurations. Then, we will discuss the results and discuss possible improvements where applicable.

5.1 Training and Evaluation Pipeline

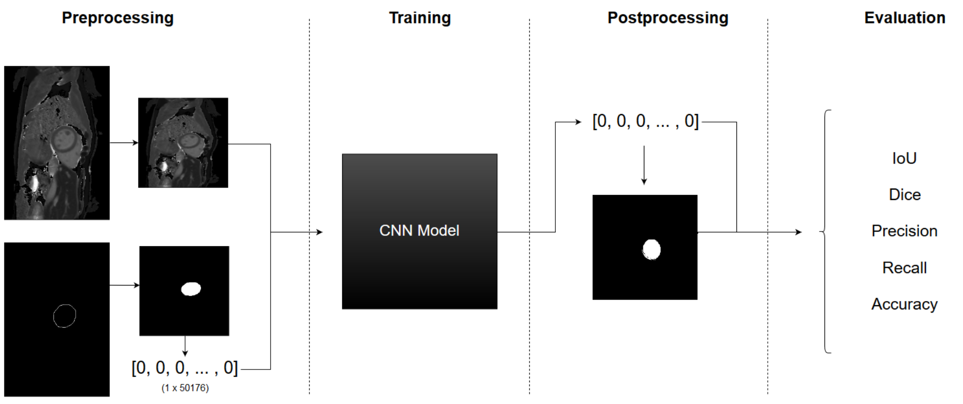


Figure 5.1: Deep Learning Model Pipeline

As a short summary, Figure 5.1 shows the overview of the four phases we go through for every experiment. First, we preprocess the images by rescaling them, and convert the contours into numpy arrays of 1 x 50176 with pixel values that are either on (1) for contour pixels, or off (0). Then we let the model train, and after the model is done, we postprocess the output by loading the predictions and turning them back into images. Finally, we pass them through several evaluation models to determine how successful and accurate the training was. In the following subsections we will give some more detail to the configurations, hyperparameters, metrics and evaluation methods that was used, then in the following section we will present the experiments.

5.1.1 Hyperparameters

Recall how in any machine learning model we separate between parameters and hyperparameters, where hyperparameters are the metrics that configure the machine learning model before the training starts, and are deciding for how the model will behave and how successful it will be. The three most common hyperparameters that typically appear in any neural network are the following:

- **Batch size** refers to the number of training examples to pass through the network and back again, per iteration.
- **Number of iterations** is the number of passes, where each pass includes the batch size of input data.
- **Epoch** is the term for passing *all* the training examples through the neural network once; that is, one time forwards and one time backwards.

For example, if you have 100 images and use a *batch size* of 20, the network will only train on and forward/backward pass 20 images at a time. Additionally, you will have five *iterations*, before you reach one *epoch* where all the images have been passed through network and back again once. Subsequently, if we wanted to run five epochs, we would then end up with 25 iterations in total. Typically, the batch size will be smaller than the number of images, as passing all the input images at once requires a lot of memory and can at occasion slow down the training. Although, the smaller the batch size is, the more it will affect the accuracy of the gradient of the network. What we typically see as the batch size gets smaller, is that it can cause the training accuracy to fluctuate a a lot, as compared to having a batch size equal to the number of inputs.

Mean Squared Error (MSE) is a loss function we used in our model. The loss function can be defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{5.1}$$

As we see in equation 5.1, what the MSE loss function does is that for each i in n predictions, we subtract the predicted output value $\hat{Y}_i$, from the observed ground truth value Y_i, which gives us the error for that prediction. Then, we square this error, and sum all of these values over all the predictions, and lastly we find the mean value by dividing the summed squares errors by the total number of predictions n.

Mean Absolute Error (MAE) The Mean Absolute Error is another loss function used, which is noticeably similar to the Mean Squared Error, as can be seen in 5.2. The main difference is that rather than squaring the absolute error $(\Delta Y) = Y_i - \hat{Y}_i$, we take the absolute value instead.

$$\frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i| \tag{5.2}$$

As a result, MAE is more delicate and more robust towards outliers, while MSE will penalise larger deviations harder. Therefore, MSE is generally more used where larger deviations are more critical, and it also converges faster.

Nesterov Momentum Adam (Nadam) Additionally, we used the optimiser Nesterov Momentum Adam (Nadam) [64], [65], which is a variant of Adam [66], [67]. Nadam is an adaptive learning rate optimiser, that utilises a smarter form of momentum. That is, for regular momentum, a problem that was discovered and addressed in a paper from 1983 by Yurii Nesterov [68] was that the momentum would in certain cases build up speed to the point where it went straight past a local minima. As a result, the Nesterov Momentum implemented a smarter approach where the network first takes a big step based on the previous momentum, then calculates the gradient and if necessary makes corrections, before it updates the parameters.

5.1.2 Model Metrics and Evaluation

In order to evaluate our model and its output, we have used an array of state of the art evaluation methods for semantic segmentation methods, that is, segmentations where one predicts the class of each pixel in an image. We chose to use several different methods in order to get a broader picture of how our model was doing, and to give an increased value of our results.

For any classification problem where there are two or more classes, the combination of predictions and actual, ground truth values can be assigned into one of four categories; true positive (TP), true negative (TN), false positive (FP), and false negative (FN). That is in our case, as shown in Table 5.1,

	Actual Contour Pixel	**Actual Non-Contour Pixel**
Pred. Contour Pixel	True Positive (TP)	False Positive (FP)
Pred. Non-Contour Pixel	False Negative (FN)	True Negative (TN)

Table 5.1: General Confusion Matrix for our Model

for any pixel, one of the four outcomes can happen; the neural network classifies the pixel as (a) a pixel inside the contour, and it in fact is a contour pixel, resulting in a TP, (b) a pixel inside the contour perimeter, however it is not, resulting in a FP, (c) a pixel that is not within the contour perimeter, however it actually is, resulting in a FN, and lastly (d) a pixel that is not inside a contour, and it is not, resulting in a TN. These four categories are the basis for all the metrics used for evaluation.

5.1.3 Metrics

The metrics of a neural network are used to make quantifiable measurements of the performance of the network. These metrics are then often used as part of evaluation methods. We have chosen to look at the following metrics when assessing our model:

Accuracy (ACC) is simply the percentage of correct guesses (TP and TN) that the model predicted. While this is a typical metric that is used for most models, it is often deemed too generic in order to give accurate feedback as to how well the model is doing. The way we calculate the accuracy can be seen in equation 5.3.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.3}$$

Precision (PREC) is a metric quantifying the percentage of the predicted classes that are relevant.

$$PREC = \frac{TP}{TP + FP} \tag{5.4}$$

Recall (REC) also known as sensitivity, is closely related to the precision. On its own it is the percentage of correctly predicted relevant results.

$$REC = \frac{TP}{TP + FN} \tag{5.5}$$

5.1.4 Evaluating and Comparing Models

When evaluating our model, we mainly used two different methods that is frequently applied to semantic segmentation problems; *Intersection over Union (IoU)* and *Dice*. That being said, due to the visual nature of our contour classification problem, we often employed a more informal first step. That is, in most cases our first evaluation step was to load the trained model and show the predicted output for a few, randomly selected inputs. Here, we simply observed the output of these examples from the model as an image, in order to give a rough evaluation of whether the model was producing something useful. A practical example of this can be seen in Section 5.2.1. If the model passed this step, we then applied IoU and Dice, that we will now describe.

Intersection over Union (IoU)

The Intersection over Union (IoU) metric, also known as the Jaccard index, is a metric that quantifies how much overlap there is between the ground truth and the prediction of a segmentation. The Jaccard index was introduced in a paper by Paul Jaccard in 1901 [69], and has since become a frequently used quantification metric in semantic segmentation.

$$IoU = \frac{target \cap prediction}{target \cup prediction} = \frac{TP}{(TP + FP + FN)} \tag{5.6}$$

As we see in 5.6, it is calculated by taking the TP values of the prediction, that is the pixel values shared by the prediction and the ground truth, and then dividing it by all the pixels present in both the prediction and the ground truth, in other words the union. This calculation, essentially, finds the percentage of overlapping pixels between the prediction and the ground truth. In order to assess an entire model, you calculate the IoU value for each class, and then average over all classes to obtain one mean IoU score.

Dice

The Dice score, also known as the Sørensen-Dice coefficient, was introduced and developed by Thorvald Sørensen and Lee Raymond Dice, in two independent papers published in 1945 and 1948, respectively. [70], [71]

$$Dice = \frac{2 \times TP}{(TP + FP) + (TP + FN)} \tag{5.7}$$

While the explanation of this metric is not as straightforward as with IoU, they can be expressed as variants of each other using common algebraic methods, as follows:

$$IoU = \frac{Dice}{2 - Dice} \tag{5.8}$$

$$Dice = \frac{2 \times IoU}{1 + IoU} \tag{5.9}$$

As a result, they are closely related and largely measure the same aspects, however IoU is always larger than Dice except from at the extrema [72], and Dice does not satisfy the triangle inequality, meanwhile IoU does. Therefore Dice is considered a semimetric, rather then a full metric. [73] All in all, the complexity of these metrics is a study of their own, however they are both frequently used as metrics for semantic segmentation problems.

5.2 Experiment 1: Automatic Generation of Contour

In Part II Method, Section 1.4, we covered the steps of how we arrived at the different results in the different experiment iterations, and in the previous sections of this chapter we covered the metrics and methods we used to evaluate the experiments. Now, we will present each experiment with their respective, deep learning configurations, and then we will display and discuss the results and discuss them in order. For a recap of the environment of the training, please refer to Section 1.2 in Part II.

5.2.1 First Iteration

As reported in the methodology chapter, our initial iteration was a ran through both the VGG16 pretrained model and the ResNet50 model, with both endocardium- and epicardium expert contours as input to the network.

Hyperparameter	**Configuration**
Pretrained Architecture	VGG16, ResNet50
Train/Test Ratio	80%/20%
Batch Size	2
Epochs	100
Optimiser	Nadam
Loss Function	MAE
Metrics	ACC, MAE

Table 5.2: Model configurations for first iteration

Results and Discussion

After we had finished letting the model train for 100 epochs, we loaded the predicted output of the model, converted the output array back into an image and was met with the result of the predicted output, shown in Figure 5.2. Despite being difficult to see, there is one pixel activated in the far right of the image, a little under halfway up. We tried making changes to the most central hyperparameters and the loss function, however despite our best efforts to improve this, the result did not improve. In fact, on the contrary, as at times we would get completely black images as predicted output.

