FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study program/ Specialization: Cybernetics and Signal Processing	Spring semester, 2019 Open / ~~Confidential~~
Author: Anders Kregnes	………Anders Kregnes…….. (signature author)

Programme coordinator: Professor Kjersti Engan

Supervisors(s): Professor Kjersti Engan, PhD Candidate Rune Wetteland

Title of Master's Thesis:
Myocardial Segmentation in LGE-CMR Images Using Deep Neural Networks

ECTS: 30

Subject headings: Myocaridal infarction, deep learning, deep neural networks, LGE-CMR images, myocardial segmentation	Pages: 70 + attachments/other: 5 + attached zip file Stavanger, 29th of June, 2019 Date/year

Myocardial Segmentation in LGE-CMR Images Using Deep Neural Networks

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

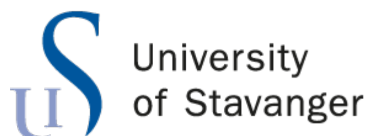
BY

ANDERS KREGNES

Master's Thesis

JUNE 2019

Under the supervision of Professor Kjersti Engan and PhD Candidate Rune Wetteland



Abstract

Cardiovascular diseases are the number one cause of death globally. 85% of these deaths are related to acute myocardial infarction or stroke.

One of the methods that are used to diagnose patients affected by myocardial infarction is myocardial segmentation in Late Gadolinium-Enhancement Cardiac Magnetic Resonance (LGE-CMR) images. The myocardial segmentation is today performed by an experienced cardiologist, manually or semi-automatically. The work is difficult and time-consuming, due to indistinct boundaries and variable pixel intensities.

The primary objective of this thesis was to develop a method for automatic segmentation of the myocardium, using deep neural networks, to aid the cardiologists. The proposed method uses the architecture of a Fully Convolutional Network. The network is trained on images and masks given by The Department of Cardiology at Stavanger University Hospital.

Two experimental layouts have been used; binary- and multiclass segmentation. In the binary experiments, masks consisting of two classes were used; the background and the myocardium. In the multiclass experiment, masks were divided into three classes; the background, healthy myocardium and myocardial scar tissue. The network was trained on 2006 images with corresponding masks, with the best model tested on 244 images from 30 patients. The predicted masks were compared to masks made by cardiologists at Stavanger University Hospital to evaluate the performance of the models.

The best developed model gave a Dice coefficient score of 0.705 with a standard deviation of 0.15. Considering the observations in this thesis, there is assumed further development of deep neural networks can improve the performance of myocardial segmentation.

Preface

This thesis marks the end of my M.Sc degree in Robotics and Signal Processing at the University of Stavanger, Department of Electrical Engineering and Computer Science.

I would like to express my gratitude to my head supervisor, Professor Kjersti Engan, for her exceptional guidance and valuable contributions during this semester. I would also like to direct my appreciation to my co-supervisor, PhD candidate Rune Wettedal, for his excellent feedback and support.

I am grateful that I have been the given opportunity to write my thesis within a field of study that fascinated me and is developing rapidly.

Finally, I thank my family, friends, and girlfriend for encouraging me throughout my five years of studies, culminating in this thesis.

Table of Contents

Abstract i

Preface ii

Table of Contents v

List of Tables vii

List of Figures x

Abbreviations xi

1 Introduction 1
- 1.1 Motivation and Problem Description 1
- 1.2 Related Work 2
- 1.3 Thesis Objective 3
- 1.4 Thesis Outline 3

2 Background 5
- 2.1 Medical Background 5
- 2.2 Deep Learning 5
  - 2.2.1 Biological and Artificial Neural Networks 7
  - 2.2.2 Activation Functions 8
  - 2.2.3 Convolutional Neural Networks 10
  - 2.2.4 Convolutional Layers 10
  - 2.2.5 Pooling Layers 11
  - 2.2.6 Fully Connected Layers 12
  - 2.2.7 Loss Functions 12
  - 2.2.8 Backpropagation and Gradient Descent 14
  - 2.2.9 Optimizers 15
  - 2.2.10 Epochs and Batch Size 16

2.2.11 Regularization . . . 17
2.2.12 Hyperparameter Optimization . . . 20
2.2.13 PyTorch . . . 21
2.3 Performance Evaluation . . . 22
2.3.1 Confusion Matrix . . . 22
2.3.2 Dice Coefficient . . . 23
2.3.3 The Jaccard Similarity Index . . . 23
2.4 Preprocessing . . . 24
2.4.1 Data Normalization . . . 24
2.4.2 Data Augmentation . . . 24

3 Data Material **27**

4 Proposed Methods **31**
4.1 Overview of the Proposed Methods . . . 31
4.2 Preprocessing . . . 32
4.3 Training . . . 35
4.3.1 Data Loading . . . 35
4.3.2 Network Architecture . . . 35
4.3.3 Choice of Hyperparameters . . . 36
4.4 Testing . . . 38
4.5 Implementations . . . 39

5 Experiments and Results **41**
5.1 Finding the Best Model . . . 41
5.2 Experiment One - Binary Segmentation . . . 42
5.2.1 Grid search . . . 42
5.2.2 Random Search . . . 46
5.2.3 Overfitting . . . 48
5.2.4 Random Search with Dropout . . . 49
5.3 Experiment Two - Multiclass Segmentation . . . 51
5.3.1 Grid Search . . . 52
5.3.2 Search Using Bayesian Optimization . . . 54
5.3.3 Evaluation of Multiclass Segmentation . . . 56
5.3.4 Comparison Between Experiment 1 and 2 . . . 57
5.3.5 Verification of Best Model . . . 57

6 Discussion **61**
6.1 Model Performance . . . 61
6.1.1 Experiment One - Binary Segmentation . . . 61
6.1.2 Experiment Two - Multiclass Segmentation . . . 61
6.1.3 Comparisons Between Binary- and Multiclass Segmentation . . . 62
6.2 Comparisons With Related Work . . . 62
6.3 Limitations . . . 62
6.4 Future Work . . . 63
6.4.1 New Network Architectures . . . 63

6.4.2 More Data Material 63
6.4.3 Training of the Deep Neural Networks 63

7 Conclusion **65**

Appendices **71**

A Results of Experiments **72**
A.1 Results of Inital Grid Search in Experiment One 72
A.2 Images From Best Found Model 73
A.3 Training- and Validation Plots for the Best Model Found 74

B Algorithms **75**
B.1 Matlab 75
B.2 Python 76

List of Tables

3.1 Distribution of data 29

5.2 Choice of hyperparameters based on grid search 46
5.3 Best performing model for ADAM and SGD with random search 48
5.4 Hyperparameters for random search with the use of dropout 49
5.5 Performance of models trained with ADAM and SGD the use of random search with dropout. 50
5.7 Multiclass segmentation - Grid search 54
5.8 Results for multiclass segmentation using Bayesian optimization. 56
5.9 Results for best model found in multiclass segmentation 57
5.10 Hyperparameters and validation performance for best binary and multi-class model 58
5.11 Hyperparameters and performance for the best found model. 58

List of Figures

1.1 Blocked coronary artery 2

2.1 Segmentation of myocardium 6
2.2 Biological neuron 7
2.3 Artificial neuron 8
2.4 ReLU activation function. 9
2.5 CNN used for classification 10
2.6 3 x 3 Convolution 11
2.7 Max-pooling 12
2.8 Gradient descent 15
2.9 Overfitting 18
2.10 Feed forward neural network with and without dropout 19
2.11 Confusion matrix and formulas used to calculate performance. 23

3.1 LGE-MRI images of a patient 28
3.2 Image of the myocardium and binary- and multiclass masks 28

4.1 System overview 32
4.2 System overview of the preprocessing. 32
4.3 Preprocessed images and masks 34
4.4 System overview of the training process 35
4.5 U-Net arcitecture 36
4.6 U-Net with dropout 38
4.7 System overview of the testing process 38

5.1 Binary mask of the myocardium 42
5.2 Validation- loss and accuracy for ADAM and SGD with cross-entropy loss function 43
5.3 Validation- loss and accuracy for ADAM and SGD with Dice loss function 44
5.4 Dice score for ADAM with the use of cross-entropy- and Dice loss. . . . 45
5.5 Dice score for SGD with the use of cross-entropy- and Dice loss. 45

5.6 Comparison between ADAM and SGD with the use of Dice loss. 46
5.7 Random search with ADAM 47
5.8 Random search with SGD 47
5.9 Validation- and training graphs with the use of ADAM and Dice loss . . . 48
5.10 ADAM with dropout 49
5.11 SGD with dropout 50
5.12 Bubble plots 51
5.13 Multiclass mask of the myocardium 52
5.14 ADAM and SGD multiclass with cross-entropy loss 53
5.15 Multiclass Dice loss 54
5.16 ADAM and SGD multiclass Bayesian 55
5.17 Samples from predicted masks performed by the best found multiclass model. 56
5.18 Visual comparison between binary and multiclass segmentation 57
5.19 Visual comparison of predicted masks and ground truth masks 59
5.20 Visual inspection of masks predicted by the best found model 59
5.21 Confusion matrix for best model found 60

A.1 Performance of ADAM and SGD with use of cross-entropy 72
A.2 Visual comparison of predicted masks and ground truth masks for one subject 73
A.3 Validation- and training graphs for best found model 74

Abbreviations

MI	=	Myocardial Infarction
GBCA	=	Gadolinium-Based Contrast Agents
SRNN	=	Shape Reconstruction Neural Network
AI	=	Artificial Intelligence
ILSVRC	=	ImageNet Large Scale Visual Recognition Challenge
NN	=	Neural Network
ANN	=	Artificial Neural Network
CNN	=	Convolutional Neural Network
FCNN	=	Fully Convolutional Neural Network
STD	=	Standard Deviation
GPU	=	Graphics Processing Unit
CPU	=	Central Processing Unit
CVD	=	Cardiovascular Disease
WHO	=	World Health Organization
LGE-CMR	=	Late Gadolinium Enhancement Cardiac Magnetic Resonance
ADAM	=	Adaptive Moment Estimation
GD	=	Gradient Descent
SGD	=	Stochastic Gradient Descent

Chapter 1

Introduction

This chapter introduces the medical condition acute myocardial infarction (MI) and the problems with diagnosis and effective treatment of patients affected by this disease. The problem of segmenting images of the myocardial muscle through images and insight into earlier research concerning this problem will be explored. The motivation for how this thesis can be a step towards developing methods for automatic segmentation of the myocardial muscle is presented, as well as the thesis objectives and an outline for the project.

1.1 Motivation and Problem Description

Cardiovascular diseases (CVDs) are said to be the number one cause of death globally, according to the World Health Organization (WHO) [1]. WHO estimates that in 2016, approximately 17.9 million people died from CVDs, 31% of the total number of deaths. 85% of those who died from CVDs died from either MI or stroke. For patients at risk of developing MI, and for those patients already affected, medical treatment is vital for prognosis and quality of life.

MI is a medical condition restricting blood supply to the cardiac muscle, also called myocardium [2]. Within a short time, the myocardium is damaged, and the cells that do not receive enough blood will die. After MI, myocardial scar tissue forms in the affected part of the heart, at the location where the artery was blocked. The size of this scar tissue is affected by the time that passes until medical treatment is administered. In Figure 1.1, we can see an illustration of a blocked artery causing an MI.

Late gadolinium-enhancement cardiac magnetic resonance (LGE-CMR) imaging is one of the methods used to diagnose patients affected by MI. By doing segmentation in images of the myocardium, cardiologists can determine the structure and functionality of the damaged muscle. These observations can be used to offer prognosis and provide guidelines for further treatment of the patient.

Today the segmentation of the myocardium is performed either manually or semi-automatically by an experienced cardiologist. The segmentation is difficult and time-consuming. With the development of new technology in image processing, computer vi-

sion, and deep learning, new methods have been proposed (see section 1.2). The purpose is to assist the cardiologist in the segmentation task and to hopefully develop ways that can make the segmentation fully automatic.

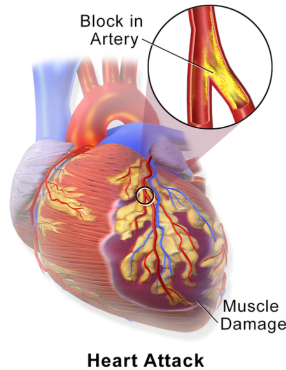


Figure 1.1: Illustration of how a blocked coronary artery causes myocardial infarction. The artery restricts blood flow to the muscle, leading to damage in the muscle tissue[1] [3].

1.2 Related Work

Recently, in June 2019, Q. Yue et al. proposed a method called SRSCN, for cardiac segmentation. SRSCN is built as a shape reconstruction neural network (SRNN) combined with a spatial constraint network (SCN) [4]. The method incorporates the shape and spatial priors of the myocardium. The structure of the SRSCN is an enhanced U-Net. By comparison, the network structure in this thesis is founded on the original proposed U-Net by O. Ronneberger et al. [5].

In November 2018, S. Moccia et al. at Universita Politecnica Delle Marce published their research which focused on the segmentation of the myocardial scar tissue [6]. The approach was to use a Fully Convolutional Neural Network (FCNN) to perform segmentation of scar tissue in LGE-CMR images.

The University of Stavanger and the Department of Cardiology at Stavanger University Hospital have been cooperating to develop technology for the segmentation of the myocardial muscle. The most recent work was published by K. Engan et al. in 2015 [7], presenting a method for performing an automatic segmentation of the endocardium (inner layer of the myocardium). In 2013 the same researchers published a method for automatic segmentation of the epicardium (outer layer of the myocardium) [8]. The methods developed are based on the usage of an algorithm for finding a posteriori probability map, categorizing the probability of a pixel to be included in the myocardium. The use of known prior information about typical heart sizes and that the blood pool and the myocardial mus-

[1] Image by Blausen Medical Communications, Inc., used under Creative Commons Attribution 3.0 Unported lisence

cle is approximately circular are exploited. The algorithms provided promising results and performed better than some traditional methods with marker-controlled watershed [9, 10].

1.3 Thesis Objective

The methods developed in this thesis will use deep neural networks (DNN) to approach the problem of performing segmentation of the myocardium automatically. The goal is to produce a model that can perform automatic segmentation of the myocardium of patients that have been affected by MI and evaluate the performance. Different metrics will measure the performance of the model by comparing the predicted masks with the masks made by an experienced cardiologist. Techniques for finding the best performing model will be applied and evaluated.

Two different approaches for segmentation are tested. One is using annotated masks of the myocardium, and the other is using annotated masks of healthy myocardium and myocardial scar tissue. The developed DNNs are trained end-to-end on images and manually segmented masks provided by the Department of Cardiology at Stavanger University Hospital.

1.4 Thesis Outline

Chapter 2: Background
In this chapter, some of the theoretical concepts within the field of deep learning will be described. The data material given by the University Hospital of Stavanger will be presented, and an introduction to how the data was used in the development of a DNN.

Chapter 3: Data Material
In this chapter, some of the theoretical concepts within the field of deep learning will be described. The data material given by the University Hospital of Stavanger will be presented, and an introduction to how the data was used in the development of a DNN.

Chapter 4: Proposed Methods
The approach to the thesis objective will be explained in this chapter, providing information and descriptions on how the system is developed and the process from beginning to end.

Chapter 5: Experiments and Results
This chapter will present how the experiments have been carried out, a presentation of the results and the performance of the final model.

Chapter 6: Discussion
The results will be discussed in this chapter, with an analysis of how well the method performed and suggestions for further improvements.

Chapter 7: Conclusion
The last chapter will give an overall evaluation and thoughts acquired from work done in this thesis

.

Chapter 2

Background

This chapter will provide theory behind MI, deep learning, and preprocessing of images.

2.1 Medical Background

The myocardium is the muscle making the contractions that ensure the distribution of blood to the body. The myocardium is positioned between the outer layer, the epicardium, and the inner layer, the endocardium. The endocardium separates the myocardium from the blood in the heart chambers, and the epicardium separates the myocardium from organs. An illustration of the marked myocardium, endocardium, and epicardium can be seen in Figure 2.1.

During an MI, the myocardium takes damage, and muscle cells die as a consequence of the reduced blood supply. After an MI, the doctors are using LGE-CMR imaging to analyze and categorize the myocardium. Gadolinium-Based Contrast Agents (GBCA) is inserted in the blood to make the scar tissue visible. Since there is no blood flow in dead tissue, the scar tissue will stand out from the healthy myocardium. In LGE-CMR images, tissue with low blood flow appears bright, and tissue with high blood flow appears dark. By inspecting the images, it is possible to find the size and location of the scars, by looking at which parts of the myocardium that are represented by high pixel values. These images provide essential information used for the prognosis and treatment of the patients. Treatments that are used include different medical treatment, angioplasty, and coronary artery bypass graft (CABG) surgery, where coronary arteries are replaced by blood vessels from another part of the body [11].

