Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

Left ventricular hypertrophy classification of MRI images

Bachelor's Thesis

Author
Máté Cseke András

Advisor
Ádám Budai

May 20, 2021

Contents

HALLGATÓI NYILATKOZAT

Alulírott *András Máté Cseke*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2021. május 20.

András Máté Cseke
hallgató

Kivonat

Az elmúlt években egyre nagyobb népszerűségre tettek szert a gépi tanulási és azon belül is a mély neurális hálokon alapuló módszerek. Szinte minden területen megfigyelhetők az általuk nyújtott előnyök és lehetőségek. Az egyik ilyen, talán az életünkre legjobban befolyással bíró ágazat az egészségügy, ahol habár manapság bizonyos részeken, mint például gyógyszertudomány, egyre nagyobb szerepet tölt be, még mindig nincs teljes mértékben kihasználva. Jelentős mennyiségű kutatások folynak például olyan orvosi diagnosztikai rendszerek fejlesztésére amelyek ezeket a módszereket kihasználva segítenek a diagnosztikai folyamat gyorsításában és pontosabb eredmények elérésében. Egy ilyen rendszer hatalmas segítséget nyújthat a megfelelő diagnózis időbeni meghatározásához és ezáltal a kulcs tényező lehet egy páciens állapotának javulásában.

A szakdolgozatom során kontraszt anyagos szív MRI képeket használva egy ilyen diagnosztikát elősegítő model készítése volt a cél. A feladat a balkamrai hypertrófia létezésének és okának minél pontosabb meghatározása volt, amelyet két különböző osztályozási problémára bontottam. Az egyik egy két osztályos probléma volt, amelyben az egészséges, vagy természetesen elváltozott, például sport szívet különböztettem meg az abnormálisan elváltozottól. A másik pedig négy osztály: normál, sport, HCM (hipertrófiás kardiomiopátia) és egyéb megkülönböztetése volt. A megoldás során figyelembe vettem a rövidtengelyről és a hosszú tengelyről készített képeket is, utóbbit három különböző nézetre bontva. A bemeneti képek javítása céljából egyedi heg-szegmentációs modellel kísérleteztem.

A teszthalmazon elért legjobb eredményem két osztályra bontás esetén 91.89%-os pontosság és 91.84%-os F1-mérték, míg négy osztályra bontás esetén 78.38%-os pontosság és 77.74%-os F1-mérték volt.

A dolgozatomban a szükséges biológiai ismeretek bemutatása után betekintést nyújtok a használt technológiákba, majd a tervezési és implementálási folyamat részletes leírása után összevetem a különböző módszerekkel elért eredményeket.

Abstract

In the recent years, artificial intelligence and deep learning based approaches in particular have become increasingly popular. The benefits and possibilities provided by these methods can be seen everywhere. Possibly one of the most important beneficiary is healthcare, where although the application of AI is steadily increasing, like e.g. in pharmaceutical science, it is still not nearly utilized to it's full potential. There is a notable amount of research and interest surrounding the development of computer-aided diagnosis systems that could use deep neural networks to speed up and even improve the diagnostic process of patients. A well tuned system like this could significantly increase the chances of properly diagnosing patients in time, and thus could make a huge difference in serious life threatening conditions.

The goal of my thesis project was to create a diagnosis aiding model specifically for the detection and classification of left ventricular hypertrophy based on contrast enhanced cardiac MRI images. The task was separated into and evaluated as two different classification problems. One of them was a binary classification, in which I differentiated healthy, or naturally enlarged (sport) hearts from the abnormally lesioned ones. And the other one was a four-class classification, where the dataset was separated into Normal, Sport, HCM (Hypertrophic cardiomyopathy) and other classes. I used both short- and long-axis images for the solution, separating the latter one into three different views, and experimented with a custom scar-segmentation approach to improve the overall performance of the model.

The final, best results achieved on the test set were 91.89% accuracy and 91.84% F1-score for the binary classification, and 78.38% accuracy and 77.74% F1-score for the four-class classification problem.

In this document I provide the necessary biological and technological background knowledge for understanding the problem and the proposed solution, discuss the planning and implementation steps in detail, and finally evaluate and compare the results achieved by different approaches.

Chapter 1

Introduction

1.1 Motivation and goals

In the recent years deep learning based approaches have become increasingly popular. More-and-more people are familiar with the term as we can see the real world use-cases in almost every field now from speech recognition through recommendation systems to the recently popular synthetic media generation (deepfakes, faceapps). Medicine is also a beneficiary of this subclass of machine learning as most problems in the field are especially complex, with many unstructured and hidden patterns. Probably the biggest and most heard-of recent breakthrough in this field was DeepMind's AlphaFold[1] system in protein folding. However, molecular biology is not the only sub-field of medicine where deep learning approaches excel. Computer-Aided Diagnosis (CAD) systems has also seen huge advances in the past few years. There have been many recent researches, which suggested very effective deep learning methods for medical image classification as part of CADs. Many things contribute to the recent surge in applications, but the most important ones are the huge amounts of data that is collected and the fast advance in hardware technology. The amount and availability of data in medicine, although generally increasing, is still low compared to other industries as information can be highly sensitive. Acquiring large enough datasets is a key part of successful deep learning based solutions and thus related research is especially complicated in this field. Due to the previously mentioned factors, it has been a great opportunity for me to join the research on cardiac hypertrophy classification at Budapest University of Technology and Economics as part of my bachelor's thesis project and get access to a relatively large cardiac MRI dataset.