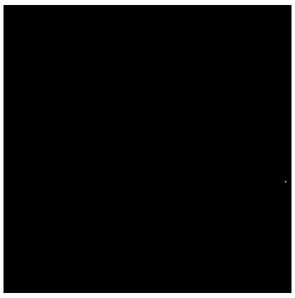


Figure 5.2: Example showing the poor contour prediction from the first deep learning iteration.

Given our results, we quickly deemed our first approach as non-viable. Our best theory as to why it failed, is because the network quickly realised that it could turn all pixels off and still have a considerably good accuracy and error, due to the uneven balance of black pixels compared to white pixels. Thereafter, it would presumably take extreme amounts of computations and time in order to find each pixel in the ground truth hollow circle. At this point, chances are that the vanishing gradient problem would start to occur, in that the gradients would shrink to values where they essentially had no impact anymore. Regardless, we did not dwell on this for long, and rather moved on to the approach we believe would fix our issue, that is, filling the contours.

5.2.2 Second Iteration

In our second iteration, we kept most of the hyperparameters and metrics the same, in order to investigate whether the actual issue in fact was the input data, and not some other factor. For the sake of being more confident that the network would have enough time to compute the errors, we did increase the number of epochs to 500. After running the full training session, we again loaded the prediction and found the results shown in Figure 5.3

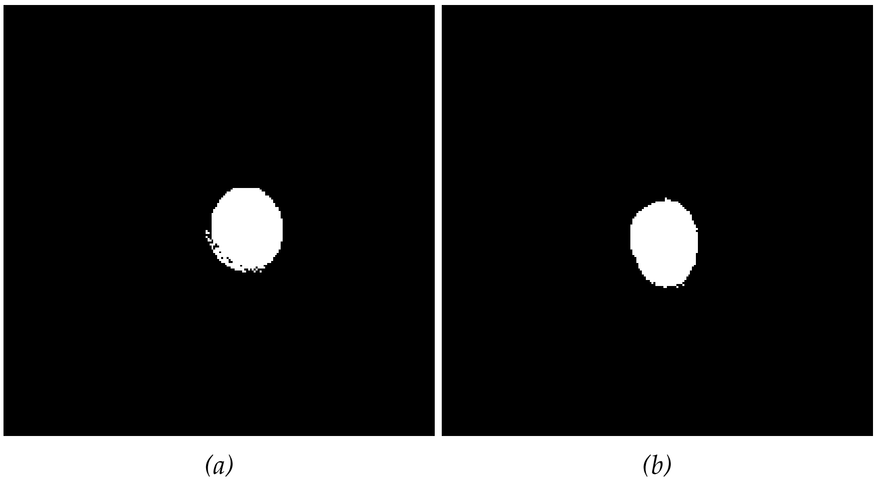

(a) *(b)*

Figure 5.3: Output predictions from our second iteration, showing examples of a (a) training set prediction and a (b) test set prediction. Note that these predictions are for two different MRI DICOM images

The results had evidently increased drastically. Despite the rough edges that would have be to addressed in post-processing, these results showed that our approach was viable after all. Even so, the rough edges were presumably due to the fact that we were training on both epicardium and endocardium at the same time. In different terms, it is likely that the model got confused, and ended up trying to compromise somewhere between the two, and thus the rough edges appeared. It follows that our next attempt to improve our accuracy was to separate endocardium and epicardium. Therefore, our next idea was to separate the endocardium- and epicardium segmentations into two separate datasets, and run them on different models to see how well the different models would perform.

Method	DICE	IoU	Precision	Recall	Accuracy
Endocardium (Left Ventricle)					
ResNet 50	0.701	0.560	0.606	0.848	0.985
VGG 16	0.676	0.526	0.652	0.550	0.982
VGG 19	0.765	0.638	0.557	0.875	0.985
DenseNet 169	0.698	0.556	0.617	0.817	0.985
InceptionResnetV2	0.702	0.583	0.718	0.900	0.984
Epicardium (Left Ventricle)					
ResNet 50	0.808	0.684	0.825	0.799	0.988
VGG 16	0.767	0.637	0.780	0.761	0.985
VGG 19	0.780	0.655	0.759	0.811	0.985
DenseNet 169	0.792	0.666	0.815	0.780	0.987
InceptionResnetV2	0.809	0.686	0.820	0.809	0.988

Table 5.3: A table showing the evaluation results of the experiments done to automatically determine the inner and outer left ventricle.

5.2.3 Further Experimentation with ResNet50, VGG16, VGG19, and InceptionResNetV2

After having separated our dataset based on endocardium and epicardium, we started running experiments on all the models covered in Section 4.2.2. As presented in Table 5.4, we observe that the all the pre-trained architectures in our model generally is more accurate when predicting the outer heart wall (epicardium), than when predicting the inner heart wall (endocardium). Additionally, that VGG 19 overall performs the best on the inner heart wall (endocardium), meanwhile InceptionResNetV2 is the performs the best on epicardium.

Results and Discussion

However, even though the InceptionResNetV2 model was the most successful, Figure 5.4 shows that it is by no means perfect yet. While these predictions were handpicked as some of the worst the model created, these results would still be unacceptable within any medical practice, be that training or diagnostic. This is a contributing factor as to why we declare that the model is a promising start, but still needs to be improved.

Conversely, the model also showed some very accurate predictions. As shown in Figure 5.5, the predicted and true contour are nearly identical, which shows that the model is learning something. As for any deep neural network training, it is severely difficult to say for certain why some images are harder to classify than others. However, one feature that stands out in

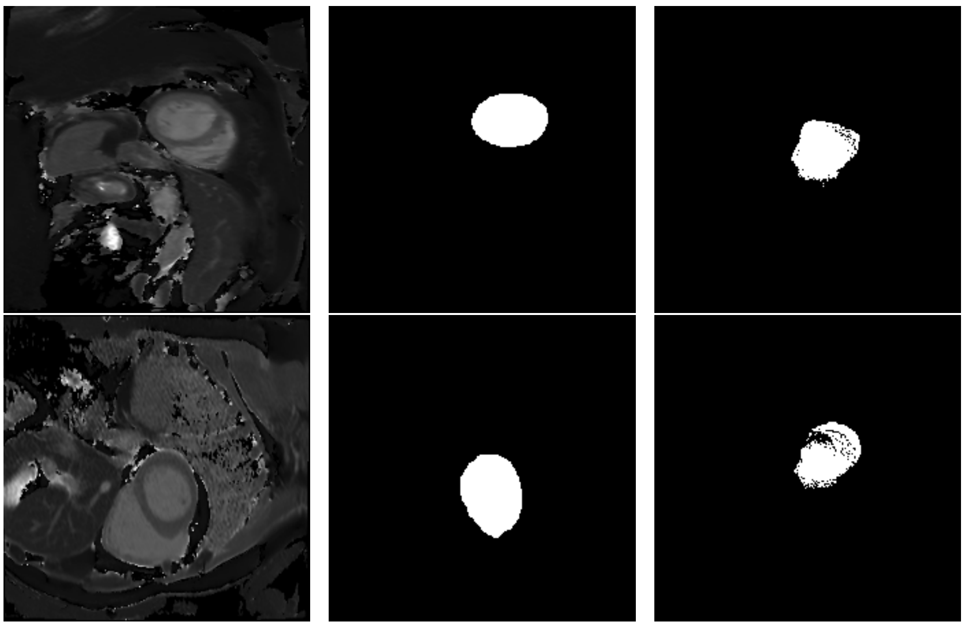


Figure 5.4: Examples of where the InceptionResNetV2 performed poorly, first column containing the MRI T1 mapping images, second column containing the human expert contours, and third column containing the model predictions

both of these examples, are that when the left- and right ventricle are surrounded by darker tissue, the model has an easier time to classify them, and vice versa. An obvious reason for this is that it is easier to distinguish contrasts that are far apart on the monochrome spectrum.

That being said, the majority of the predicted contours land somewhere inbetween these two extremes, where there are smaller errors that would have to be improved. Because of this, we consider this approach to be overall fairly successful.

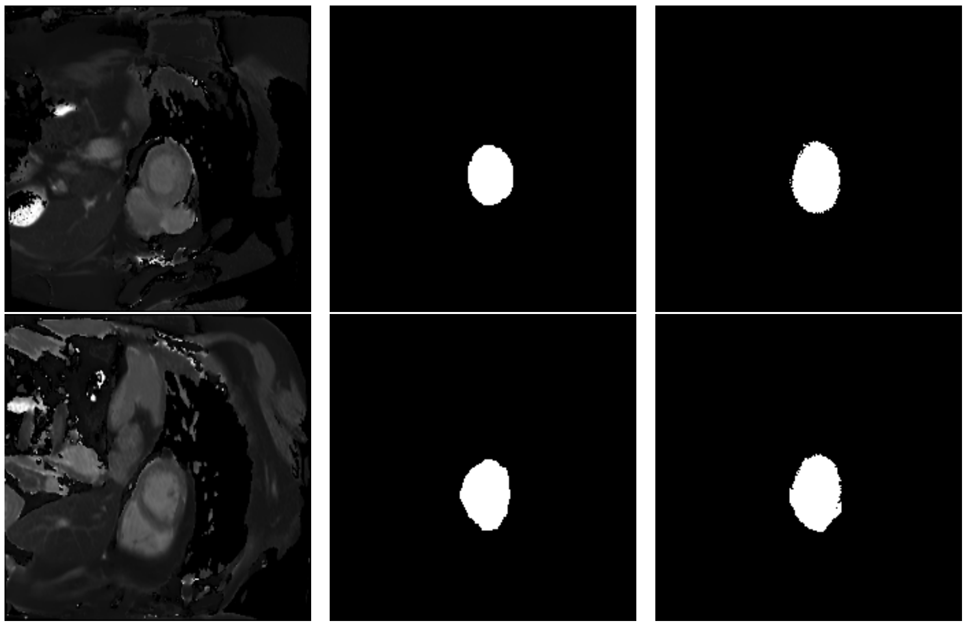


Figure 5.5: Examples of where the InceptionResNetV2 performed well, first column containing the MRI T1 mapping images, second column containing the human expert contours, and third column containing the model predictions

5.3 Experiment 2: Generate Feedback from Trainee Contour

We decided to also explore the supervised learning problem of generating expert contours from a DICOM image *and* a trainee contour. The idea was that based on the trainee contour, the network could train the model to attempt to make an "expert" contour, and then compare it to actual expert contours during validation. Our proposed network used to generate these contours is a modified version of the VGG16 [58] model, implemented in Keras. Also here, we ran the model over the same popular CNN architectures as in our last supervised learning problem, such as architectures based on ResNet, Inception, and DenseNet [74].