2.2 Deep Learning

Machine learning is a subfield of artificial intelligence (AI), where the purpose is to make algorithms learn by the use of data. By detecting patterns in the data, the algorithms can distinguish different types of data from one another. A simple example of this can be to

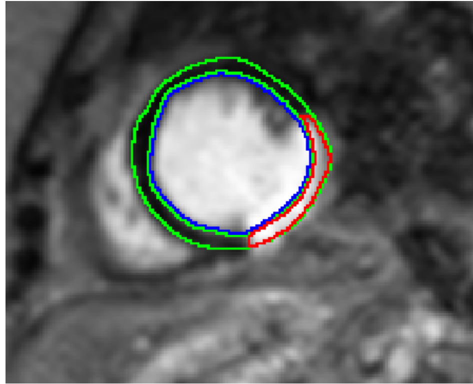


Figure 2.1: Image showing segmentation of the myocardium made by a cardiologist. The outer green contour marks the epicardium, the blue contour marks the endocardium, and the red contour marks the scar tissue. The area inside the blue contour is the blood pool, and the area between the two green lines marks the myocardium.

classify if one animal is a dog or a cat by making the algorithms learn from features such as weight, height, and color. These algorithms are an example of machine learning used for classification, where a label is assigned to the different classes. Another method in machine learning is to use features such as location, number of bedrooms, the size, and the year built to give a prediction of how much an apartment is worth. In this example, the predicted outcome is a continuous variable. Techniques predicting continuous variables are called regression. The features are the key to make these models since they are the objective factors distinguishing the objects. For making good models, it is vital to have sufficient data material.

Deep learning is a subfield of machine learning. Many consider deep learning as new technology, but the field of study has been developed under different terms from the birth in the 1940s [12]. Interest in deep learning has varied from the launch to present day, and the term deep learning was introduced in 2006. Formerly the area of study was included in the terms cybernetics and later connectionism. With the development of DNN and the use of multiple convolutional layers that can extract more high-level features (See 2.2.4), the term deep learning became natural to use. One of the vital resources for success in deep learning is the great development of powerful hardware, such as Graphics Processing Units (GPUs). Unlike Central Processing Units, GPUs use parallel computing. Parallel computing is well suited for matrix-vector multiplications, a key component in the construction of DNN. Another critical factor is the increasing access to data made available for model creation. The data size is a crucial part of developing deep learning models with good performance, as the more samples the DNN is exposed to during training, the better it can be at performing on new, unseen data. Through the development of more powerful hardware, improved algorithms, and access to more data, the impact of DL in society is likely to grow.

In recent years the implementation of DNN has been focused on various applications such as image classification, image segmentation, and natural language processing. In the

annual arranged ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [13], a team from the University of Toronto made a DNN that outperformed traditional methods used in image processing. The challenge was to classify objects in images with and without localization and the team from Toronto achieved a winning top-5 test error rate of 15.3% with the use of DNN. The top-5 test error rate measures how efficiently the system predicts a class to be among the top five predicted classes. An error rate of 15.3% suggests that the correct class was not among the top five classes in 15.3% of the tested images. The second best contestant used methods from traditional image processing and achieved a top-5 test error rate of 26.3% [14].

2.2.1 Biological and Artificial Neural Networks

Inspiration for the first learning algorithms in deep learning was based on how we believe biological learning happens in the brain of an animal [12]. The biological neural system contains billions of connected neurons, communicating through electrical impulses. These can be activated by internal processes in the body or external sources that stimulate the human-bodies senses. The neurons create new connections between one another to adapt to stimuli the brain is exposed to, and the creation of paths enables the brain to memorize. If the brain is exposed to the same stimuli multiple times, long-term paths of connections are created.

A biological neuron, as seen in Figure 2.2, is built by dendrites receiving electric signals from the axons terminals, called boutons, from other neurons, and sending messages from its boutons to dendrites in different neurons. We can regard the signals received in the dendrites as inputs and the signals sent from the boutons as outputs.

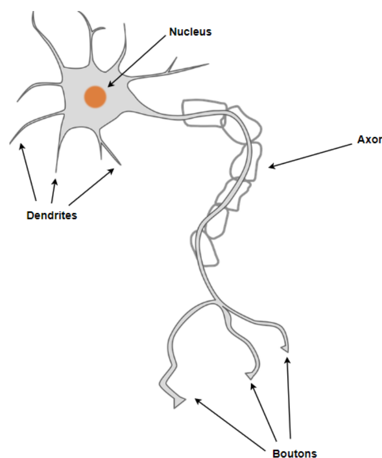


Figure 2.2: Illustration of the biological neuron[1] [15].

[1] Image used under Creative Commons Zerop CCO 1.0 Universal licence. The image has been edited by adding text

Artificial neurons (See Figure 2.3), also called perceptrons, are inspired by the design of the biological neuron [16]. The inputs can be compared to the dendrites and the output to the boutons.

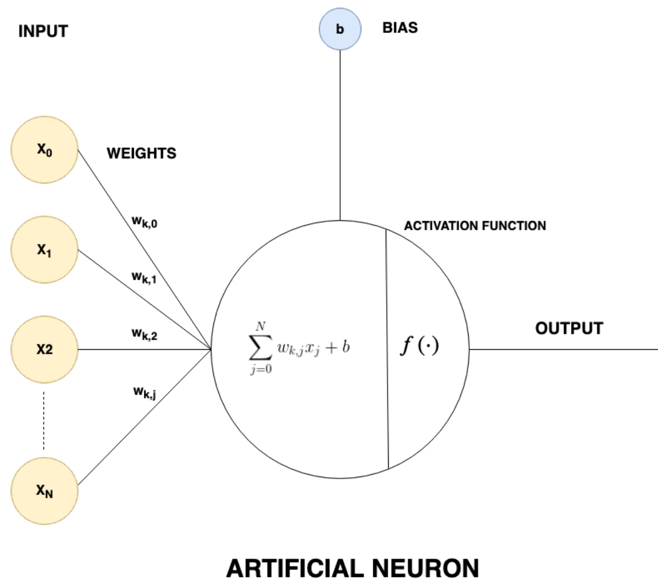


Figure 2.3: Illustration of the artificial neuron.

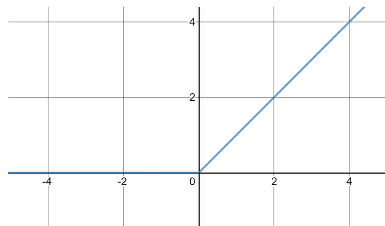
DNN's are a commonly used term for deep learning, as a result of the mentioned inspiration of the biological neural network. The algorithms are thought to mimic organic learning. However, today, our understanding of the brain structure is not sufficient enough to categorically define the functionality of the brain due to its complexity. Mapping the activity in the neural system is very challenging; therefore, the approach used today is to see DNN as a model of learning, inspired by the brain, without creating a clone. We can draw an analogy between the creation of DNN and the invention of airplanes. In the beginning, we aimed to model the way birds fly, but eventually, we developed machines that fulfilled the purpose in a different and more powerful way.

2.2.2 Activation Functions

Activation functions are nonlinear transformation done on the input signals of an artificial neuron. The functions control the activation of different nodes in a layer. The choice and use of activation functions are essential for creating well-performing models. From Figure 2.3 we saw that the formula for the output is determined by the sum of the weights multiplied by the input plus bias: $f(\cdot) = \sum_{j}^{N} w_{k,j} + b$. The calculation determines if the neuron fires or not, or if the output should be strong or weak. The weights and bias are learnable model parameters, updated during the training of the DNN. It is important to acknowledge that activation functions must be differentiable in order to execute backpropagation (see section 2.2.8).

ReLU Activation Function

The *Rectified linear unit* (ReLU) activation function is one of the most used activation function in DL today. In Figure 2.4, we can see that the neuron is activated when the calculations of input result in a positive output. The strength of the output is proportional to the value of the output. The advantage of using the ReLU activation function is that it constrains the number of neurons that fire, thus making the network more efficient and less computational heavy to train. Additionally, its properties make it efficient to update the weights when performing backpropagation (See section 2.2.8). ReLU is in general recommended for us in hidden layers in CNN's.



$$f(x) = \max(0, x)$$

Figure 2.4: ReLU activation function.

Softmax

The softmax activation function is commonly used in the output layer of a DNN used for classification or segmentation. The function gives a probability distribution where each class is assigned a decimal between zero and one, and where the probabilities are equal to one. The decimal represents the likelihood for the output to belong to the respective class. In classification problems, the prediction becomes the label given the highest probability by the softmax function. In segmentation problems, each pixel is predicted by choosing the class given the highest probability for the specific pixel. In segmentation networks, each class is represented by a channel. The pixels in each channel are given a probability by the softmax function, which lays the foundation for which label is assigned to the pixels in the predicted map. The predicted map is, in most cases a one-channel image, where each pixel is labeled as the class given the highest probability by the softmax function. In Equation 2.1 the equation for the softmax is given.

$$f(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{N} e^{x_i}}, for\ j = 1, 2, 3..., N_{classes} \tag{2.1}$$

2.2.3 Convolutional Neural Networks

An example of an ANN is the Convolutional neural network (CNN) (See Figure 2.5). A CNN consists of an input layer, a chosen number of hidden layers, and an output layer. Each layer has a set of neurons. When all neurons in a layer connected to all the neurons in the next and previous layer, the layer is called a fully connected layer.

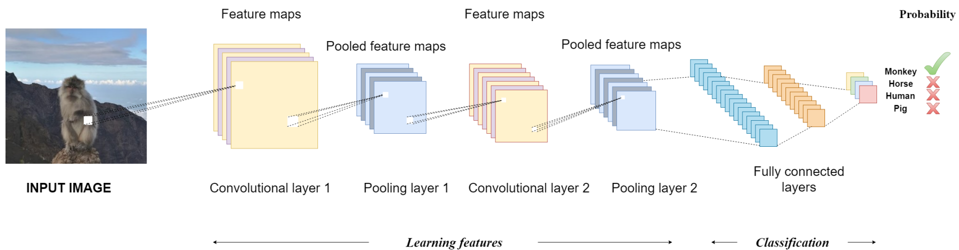


Figure 2.5: A CNN used for classification. The network takes an image as input and predicts the label of the object in the image.

2.2.4 Convolutional Layers

Convolutional layers are the components in an ANN that are making feature maps for extracting information on different levels in an image, such as edges, corners, shapes, etc. Kernels (also called filters) stride across the image, making convolutional operations (see Equation 2.3). The operation is comparable to sliding a flashlight over a surface. These convolutional operations map new values to a new matrix, called a feature map. The kernel contains specific learnable values, also called weights, that being updated by the model during training. In the first layers, the kernels can be used to extract low-level features such as horizontal or vertical edges. In the next layers, the kernels can find mid-level features such as corners, color, and gradient orientation. In deeper layers, high-level features such as noses, eyes, wheels, roads, poles, cells, and other structures can be found.

When applying convolutional operations on an RGB-image, which has three channels, three kernels are used, one for each channel. The operation of the dot product on the three channels results in one single feature map as output. So if a convolution is performed with six kernels, the result will be six separate feature maps. Adding bias to the operation is often used to determine the threshold of activation.

An illustration of a convolution operation made by a 3x3x1 kernel on a 9x9x1 image can be seen in Figure 2.6. The kernel, marked in red, strides across the image and applies a mathematical operation by taking the sum of an element-wise multiplication on the pixels in each window (see Equation 2.3). The output is added to a feature map, and the green pixel is the result of the first operation. The kernel moves a defined step at a time. In this example, each step is equal to one (stride=1), making the input image become a feature map of size 7x7. If a step size of two was used (stride=2), the feature map would have been of size 5x5. To determine the size of the resulting feature map *padding* can be applied. The use of padding adds pixels in the border of the input and makes it possible to maintain or increase the dimension of the resulting feature map, as well as retaining information

from the corner pixels of the input. The most common technique for padding in CNN's is *zero padding*. The operation adds zeros in the border of the input, making us able to get the desired resolution of the output. The output size is described by:

$$\begin{aligned} Output\ width &= \frac{W - F_w + 2P}{Stride_w} + 1 \\ Output\ height &= \frac{H - F_h + 2P}{Stride_h} + 1 \end{aligned} \tag{2.2}$$

Equation 2.2: W = width of the input, H = height of the input, F_w = width of the kernel, F_h = height of the kernel, P = padding size, $Stride_w$ = horizontal stride, $Stride_h$ = vertical stride

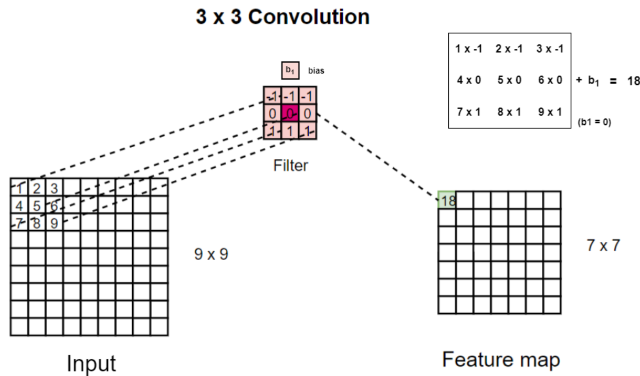


Figure 2.6: Illustration of a 3 x 3 convolution on an image. The kernel showed in this example is called a Prewitt filter, used for vertical edge-detection.

$$g(x,y) = \omega * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \omega(s,t) f(x-s, y-t) \tag{2.3}$$

Equation 2.3: Mathematical equation for the convolutional operation. $f(x,y)$ is the input image, ω is the filter and $g(x,y)$ is the resulting feature map after the convolution.

2.2.5 Pooling Layers

It is common to use pooling layers after some of the convolutional layers in a CNN. Pooling layers are used for reducing the dimension of the inputs in the hidden layers. The use of pooling layers is an example of nonlinear sampling. The pooling makes the representations less computational heavy and easier to process. Additionally, in many cases, it is beneficial to make a more abstract representation of the input to help with the learning

process. Two types of pooling commonly applied are average pooling and max-pooling. The pooling is performed by a kernel sliding over the input and extracting a value that is mapped to an output. Average pooling computes the average value of all pixels within the window covered by the kernel and assigns this value to the new feature map. The max-pooling operation consists of identifying the maximal value in the window and assigning it to the feature map. Max-pooling is the most used pooling operator in DNN because the operation discards noisy parts of the input. The pooling operations are illustrated in *Fig. 2.7*. The input is a matrix of size 6 x 6, and the output of size 2 x 2. The kernel is of size 3 x 3, and the stride is set to 3.

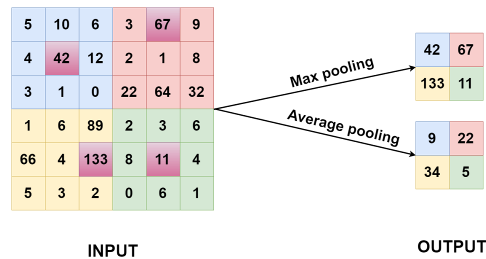


Figure 2.7: Figure showing a max-pooling operation with 3x3 kernel and stride 3.

2.2.6 Fully Connected Layers

Fully connected layers are put at the end of the network and use the high-level features from the convolutional and pooling layers to make a prediction. In DNN used for classification, the output will be the prediction of which class the input image represents, and the fully connected layer makes the decision based on the high-level features received from all neurons in the previous layer.

2.2.7 Loss Functions

A loss function is used to evaluate the performance of a DNN by determining how well the algorithms fit the data. The loss function takes the output of a model and compares it with the target we want to predict. It calculates a value showing how far the predicted output deviates from the target, which is used to update the weights of the network in the next phase. The goal is to minimize the loss and optimize the performance.

In DNN's, two main categories of loss functions are used; regression loss functions and classification loss functions. Classification loss functions are often used for object detection in images and segmentation problems when we have N number of labels. Regression loss functions are used when we are predicting real value quantities, such as the price of goods. In this thesis, two different loss functions have been used; the cross-entropy loss function and dice loss.

Cross-entropy Loss Function

The cross-entropy loss function is widely used in DL today. The function compares pixel-wise the predicted probability for each class with the label in the target. The scores for the pixels are summed and averaged. Due to the calculation of the average, the loss is distributed equally for all the pixels; thus, class imbalance is not taken into account. When working with unbalanced data, the weighted cross-entropy function is commonly used. Each class is assigned a weight, which scales the contribution from each class to the total loss. This makes it possible to penalize misclassification of classes that are more of interest to predict. The formulas for the cross-entropy loss function and the weighted cross-entropy loss function are given in Equations 2.4 and 2.5 [17].

$$\operatorname{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right) \tag{2.4}$$

Equation 2.4: Cross-entropy loss function

$$\operatorname{loss}(x, class) = weight[class]\left(-x[class] + \log\left(\sum_j \exp(x[j])\right)\right) \tag{2.5}$$

Equation 2.5: Weighted cross-entropy loss function

Dice Loss Function

The use of Generalized Dice overlap is a popular loss function to implement in DNN for segmentation in medical images [18]. In medical images, the pixels of objects of interest, are often underrepresented, representing a small fraction of the total image. Dice loss uses the measurement of the Dice Similarity Coefficient (DSC) (see section 2.3.2). During training, the algorithm seeks to optimize performance by minimizing (1 - DSC). The formula for calculating the dice loss can be seen in Equation 2.6.

$$DL = 1 - \frac{\sum_{n=1}^{N} p_n r_n + \epsilon}{\sum_{n=1}^{N} p_n + r_n + \epsilon} - \frac{\sum_{n=1}^{N} (1-p_n)(1-r_n) + \epsilon}{\sum_{n=1}^{N} 2 - p_n - r_n + \epsilon}$$

N = number of images
$r_n = voxel\ values\ for\ the\ ground\ truth$
$p_n = voxel\ values\ for\ the\ predicted\ probabilistic\ map\ of\ the\ foreground$
$\epsilon = constant\ to\ avoid\ dividing\ by\ zero$

(2.6)

Equation 2.6: Equation for calculating the Dice loss over N images [18].