The preprocessing and configuration of this network is also similar, however the input the model expects has changed. Instead, what the model expects as an input is an image consisting of four channels. The first three channels consists of the R (red), G (green), and B (blue) colour channels of the extracted DICOM MRI image, like before. However, the fourth image channel is represented by the trainee contour. Like in our other model, the input image with all four channels is then resized into the size of $224 \times 224 \times 4$, before being passed into the model. The model then outputs a vector with the size of 50176 (224×224), which represents the "expert" segmentation. Each value in the vector still represents one pixel of the output segmentation, and each pixel can be either "on" or "off".

For training, we used a variety of different architectures, however this time we trained it from scratch. We replace the classification block of the original model, with a custom block consisting of one 2D global average layer, a normalisation layer which squashes the input values between 0 to 1, then a final fully-connected layer consisting of 50176 nodes (one for each pixel). The model then is trained using MAE to calculate loss and Nadam [64] to optimise the weights with a learning rate of 0.004. The output of the network was then compared towards the expert contour from our dataset, in order to let the network average an "expert" consensus. Additionally, to ensure robust results, each experiment was run using 3-fold cross-validation. A diagram explaining the entire training process can be seen in Figure 5.6.

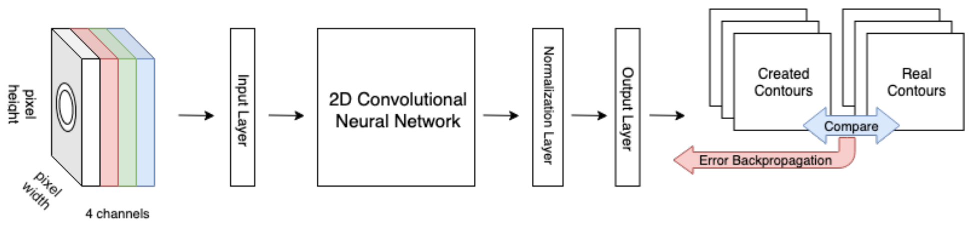


Figure 5.6: A diagram showing the entire process for training the trainee to expert segmentation model.

Method	DICE	IoU	Precision	Recall	Accuracy
Endocardium (Left Ventricle)					
ResNet 50	0.794	0.663	0.680	0.967	0.989
VGG 16	0.737	0.587	0.620	0.922	0.986
VGG 19	0.763	0.623	0.642	0.965	0.987
DenseNet 169	0.786	0.652	0.667	0.975	0.989
InceptionResnetV2	0.789	0.659	0.675	0.969	0.989
Epicardium (Left Ventricle)					
ResNet 50	0.888	0.803	0.896	0.888	0.992
VGG 16	0.771	0.629	0.658	0.942	0.988
VGG 19	0.782	0.646	0.664	0.967	0.988
DenseNet 169	0.890	0.666	0.815	0.780	0.987
InceptionResnetV2	0.727	0.586	0.639	0.884	0.984

Table 5.4: A table showing the evaluation results of the experiments done to automatically determine the inner and outer left ventricle.

While the contours that were good for the previous section did not see any major improvements following this approach, many of the bad predicted segmentations now looked a lot better, as shown in Figure 5.7. This is not too surprising as this new approach includes information taken from the student contours who although are in training, do have some knowledge about the segmentation of the left ventricle. Additionally, as we see in

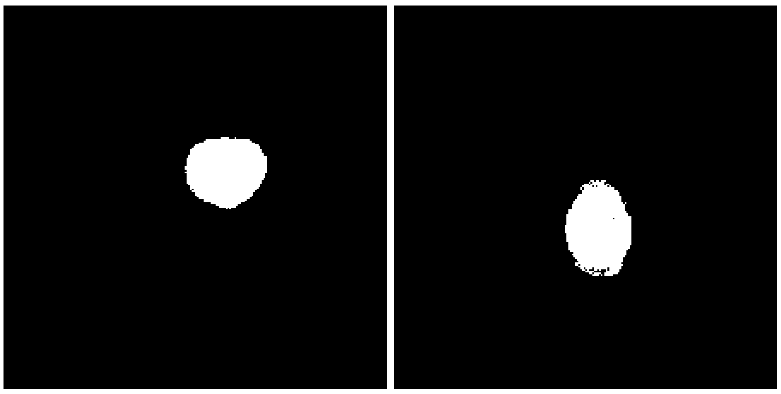


Figure 5.7: Figure showing the that predictions that were originally bad in Figure 5.4, has now improved considerably, where the left image corresponds to the prediction in the first row, and the right image corresponds to the prediction in the first column

Table 5.4, the metrics are approximately 0.10 higher on average than in our other model. Following this, we believe that the model with both a DICOM and a contour as input might be the superior approach, as indicated by the results presented here. However, a proof of this theory are left to the future work.

5.4 Summary

In this chapter we have presented the experiments carried out on our deep convolutional neural network architecture and models. We have demonstrated that the models are already generating promising results, and we believe it is a possibility that the model can be applied for automatic feedback in the training purposes in the future. However, because our data is limited, and we do not have the resources nor time to carry out a qualitative study, we are unable to draw any final conclusions on whether the prior hypothesis is true. Instead, we suggest the following step of these experiments are to conduct a qualitative study to really make a conclusion on this for your approach, with the main question; Do the trainees get better due to the automatic feedback or not?

Part IV

Conclusion and Future Work

Chapter 6

Summary and Conclusion

As discussed, in recent times, the effort to make a more standardised training process within the medical image analysis field has be steadily increasing. Following this, the field welcomes ideas and tools to aid this standardisation. After all, the training of medical analysts can have critical consequences for clinical cardiovascular diagnosis. In order to aid this movement, we have investigated both a software system and a deep learning architecture throughout this thesis, in order to provide tools to quantify and standardise training, and improve the training process as a whole. This lead us to the following research questions:

- ***How can we design a tool to support training in T1 mapping analysis based on a digitised approach?***
- ***Can CNNs create automatic feedback based on non-expert annotations?***

On the quest to answer the research questions, we developed a software system called Fabulinus. It is a software tool meant to aid trainees in training of contouring of the endocardium and epicardium of the left ventricle of the heart, currently focusing only on images from T1 mapping MRI. The system automatically evaluates the trainees segmentation by comparing it against both expert contours as well as deep CNN generated "expert" contours. With the feedback from the deep CNN, it allows the system to provide immediate feedback on the trainees submitted segmentation, and has the potential to greatly reduce the need of manually contoured examples from experts in the future.

More specifically, regarding the first problem statement, thanks to our cooperation wtih OCMR, we have found several problems that can be supported by a digitised approach. In particular, we have aimed to offload the expert teacher by enabling the trainees compare their own analyses before the one-on-one feedback. Conclusively, we have created a software with many features that we know are needed in the training process. While it cannot be used in a training environment yet, we have also received positive feedback from various experts at OCMR, that the direction that Fabulinus is headed is very promising. That being said, there still has to be conducted rigorous validation studies, either qualitative and quantitative, to provide absolute results.

For the second problem statement, our work strongly supports and strengthens this question as to something that is very much possible. The CNN automatically generates contours that on occasion are very similar to the human expert contours. Additionally, our results suggest that our specliased approach using both trainee contours *and* the DICOM image as input could be a superior approach to generating "expert" contours for feedback, which we consider a strong point of the research conducted in this thesis. However, in order to be certain, also here validation studies such as qualitative studies need to be performed, and the models should be applied on other datasets in order to verify the generalisability of the approach.

6.1 Future work

While the possible directions the work conducted in this thesis could go are numerous, we have narrowed it down to some main points that we consider to be most useful for the system at this stage. Although, this is by no means an extensive list, and as with all research, we encourage anyone who want to take this idea further to do so, regardless of whether it follows the following suggestions.

6.1.1 Fabulinus

Through our work with Fabulinus we have already produced a good foundation. However, in order to quantify if it would be useful, there should be carried out a comparative study, including experts and trainees, in order to explore whether the software actually improves the results of the training, and if it is useful to the trainees.

We also suggest adding the following functionalities:

Expert Dashboard As briefly discussed in Section 3.1.1, the expert should have a dashboard with some additional functionalities. These include: an overview of all uploaded trainee analyses, and access to, and ability to edit or remove, all available expert segmentations and CNN segmentations. There should be a clear distinction between what expert segmentations are from human expert and which are CNN generated. Finally, it would also be useful to generate a report collecting data from all the available trainee analyses.

Prevent Operators Bias In order to prevent trainees from simply uploading their first contour into Fabulinus and instantly getting the "correct" results to base their next repetitions on, a restriction should be implemented that ensures that the trainee has done at least two repetitions of an MRI stack before they can compare their results.

Explore the Team Approach The future goal of this software should be to shift the focus from the status quo of training sessions, that are heavily reliant on one-to-one feedback. Instead, many in the field welcome a more team-based approach. For example, if the expert and trainees could cooperate over a system like Fabulinus, through their respective dashboards, the ability to transfer the focus to a more team-based approach would begin. Following this, the tool could even be extended to be used in teams, rather than only in training sessions, where the teams could use the dashboard collaboratively to compare and assess the differences in analyses in real time.

6.1.2 Deep Learning Architecture

When it comes to our deep learning architecture and model, the main next steps are to take our results and see if they can be generalised, especially with respects to different datasets. More importantly, modifications need to be made to our architecture in order to improve the quality of the output contours. In addition, a qualitative study should be carried out in order to properly determine whether the results are useful.

Qualitative Study As mentioned in Section 5.4, the next step for the deep learning architecture is to carry out a proper qualitative study. In brief, we recommend that the study should include both experts and trainees, where a group of trainees initially do a training session under one expert supervision, without the deep learning feedback available. Then, the quality of the training and time should be assessed in cooperation with 2-3 experts that did not take part in the training session. Following this, one should do an equivalent repetition with the deep learning feedback present on the same participants, and then assess the quality again and quantify the differences.

Run the Model on Other Datasets In order to generalise our model, it should also be run on different datasets, as our dataset is fairly small. For example, the datasets available in public segmentation competitions could be applied, such as the one mentioned in Section 2.7.6.

Generate Consensus Within Team Another model that could be useful to train, is one that takes a number of contours from image analysts and attempts to make an average between them to generate consensus within a team. This could potentially expand the number of people the model could be useful for outside of the training environment.

Bibliography

[1] W. H. Organization. (2017). Cardiovascular diseases (CVDs), [Online]. Available: https://www.who.int/en/news-room/fact-sheets/detai%20/cardiovascular-diseases-(cvds) (visited on 30/04/2019).

[2] H. S. Thomsen, P. Marckmann and V. B. Logager, 'Nephrogenic systemic fibrosis (NSF): A late adverse reaction to some of the gadolinium based contrast agents', *Cancer Imaging*, vol. 7, no. 1, p. 130, Sep. 2007. DOI: 10.1102/1470-7330.2007.0019. [Online]. Available: https://doi.org/10.1102/1470-7330.2007.0019.

[3] V. V. Valindria *et al.*, 'Reverse Classification Accuracy: Predicting segmentation performance in the absence of ground truth', *IEEE Transactions on Medical Imaging*, vol. 36, no. 8, pp. 1597–1606, Aug. 2017, ISSN: 1558-254X. DOI: 10.1109/TMI.2017.2665165. [Online]. Available: https://doi.org/10.1109/TMI.2017.2665165 (visited on 01/05/2018).

[4] E. Konukoglu, B. Glocker, D. H. Ye, A. Criminisi and K. M. Pohl, 'Discriminative segmentation-based evaluation through shape dissimilarity', *IEEE Transactions on Medical Imaging*, vol. 31, no. 12, pp. 2278–2289, Dec. 2012, ISSN: 1558-254X. DOI: 10.1109/TMI.2012.2216281. [Online]. Available: https://doi.org/10.1109/TMI.2012.2216281 (visited on 03/05/2018).

[5] D. R. Messroghli *et al.*, 'Clinical recommendations for cardiovascular magnetic resonance mapping of T1, T2, T2* and extracellular volume: A consensus statement by the Society for Cardiovascular Magnetic Resonance (SCMR) endorsed by the European Association for Cardiovascular Imaging (EACVI)', *Journal of Cardiovascular Magnetic Resonance*, vol. 19, no. 1, p. 75, Oct. 2017, ISSN: 1532-429X. DOI: 10.1186/s12968-017-0389-8. [Online]. Available: https://doi.org/10.1186/s12968-017-0389-8 (visited on 24/04/2019).

[6] Radcliffe Department of Medicine. (2019). About OCMR, [Online]. Available: https://www.rdm.ox.ac.uk/about/our-clinical-facili%20ies-and-mrc-units/oxford-centre-for-clinical-magnetic-resonan%20e-research/about-ocmr (visited on 30/04/2019).

[7] D. E. Comer, D. Gries, M. C. Mulder, A. Tucker, A. J. Turner and P. R. Young, 'Computing as a discipline', *Communications of the ACM*, vol. 32, no. 1, P. J. Denning, Ed., pp. 9–23, Jan. 1989, ISSN: 0001-0782.

DOI: 10.1145/63238.63239. [Online]. Available: http://doi.acm.org/10.1145/63238.63239.

[8] The National Heart, Lung, and Blood Institute. (n.d.). How the Heart Works - Anatomy of the Heart, [Online]. Available: https://www.nhlbi.nih.gov/node/3710 (visited on 27/04/2018).

[9] T. Taylor. (2018). Human Heart – Diagram and Anatomy of the Heart, Innerbody, [Online]. Available: http://www.innerbody.com/image/card01.html (visited on 27/04/2018).

[10] C.-J. Lin, C.-Y. Lin, C.-H. Chen, B. Zhou and C.-P. Chang, 'Partitioning the heart: Mechanisms of cardiac septation and valve development', *Development*, vol. 139, no. 18, pp. 3277–3299, Aug. 2012, ISSN: 1477-9129. DOI: 10.1242/dev.063495. [Online]. Available: https://doi.org/10.1242/dev.063495 (visited on 29/05/2018).

[11] A. Berger, 'Magnetic resonance imaging', *British Medical Journal*, vol. 324, no. 7328, p. 35, Jan. 2002, ISSN: 0959-8138. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1121941/ (visited on 29/04/2018).

[12] V. Carapella, 'Impact of tissue microstructure on a model of cardiac electromechanics based on MRI data', PhD thesis, Oxford University, UK, 2013. [Online]. Available: https://ora.ox.ac.uk/objects/uuid:69d28c8c-832b-4ac4-aa48-3d0613708515 (visited on 23/05/2019).

[13] C. Petitjean and J.-N. Dacher, 'A review of segmentation methods in short axis cardiac MR images', *Medical Image Analysis*, vol. 15, no. 2, pp. 169–184, Apr. 2011. DOI: 10.1016/j.media.2010.12.004. [Online]. Available: https://doi.org/10.1016/j.media.2010.12.004 (visited on 01/05/2018).

[14] R. J. v. d. Geest, 'Automated image analysis techniques for cardiovascular magnetic resonance imaging', PhD thesis, Leiden University, NL, 2011. [Online]. Available: https://openaccess.leidenuniv.nl/handle/1887/16643 (visited on 23/05/2019).

[15] D. T. Ginat, M. W. Fong, D. J. Tuttle, S. K. Hobbs and R. C. Vyas, 'Cardiac imaging: Part 1, MR pulse sequences, imaging planes, and basic anatomy', *American Journal of Roentgenology*, vol. 197, no. 4, pp. 808–815, Oct. 2011. DOI: 10.2214/AJR.10.7231. [Online]. Available: https://doi.org/10.2214/AJR.10.7231 (visited on 10/05/2018).

[16] S. E. Petersen *et al.*, 'UK Biobank's cardiovascular magnetic resonance protocol', *Journal of Cardiovascular Magnetic Resonance*, vol. 18, no. 1, p. 8, Feb. 2016. DOI: 10.1186/s12968-016-0227-4. [Online]. Available: https://doi.org/10.1186/s12968-016-0227-4 (visited on 01/05/2019).

[17] V. M. Ferreira *et al.*, 'Native T1-mapping detects the location, extent and patterns of acute myocarditis without the need for gadolinium contrast agents', *Journal of Cardiovascular Magnetic Resonance*, vol. 16, no. 1, p. 36, May 2014. DOI: 10.1186/1532-429X-16-36. [Online]. Available: https://doi.org/10.1186/1532-429X-16-36 (visited on 01/05/2019).

[18] A. J. Taylor, M. Salerno, R. Dharmakumar and M. Jerosch-Herold, 'T1 mapping: Basic techniques and clinical applications', *JACC: Cardiovascular Imaging*, vol. 9, no. 1, pp. 67–81, Nov. 2016. DOI: 10.1016/j.jcmg.2015.11.005. [Online]. Available: https://doi.org/10.1016/j.jcmg.2015.11.005 (visited on 02/05/2019).

[19] R. A. Pooley, 'Fundamental physics of MR imaging', *RadioGraphics*, vol. 25, no. 4, pp. 1087–1099, Jun. 2005. DOI: 10.1148/rg.254055027. [Online]. Available: https://doi.org/10.1148/rg.254055027 (visited on 02/05/2019).

[20] National Electrical Manufacturers Association. (2019). Digital Imaging and Communications in Medicine (DICOM), [Online]. Available: https://www.dicomstandard.org/ (visited on 22/04/2019).

[21] M. Larobina and L. Murino, 'Medical Image File Formats', *Journal of Digital Imaging*, vol. 27, no. 2, pp. 200–206, Apr. 2014, ISSN: 1618-727X. DOI: 10.1007/s10278-013-9657-9. [Online]. Available: https://doi.org/10.1007/s10278-013-9657-9 (visited on 14/05/2019).

[22] R. W. Cox *et al.* (2007). NIfTI-1 data format documentation, [Online]. Available: https://nifti.nimh.nih.gov/nifti-1/ (visited on 02/05/2019).

[23] R. E. Klabunde. (2015). Regulation of Stroke Volume, [Online]. Available: http://www.cvphysiology.com/Cardiac%20Function/CF002 (visited on 27/04/2018).

[24] L. Agoston-Coldea and S. Lupu, 'Right chambers quantification in clinical practice: Echocardiography compared with cardiac magnetic resonance imaging', in *Hot Topics in Echocardiography*, IntechOpen, 2013. DOI: 10.5772/55832. [Online]. Available: https://doi.org/10.5772/55832 (visited on 23/05/2019).

[25] Circle Cardiovascular Imaging Inc. (2019). Cardiac MRI and CT Software, Circle Cardiovascular Imaging Inc., [Online]. Available: https://www.circlecvi.com/ (visited on 13/04/2019).

[26] Medis Medical Imaging Systems. (2019). QMass: Post-processing software, [Online]. Available: https://www.medis.nl/products/qmass (visited on 13/04/2019).

[27] C. T. Rueden *et al.*, 'ImageJ2: ImageJ for the next generation of scientific image data', *BMC Bioinformatics*, vol. 18, no. 1, p. 529, Nov. 2017. DOI: 10.1186/s12859-017-1934-z. [Online]. Available: https://doi.org/10.1186/s12859-017-1934-z (visited on 13/04/2019).

[28] D. Fagella. (Feb. 2019). What is Machine Learning? - An Informed Definition, TechEmergence, [Online]. Available: https://www.techemergence.com/what-is-machine-learning/ (visited on 22/05/2019).

[29] A. Ng. (n.d.). Machine learning, Stanford University, [Online]. Available: https://www.coursera.org/learn/machine-learning (visited on 27/04/2018).

[30] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, Inc., 1997, p. 2, ISBN: 0070428077.

[31] A. Karpathy *et al.* (2019). Neural network architectures in CS231n: Convolutional neural networks for visual recognition, Standford University, [Online]. Available: http://cs231n.github.io/neural-networks-1/ (visited on 01/05/2018).