2.2.8 Backpropagation and Gradient Descent

The training process is targeted to make use of the error between the target value and output value, enabling a change to the weights and biases to reduce the error and increase the performance. This is achieved by using gradient-optimization in a process often referred to as *gradient descent (GD)*. For the gradient-optimization process, algorithms called backpropagation are used to calculate the gradients, which are used to update the parameters in the DNN [19]. The algorithms use the error calculated by the loss function. When a signal initialized by feeding the network with an input has propagated through the entire network, the next step is to utilize the result given by the loss function to backpropagate through the network and update the weights, layer by layer. The idea is to calculate the gradients at a given point in order to identify the gradient that would decrease the error the most, enabling the performance of the model to increase.

Figure 2.8 is an illustration of how GD optimization works in a 2-dimensional space. The operation can be compared to a ball rolling down a hill where the goal is for the ball to reach the lowest point in the valley. Along the y-axis we find the cost *J(w)* and along the x-axis, we find the weights *w*. The landscape illustrates the different configurations of the weights. It is optimal when the weights are configured such that the cost is at its minimum. This configuration is called the global minimum. GD uses the calculation of the gradients, of the configuration of weights, at a given point. The gradient is the derivative of the slope at that time. When the steepest (most negative) gradient is calculated, the system will make a step in that direction and update the weights to lower the error. The ”valley” contains suboptimal configurations of weights, also known as local minimums. These are configurations where the system does not improve with further training, even though the optimal solution is not found. In Figure, 2.8 three different learning rates are illustrated as curves in red, blue, and green.

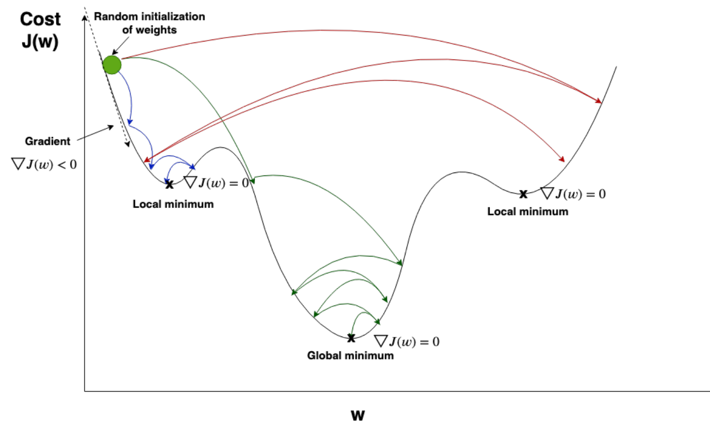


Figure 2.8: Illustration of the gradient-optimization. The process is comparable to a ball rolling down a valley trying to reach the lowest position, where the cost is at its minimum. The curves in red, blue, and green illustrate the learning rates. By finding the optimal learning rate, the global minimum can be found.

Learning Rate

The learning rate (η), or the step size, is a parameter used for determining how extensively the weights should be updated. η is multiplied by the calculated gradient, making it possible to scale how much w_{i+1} should be adjusted (See Equation 2.7). For example, setting $\eta = 0.1$ is giving a value equal to 10% of the original gradient.

$$w_{i+1} = w_i - \eta \cdot \frac{\partial}{\partial w_i} J(w_i) \tag{2.7}$$

Equation 2.7: Formula for updating weights

In Figure 2.8 we can intuitively see that η determines how quickly the weights are updated in the direction of the gradient. In this simple example, too high η (red) is not optimal - the weights are adjusted too much per iteration, and the loss does not converge. Too low η is not optimal - the weights are adjusted insufficiently in each iteration, and the configuration concludes with a local minimum. The optimal η (green) adjusts the weights appropriately, and the configuration reaches a global minimum.

2.2.9 Optimizers

Optimizers are algorithms designed to update the weights to minimize the loss calculated by the loss function. The optimizer uses η as a parameter to decide on which scale the weights should be updated. In some optimizing algorithms, the parameter momentum is added as a hyperparameter. Momentum can be thought of as a factor that accelerates the

GD in the steepest direction of a curve. An analogy can be drawn to Figure 2.8 where the ball gains momentum whilst rolling down a hill. To update the weights, algorithms with momentum exploit the existing steps gradients and gradients from earlier steps. The idea is to make the loss converge faster and find the best model parameters in less time.

One conventional algorithm for doing the optimization in the learning process and find the minimum loss is the stochastic gradient descent (SGD). Most optimizers used in DL today are based on SGD, which are proven to perform well. Two variants are the classic SGD and the modified variant ADAM optimizer introduced by Kingma et al. in 2014 [20]. The ”classic” SGD-optimizer and the ADAM optimizer have been explored in this thesis.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an optimizer using a stochastic approximation of the gradient descent optimization [21]. It is called stochastic because it uses a randomized selection of samples to calculate the gradients. The algorithm performs weight updates for each training sample and is often combined with momentum for making use of the gradients calculated in previous steps.

Adaptive Moment Estimation

Adaptive Moment Estimation (ADAM) is an optimizing algorithm designed to compute adaptive learning rates for each parameter [20]. It makes use of the first and second moment of the gradient and uses an exponential moving average of the gradients to scale the learning rates. ADAM is an efficient algorithm that requires little use of memory.

2.2.10 Epochs and Batch Size

Epochs are the number of times a full training set is propagated through a DNN, with corresponding updates of weights. The risk of overfitting (See section 2.2.11) increases as more epochs one run since the complexity of the model rises, and the weights are excessively altered to fit the training data. A technique to manage this is an implementation of saving the best model. Saving the best model means to save the weights only if the validation loss calculated after an epoch is lower than the lowest registered validation loss from previous epochs (see Algorithm 1). Saving the weights can also be done when finding a new top validation accuracy. The measurement used often depends on the objective. The batch size indicates the number of training samples that are passed through the DNN for each weight-update. This is also a hyperparameter and is often determined by the limitation of the memory on the GPU.

Algorithm 1 Saving of best model

Initialization: Initialize epochs, patience, best val loss
```
forall epochs do
    Train model
    validate model
    if current val loss < best val loss then
        best val loss ← current val loss
        overwrite best model
    else
        patience ← patience + 1
    end
    if patience > limit then
        stop training
    end
end
```

2.2.11 Regularization

Overfitting

A lack of data is a common problem when training DNN's. With limited accessible data, the network *overfits* the training data prematurely, making it difficult to achieve good results. When the training set is small, the network will quickly learn the features in the data fast, and therefore perform poorly when exposed to new data. Another issue when training a network is the model complexity, which is a factor of how deep the network is, in other words, how many hidden layers there are. With many convolutional layers, the network may become too complex, reducing the performance. It is, therefore, vital to balance the quantity of data and depth of the DNN when designing a system. Regularization is a set of techniques that are often applied to prevent overfitting.

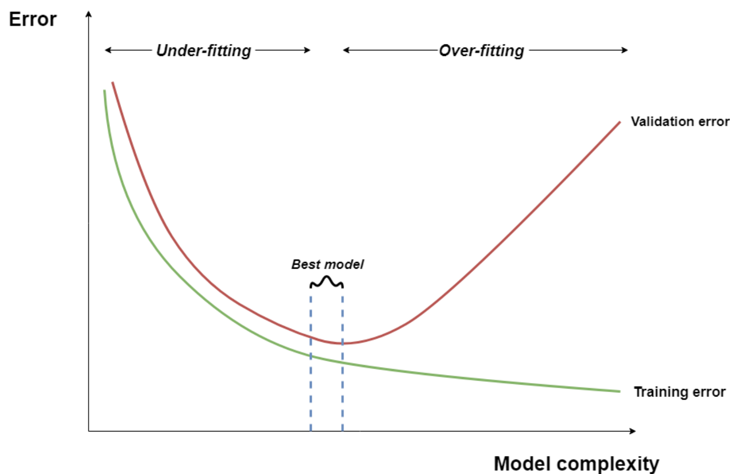


Figure 2.9: Illustration of overfitting. The complexity of the model increases whilst training. After some time the model learns features that are specific for the training data, adjusting excessively to the training data and becomes less capable of recognizing general features, or in other words, we can say that the model fits the training data. It is important to find the weights when the validation loss is at its lowest, before it diverges from the training loss, to find the best performing model. The area below *best model* in the plot is the range of optimal complexity.

Dropout

Dropout is a method that is used for avoiding overfitting and generalizing the data when training a DNN. A. Krizhevsky et al., at the University of Toronto, have shown that by introducing dropout to DNN's the performance can be improved significantly in different applications, such as analysis of computationally biological data, object classification and digit recognition [22]. The idea behind the technique is to drop nodes in the network, and the connections to other nodes, randomly. The implementation of dropout is performed by setting a parameter that defines the probability of how many nodes are randomly skipped in a layer during one forward- and backward pass. To ignore a node, the weights are multiplied with zero. When running multiple forward- and backward passes, different combinations of nodes will be skipped at each run, and different sub-networks of the original network architecture are created. By introducing dropout, random noise is added to the network, enhancing the models capacity to learn general features in the data, and possibly improving the quality of the features. When looking at the n hidden units in a layer, and introducing a probability p for skipping a unit, the number of units that will be present in the layer are $n(1-p)$. When applying dropout, another hyperparameter is introduced that must be tuned. The dropout rate, in combination with other hyperparameters, such as learning rate and choice of optimizer and loss function, increases the complexity of finding the best model.

The number of hidden units n needs to be considered when searching for an optimal drop-rate p. Dropping many nodes might increase the training time and result in underfitting. On the contrary, dropping too few nodes may not avoid overfitting. The size of the dataset also plays a part. Krizhevsky et al. tested datasets of different sizes and found that

with limited or vast datasets, the use of dropout gave little or no effect. A downside of applying dropout is the subsequent increase in training time. Krizhevsky et al. experienced that typically training would take between two to three times longer.
When testing the performance of trained networks on unseen data, dropout is not used. When predicting the test data, the weights in the network are scaled by multiplying the weights with the same p initialized during training. This multiplication is used to counteract the effect the use of dropout made in the training process. The outputs will then be the same as the expected predicted outputs during training.

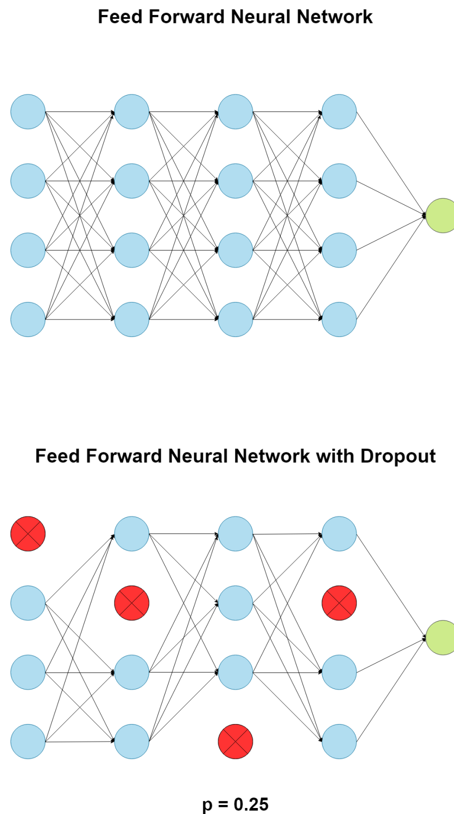


Figure 2.10: The figure shows an example of applying dropout to a DNN with three hidden layers and four hidden nodes in each layer. The units in the original network will be dropped with a probability of p=*0.25*.

Batch normalization

Batch normalization *BN* is a technique that has become popular to use in DNN's [23]. S. Ioffe et al. have shown that implementing BN in their DNN's has improved the training speed and performance [24]. The method builds on the same principle as normalizing the input samples that are fed into the network (see section 2.4.1). BN is applied to a batch before a layer, providing a common distribution for pixels in the input. Each node

in a layer is transformed to have a mean of zero and a variance of one. Compared to the calculation done when normalizing the images in the preprocessing (see section 2.11), there are used learnable parameters that scale and shift the normalized values. Introducing such learnable parameters is necessary since the parameters in the layers change during training and shifts the distributions of parameters passed on to the next layers. Using BN has proved to make DNN's more robust, therefore by adjusting hyperparameters, such as the learning rate, does not considerably affect the performance. It has been found that by using BN, the implementation of dropout does not necessarily improve performance since the BN introduces noise to the network [12, 24]. X. Li et al. experienced through their research that using a combination of dropout and BN might even decrease the performance of the DNN [25]. They suggest that by modifying where the dropout layers are inserted and different scale of variance, might be beneficial to improve performance. The formula for normalizing a pixel in an image, with the use of learnable parameters can be seen in Equation 2.8.

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon} * \gamma + \beta} \tag{2.8}$$

Equation 2.8: Formula for normalizing a pixel. y is the output, $\gamma\, and\, \beta$ are learnable parameters used to scale and shift the distribution, and ϵ is a constant added for numerical stability.

2.2.12 Hyperparameter Optimization

Hyperparameter optimization, or hyperparameter tuning, is the search for finding the best as possible combinations of hyperparameters for a model. The aim is to determine the set of hyperparameters that provide the best result when testing the models on a validation set. The hyperparameters are all tunable variables, such as the optimization function, the loss function, the use of weighted loss, network architecture, data augmentation, and the number of epochs run. These parameters will here forth, be referred to as x. For a learning algorithm *A*, the objective is to find parameters *x*, such that a loss function *f* is minimized. This is performed by evaluating *f* against different sample values of x. The way the hyperparameters are combined is essential for the performance of a DNN. The search is performed experimentally by training multiple combinations of parameters and evaluate the performance of the different models. Different strategies have been developed for finding the optimal hyperparameters, such as grid search, random search, and Bayesian optimization. In general, hyperparameter optimization can be described as $x_{opt} = \underset{x \in X}{argmin}\, f(x)$, where x_{opt} is the optimal set of x. X is the domain of the possible hyperparameters.

Grid Search

Grid search is a popular strategy used for hyperparameter tuning. The technique is applied by using a preset structure of parameters to be tested and evaluating the performance of the trained models against different combinations. The model with the best performance, often in terms of validation accuracy, is considered to be the best. A downside of grid search

is the quantity of possible interchanging combinations. By adding one more variable, the combinations expand exponentially, thus increasing time-consumption. Using grid search consequently becomes impractical in many circumstances.

Random Search

Random search is another strategy for hyperparameter optimization and is often preferred over grid search. J. Bergstra et al. has shown through research that when training DNN, random search finds the best models in most cases, and in a shorter time compared to the use of grid search [26]. The method is implemented by defining a distribution of desired hyperparameters and randomizing the values on each combination.

Bayesian Optimization

Bayesian optimization is an approach to hyperparameter optimization by using previous observations of *f*, or performance *P*, for some models *M*, to determine a new set of hyperparameters Θ to utilize *f* or *P* [27]. The intention is to use a more sophisticated and automatic method compared to grid search and random search, to find the optimal hyperparameters in less time. Bayesian optimization is the idea of making a probability model of an objective function *f*, selecting the most promising hyperparameters, and evaluating the result in the true *f*.

The technique uses a description of *f* as a Gaussian process to return a mean and variance of a normal distribution over possible values of *f*. Pseudo-code for Bayesian optimization is shown in Algorithm 2 and reprinted from research published by P. I. Frazier [27].

Algorithm 2 Pseudo-code for Bayesian optimization

Place a Gaussian process prior on *f*
Observe f at n_0 points according to an initial space-filling experimental design. Set $n = n_0$.
while $n \leq N$ **do**
  Update the posterior probability distribution on *f* using all available data
  Let x_n be a maximizer of the acquisition function over x, where the acquisition function is computed using the current posterior distribution. Observe $y_n = f(x_n)$.
  Increment n
end
Return a solution: Either the point evaluated with the largest f(x), or the point with the largest posterior mean.

2.2.13 PyTorch

PyTorch is an open-source framework used for machine learning and deep learning, designed for the programming language Python, and based on the library Torch. The framework was developed and is maintained by Facebook AI Research, but has many other contributors from companies and individuals. PyTorch is, among others, used by Stanford

University, IBM, Apple and Siemens for education, research and business. PyTorch is primarily a tool built for handling tensors. Tensors are similar to multidimensional arrays, which are well suited for handling various data, such as images. With the use of PyTorch, tensors can easily and efficiently be moved back and forth from the CPU to the GPU.

2.3 Performance Evaluation

Performance evaluation is the process, through which analysis is performed, to determine the best performing model. Often data validation sets are evaluated to tune hyperparameters. The validation data set contains ”unseen” data, i.e., any data not used to update the weights. This performance-evaluation can then be used as the basis for further developments. Evaluating performance is not straightforward and largely depends on the problem.

2.3.1 Confusion Matrix

One of the most used methods to evaluate the performance is the confusion matrix. The confusion matrix gives scores for each class/label that is present in the problem. In Figure 2.11, we see an example of a confusion matrix for a two-class problem. For example, one class is labeled one and the other zero. True positive corresponds to how many actual ones have been predicted as one, whereas true negative corresponds to how many actual true zeros have been predicted as zero, whilst false positive corresponds to how many true zeros have been predicted as one, and false negative corresponds to how many ones have been predicted as zero.

With the confusion matrix forming the framework, it is possible to calculate different scores of interest. Which performance metrics that give a good description, depends on the actual problem. Precision for a class tells the ratio for how many samples are correctly predicted of the total number of predicted samples representing the class. Recall describes how many samples are correctly classified of the total number of true labels for the specific class. F1-score is a combination of precision and recall and is often used as a measurement when the data is unbalanced, meaning one or more classes are over-represented compared to other classes.

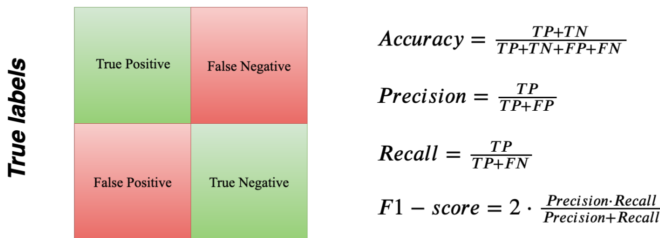


Figure 2.11: Confusion matrix and formulas used to calculate performance.

2.3.2 Dice Coefficient

The Dice coefficient, or Dice score, is a popular metric for evaluating performance in problems within object detection in images and image segmentation when comparing an output of some algorithm with the ground truth. The dice coefficient in image processing can be seen as a ratio of how much an image A overlaps another image B. This is done by calculating the intersection between A and B, multiply with 2, and divide by the total number of pixels. In binary segmentation tasks, the Dice coefficient and the F1-score are the same. In multiclass segmentation, the dice score can be calculated for each class separately. The range of the index goes from zero to one. If the score is zero, there is no overlap, and if the score is one, the images overlap completely. The calculation of the Dice score is seen in Equation 2.9

$$Dice(A, B) = 2 \cdot \frac{|A| \cap |B|}{|A| + |B|} \tag{2.9}$$

Equation 2.9

2.3.3 The Jaccard Similarity Index

The Jaccard similarity index, also known as *Intersection over Union score (IoU)*, is another metric often used when working with image segmentation problems. The calculation is similar to the calculation of the Dice coefficient, except that when using IoU, the true positives are only counted once. The range of the index goes from zero to one, where a higher number gives a better performance, the same as the Dice coefficient. The calculation of the Jaccard index is seen Equation 2.10

$$Jaccard(A, B) = \frac{|A| \cap |B|}{|A| \cup |B|} \tag{2.10}$$

Equation 2.10

2.4 Preprocessing

2.4.1 Data Normalization

Before feeding images to the input of a DNN, it is common practice to normalize the input images. One way of doing this is to calculate the mean and standard deviation for the values of each pixel in the total number of samples in the training set. In the next step, the mean is subtracted from the pixel-values and divided by the standard deviation in each image when it is uploaded. The purpose is to scale the images to a common distribution, making it easier for the network to find patterns and extract relevant features. The convergence of the loss in the training process will by this improve. The distributions of the total samples are often made as a Gaussian distribution with zero mean. The equation for normalizing a pixel is shown in Equation 2.11

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}} \tag{2.11}$$

Equation 2.11: Equation for normalization of a pixel in an image. The output y is the result of subtracting the mean for the total number of pixels from the pixel x and divide by the standard deviation.

2.4.2 Data Augmentation

Data augmentation is a set of operations that are used in DNN's for making the most of the available data set. One problem we might encounter is that the training-data appear very similar. An example can be when we use a CNN to classify two different types of cars in a set of images. One of the cars is always faced to the left, and the other car is always faced to the right. Even though the data set is large, the network will have difficulties classifying the cars in unseen images. The vehicles might appear faced in new directions or positions.

Another problem could be if we have a data set of small size, where the potential for overfitting is large. Data augmentation can be used to increase the data set by modifying the original data to appear in different ways. The use of data augmentation has shown to improve the performance in multiple problems [28]. Commonly used operations are to rotate, flip, crop, translate, mirror, or scale images. Besides, a method called elastic deformation has shown to improve models [29]. Elastic deformation is a technique for making a change of shape and contours of objects in an image.

Two different methods of data augmentation are often used. The first is to do offline data augmentation before we start to train the DNN. The augmented data is added to the original data, and we have increased the data set. The other method is called online data

augmentation. The technique is to do randomized augmentation of the samples during training, for each batch that is loaded.

Chapter 3

Data Material

The Department of Cardiology at Stavanger University Hospital gave the data material used in this thesis. The content consists of LGE-CMR images of patients who have been affected by MI and associated segmentation masks made manually by a cardiologist. CMR was performed using 1.5 T Phillips Intera R 8.3, with a pixel size of typical $0.8 \times 0.8mm^2$, slices of $15mm$ thickness, without interslice gaps. The images were given as DICOM-files (*Digital Imaging and Communication in Medicine*), in 512 x 512 resolution. The DICOM-format is the standard format for handling and working with digital medical images. The segmented masks were given as MAT-files and contained pixel coordinates of the manually marked *regions of interests (ROIs)*. The marked areas were the myocardium, endocardium, epicardium and the myocardial scars. For each patient, there were taken images at four different points; after one day, one week, one month and one year. The images used were MRI-images taken after one month and after one year because the scars have not the permanent shape after one week. LGE-MRI slices of a patient, taken one year after the MI, can be seen in Figure 3.1. In Figure 3.2, it is shown one slice, and the corresponding masks of the myocardium and the scar tissue.

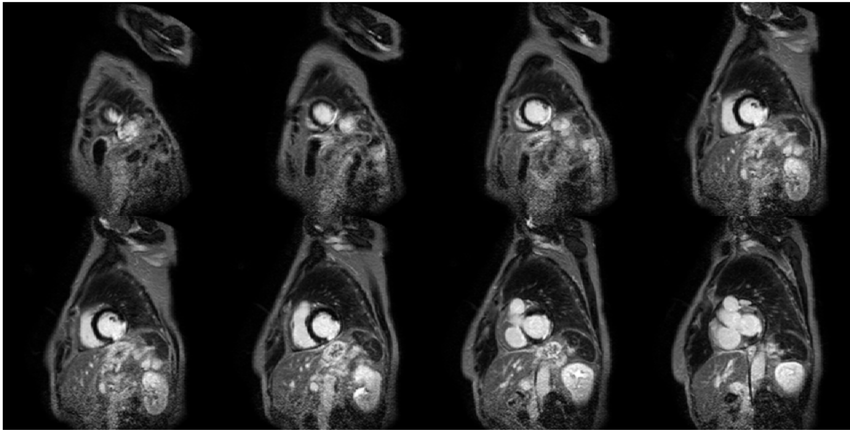
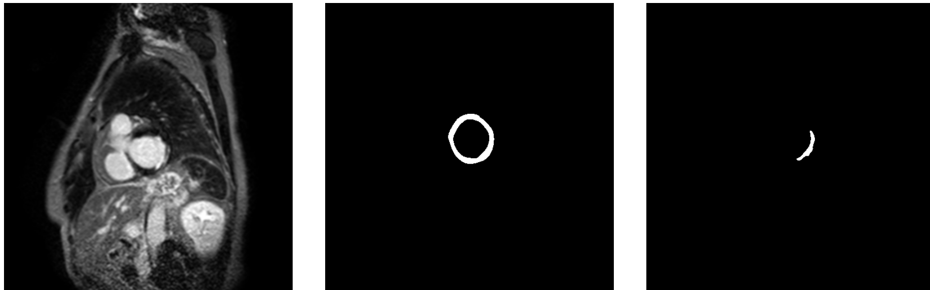


Figure 3.1: LGE-MRI slices taken of the heart of one patient with a myocardial scar. The images are showing the heart from different depths. The blood pool is the circular and bright area approximately in the middle of a slice. The myocardium is the dark area surrounding the blood pool. The myocardium in a healthy subject would have appeared as a complete ring, but a damaged myocardium has a bright part visualizing the scar tissue. The myocardial scar tissue appears bright due to reduced blood flow.



(a) Original image of one patient. **(b)** Mask of the myocardium. **(c)** Mask of the myocardial scar.

Figure 3.2: The figure shows one slice of the myocardium of a patient, with the corresponding masks of the myocardium and the scar tissue.

Data from 272 patients were used, with an average of approximately nine image-slices of the myocardium for each patient. The total number of images was 2523. The data was split in a training set *X*, a validation set *Y*, and a test set *Z*, in a the ratio 80 %, 11 %, and 9%, respectively, as seen in 3.1

One of the significant issues explored during this project is how to handle the unbalanced distribution of the classes in the images. The background is overrepresented by appearing as 96.8% of the total number of pixels, averaged over all images. The healthy

myocardium is represented by 2.7% of the pixels and the myocardial scar as 0.5%.

Table 3.1: Split of data set

Distribution of data			
Dataset	Patients	Images	Percentage
Training-set, **X**	214	2006	80%
Validation-set, **Y**	28	273	11%
Test-set, **Z**	30	244	9%
Total	*272*	*2523*	100%

Images of the same patient were used exclusively in either dataset X, Y, or Z. It is of importance to do this separation for avoiding a biased evaluation of the models. If one uses images from the same patient during training and validation, it might learn features specific for the training-set rather than features in the general distribution of data.

Chapter 4

Proposed Methods

4.1 Overview of the Proposed Methods

The proposed method is to divide the experimental part into two. The first experiment will be to use binary masks as a two-class segmentation problem. The masks will then be the segmented myocardium without scar tissue. The second experiment will be to approach the problem as a multiclass segmentation. The myocardium is then split into healthy tissue and scar tissue.

When doing segmentation of the myocardium, the interest is to locate both the healthy myocardium and the myocardial scar. Finding the myocardium is one step towards finding both, and there exist post-processing methods for this. In this thesis, the main objective is to find the epicardium and endocardium contours. By approaching the problem in two different ways, it might be precise, which is the most promising.

In Figure 4.1, we can see an overview of the layout for the process. The process is split into three parts; preprocessing, training and testing

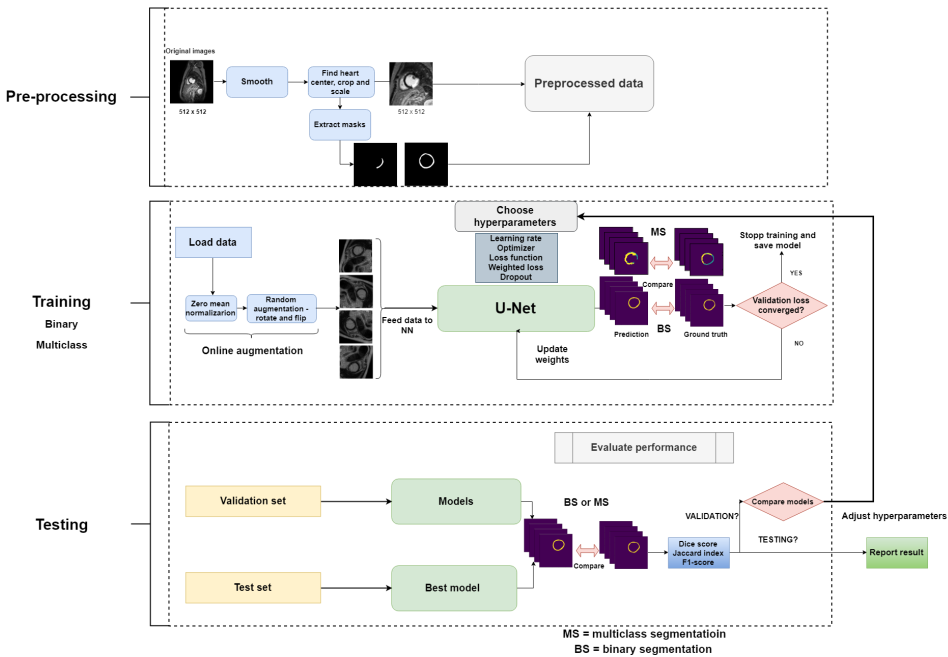


Figure 4.1: Overview of the approach for myocardial segmentation using DNN.

4.2 Preprocessing

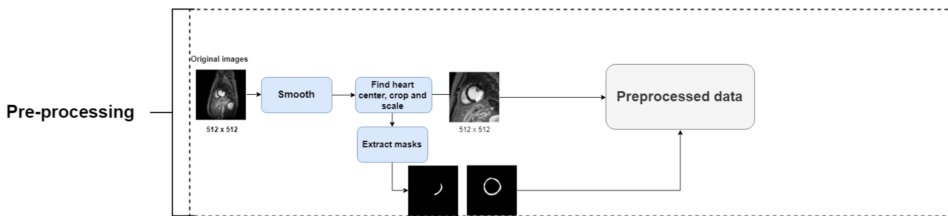


Figure 4.2: System overview of the preprocessing.

The preprocessing of the data was done in MATLAB and Python. The University of Stavanger provided some of the algorithms that were used; these were produced during the research by Engan et al.[7]. Additionally, algorithms were made by the author, used in combination with the provided algorithms. It was necessary to apply to preprocess on the given data for the implementation in the DNN. The images were converted from DICOM-format to PNG *Portable Network Graphics*, which allowed visualizing the images during the preprocessing easily. An alternative could have been to save the data as MAT-files. These files take less storage on the disk and give faster uploading to the program made for DNN in Python, but cannot be visualized instantly as images. Due to the use of powerful

hardware and relatively small size of the data, the consideration was that the size of the files did not have a significant impact on the computational time when training the models.

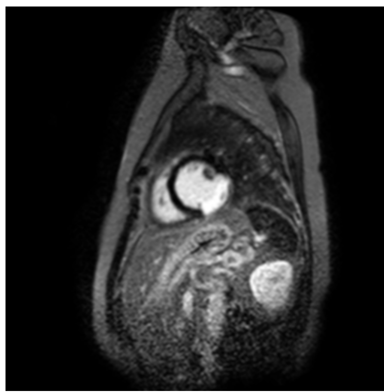
Algorithms for finding the heart center and for cropping the images and masks were used to get the data on the desired format. The intention was to make the images concentrate around the myocardial muscle, removing information that is irrelevant for the segmentation procedure. In this process, the manually marked coordinates of the segmentation masks were combined with the images to create binary masks of the myocardium, endocardium, epicardium, and the myocardial scar. The resolution of the images after the cropping was 232 x 232. To make the images applicable to the created NN architecture, they were rescaled to resolution 512 x 512. Due to noise in the original images, a Gaussian filter was used for smoothing. See Algorithm 3 for pseudocode and Figure 4.3 for the preprocessed image and masks.

Algorithm 3 Preprocessing images and masks

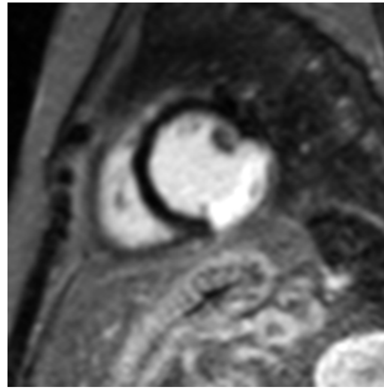
Initialization:
1: *Image, I*
2: *Patient, p*
3: *Slice, j*

forall p **do**
    forall $I^{j}_{p,DICOM}(x,y)$ **do**
        1: *Crop and scale* $\rightarrow I^{j}_{p}(x,y)_{C,S}$
        2: *Gaussian smoothing (kernel = 2)* $\rightarrow I^{j}_{p}(x,y)_{S}$
        3: *Extract masks* $\rightarrow Myocard^{j}_{p}(x,y)$, $Scar^{j}_{p}(x,y)$
    end
end

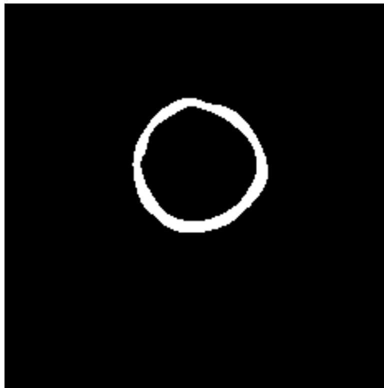
Result: $I^{j}_{p}(x,y), Myocard^{j}_{p}(x,y), Scar^{j}_{p}(x,y)$



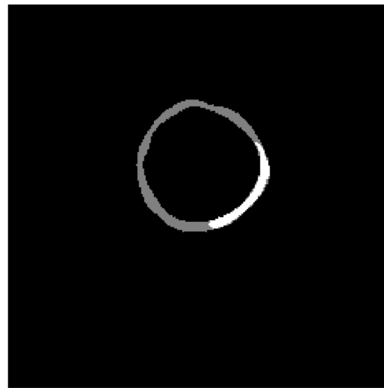
(a) Original image.



(b) Preprocessed image.



(c) Preprocessed binary mask.



(d) Preprocessed multiclass mask.

Figure 4.3: One LGE-CMR slice of a subject, showing the original image, the preprocessed image, and the binary- and multiclass masks. All images are in grayscale format. The pixel values in the binary mask are zero for the background and on for the myocardium. The pixel values in the multiclass mask are zero for the background, one for the healthy myocardium and two for the myocardial scar.