[32] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[33] Y. LeCun, Y. Bengio and G. Hinton, 'Deep learning', *nature*, vol. 521, no. 7553, pp. 436–440, May 2015. DOI: 10.1038/nature14539. [Online]. Available: https://doi.org/10.1038/nature14539 (visited on 01/05/2018).

[34] H. Hu *et al.*, 'Automatic segmentation of the left ventricle in cardiac MRI using local binary fitting model and dynamic programming techniques', *PLOS ONE*, vol. 9, no. 12, pp. 1–17, Dec. 2014. DOI: 10.1371/journal.pone.0114760. [Online]. Available: https://doi.org/10.1371/journal.pone.0114760 (visited on 23/05/2019).

[35] M. A. Abdelfadeel, S. ElShehaby and M. S. Abougabal, 'Automatic segmentation of left ventricle in cardiac MRI using maximally stable extremal regions', in *2014 Cairo International Biomedical Engineering Conference (CIBEC)*, Dec. 2014, pp. 145–148. DOI: 10.1109/CIBEC.2014.7020940. [Online]. Available: https://doi.org/10.1109/CIBEC.2014.7020940 (visited on 23/05/2019).

[36] P. Radau, Y. Lu, K. Connelly, G. Paul, A. Dick and G. Wright, 'Evaluation framework for algorithms segmenting short axis cardiac MRI', *The MIDAS Journal: Cardiac MR Left Ventricle Segmentation Challenge*, vol. 49, Jul. 2009. [Online]. Available: http://hdl.handle.net/10380/3070 (visited on 23/05/2019).

[37] M. Zreik, T. Leiner, B. D. de Vos, R. W. van Hamersvelt, M. A. Viergever and I. Išgum, 'Automatic segmentation of the left ventricle in cardiac ct angiography using convolutional neural networks', in *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, Apr. 2016, pp. 40–43. DOI: 10.1109/ISBI.2016.7493206. [Online]. Available: https://doi.org/10.1109/ISBI.2016.7493206 (visited on 23/05/2019).

[38] Node.js Foundation, *Node.js*, [Computer Software], 2019. [Online]. Available: https://nodejs.org/en/.

[39] T. Christie and Django REST Framework contributors, *Django REST framework (Version 3.9)*, [Computer Software], 2019. [Online]. Available: https://www.django-rest-framework.org/.

[40] Django Software Foundation, *Django (Version 1.5)*, [Computer Software], 2013. [Online]. Available: https://djangoproject.com/.

[41] Adobe, *Adobe XD (Version 15.0)*, [Computer Software], 2019. [Online]. Available: https://www.adobe.com/no/products/xd.html.

[42] Django Software Foundation, *PostgreSQL*, [Computer Software], 2019. [Online]. Available: https://www.postgresql.org/.

[43] Facebook, *React*, [Computer Software], 2019. [Online]. Available: https://reactjs.org/.

[44] ——, *Create React App*, [Computer Software], 2019. [Online]. Available: https://facebook.github.io/create-react-app/.

[45] T. Koppers, S. Larkin, J. Ewald, J. Vepsäläinen, K. Kluskens and Webpack contributors, *webpack*, [Computer Software], 2019. [Online]. Available: https://webpack.js.org/.

[46] S. McKenzie and Babel contributors, *Babel*, [Computer Software], 2019. [Online]. Available: https://babeljs.io/.

[47] S. K. Piechnik *et al.*, 'Shortened Modified Look-Locker Inversion recovery (ShMOLLI) for clinical myocardial T1-mapping at 1.5 and 3 T within a 9 heartbeat breathhold', *Journal of Cardiovascular Magnetic Resonance*, vol. 12, no. 1, p. 69, Nov. 2010, ISSN: 1532-429X. DOI: 10.1186/1532-429X-12-69. [Online]. Available: https://doi.org/10.1186/1532-429X-12-69 (visited on 24/04/2019).

[48] D. Mason and pydicom contributors. (2018). pydicom documentation, [Online]. Available: https://pydicom.github.io/pydicom/stable/index.html (visited on 16/04/2019).

[49] Harris Geospatial Solutions, Inc. (2018). IDL - Extract Meaningful Visualizations From Complex Numerical Data, [Online]. Available: https://www.harrisgeospatial.com/Software-Technology/IDL (visited on 16/04/2019).

[50] Anaconda, Inc, *Anaconda Software Distribution*, [Computer Software], 2019. [Online]. Available: https://www.anaconda.com/ (visited on 19/04/2019).

[51] M. Abadi *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous distributed systems*, Software available from tensorflow.org, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467 (visited on 19/04/2019).

[52] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[53] F. Seide and A. Agarwal, 'CNTK: Microsoft's open-source deep-learning toolkit', Aug. 2016, pp. 2135–2135. DOI: 10.1145/2939672.2945397. [Online]. Available: http://doi.acm.org/10.1145/2939672.2945397 (visited on 02/05/2019).

[54] R. Al-Rfou *et al.*, 'Theano: A Python framework for fast computation of mathematical expressions', *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688 (visited on 02/05/2019).

[55] J. Hale. (Sep. 2018). Deep learning framework power scores 2018, [Online]. Available: https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a (visited on 04/05/2019).

[56] F. Chollet *et al.* (2019). Applications, [Online]. Available: https://keras.io/applications/ (visited on 16/04/2019).

[57] O. Russakovsky *et al.*, 'ImageNet Large Scale Visual Recognition Challenge', *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, Apr. 2015. DOI: 10.1007/s11263-015-0816-y. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y (visited on 16/04/2019).

[58] K. Simonyan and A. Zisserman, 'Very deep convolutional networks for large-scale image recognition', *arXiv preprint arXiv:1409.1556*, Apr. 2014. [Online]. Available: https://arxiv.org/abs/1409.1556 (visited on 08/05/2019).

[59] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition', *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385 (visited on 12/05/2019).

[60] C. Szegedy *et al.*, 'Going deeper with convolutions', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html (visited on 12/05/2019).

[61] R. K. Srivastava, K. Greff and J. Schmidhuber, 'Highway networks', *CoRR*, vol. abs/1505.00387, May 2015. arXiv: 1505.00387. [Online]. Available: http://arxiv.org/abs/1505.00387 (visited on 12/05/2019).

[62] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'ImageNet classification with deep convolutional neural networks', in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 24/04/2019).

[63] F. Chollet *et al.* (2015). Writing your own Keras layers, [Online]. Available: https://keras.io/layers/writing-your-own-keras-layers/ (visited on 16/04/2019).

[64] T. Dozat, 'Incorporating Nesterov Momentum into Adam.', 2016. [Online]. Available: http://cs229.stanford.edu/proj2015/054_report.pdf (visited on 13/05/2019).

[65] I. Sutskever, J. Martens, G. Dahl and G. Hinton, 'On the importance of initialization and momentum in deep learning', in *International conference on machine learning*, 2013, pp. 1139–1147. [Online]. Available: http://www.cs.toronto.edu/~fritz/absps/momentum.pdf (visited on 13/05/2019).

[66] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980*, Dec. 2014. [Online]. Available: https://arxiv.org/abs/1412.6980v8 (visited on 13/05/2019).

[67] S. J. Reddi, S. Kale and S. Kumar, 'On the convergence of Adam and beyond', *arXiv preprint arXiv:1904.09237*, Apr. 2019. [Online]. Available: https://arxiv.org/abs/1904.09237 (visited on 13/05/2019).

[68] Y. Nesterov, 'A method of solving a convex programming problem with convergence rate $O(1/k^2)$', *Soviet Mathematics Doklady*, vol. 27, no. 2, pp. 372–376, 1983.

[69] P. Jaccard, 'Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines', *Bulletin de la Société vaudoise des sciences naturelles*, vol. 37, pp. 241–272, 1901.

[70] L. R. Dice, 'Measures of the amount of ecologic association between species', *Ecology*, vol. 26, no. 3, pp. 297–302, Jul. 1945. DOI: 10.2307/1932409. [Online]. Available: https://doi.org/10.2307/1932409 (visited on 13/05/2019).

[71] T. Sørensen, 'A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons', *Biologiske Skrifter*, vol. 5, pp. 1–34, 1948.

[72] A. A. Taha and A. Hanbury, 'Metrics for evaluating 3D medical image segmentation: Analysis, selection, and tool', *BMC Medical Imaging*, vol. 15, no. 1, p. 29, Jul. 2015. DOI: s12880-015-0068-x. [Online]. Available: https://doi.org/10.1186/s12880-015-0068-x (visited on 13/05/2019).

[73] E. Gallagher, 'Compah documentation', *User's Guide and application published at: http://www. es. umb. edu/edgwebp. htm*, Apr. 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.1334&rep=rep1&type=pdf (visited on 13/05/2019).

[74] G. Huang, Z. Liu, L. v. d. Maaten and K. Q. Weinberger, 'Densely connected convolutional networks', in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 2261–2269. DOI: 10.1109/CVPR.2017.243. [Online]. Available: https://doi.org/10.1109/CVPR.2017.243 (visited on 23/05/2019).

Appendices

Appendix A

Useful Terminology

The terms related to the background material of this project are defined as they are introduced in the following chapter. I also use the following terms, but they are not defined in the chapter.

Image Analyst Anyone who carries out cine-MRI scans, analyses the MRI images and contours them. These include but are not limited to clinicians, radiographers, image analysts and experts in for example computational modelling.

Trainee An observer in training.

Corelab A specialised unit within a clinical research team that works on various aspects of research support, including data analysis, software testing, training in image analysis, quality assessment of clinical datasets and ad-hoc software development.

Appendix B

Code Examples

B.1 Debatable DICOM handling

```python
import pydicom
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from scipy import io as sp_io

def store_dicom_image(test_dcm, endo, epi):
    ref_ds = pydicom.read_file(test_dcm)
    print(ref_ds.pixel_array.shape, ref_ds.pixel_array.dtype)
    print(endo.shape, endo.dtype)
    print(epi.shape, epi.dtype)

    reversed_endo = endo[::-1]
    reversed_epi = epi[::-1]

    fig, ax = plt.subplots()
    # pixel_array is the image
    ax.imshow(ref_ds.pixel_array[::-1],
              cmap='gray')
    ax.plot(reversed_endo[:, 0], reversed_endo[:,1], 'r')
    ax.plot(reversed_epi[:, 0], reversed_epi[:,1], 'b')
    plt.gca().invert_yaxis()

    # plt.show()
    # All of the below is needed to completely remove whitespace, borders etc.
    # around the graph when stored as an image
    plt.gca().set_axis_off()
    fig.subplots_adjust(top = 1,
                        bottom = 0,
                        right = 1,
                        left = 0,
                        hspace = 0,
                        wspace = 0)
    plt.margins(0,0)
    plt.gca().xaxis.set_major_locator(ticker.NullLocator())
    plt.gca().yaxis.set_major_locator(ticker.NullLocator())

    plt.savefig("modified_img.png",
            pad_inches=0,
            bbox_inches='tight'
        )
```

Listing 7: Lengthy code extraction of DICOM using matplotlib

Appendix C

Papers

C.1 A Web-Based Software for Training and Quality Assessment in the Image Analysis Workflow for Cardiac T1 Mapping MRI

A Web-Based Software for Training and Quality Assessment in the Image Analysis Workflow for Cardiac T1 Mapping MRI

Edvarda Eriksen[*][†], Steven A. Hicks[‡][§], Michael A Riegler[‡], Pål Halvorsen[‡][§], and Valentina Carapella[¶]
[*]Simula Research Laboratory, Norway
[†]University of Oslo, Norway
[‡]SimulaMet, Norway
[§]Oslo Metropolitan University, Norway
[¶]King's College London, United Kingdom
Contact email: edvarda.eriksen@outlook.com

***Abstract*—Medical practice makes significant use of imaging scans such as Ultrasound or Magnetic Resonance Imaging (MRI) as a diagnostic tool. They are employed in visual inspection or quantification of medical parameters computed from the images in post-processing. However, the value of such parameters depends greatly on user's variability, device and algorithmic differences. In this paper we focus on quantifying the variability due to the human factor, which can be largely addressed by structured training of the human operator. We focus on a specific emerging cardiovascular MRI methodology, the T1 mapping, that has proven useful to identify a range of pathological alterations of the myocardial tissue structure. Training, especially in emerging techniques, is typically not standardized, varying dramatically across medical centres and research teams. Additionally, training assessment is largely based on qualitative approaches. The aim of our work is to provide a software tool combining traditional clinical metrics and convolutional neural networks (CNNs) to aid the training process by gathering contours from multiple trainees, quantifying discrepancy from local gold standard or standardized guidelines, classifying trainees output based on critical parameters that affect contours variability.**

***Index Terms*—Cardiovascular MRI, T1 mapping MRI, quality assessment, deep learning, image analysis training, standardisation.**

I. Introduction

Training medical personnel in data analysis related work is a very important task. Especially due to the fact that machine learning is playing an increasing role withing the medical field [1]. Work related to this task ranges from analyzing image or senors data using computer aided diagnosis tools to simple tasks such as image annotation or segmentation that allows for training of new and better algorithms. Due to constant lack of experts (doctors) in the medical area, it is often very hard to get these tasks done [2]. Therefore, an important milestone is to train new personnel efficiently. Using automatic methods to support training of medical experts is a rather neglected field. There does not exist much work on how machine learning could be used to make the training of junior doctors more efficient. In this work, we are proposing a system that is targeting this problem. As a use case, we chose the very common T1 mapping in cardiac Magnetic Resonance Imagings (MRIs) which required a large amount of student training effort.

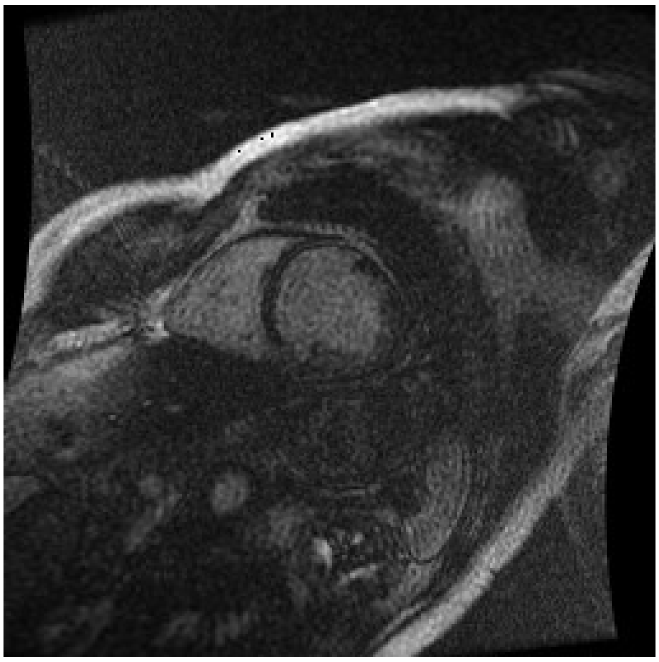


Fig. 1: Example MRI taken from the Sunnybrook Cardiac Data (SCD) [3] dataset. Note that the presented image has been resized for the purpose of this figure.

T1 mapping in cardiac MRIs is a rapidly emerging technique in the clinical setting to identify microstructural defects associated with a range of cardiovascular disease [4]. It achieves this by quantifying the spin-lattice relaxation time (T1) of protons in the water molecules of the biological tissue. This is measured by hitting the protons with a radio frequency signal and seeing how long it takes (in milliseconds) for the protons to return to their original state during an MRI procedure. T1 times are determined by the proportion of water content in tissue, in addition to its compartmentalization, so that each tissue type (such as blood or fat) show a range of different T1 values. These ranges, however, change drastically at different magnetic field strengths and may also depend on the specific MRI sequence used to measure them [5]. T1 mapping is the process of mapping T1 times to the individual pixels for a given MRI. These mappings are used to visualize the MRI image to easily distinguish between the different types of tissue including blood, fat and muscle. Standardization of image acquisition, post-processing and interpretation is crucial to ensure the consistency and reproducibility of any medical image analysis process, as well as guaranteeing a common ground for clinical assessment. Clinical use of cardiovascular T1 mapping is no exception, and a standardization task force has already provided the initial guidelines [6].

A T1 mapping of cardiac MRIs is a process which requires a segmentation of the inner and outer left ventricle of the heart (Figure 1 shows an example MRI image of the left ventricle). These segmentations are used to calculate the wall thickness (WT) and the T1 value, which both set the basis for detecting a range of abnormalities. Placing these segmentations requires trained observers who understand how to interpret and analyze a cardiac MRI. Observers are commonly trained through a training program consisting of segmentation tasks on an expert segmented training dataset, which is made up of segmentations made by a consensus of multiple expert observers. However, labeled datasets are limited, and evaluating each students segmentations is time consuming and lack the immediate feedback needed for rapid improvement. Therefore, we propose a system which aids in the training of new observers. For the system to be truly useful, it needs to incorporate the following requirements:

1) Produce a standardized report of the training progress and completion for accreditation for an individual trainee, or team of trainees.
2) Include a centralized approach to compare the performance of a single trainee, as well as a team of students, against the accepted reference for a training dataset.
3) Automatic generation of average contours either from expert analysis (typically called the *consensus*) or the average of team results.

With these requirements in mind, we present a software tool which aids the training of new observers by automatically evaluating their work by comparing against an automatically generated "expert" consensus using deep convolutional neural networks (CNNs).

The novelty of our software is two-fold. First, it focuses on the evaluation of trainees working on T1 mappings in cardiac MRIs, not just from an individual point of view, but also from the viewpoint of multiple operators (who typically work in a clinical research unit). This combination of individual and team evaluation stems from the fundamental need for consistency in the performed analysis, not just with respect to general guidelines, but also within a clinical unit. Second, the expert observer segmentations used to evaluate a trainees (or group) segmentation is automatically generated by a deep CNN. This CNN is modeled and trained to imitate the knowledge of several expert observers. By using pre-existing training datasets, we teach the model to estimate an outline of the outer (epicardium) and inner (endocardium) wall of the myocardium for a given MRI together with a trainees contour. The here presented approach allows for instant feedback to the trainees without having an expert annotating new MRI images since the CNN can get the expert feedback from the not perfect student annotation. Furthermore, another advantage is that a training facility would not have to rely on expert contoured MRI datasets.

In the process of analyzing T1 mappings in cardiac MRIs, two measurements are typically recorded; The average myocardial T1 value measured in milliseconds, and the average myocardial WT measured in pixels or millimetres. The value of T1 is the most clinically relevant in these types of scans, whereas WT is mostly employed as a quality assessment metric, its value being loosely correlated with the average T1. Regarding the underlying research carried out alongside the software development, we present a combination of traditional metrics of evaluation (such as average myocardial T1 and WT) that are easily understandable to a clinical audience.

The rest of this paper is structured as follows. In Section II, we give a brief look at some previous works done in this field. This section mostly focuses on work done for the automatic segmentation of the left ventricular myocardial ring. Section III further details how the developed software may support the cardiovascular magnetic resonance imaging (CMR) community as a whole, and the main contributions we aim to deliver with this research. Section IV gives a description of our developed software, showing how it may be used for the training of new observers. Then, in Section V, we look at the architecture and implementation details of this software, and discuss in detail how it works. Section VI shows the experiments performed for the underlying "expert" observation CNN model used to evaluated trainee segmentations. Lastly, Section VII concludes this paper with a summary of our findings and a brief discussion on future work.