4.3 Training

This section presents an overview of the training of the models. An illustration is shown in Figure 4.4

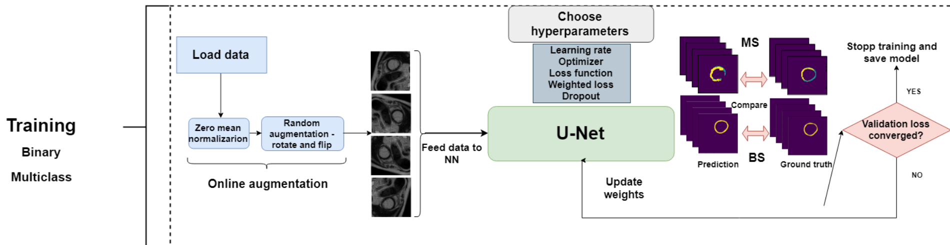


Figure 4.4: System overview of the training process.

4.3.1 Data Loading

During the training, there was performed zero-mean normalization and online data augmentation on the training-set, X. The applied data augmentation was to randomly rotate or flip the images and masks with a ratio of 0.3. With a batch size of 10, this gives an average of 3.3 augmented samples in each batch.

4.3.2 Network Architecture

U-Net

The architecture that has been used in this thesis is an FCNN called U-Net. The network was developed at the University of Freiburg in 2015 and was designed for the biomedical image segmentation of neuronal structures [5]. The network has become popular for many applications in segmentation problems due to the well-proven performance in terms of precision and training- and segmentation-time [6, 4]. Compared to several other network-structures used on segmentation tasks, the U-Net does not need as much training samples. For this reason, the U-Net is often used in problems working with biomedical images; the data available is often limited.

The architecture of the U-Net can be seen in Figure 4.5. The network consists of a contracting part (reducing the resolution), and an expansive part (increasing the resolution). The contracting part of the network is often referred to as the encoder and the expansive part as the decoder. The encoder is capturing the desired features, and the decoder enables the localization of the features.

The U-Net consists of convolutional layers with 3x3 convolutional operations followed by batch normalization and a ReLU activation function (green arrows). 2x2 max-pooling layers are used for downsampling (red arrows). In the expansive part of the network, up-convolution is used (blue arrows). The yellow arrows indicate the copies of feature maps from the contracting part that are merged with the feature maps with the same size at the expansive part. This merging operation is used for making the network better at

determining where the features of interest are located. The total number of convolutional layers is 23, and the number of max-pool layers is four. The U-Net does not have any fully connected layers.

The original U-Net has been slightly modified in the program developed in this thesis. The resolution is 512x512 for both the input and the output. The modification is done for practical reasons and was performed by using padded convolutions. In the multiclass segmentation, the output is changed to consist of three channels. Additionally, experiments are done with implementing dropout in the architecture (see section 4.3.3).

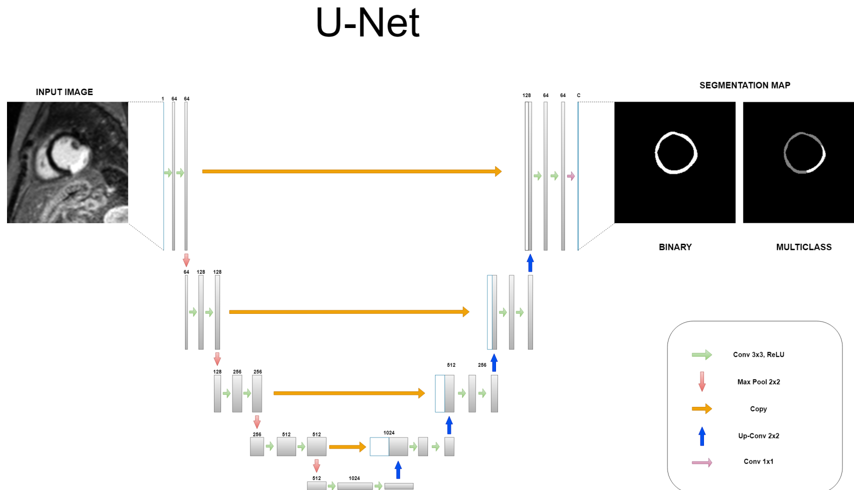


Figure 4.5: The figure is illustrating the U-Net architecture. The network in the figure takes a grayscale image as input and gives a segmentation map of two or three channels as output.

4.3.3 Choice of Hyperparameters

Adjustment of the hyperparameters has been made experimentally by training multiple models with different combinations of optimizers, loss functions, learning rates, weighted loss, and dropout. The performance for each model was evaluated by testing the model on the validation set.

Grid search and random search were used as strategies for hyperparameter optimization in the experiment with binary segmentation. In the multiclass segmentation, grid search and Bayesian optimization were used.

Optimizers and Learning Rate

The optimizers that have been used are ADAM and SGD. The momentum was set to 0.9 with the use of SGD. The learning rates were set in the initial grid search, and further adjustments of the learning rates were performed based on the observed results.

Loss Functions

The cross-entropy loss function and Dice loss have been used in the experiments in this thesis. Experiments have been performed to evaluate the functionality and performance. Weighted cross-entropy loss (see section 2.2.7) was used to train some models, to penalize miss classifications of the myocardial muscle and the myocardial scar. On average, 96.8% of the pixels are background, 2.7% pixels are myocardium, and 0.05% pixels are the myocardial scar, making the dataset highly unbalanced. A weight was assigned to each class in the experiments using cross-entropy loss. The weight is multiplied by the contribution of the loss for the specific class. A larger weight means in practice that the misclassifications of a given class are penalized more, and provides a higher contribution to the total loss. The equation for the weighted cross-entropy is shown in Equation 4.1, and the equation used to calculate the weights is presented in Equation 4.2.

$$loss(x, class) = w[class]\left(-x[class] + log\left(\sum_j e^{(x[j])}\right)\right) \tag{4.1}$$

Equation 4.1: Calculation of weighted loss

$$\begin{gathered} c1 = 0.968, \qquad c2 = 0.027, \qquad c3 = 0.005 \\ scale \in [0, 1] \\ w[1] = \tfrac{scale}{c1} \\ w[2] = w[3] = \tfrac{1-scale}{c2+c3} \end{gathered} \tag{4.2}$$

Equation 4.2: Calculation of the weights used in the cross-entropy loss function. c1, c2, c3 is the ratio of the different classes represented in the ground truth masks. w[1], w[2], and w[3] are the weights calculated for each class. The scale is set to a decimal between 0 and 1. If the scale is large, the penalization of miss classifying class 2 and class 3 gets bigger. For the binary problem two weights are calculated, one for the background and one for the myocardial muscle (c1 = 0.97, c2 = 0.03)

Dropout

Dropout was implemented in the last experiments of the binary segmentation, to explore if the technique could reduce overfitting, and increase the performance. Dropout was implemented in eight convolutional layers, after batch normalization and ReLU. The new hyperparameter, dropout rate, was set to be a decimal between 0.05 and 0.5, based on observations made by A. Krizhevsky et al. [22], and X. Li et al. [25]. The enhanced architecture can be seen in Figure 4.6.

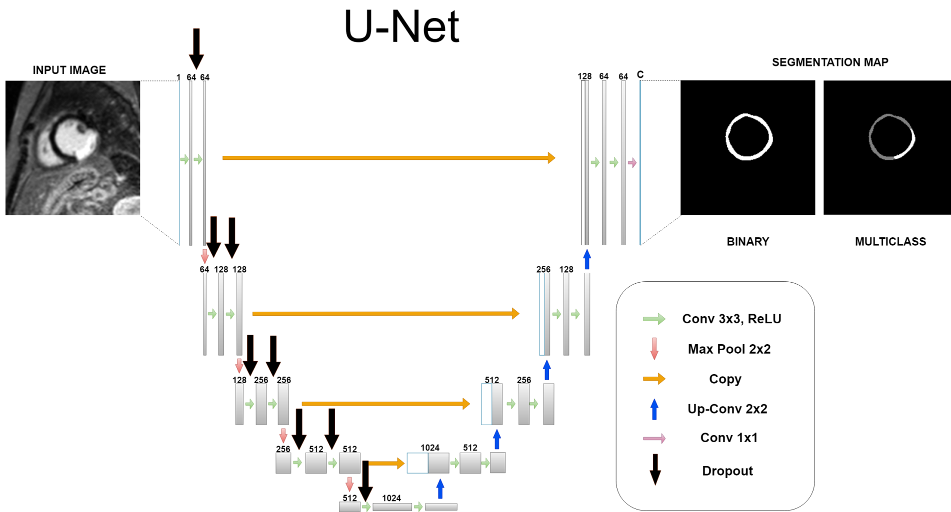


Figure 4.6: U-Net with dropout

4.4 Testing

An overview of the process of testing is shown in 4.7. To evaluate the produced models, they are tested on the validation data set, and the performance is compared. The performances of the models are used to choose the hyperparameters for the proceeding experiments.

The best-obtained model is evaluated on the test data set. The test data set is used to provide an unbiased evaluation of the model. Although the validation data set is not used to update the weights, it is used for finding the best model. The choice of hyperparameters is based on the performance measured on the validation data set. For that reason, using the validation set to report the final result, is not valid. The test data set consists of unseen data with the same representation as to the data used for training. It is only used once, on a completely trained model.

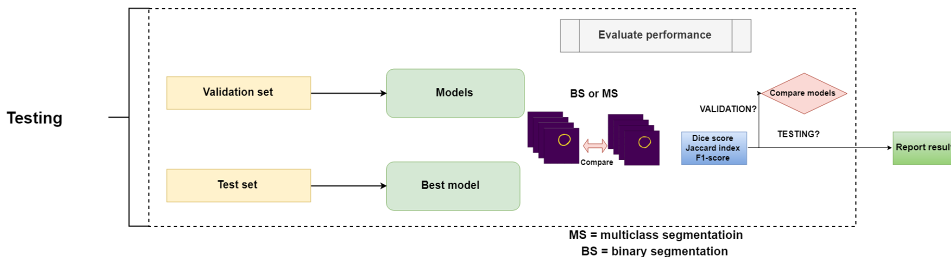


Figure 4.7: System overview of the testing process

4.5 Implementations

Matlab was used as the software for the preprocessing part [30]. Algorithms produced by K. Engan et al. [7] was combined with self-made code. The programming language used for designing the DNN was python in combination with the DL framework PyTorch [31] [17]. The developed program was made as a combination of embedded, external, and self-made code. An overview of the implementation of the methods can be seen in Table 4.1

Table 4.1: Implementation of algorithms[1]

Implemented algorithms			
Method	**Embedded**	**External**	**Self made**
Preprocessing		X	X
Load data		X	X
Training		X	X
Data augmentation			X
Optimizers and loss functions	X	X	
U-Net		X	X (modified)
Testing			X
Performance metrics		X	X

[1]Some code was reused or inspired by ugent-korea from their project *unet-pytorch-segmentation*, found on GitHub, and used under the MIT lisence.

Chapter 5

Experiments and Results

This chapter presents experiments and the results obtained in this thesis. The objective was to develop a DNN model for myocardial segmentation, and find hyperparameters that give the best performance. Binary- and multiclass segmentation have been explored.

Results from the best found binary- and multiclass models are presented, and the choices of hyperparameters are discussed. The verification of the best performing model is done by testing the model on the test data set.

The presentation of the results in the graphs is given as pixel-wise accuracy (see equation *accuracy* in Figure 2.11), logged during validation in the training process. Additionally, histograms are presented for the saved models tested on the validation set. The performance is measured as Dice score.

5.1 Finding the Best Model

For developing the best DNN model, multiple combinations of hyperparameters have been tested. Due to the many possible combinations, some of the variables were fixed. The U-Net architecture (see section 4.3.2) has been used throughout all experiments, with and without the implementation of dropout. All experiments were run with a maximum of 100 epochs. From experience, the models did not further improve after this limit. The batch size was kept consistent top ten, due to memory limitations on the GPU.

Optimizer, loss function, learning rate, weighted loss, and dropout were tuned. The best model for each set of hyperparameters was saved. The training was stopped when the validation loss did not converge any further. The patience was set to ten, meaning to finish the training if there has not been any improvement in the last ten epochs. The weights were saved only if the present validation loss was lower than the previous lowest. A full overview of the hyperparameters can be seen in table **??**

Dice similarity coefficient and the Jaccard similarity index was used to calculate the performance in the binary segmentation. In the multiclass segmentation, F1-score for class one and class two were calculated as well.

Combinations of hyperparameters		
Hyperparameter	**Fixed**	**Variants**
Learning rate	No	$[1 \times 10^{-5} - 2]$
Optimizer	No	ADAM and SGD with fixed momentum (0.9)
Loss function	No	Cross-entropy loss and Dice loss
Weighted loss	No	Scaled by class
Drop rate	No	[0.05-0.5]
Network architecture	No	With and without dropout
Data augmentation	Yes	Random roation, horizontal and vertical flip
Batch size	Yes	10
Epochs	Yes	Hard limit of 100 epochs
Patience	Yes	10 epohs

5.2 Experiment One - Binary Segmentation

The first experimental layout was to use a binary approach to the segmentation task. The ground truth masks were made by merging the mask with the healthy myocardium with the mask representing the scar tissue. With this, the pixel values in the ground truth masks are divided into two classes; background (class one) and myocardium (class two). Background-pixels are set to values of zero value, and the myocardium-pixels are set to values of one. Figure 5.1 visualize a ground truth mask used for binary segmentation.

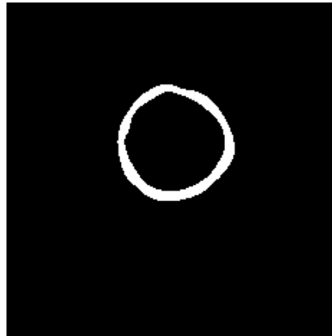


Figure 5.1: Binary mask of the myocardium

5.2.1 Grid search

Grid search was initially used to search for combinations of learning rates, optimizers, and loss functions, to get guidance for choices in further experiments. The range of learning rates was set from 1×10^{-5} to 2 with the use of SGD. For the ADAM optimizer, the range was set from 1×10^{-5} to 0.1. The initialization was based on recommendations presented by Y. Bengio [32] and I. Goodfellow et al. [12]. The weighted loss was used with scales of 0.2, 0.5, and 0.8. The calculation of weights is shown in Equation 4.2. Table 5.1 shows a review of the used hyperparameters.

Table 5.1: Hyperparameters when doing grid search

Combinations of hyperparameters			
Optimizers	**Learning rate**	**Loss functions**	**Scaling**
ADAM	1×10^{-5} - 0.1	Cross-entropy and Dice loss	0.2, 0.5, 0.8
SGD	1×10^{-5} - 2	Cross-entropy and Dice loss	0.2, 0.5, 0.8

Cross-Entropy Loss

Experiments with cross-entropy loss in combination with ADAM and SGD were performed. In Figure 5.2, we can see the curves of the validation- loss, and accuracy for the two best performing models for each of the optimizers.

The results of experiments with weighted loss gave significantly worse results with the tested parameters. The performance of all tested combinations with cross-entropy as loss function, with and without the use of weighted loss, can be seen in Appendix A.1.

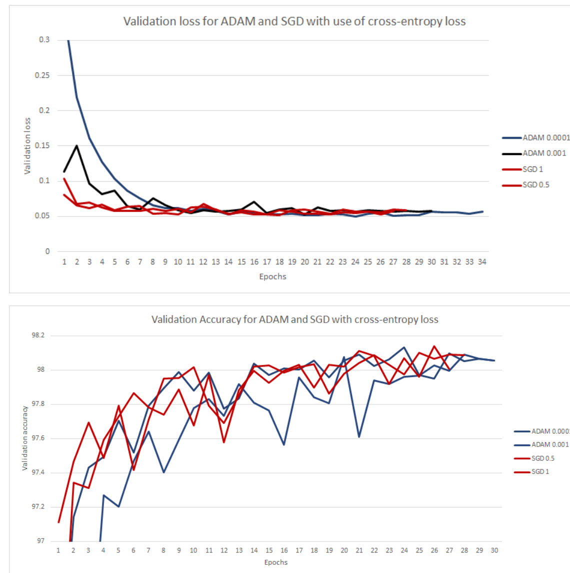


Figure 5.2: Validation- loss and accuracy when using cross-entropy loss in combination with ADAM and SGD. The plot shows the two best models found for both optimizers.

Dice Loss

The same experiments, as described in the previous section, were performed with the use of the Dice loss function. All other hyperparameters were kept the same. In Figure 5.3, plots of validation- loss, and accuracy during training are presented.

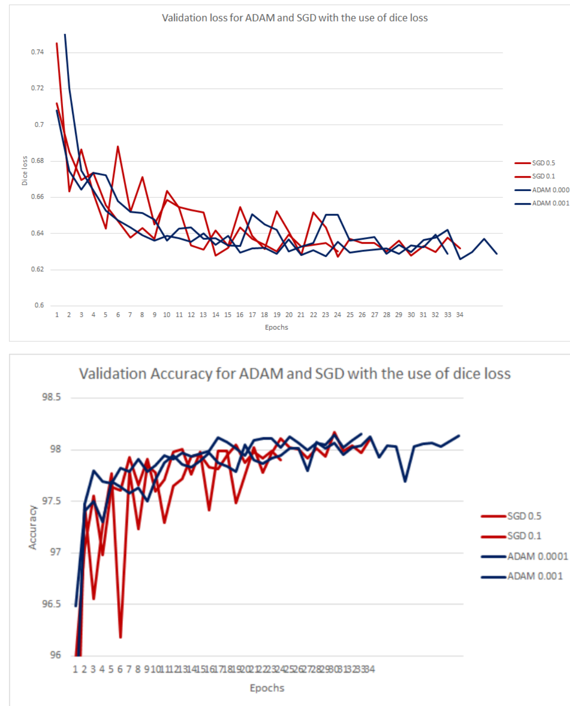


Figure 5.3: Validation- loss and accuracy when using Dice loss in combination with ADAM and SGD. The two best models found are shown.

Comparisons Between Cross-Entropy Loss and Dice Loss

The results of the grid search shows a tendency that the Dice loss function performs slightly better than the cross-entropy loss function in average. This can be seen in Figures 5.4 and 5.5.

The Dice loss functions give better results for both the use of ADAM and SGD. These results led to the choice of dropping further experiments using cross-entropy loss in experiment one.

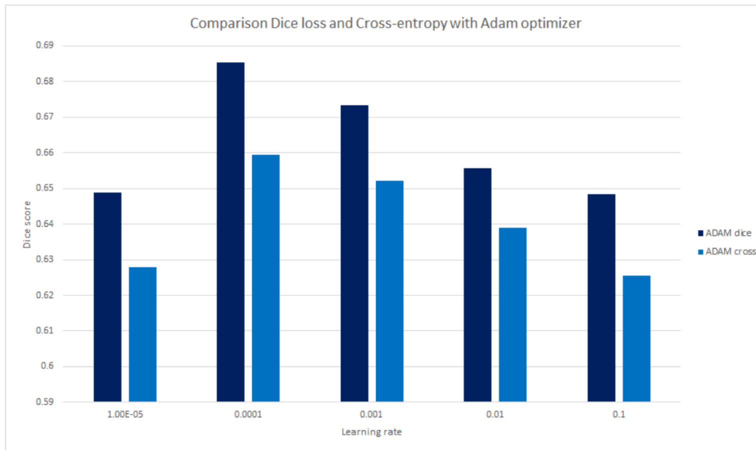


Figure 5.4: Dice score for ADAM with the use of cross-entropy- and Dice loss.

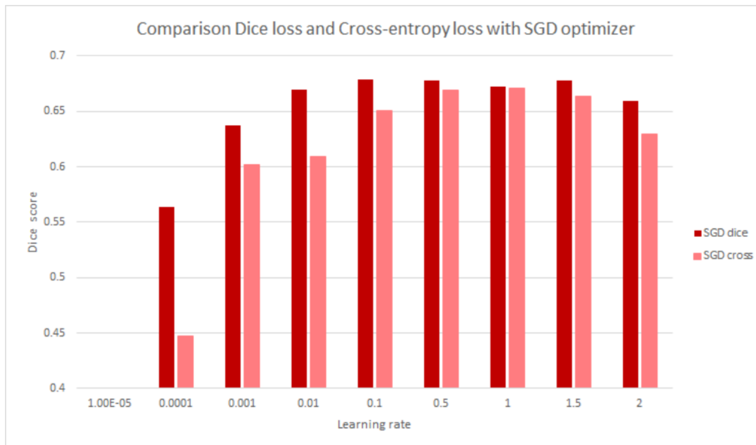


Figure 5.5: Dice score for SGD with the use of cross-entropy- and Dice loss.

There is no evident optimizer that gives the best result in the initial grid search. SGD performs better with higher learning rates (0.1 to 1.5), while ADAM performs better with lower learning rates (0.0001 to 0.001). From the observations made, it was decided to make use of both ADAM and SGD in the following experiments. A comparison of Dice score for SGD and ADAM with the use of Dice loss can be seen in Figure 5.6.

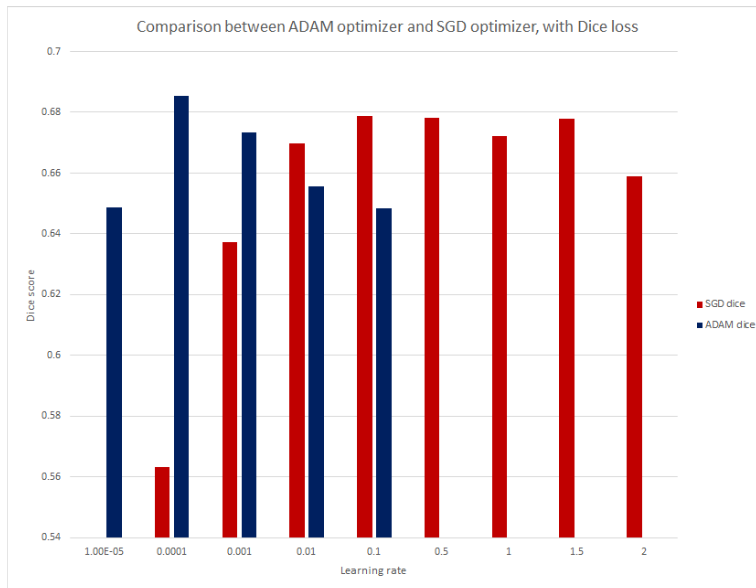


Figure 5.6: Comparison between ADAM and SGD with the use of Dice loss.

5.2.2 Random Search

Table 5.2: Choice of hyperparameters based on grid search

Hyperparameters used in random search		
Optimizers	**Learning rate**	**Loss functions**
ADAM	Rand [$1 \times e^{-5}$ - 0.005]	Dice loss
SGD	[0.1 to 1.5]	Dice loss

Experiments with the random search were done to explore the possibility of finding better performing models. The random search was performed with Dice loss in combination with ADAM and SGD. Bar graphs of the performance are presented in Figures 5.7 and 5.8. By looking at the results, we see that the use of ADAM again performs slightly better compared to the use of SGD, in terms of average Dice score and best Dice score. Notice that scaling of the axis differs, which might give the impression that the models trained with SGD are more stable in performance. The results for the best performing model for both optimizers can be seen in Table 5.3.

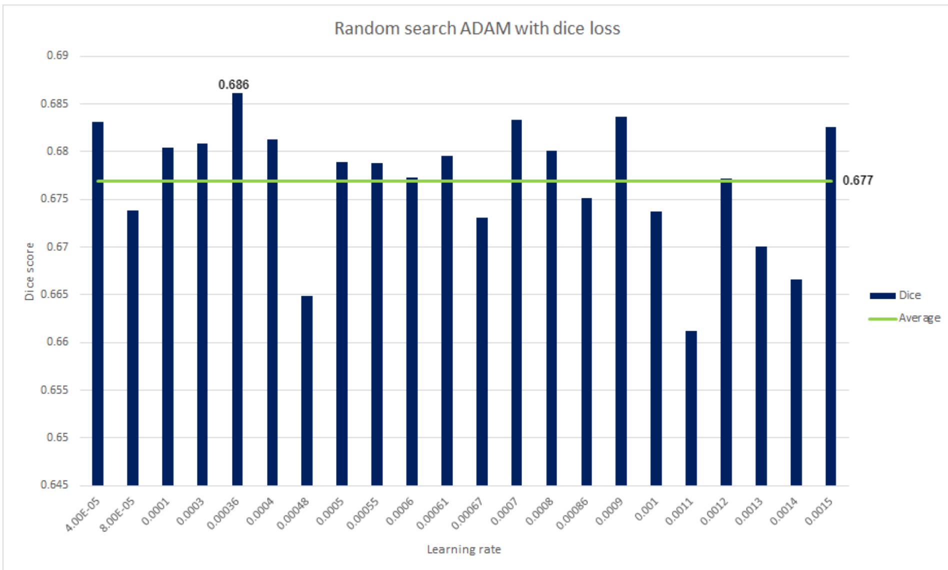


Figure 5.7: Random search with ADAM

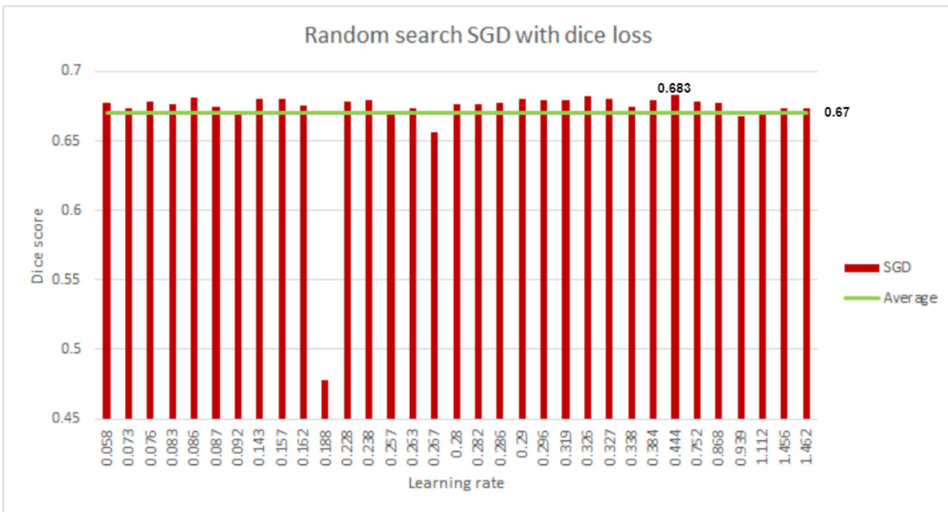


Figure 5.8: Random search with SGD. In the graph, we can see an outlier, in the model trained with learning rate 0.188. The initialization of weights is most likely to be the reason for the poor performance. The consequence is that the model reaches a local minimum.

Table 5.3: Best performing model for ADAM and SGD with random search

	ADAM	**SGD**
Loss function	Dice	Dice
Learning rate	$[4 \times 10^{-5} - 0.0015]$	[0.058-1.4624]
Number of models	22	33
Best Dice	0.686 (0.18)	0.683 (0.17)
Average Dice	0.677	0.670

5.2.3 Overfitting

In Figure A.3, we can see graphs for the training process of the best model in the initial tests; ADAM optimizer, Dice loss, and learning rate = 0.0001. The model is overfitting. The validation loss converges when the training loss keeps decreasing. The same is the case for the accuracy; validation accuracy converges when the training accuracy increases. Overfitting was observed in all experiments with the use of grid search and random search, for all combinations of hyperparameters. These observations make it reasonable to apply some form of regularization to prevent the models from overfitting the training data. The technique used is dropout, which follows in section 5.2.4.

Figure 5.9: Validation- and training graphs with the use of ADAM and Dice loss, learning rate=0.0001

5.2.4 Random Search with Dropout

The final experiments for the binary segmentation were to implement dropout in the U-Net architecture, to see if the use of dropout potentially could reduce overfitting (see section 4.3.3 for a description of the enhanced architecture). The experiments were done using both ADAM and SGD, in combination with Dice loss. The determined range for the learning rates was based on observation from the previous experiments. The range of the drop rate was set to 0.05 - 0.5, upon recommendation by X. Li et al. [25] and A. Krizhevsky et al. [22]. An overview of the parameters can be seen in Table .

Table 5.4: Hyperparameters for random search with the use of dropout

Hyperparameters used in random search with dropout			
Optimizers	**Learning rate**	**Loss function**	**Drop rate**
ADAM	Rand $[1 \times e^{-5}$ - 0.005]	Dice loss	Rand $[0.05 - 0.5]$
SGD	Rand $[0.1 - 1.5]$	Dice loss	Rand $[0.05 - 0.5]$

Figures 5.10 and 5.11 presents the performance of the trained models with the use of dropout. The models trained with ADAM perform better on average than models trained with SGD. Compared to the results obtained with random search, the average performance is the same for ADAM, and slightly worse for SGD. An overview of the results is presented in Table 5.5.

Figure 5.10: ADAM with dropout

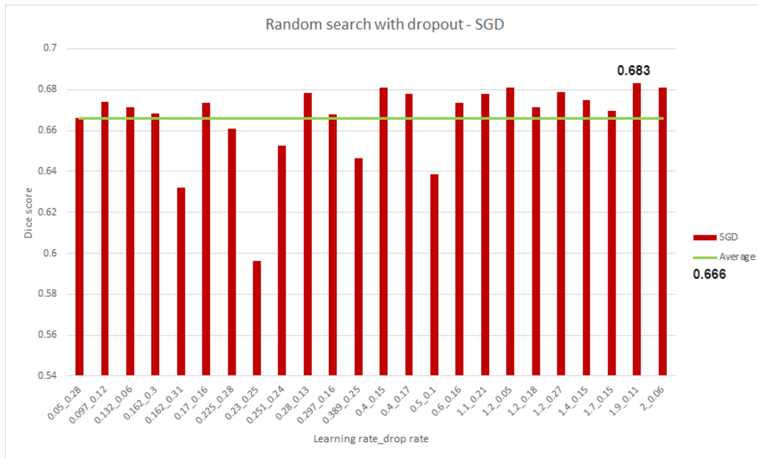
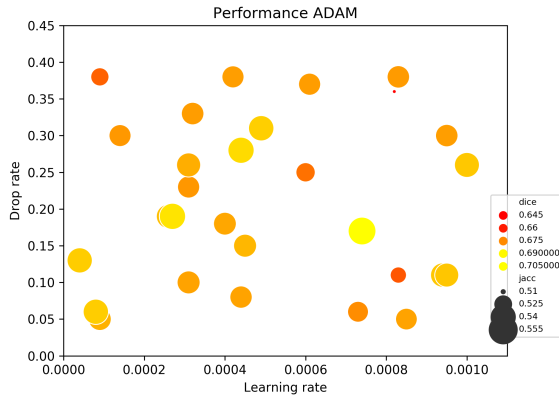


Figure 5.11: SGD with dropout

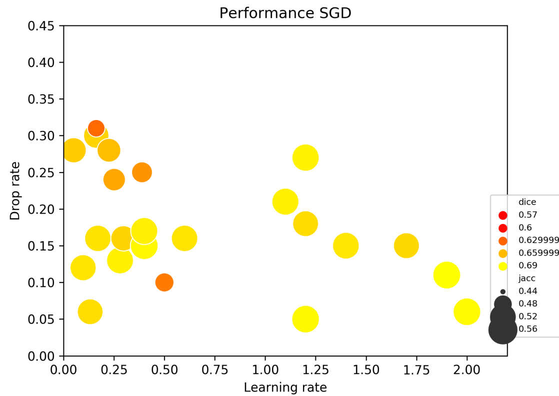
Table 5.5: Performance of models trained with ADAM and SGD the use of random search with dropout.

	ADAM	**SGD**
Loss function	Dice loss	Dice loss
Learning rate	$[9 \times 10^{-5} - 1.4 \times 10^{-4}]$	$[0.05 - 2]$
Drop rate	$[0.05 - 2]$	$[0.05 - 0.31]$
Number of models	29	24
Best Dice	<u>0.691</u> (0.17)	<u>0.683</u> (0.17)
Average Dice	<u>0.677</u>	<u>0.666</u>
Best Jaccard	<u>0.550</u> (0.175)	<u>0.539</u> (0.30)
Average Jaccard	<u>0.537</u>	<u>0.522</u>

In Figure 5.12, we can see bubble plots for evaluating the use of dropout. The learning rate is along the x-axis, and the drop rate is along the y-axis. The colors of the bubbles indicate performance measured in Dice score. A brighter color means better performance. The size of the bubbles indicates the performance measured as the Jaccard index, where large bubbles mean better performance. From observations, it is difficult to see any correlation between learning rate, drop rate, and performance.



(a) Bubble plot for ADAM optimizer.



(b) Bubble plot for SGD optimizer.

Figure 5.12: Bubble plots for evaluating learning rate and drop rate. Brighter and larger bubbles means better performance.

5.3 Experiment Two - Multiclass Segmentation

The second experimental layout was multiclass segmentation, with ground truth masks consisting of three classes; the background (class one), healthy myocardium (class two) and the myocardial scar (class three). Pixel values for class one are set to zero, pixel values for class two are set to one, and pixel values for class three are set to two. In Figure 5.13, we see one example of the ground truth masks used.

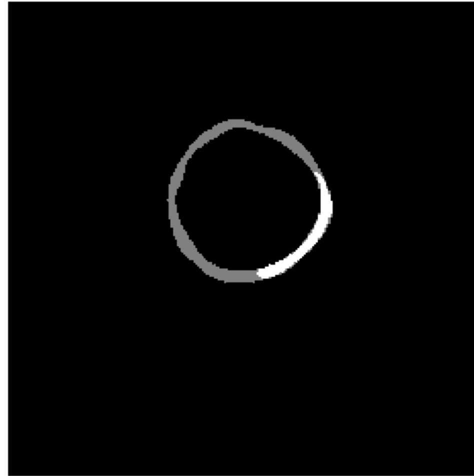


Figure 5.13: Multiclass mask of healthy myocardium and myocardial scar tissue.

5.3.1 Grid Search

Initially, grid search was used for hyperparameter optimization. ADAM- and SGD were used as optimizers, and cross-entropy and Dice loss were used as loss functions. As in experiment one, the objective was to search for the best initial combinations of these hyperparameters. Additionally, it was performed tests with weighted cross-entropy loss. The purpose was to penalize misclassification of class two and class three, due to the unbalance of the classes in the ground truth. The values of the scales were increased, based on the observations in experiment one.