II. Related Works

Our software aims to aid students in training to become expert observers of T1 mappings in cardiac MRIs. It does this by providing immediate feedback on the students performance and providing a progress report for individual and classes of students. Image post-processing of T1 mapping in MRIs requires the delineation of a region of interest (ROI), that is, placing contours. The most common case is that of the left ventricular myocardial ring. Depending on the software, contours can be set manually or (semi)-automatically. However, manual inspection should be carried out to ensure that the contours do not include regions outside of the myocardium (such as the outer pericardial fat or inner blood pool). More thorough quality assessment also labels regions in the my-

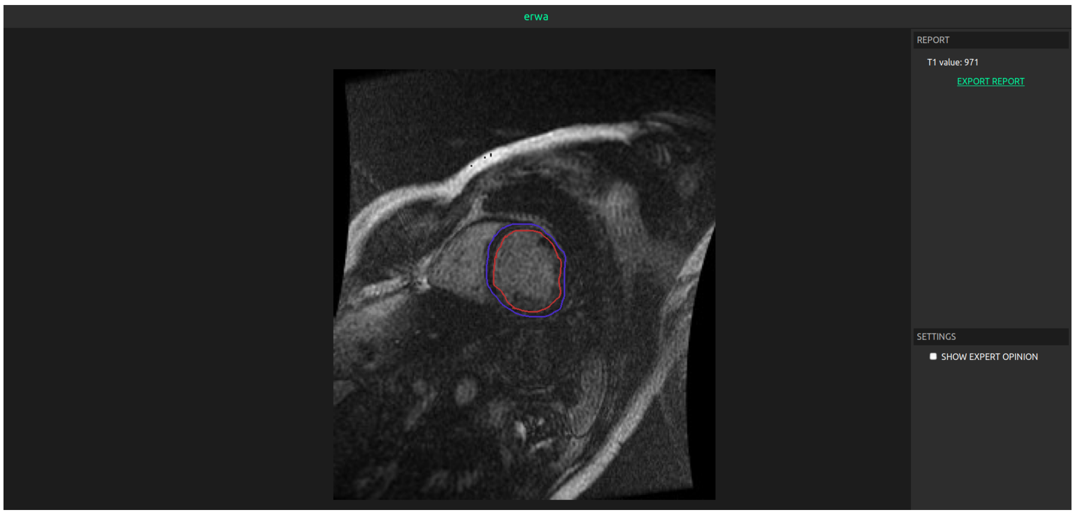


Fig. 2: Screenshot of the Trainees Dashboard after the T1 Analysis is uploaded.

ocardium affected by imaging artefacts in order to exclude them from the analysis.

Over the years, multiple approaches for placing these contours automatically have been proposed. In 2014, Hu et al. [7] proposed a method for automatically segmenting the left ventricular myocardial using local binary fitting models and dynamic programming techniques. Overall, their method shows good results, yet they struggles to segment the overlap between intensity distributions within the cardiac regions. Abdelfadeel et al. [8] use maximally stable extremal regions to segment the left ventricle of cardiac MRIs. Their achieves a DICE metric of 0.88 on the Medical Image Computing and Computer Assisted Intervention (MICCAI) 2009 challenge database [3]. Similar to our method, Zreik et al. [9] propose a method based on deep CNNs, where they try to segment the left ventricle in cardiac CT images. Their method uses a combination of three different CNNs, each detecting the presence of the left ventricle in the axial, coronal, and sagittal independently. Our work, however, differs from these approaches as we try to measure the quality of a student contour by generating an expert consensus based on the student contour and MRI image using a deep CNN.

III. The Research Contribution

The development of the software tool started from the specific requirements gathered from a team of researchers working in the field. Thus, it is first of all a tool for research support. In this respect, by allowing a more efficient and consistent monitoring of quality assessment in cardiac MRI image analysis, it has a fundamental impact on the quality of the research produced at that centre. However, this has given us the opportunity to carry out novel research in the field of quality assessment using deep learning methods, specifically to automate the generation of contours based on experts observer analysis (in other words, generating an expert consensus). This approach to quality assessment based on the comparison of contours, rather than point measures such as T1 or WT, has not yet been fully explored in the CMR community, with some exceptions [10]. We believe that this aspect of our work will contribute to a more quantitative, detailed and standardized approach to quality assessment of image analysis done on T1 mapping in cardiac MRIs.

IV. System Description and Usage

We have divided the application into two user groups: (a) experts / supervisors and (b) trainees. The system expects both users to upload files of the file types Digital Imaging and Communications in Medicine (DICOM) [11], that contain the original T1 mapping MRI data, and Interactive Data Language (IDL) SAV Files [12], that contain the image segmentation, that is the black and white binary mask representing the ROI enclosed by the contours (specifically, the image segmentation). The system is implemented as a web-based application, comprising of a back-end server and front-end web interface (which will be further discussed in Section V).

A. *The Supervisor System*

The supervisor (expert) can upload a pre-generated consensus of experts into the system, which will give the trainees the ability to compare their own analysis with that of the gold-standard. In addition, the supervisor can enable an automatically generated consensus to be an evaluation option for the trainees. This allows for the use of non-contoured datasets which increases the overall training material. The system will also allow supervisors to view the overall progress of a class of trainees by getting an automatically generated report about their collective progress. The report is produced in a PDF format and provides various graphs and summary metrics about individual trainees and the class as a whole.

B. *The Trainee System*

As previously stated, the main purpose of this software is to train observers in contouring the endocardium and epicardium

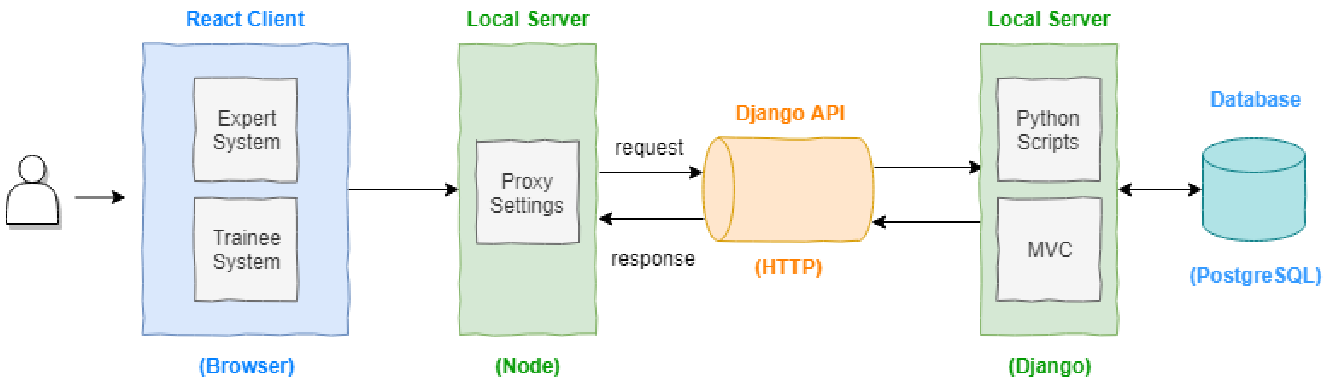


Fig. 3: A diagram showing how data is transferred across the system, starting with the user and ending with the Django based server. From the left, we see that a user uploads a segmentation together with the associated training image. This is then passed through the local Node based proxy server to the centralized Django server. The image together with the contour is stored in the database and evaluated by the CNN ”expert”. The evaluation is then passed back to the user where he/she is presented with their performance metrics.

of the left ventricle, and giving immediate feedback by comparing against an automatically generated ”expert” consensus produced by a deep CNN. For an example use case scenario, we imagine a trainee who has been working on contouring an MRI as part of their training dataset. The trainee uploads the T1 mapping of the MRI (or DICOM file) along with their produced segmentation into the system. The system will then start extracting key values from the DICOM [11] file, of which the trainee will be presented with a visual representation of their work, as well as the T1-value and the WT. Then, to get an understanding of their progress, the trainee may compare against the systems generated ”expert” contour consensus by getting a visual representation of the overlay between the two contours (trainee produced contour and CNN produced contour), showcasing the regions of discrepancy between the two. Furthermore, the user will get some key metrics regarding how well their contour compares to the one generated by the system. These evaluation metrics will be used to clearly state how well the trainees segmentation compares to that of the consensus.

The trainee can also generate a basic T1 training report to track progress of their repeated analysis over the same dataset. This is produced in a PDF format and provides an overview of the progress with graphs and summary metrics showing the discrepancies between their analysis and the consensus. This report may also be used to track the progress of a team of trainees, showing how a class improves over time.

V. Software Architecture and Implementation

Due to the sensitive nature of the data used for training, as well as the geographical localization of our target demographic, the system is only meant to be run over a local area network (LAN). That is, it is not meant to be available outside the area of deployment. With this in mind, our web application is managed by two locally run servers. One is based on Node.js [13] acting as an intermediate proxy which is responsible for handling the front-end logic, and the other is based on the python web-framework Django [14] which interacts with the database and deep learning model. A screenshot of the web-based graphical user interface (GUI) is shown in Figure 2, and the overall architecture is shown in Figure 3.

The web-based client is built using React [15] (a popular JavaScript library used to build front-end interfaces), which communicates directly with the Node.js based server. This server acts as a proxy that redirects all HTTP requests and responses to the Django server. From here, Django may retrieve images from the database or evaluate uploaded contours against the contours produced by the CNN model or expert consensus. Furthermore, the Django-based server is also responsible for calculating the evaluation metrics used to assess a trainees performance on contouring the left ventricle. The underlying CNN is implemented using the deep learning framework Keras [16] with a TensorFlow [17] back-end. The CNNs purpose is to automatically infer the endocardium and epicardium contours based off a given MRI and student contour.

VI. Automated Feedback Using CNNs

Part of what makes the system novel is the automatic generation of an ”expert” contour consensus using deep CNNs. The purpose of automating this process is to fulfill the need for quick and consistent feedback to the trainees using the system, in addition to not needing a fully expert contoured dataset. This section will cover the various experiments done in producing this ”expert” observer CNN, and a qualitative analysis on the models produced contours. But first, we will give a short description on the dataset used to train our models.