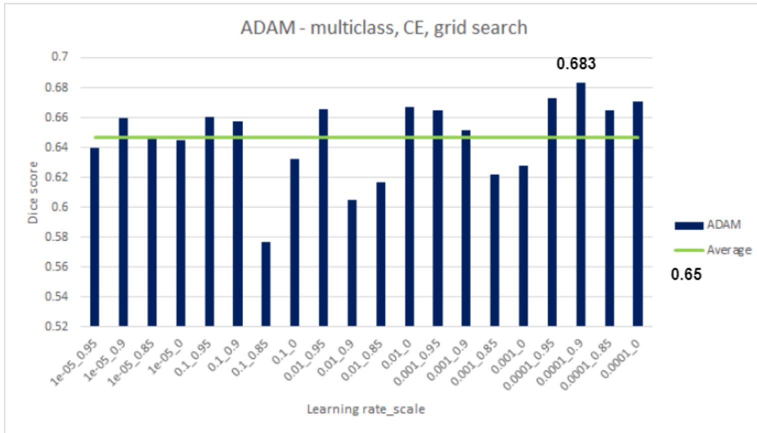
Table 5.6: Hyperparameters for grid search in experiment two

Combinations of hyperparameters			
Optimizers	**lr**	**Loss functions**	**Scaling**
ADAM	$1 \times e^{-5}$ - 0.1	Cross-entropy and Dice loss	0.85, 0.90, 0.95
SGD	$1 \times e^{-5}$ - 2	Cross-entropy and Dice loss	0.85, 0.90, 0.95

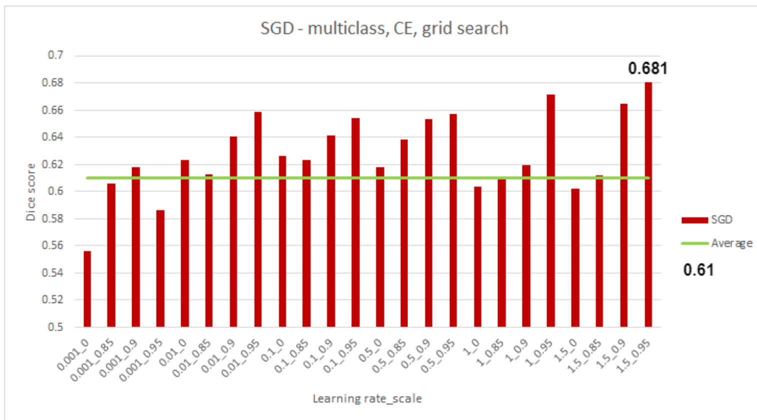
Cross-entropy Loss

Initially, experiments with the use of cross-entropy loss in combination with ADAM and SGD were performed. The results can be seen as bar charts in Figure 5.14. Two different approaches were performed to evaluate the performance. The thesis objective was to compare the binary segmentation with the multiclass segmentation. For that reason, the predicted multiclass masks were merged to form binary masks by defining the healthy myocardium and the scar tissue as one class. Dice score was used to evaluate the performance, as in the binary segmentation. Additionally, the results of multiclass segmentation were evaluated with F1-score for class one and class three.

In the bar charts in Figure 5.14, we see the results for the initial grid search. The best result performed with the use of ADAM was Dice 0.683 (0.17). The best result achieved with the use of SGD was Dice 0.681 (0.17). An overview of the results is presented in Table 5.5.



(a) ADAM and cross-entropy loss.



(b) SGD and cross-entropy loss.

Figure 5.14: Grid search - performance of the models trained with ADAM and SGD, in combination with cross-entropy loss

Table 5.7: Multiclass segmentation - Grid search

	ADAM	**SGD**
Loss function	Cross-entropy	Cross-entropy
Learning rate	$[1 \times 10^{-5} - 0.1]$	$[0.001 - 1.5]$
Scale	$[0.85, 0.90, 0.95]$	$[0.85, 0.90, 0.05]$
Number of models	20	28
Best Dice	<u>0.683</u> (0.17)	<u>0.681</u> (0.17)
Average Dice	<u>0.65</u>	<u>0.61</u>

Dice Loss

To obtain satisfactory results using Dice loss proved to be difficult with multiclass segmentation. We can see from Figure 5.15 that the DNN struggles to pinpoint classes two and three. Based on these observations, performance with Dice loss was not explored in the following experiments. The Dice loss function used the average Dice coefficient for class one and class two to calculate the loss.

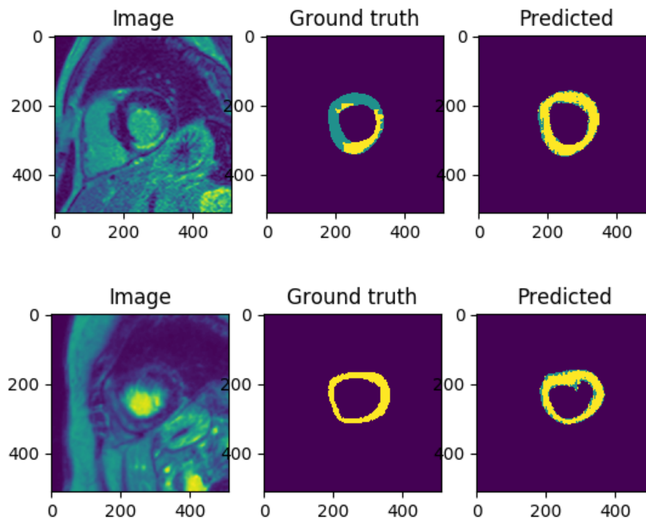


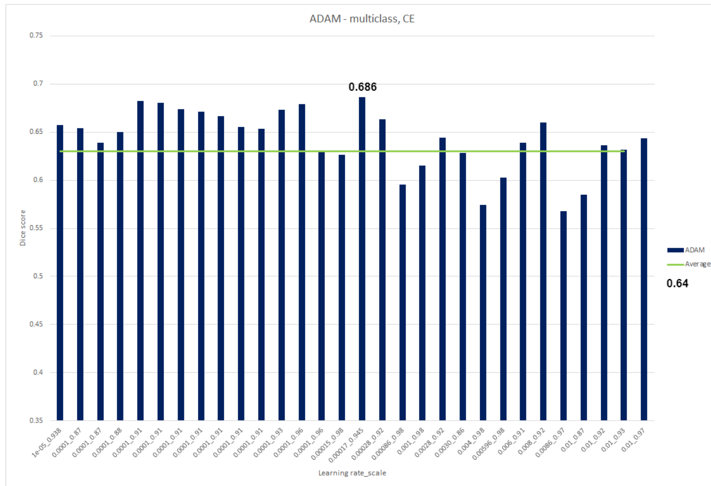
Figure 5.15: Figures from the validation of multiclass segmentation with the use of Dice loss. The predicted masks are made with a model trained with ADAM and learning rate 0.0001

5.3.2 Search Using Bayesian Optimization

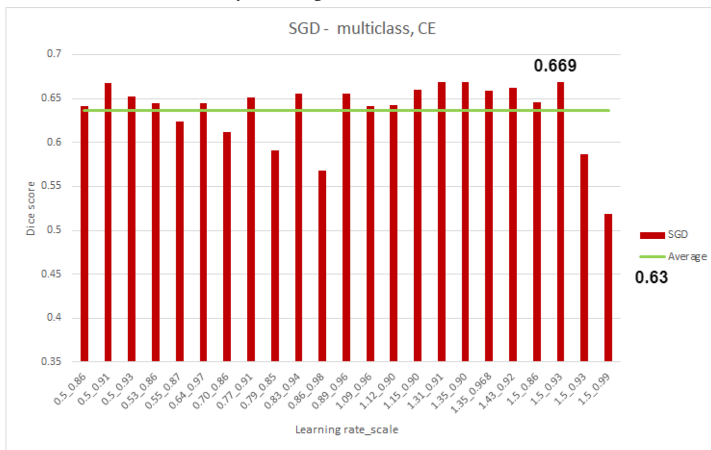
The final experiments were performed with Bayesian optimization. Bayesian optimization was implemented by using the results from grid-search as prior observations. The algorithms were created to make a probability model of an objective function made to optimize the performance. Measurements of the Dice coefficient was chosen to be optimized, by implementing a minimizing function taking (1 - Dice coefficient) as input argument.

From the search for best hyperparameters, the objective function ended training multiple models with learning rate 0.0001, and scale around 0.90, when using ADAM. With the use of SGD, the objective function ended up using learning rates around 1.30 and scale in the range of 0.90 to 0.97. The results for the experiments performed with Bayesian optimization are presented in bar graphs in Figure 5.16.

An overview of the hyperparameters used and the measured performance can be seen in Table 5.8



(a) Bayesian optimization with ADAM



(b) Bayesian optimization with SGD

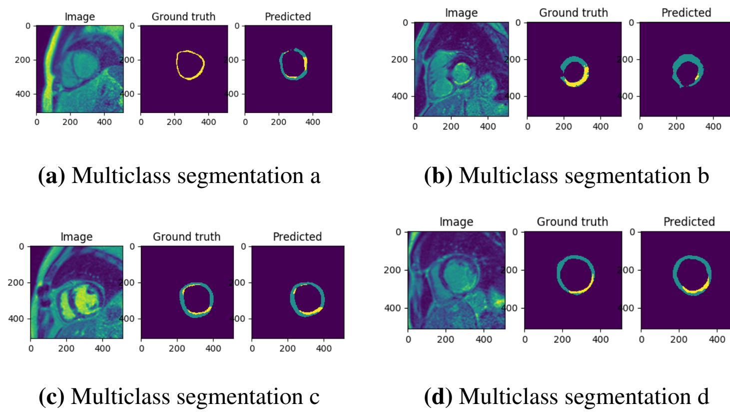
Figure 5.16: Performance of ADAM and SGD with Bayesian optimization.

Table 5.8: Results for multiclass segmentation using Bayesian optimization.

	ADAM	**SGD**
Loss function	Cross-entropy	Cross-entropy
Learning rate	$[1 \times 10^{-5} - 0.01]$	$[0.5 - 1.5]$
Scale	$[0.85 - 0.99]$	$[0.85 - 0.99]$
Number of models	30	23
Best Dice	0.683 (0.17)	0.681 (0.17)
Average Dice	0.65	0.61

5.3.3 Evaluation of Multiclass Segmentation

In Figure 5.13, it is shown samples from the predicted masks performed by the best found multiclass model. The predictions were generated by testing the model on the validation data set. As we can see, the model has difficulties distinguishing between the healthy myocardium and the myocardial scar in Figure (a), and it does not detect the scar tissue in Figure (b). In Figure (c) and (d), we see predictions where the model performs better and localizes some of the scar tissue. An overview of the hyperparameters and results for the best found model is presented in Table 5.9.



(a) Multiclass segmentation a

(b) Multiclass segmentation b

(c) Multiclass segmentation c

(d) Multiclass segmentation d

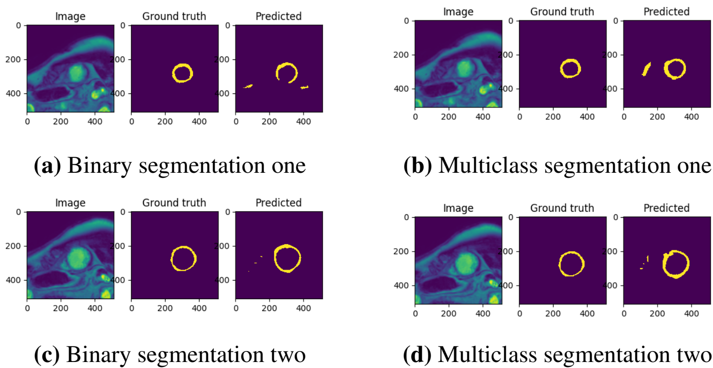
Figure 5.17: Samples from predicted masks performed by the best found multiclass model.

Table 5.9: Results for best model found in multiclass segmentation

	Multiclass
Optimizer	ADAM
Loss function	CE
Learning rate	0.00017
Scale	0.92
Drop rate	None
Dice myocardium	0.686 (0.17)
Jaccard myocardium	0.544 (0.19)
F1-score class 2	0.667 (0.18)
F1-score class 3	0.289 (0.27)

5.3.4 Comparison Between Experiment 1 and 2

In Figure 5.18, we can see a comparison between the best binary and multiclass model, tested on the validation set. The two LGE-CMR images are slices of the same subject. We see that in one of the predictions, the binary model performs best, and in the other, the multiclass gives the best result. The hyperparameters and validation performance for each model are shown in Table 5.11



(a) Binary segmentation one

(b) Multiclass segmentation one

(c) Binary segmentation two

(d) Multiclass segmentation two

Figure 5.18: Samples from predicted masks with the best found binary- and multiclass model. From (a) and (b) we can clearly see that multiclass segmentation performs better than binary segmentation. In (c) and (d) we witness the opposite

5.3.5 Verification of Best Model

The best model was found with the use of binary segmentation. The hyperparameters and performance of the model are listed in Table 5.11. The model obtained the Dice score 0.705 (0.15) when tested on the test data set.

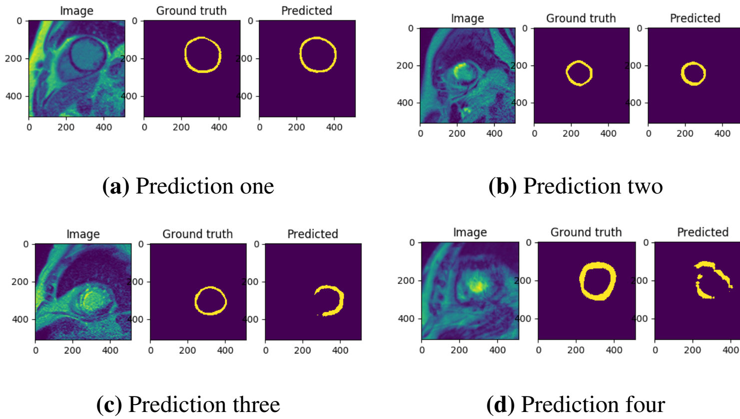
Table 5.10: Hyperparameters and validation performance for best binary and multiclass model

	Binary	**Multiclass**
Optimizer	ADAM	ADAM
Loss function	Dice	CE
Learning rate	0.0074	0.00017
Scale	None	0.92
Drop rate	0.17	None
Dice	0.691 (0.17)	0.686 (0.17)
Jaccard	0.550 (0.18)	0.544 (0.19)

Table 5.11: Hyperparameters and performance for the best found model.

Best found model	
Optimizer	ADAM
Loss function	Dice
Learning rate	0.00074
Scale	None
Drop rate	0.17
Epochs	57
Training time	2:45:31
Dice	0.705 (0.15)
Jaccard	0.560 (0.16)

Samples from predicted masks can be seen in Figure 5.19. In the Figure, two examples of myocardial segmentation with good performance, and two examples with bad performance are shown. In Figures 5.20 and we see a comparison between samples of predicted masks and manually segmented masks. All slices, for a patient, with manual notation and predicted masks, are presented in Appendix A.2. Results presented as a confusion matrix is displayed in Figure 5.21



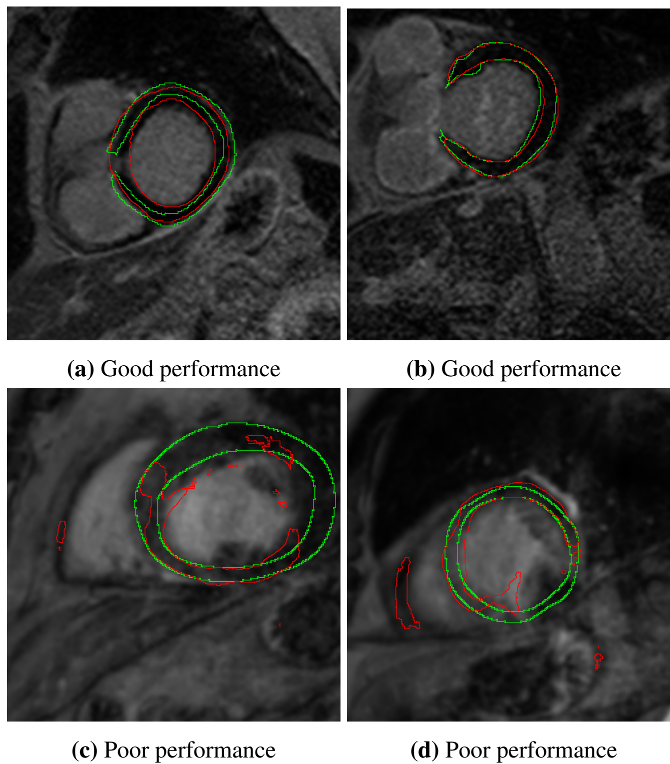
(a) Prediction one

(b) Prediction two

(c) Prediction three

(d) Prediction four

Figure 5.19: Samples from predicted masks tested on the test set with the best found model.



(a) Good performance

(b) Good performance

(c) Poor performance

(d) Poor performance

Figure 5.20: Samples from predicted masks performed by the best found model. Images (a) and (b) show examples of good performance, and images (c) and (d) show examples of poor performance. Manually marked masks are shown with green contour, predicted masks are shown with red contour.

Figure 5.21: Confusion matrix for the best model found.

Chapter 6

Discussion

This chapter will give a review of Chapter 5. Achieved results, comparisons with related work, encountered limitations, and possible improvements will be discussed. Suggestions for future work will also be presented.

6.1 Model Performance

This chapter summarizes and discusses the experiments, and results obtained in Chapter 5.