A. Dataset Details and Preprocessing

The dataset consists of 42 fully anonymized single mid-ventricular short-axis native T1-maps, half of which come

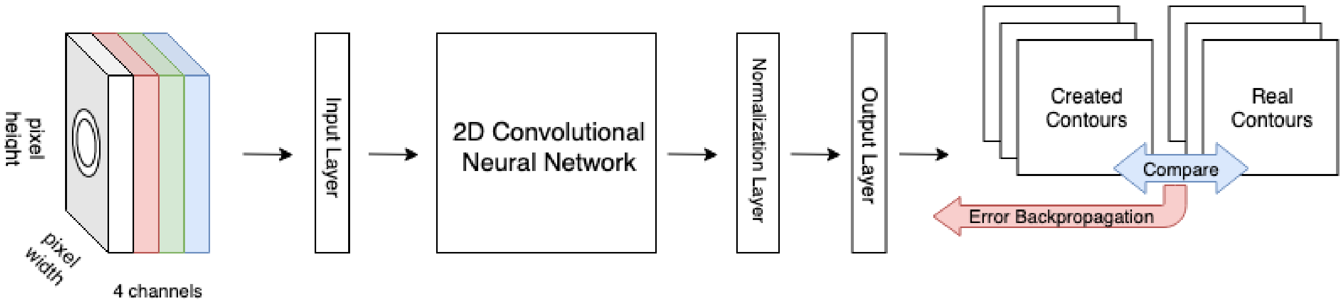


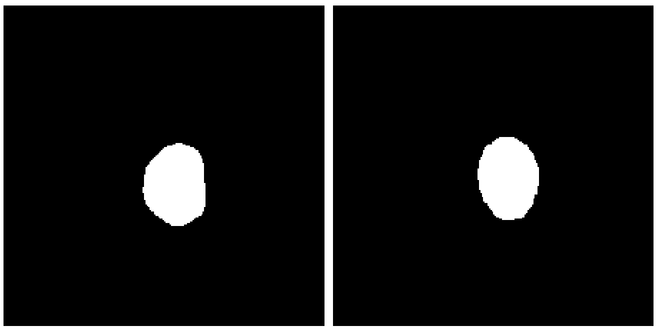
Fig. 4: A diagram showing the entire process for training the "expert" consensus CNN used to evaluated trainees. Starting from the left, we see that the input image (comprised of the student contour and MRI) is resized to $224 \times 224 \times 4$ before being put into the model. Then, the model outputs a vector with size $52,176$, which is then reshaped into the same shape as the input image.

from healthy volunteers, while the other half from patients with an acute myocardial injury. Each T1-map comes as a DICOM file (a standard for medical imaging data). A typical DICOM file contains a wide variety of different information ranging from simple meta-data (such as information about the MRI sequence or details about the patient), to full MRI images. Because we are only interested in the MRI data, the MRI images had to be extracted before training the neural network. This was done using the python library called pydicom [18], which is a popular library for working with DICOM files. The extracted images vary in size, ranging from 384×264 to 384×344 pixels.

For the ground truth contours, the dataset includes several contours per DICOM file. Each MRI has been contoured by several experts to form a consensus between them. Overall, there are two sets of contours. One for the inner myocardial wall (endocardium), and one for the outer (epicardium). These contours come in a special file written in a proprietary programming language called IDL. The contents of these files is coordinates which correspond the the contours of the related MRI image. For training purposes, we converted these files to a pixel activated vector which was used as the ground truth for our CNN experiments.

B. *CNN Architecture and Training Configuration*

The network used to generate segmentations is a modified version of the VGG-16 [19] based model implemented in Keras. To decide on which architecture to use, we experimented with numerous of the Keras-based implementations of popular CNN architectures (such as, ResNet [20], Inception [21], DensNet [22]), but we ended up using VGG-16 as it showed the best performance. For input, the model expects an image consisting of four channels. The first three channels consists of the R (red), G (green), and B (blue) color channels of the extracted MRI image (extracted as previously described). The fourth image channel is represented by the student contour. The input image is then resized $224 \times 224 \times 4$ before being passed into the model, which outputs a vector with the size $50,176$ (224×224) which represents the segmentation of the left ventricular myocardial ring. Each value in the vector represents one pixel of the output segmentation, and each pixel can be either "on" or "off", meaning it consists of $50,176$ binary values being either 1 or 0. To get the final segmentation, we chop the output vector into 224 pieces and stack them on top of each other before turning "on" pixels white and "off" pixels black.



(a) CNN system (b) Expert observer

Fig. 5: The segmentation results produced by the system's CNN and an expert observer.

For training, we used the Keras based implementation of VGG-16 and trained it from scratch. We replace the classification block of the original model with a custom block consisting of one 2D global average layer, a normalization layer which squashes the input values between 0 to 1, then a final fully-connected layer consisting of $50,176$ nodes (one for each pixel). The model was trained using mean absolute error (MAE) to calculate loss and Nadam [23] to optimize the weights with a learning rate of 0.004 which decayed at a rate of 0.900 for β_1 and 0.999 for β_2. The ground truth is created from expert segmentations made by multiple observers. This includes several expert segmentations for a single MRI image, which are all used in the training process to let the network average an "expert" consensus. To ensure robust results, each experiment was run using 3-fold cross-validation. A diagram explaining the entire training process can be seen in Figure 4.

Figure 5 shows some examples segmentations produced by the underlying CNN model compared to that of the actual expert produced segmentation. We see that the CNN produced segmentation is quite similar to that of the expert, but still requires some improvements before being deployed into a fully functional system.

VII. Conclusion and Future Work

In this paper, we presented a software tool meant to aid students in training for contouring the inner and outer wall of the left ventricle of the heart. The system automatically evaluates the trainees segmentation by comparing it against an "expert" consensus generated by a deep CNN. This allows for immediate feedback on their submitted segmentation and removes the need for a fully contoured expert dataset. Furthermore, the system presents an overview of the student progress using a variety of different metrics and graphs. The underlying CNN used for generating the "expert" segmentation is based on a VGG-16 architecture.

For future work, we aim to support more options for generating an expert consensus, in addition to improving on the existing method. Furthermore, we intend to get this software into the hands of real trainees and observers to gain further feedback and a real-world evaluation of this tool.

References

[1] E. J. Topol, "High-performance medicine: the convergence of human and artificial intelligence," *Nature medicine*, vol. 25, no. 1, p. 44, 2019.

[2] M. Riegler, M. Lux, C. Griwodz, C. Spampinato, T. de Lange, S. L. Eskeland, K. Pogorelov, W. Tavanapong, P. T. Schmidt, C. Gurrin *et al.*, "Multimedia and medicine: Teammates for better disease detection and survival," in *Proceedings of the 24th ACM international conference on Multimedia*. ACM, 2016, pp. 968–977.

[3] P. Radau, Y. Lu, K. Connelly, G. Paul, A. Dick, and G. Wright, "Evaluation framework for algorithms segmenting short axis cardiac mri." 07 2009.

[4] D. R. Messroghli, J. C. Moon, V. M. Ferreira, L. Grosse-Wortmann, T. He, P. Kellman, J. Mascherbauer, R. Nezafat, M. Salerno, E. B. Schelbert, A. J. Taylor, R. Thompson, M. Ugander, R. B. van Heeswijk, and M. G. Friedrich, "Clinical recommendations for cardiovascular magnetic resonance mapping of t1, t2, t2* and extracellular volume: A consensus statement by the society for cardiovascular magnetic resonance (scmr) endorsed by the european association for cardiovascular imaging (eacvi)," *Journal of Cardiovascular Magnetic Resonance*, vol. 19, no. 1, p. 75, Oct 2017. [Online]. Available: https://doi.org/10.1186/s12968-017-0389-8

[5] S. K. Piechnik, V. M. Ferreira, E. Dall'Armellina, L. E. Cochlin, A. Greiser, S. Neubauer, and M. D. Robson, "Shortened modified look-locker inversion recovery (shmolli) for clinical myocardial t1-mapping at 1.5 and 3 t within a 9 heartbeat breathhold," *Journal of Cardiovascular Magnetic Resonance*, vol. 12, no. 1, p. 69, Nov 2010. [Online]. Available: https://doi.org/10.1186/1532-429X-12-69

[6] J. C. Moon, D. R. Messroghli, P. Kellman, S. K. Piechnik, M. D. Robson, M. Ugander, P. D. Gatehouse, A. E. Arai, M. G. Friedrich, S. Neubauer, J. Schulz-Menger, and E. B. Schelbert, "Myocardial t1 mapping and extracellular volume quantification: a society for cardiovascular magnetic resonance (scmr) and cmr working group of the european society of cardiology consensus statement," *Journal of Cardiovascular Magnetic Resonance*, vol. 15, no. 1, p. 92, Oct 2013. [Online]. Available: https://doi.org/10.1186/1532-429X-15-92

[7] H. Hu, Z. Gao, L. Liu, H. Liu, J. Gao, S. Xu, W. Li, and L. Huang, "Automatic segmentation of the left ventricle in cardiac mri using local binary fitting model and dynamic programming techniques," *PLOS ONE*, vol. 9, no. 12, pp. 1–17, 12 2014. [Online]. Available: https://doi.org/10.1371/journal.pone.0114760

[8] M. A. Abdelfadeel, S. ElShehaby, and M. S. Abougabal, "Automatic segmentation of left ventricle in cardiac mri using maximally stable extremal regions," in *2014 Cairo International Biomedical Engineering Conference (CIBEC)*, Dec 2014, pp. 145–148.

[9] M. Zreik, T. Leiner, B. D. de Vos, R. W. van Hamersvelt, M. A. Viergever, and I. Igum, "Automatic segmentation of the left ventricle in cardiac ct angiography using convolutional neural networks," in *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, April 2016, pp. 40–43.

[10] A. Suinesiaputra, D. A. Bluemke, B. R. Cowan, M. G. Friedrich, C. M. Kramer, R. Kwong, S. Plein, J. Schulz-Menger, J. J. Westenberg, A. A. Young, and E. Nagel, "Quantification of lv function and mass by cardiovascular magnetic resonance: multi-center variability and consensus contours." *Journal of cardiovascular magnetic resonance*, p. 63, 2015.

[11] N. E. M. Association. (2019) Dicom standard. [Online]. Available: https://www.dicomstandard.org/

[12] H. G. S. Inc. (2019) The save procedure. [Online]. Available: https://www.harrisgeospatial.com/docs/SAVE.html

[13] N. Foundation. (2019) Node.js. [Online]. Available: https://nodejs.org/en/

[14] D. S. Foundation. (2019) Django - the web framework for perfectionists with deadlines. [Online]. Available: https://www.djangoproject.com/

[15] F. Inc. (2019) React - a javascript library for building user interfaces. [Online]. Available: https://reactjs.org/

[16] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[18] D. Mason, scaramallion, rhaxton, mrbean bremen, J. Suever, V. Sochat, G. Lemaitre, D. P. Orfanos, A. Panchal, J. Massich, A. Rothberg, K. van Golen, J. Kerns, T. Robitaille, M. Shun-Shin, moloney, pawelzajdel, M. Mattes, F. C. Morency, huicpc0207, colonelfazackerley, M. D. Herrmann, K. S. Hahn, H. Meine, I. de Bruijn, E. Stevens, D. Barreto, C. Bryant, A. Fedorov, and A. Klimont, "pydicom/pydicom: 1.2.2," Jan. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2541240

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[21] C. Szegedy, , , P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1–9.

[22] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 2261–2269.

[23] T. Dozat, "Incorporating nesterov momentum into adam," 2015.