6.1.1 Experiment One - Binary Segmentation

In the initial grid search, it was found that using Dice loss overall gave better results compared to using cross-entropy loss. These results led to the choice of not using cross-entropy loss in the following experiments. Regardless, with different use of scaling, one should not exclude the possibility that the cross-entropy loss function could have given the best model if further explored.

Regarding the optimizers, it is difficult to establish, which produces the best performance. Overall, ADAM performed slightly better.Howaver, there are too many variables to be considered for making a definite conclusion.

In general, the implementation of dropout did not significantly impact the performance of the models. No patterns of optimal combinations for learning rate and drop rate were identified. The results implied that using dropout did not prevent overfitting. The best model was found with the implementation of dropout, but this is likely to be a coincidence.

6.1.2 Experiment Two - Multiclass Segmentation

The initial experiments showed that the use of the implemented Dice loss function gave poor performance. The models had problems with separating the healthy myocardium and the myocardial scar, hence Dice loss was not used in the following multiclass experiments.

Experiments with grid search and Bayesian optimization showed that models trained using weighted cross-entropy loss in overall performed better than not using weights. The observations are reasonable due to the class imbalance.

Testing of the best model proved that, on average, segmentation of the myocardial scar tissue performed poorly, despite it managing to localize parts of the scar tissue in some images.

6.1.3 Comparisons Between Binary- and Multiclass Segmentation

The best binary model proved to perform slightly better than the best performing multiclass model. On average the models trained on binary masks performed better than models trained on multiclass masks. Nevertheless, it is challenging to affirmatively conclude which approach is desirable. The best multiclass model obtained the Dice score 0.686 (0.17) for segmentation of the myocardium when tested on the validation data set. In comparison, the best binary model achieved a Dice score of 0.691 (0.17).

6.2 Comparisons With Related Work

Q. Yue et al. proposed in June 2019 a method for myocardial segmentation, using a technique called SRSCN [4]. The method uses an enhanced version of the U-Net and a combination of cross-entropy loss and dice loss as the loss function. By performing data augmentation, they obtained a data set of 20 405 images and masks. The experiments found that SRSCN outperformed the use of U-Net, by 0.08 in generalized Dice score. SRSCN obtained a Dice score of 0.758 (0.227) for myocardial segmentation, with data collected from 45 patients.

S. Moccia et al. reported in 2018 a result of 0.71 median Dice for segmentation of myocardial scars. The method used an FCNN with comparable architecture as the U-Net.

In 2015, K. Engan et al. reported a result of mean Dice 0.87 for segmentation of the endocardium, and mean Dice 0.90 for segmentation of the epicardium, when using the methods mentioned in section 1.2.

The best performing model produced in this thesis gave mean Dice of 0.705 (0.15) and mean Jaccard 0.560 (0.16). The research by Q. Yue et al. might make for the best comparison of the related work presented, even though direct comparisons must not be drawn since the data sets are distinct. The results by S. Moccia et al., and K. Engan et al. are not fully transmissible to the results presented in this thesis as the targets for segmentation were different.

6.3 Limitations

The data set applied for this thesis contains 2526 images, with corresponding masks, from 272 patients. For training DNN, this data set is considered small and is likely to limit performance. A challenge of using a small data set is the increased possibility of overfitting, making it harder for the DNN to learn the necessary features to perform effectively on unseen data. Online data augmentation was implemented to expose the DNN to images in

different ways. For each loaded batch, 33 % of all images and masks in the training set were either flipped or rotated. This is considered a narrow use of data augmentation and might have limited the performance.

For the training of the DNN, the data set was divided into a training data set, a validation data set, and a test data set. This division might not be fortunate to provide robust results. The risk of getting biased results increases as the features in the images might be specific for each data set. The DNN might, therefore, have problems learning general features. Cross-validation could be used as an alternative method. However, the testing of the best model on the validation data and test data produced similar results, indicating that the verification of the model is reliable.

6.4 Future Work

This chapter presents recommendations of techniques that might be worth exploring in future work.

6.4.1 New Network Architectures

U-Net has been used in all experiments in this thesis. Experiments with different network architectures could be utilized to develop better models. A wider range of hyperparameters could be explored, and other methods for handling the imbalance of the classes in the data might be beneficial.

6.4.2 More Data Material

One of the essential issues when training a DNN is the amount of data available. By retrieving more data, the likelihood is substantial for improving the performance of the models.

An alternative way of creating a bigger data set could be to perform further data augmentation, in addition to randomized rotation and flipping. Techniques that can be used are randomized mirroring, adding noise, cropping, and elastic deformation. This can be done by, for instance, adding noise to the images, and apply random elastic deformation to the data. Elastic deformation is commonly used and was implemented by Ronneberger et al. in the original U-Net [5].

6.4.3 Training of the Deep Neural Networks

In this thesis, the data material has been split into three data sets; training, validation, and test. K- fold cross validation is an alternative method that could be used, which might give a more robust and less biased estimation of the models [33]. Transfer learning is a technique that might be worth exploring. The method is based on using pre-trained DNNs, trained on large data sets. The pre-trained DNNs could further be trained on the LGE-CMR images.

Chapter 7

Conclusion

The objective of this thesis was to propose a method for automatic myocardial segmentation in LGE-CMR images. The developed method was using an FCNN architecture and was trained end-to-end with a training set consisting of 2006 images and masks from 214 patients affected by MI.

Experiments with two different approaches were performed. Experiment one was to train the DNN with masks of the myocardium, and experiment two was using masks with the healthy myocardium and myocardial scar tissue to train the DNN.

The best model was obtained by binary segmentation. The model got a final result of a mean Dice score 0.705 with a standard deviation of 0.15, and a mean Jaccard index 0.560 with a standard deviation of 0.16. The model was evaluated using 244 images from 30 patients affected by myocardial infarction.

When evaluating the obtained results in this thesis, it is considered that the use of DNN for myocardial segmentation is a promising method worth exploring further.

Bibliography

[1] World Health Organization. *Cardiovascular diseases (CVDs)*. [Online; accessed April 30, 2019]. 2017. URL: https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[2] Kristian Thygesen et al. “Fourth universal definition of myocardial infarction (2018)”. In: *European Heart Journal* 40.3 (Aug. 2018), pp. 237–269. ISSN: 0195-668X. DOI: 10.1093/eurheartj/ehy462. eprint: http://oup.prod.sis.lan/eurheartj/article-pdf/40/3/237/28457750/ehy462.pdf. URL: https://doi.org/10.1093/eurheartj/ehy462.

[3] Inc. Blausen Medical Communications. *Myocardial Infarction or Heart Attack*. URL: https://commons.wikimedia.org/wiki/File:Blausen_0463_HeartAttack.png. (accessed: 09.05.2019).

[4] Qian Yue et al. “Cardiac Segmentation from LGE MRI Using Deep Neural Network Incorporating Shape and Spatial Priors”. In: *arXiv preprint arXiv:1906.07347* (2019).

[5] O. Ronneberger, P. Fisher, and T.Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351. LNCS. (available on arXiv:1505.04597 [cs.CV]). Springer, 2015, pp. 234–241. URL: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

[6] Sara Moccia et al. “Automated Scar Segmentation From Cardiac Magnetic Resonance-Late Gadolinium Enhancement Images Using a Deep-Learning Approach”. In: Dec. 2018. DOI: 10.22489/CinC.2018.278.

[7] Kjersti Engan et al. “Segmentation of LG Enhanced Cardiac MRI”. In: Jan. 2015, pp. 47–55. DOI: 10.5220/0005169200470055.

[8] K. Engan et al. “Automatic segmentation of the epicardium in late gadolinium enhanced cardiac MR images”. In: *Computing in Cardiology 2013*. Sept. 2013, pp. 631–634.

[9] Fernand Meyer. “Meyer, F.: Topographic distance and watershed lines. Signal Process. 38, 113-125”. In: *Signal Processing* 38 (July 1994), pp. 113–125. DOI: 10.1016/0165-1684(94)90060-4.

[10] Soille Pierre. *”Morphological image analysis : principles and applications”*. English. 2nd ed., corrected. Previous ed.: 1999. Berlin : Springer, 2004. ISBN: 3540429883 (alk. paper).

[11] Winnie Yu Brindles Lee Macon and Lauren Reed-Guy. *Acute Myocardial Infarction*. URL: https://www.healthline.com/health/acute-myocardial-infarction. (accessed: 18.06.2019).

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016, pp. 12–20, 255–264.

[13] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386.

[15] Max Pixel. *Dendrites Soma Axon Brain Nerve Neuron Cell*. URL: https://www.maxpixel.net/Dendrites-Soma-Axon-Brain-Nerve-Neuron-Cell-1294021.

[16] Maximilian Riesenhuber and Tomaso Poggio. “Hierarchical models of object recognition in cortex”. In: *Nature Neuroscience* 2 (1999), pp. 1019–1025.

[17] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: (2017).

[18] Carole H. Sudre et al. “Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations”. In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Ed. by M. Jorge Cardoso et al. Cham: Springer International Publishing, 2017, pp. 240–248. ISBN: 978-3-319-67558-9.

[19] H. Leung and S. Haykin. “The complex backpropagation algorithm”. In: *IEEE Transactions on Signal Processing* 39.9 (Sept. 1991), pp. 2101–2104. ISSN: 1053-587X. DOI: 10.1109/78.134446.

[20] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[21] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[22] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[23] Johan Bjorck, Carla P. Gomes, and Bart Selman. “Understanding Batch Normalization”. In: *CoRR* abs/1806.02375 (2018). arXiv: 1806.02375. URL: http://arxiv.org/abs/1806.02375.

[24] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[25] Xiang Li et al. "Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift". In: *CoRR* abs/1801.05134 (2018). arXiv: 1801.05134. URL: http://arxiv.org/abs/1801.05134.

[26] James Bergstra and Yoshua Bengio. "Random Search for Hyper-parameter Optimization". In: *J. Mach. Learn. Res.* 13.1 (Feb. 2012), pp. 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2503308.2188395.

[27] Peter I Frazier. "A tutorial on Bayesian optimization". In: *arXiv preprint arXiv:1807.02811* (2018).

[28] Luis Perez and Jason Wang. "The Effectiveness of Data Augmentation in Image Classification using Deep Learning". In: *CoRR* abs/1712.04621 (2017). arXiv: 1712.04621. URL: http://arxiv.org/abs/1712.04621.

[29] Sebastien C. Wong et al. "Understanding data augmentation for classification: when to warp?" In: *CoRR* abs/1609.08764 (2016). arXiv: 1609.08764. URL: http://arxiv.org/abs/1609.08764.

[30] MATLAB. *version 9.6.0.1062519 (R2019a)*. Natick, Massachusetts: The MathWorks Inc., 2019.

[31] Guido Van Rossum and Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[32] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *CoRR* abs/1206.5533 (2012). arXiv: 1206.5533. URL: http://arxiv.org/abs/1206.5533.

[33] Sudhir Varma and Richard Simon. "Bias in Error Estimation When Using Cross-Validation for Model Selection.fffdfffdfffd BMC Bioinformatics, 7(1), 91". In: *BMC bioinformatics* 7 (Feb. 2006), p. 91. DOI: 10.1186/1471-2105-7-91.

Appendices

Appendix A

Results of Experiments

A.1 Results of Inital Grid Search in Experiment One

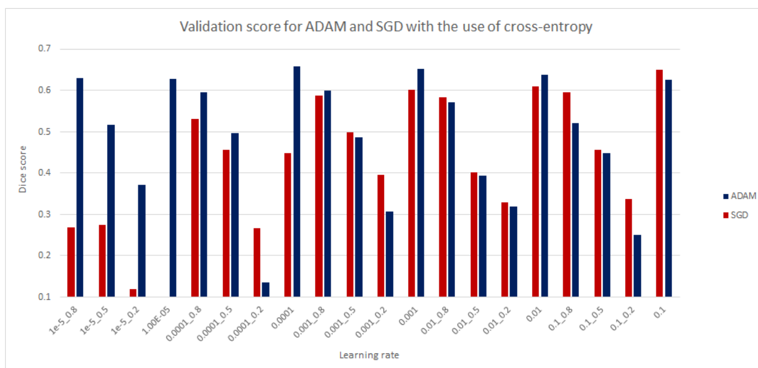


Figure A.1: The performance of the ADAM and SGD optimizers with the use of cross-entropy loss function. The models are trained with the learning rates $1 \times e^{-5}, 0.0001, 0.001, 0.01, 0.1$, with and without weighted loss (Scales 0.2, 0.5, 0.8).

A.2 Images From Best Found Model

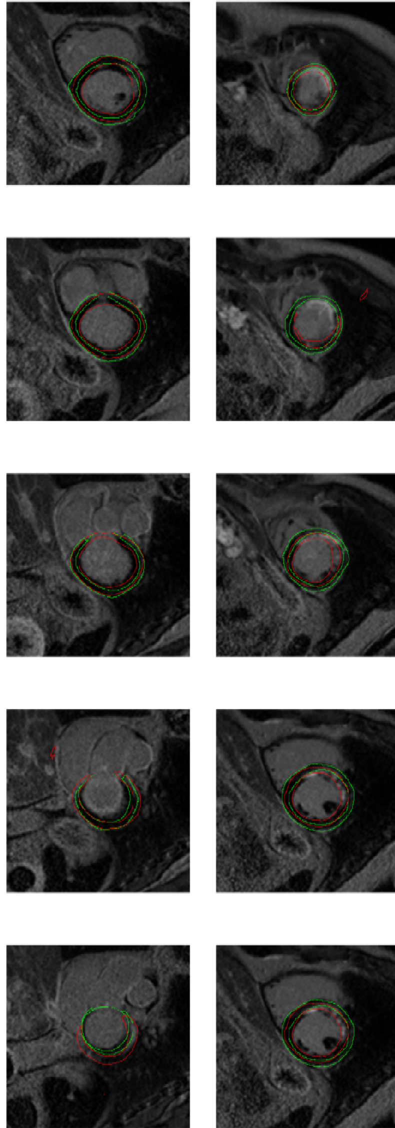


Figure A.2: Predicted masks from slices of an example patient. Masks in green contour are made by a cardiologist, predicted masks hare shown in red contour.

A.3 Training- and Validation Plots for the Best Model Found

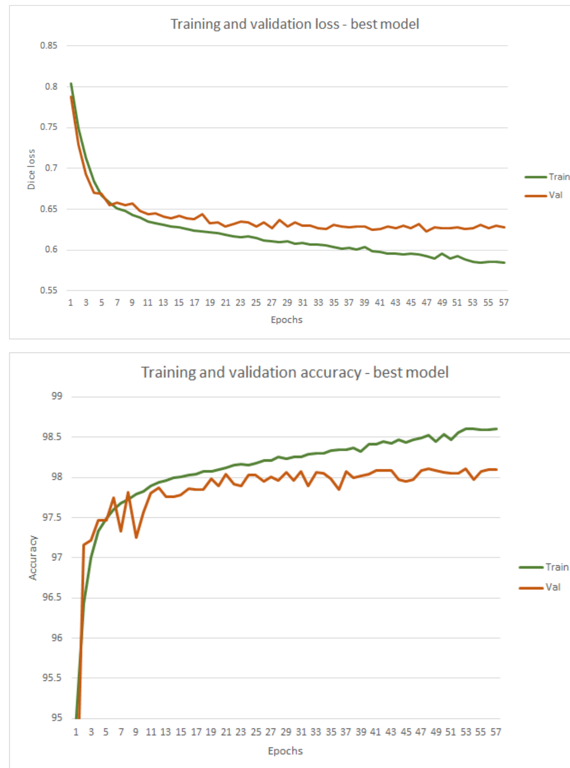


Figure A.3: Validation and- training grapsh for the best found model.

Appendix B

Algorithms

B.1 Matlab

Get_images_and_masks.m

The script preprocesses the images and the corresponding masks; healthy myocardium and myocardial scar, and saves them to a local folder. It uses external algorithm created by K. Engan et al. for loading DICOM-files and .MAT files of annotations, finding heart center, and cropping. If desired, the masks for the endocarium and epicardium can also be extracted. The main scripts are listed under. The subscripts of these can be found in the attached zip file.

External:

- dbread.m
- organizeimage.m
- crop_heart_v2016.m
- Segment_prob_2016.m

Self-made:

- get_images_and_masks.m
- figures.m

B.2 Python

The following packages were required for scripts produced:

- Pytorch
- Numpy
- OS
- DateTime
- PIL
- Pandas
- Sckit-learn
- Matplotlib
- CSV

Scripts

- **normalization.py**
  Calculates mean and standard deviation for all pixels in the data set. The mean and standard deviation are used to normalize the pixels when the images are loaded in the script loader.py
- **calc_balance.py**
  Calculates the average distributions of the classes in ground truth masks.
- **loader.py**
  Loads the images and masks from folders, converts from numpy to tensors. flip.py is called randomly to flip and rotate images and masks.
- **dice_loss.py**
  Dice loss function.
- **main.py**
  Script for training the DNN.
- **unet.py**
  Architecture of U-Net.
- **unet_dropout.py**
  Enhanced U-Net with implemented dropout.
- **modules.py**
  Script used for training.

- **test.py**
  Script for testing the performance of produced models.
- **metrics.py**
  Script for calculating Dice score, Jaccard index, F1-score, and confusion matrix.