The project's goal was to create a classification model that can benefit from the 3D nature of MRI recordings and give predictions based on multiple axis views of the heart. The project involved processing the raw dataset, then designing and adjusting the model to reach an optimal result for a 4 class classification problem separating patients into Normal, Sport, HCM (Hypertrophic cardiomyopathy) and Other (Amyloidosis, Aortastenosis, Fabry, Endomyocardial fibrosis) categories. The best performing model was then evaluated on a binary: Normal and Abnormal classification problem as well.

1.2 The structure of the document

The thesis is structured the following way to build up the necessary knowledge and background for understanding the problem and suggested methods:

- Biological background and the current state of medicine. Includes a brief overview on the anatomy of the heart, what cardiac hypertrophy is and the importance of diagnosing it's cause, and an introduction to MRI.
- Insight on the state of the art solutions in deep learning based medical image classification. Basics and related history of deep learning.
- Brief introduction to the DICOM standard and the used dataset. Explanation of the model concept.
- Detailed description of the data processing tasks, including statistics and all the necessary information to understand the input requirements of the model.
- Implementation of the model, discussion on the chosen methods, technologies and different possibilities.
- Evaluation of the model, comparing the results of different approaches.

Chapter 2

Overview of related medicine

2.1 Basic anatomy of the heart

The heart is the core of our bodies circulatory system. It is an especially muscular organ, which job is to pump blood through the blood vessels and provide oxygen and nutrients to the whole body, while simultaneously clearing it from metabolic waste. It has four chambers each having their own responsibilities, but working in perfect harmony. The two upper chambers, called atria, are responsible for receiving blood, while the two lower chambers, called ventricles' job is to discharge it. The left side of the heart receives oxygenated blood from the lungs and pumps it to the rest of the body, while the right side receives deoxygenated blood and sends it to the lungs. The two sides are separated by a tissue called septum and the upper and lower chambers are separated by valves. The heart wall is divided into three layers, the endocardium, which is the innnermost layer of tissue that surrounds and protects the valves and chambers, the myocardium, which is the muscular middle layer responsible for the pumping movements, and finally the epicardium, the outer protective layer. [9] [19]

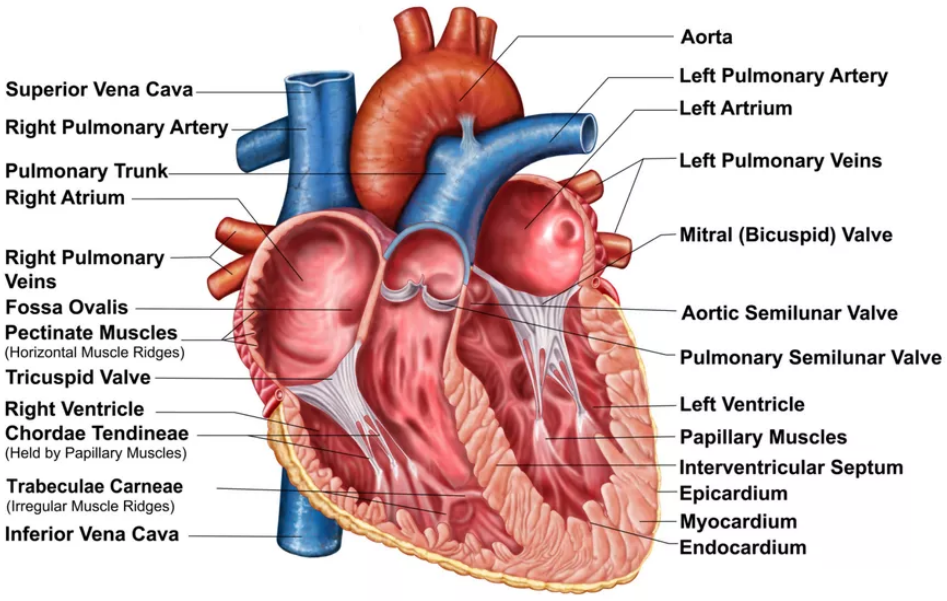

Figure 2.1: Detailed structure of the heart. [4]

2.2 Hypertrophy

Muscular hypertrophy is the growth of muscle tissue due to the component cells increasing in size. There are two types, sarcoplasmic hypertrophy, which is the increase in the muscles glycogen storage, and myofibrillar hypertrophy, which is the growth of the contractile portion of the muscle, called myofibril. Sarcoplasmic improves the endurance, while myofibrillar hypertrophy increases the actual strength and speed of the muscle. This increase in size and density is what makes us stronger and able to withstand bigger physical stress. This would mean that muscle hypertrophy is a good thing, well it is for skeletal, but not so much for cardiac and smooth muscle (e.g. intestines) tissue. These three types differ greatly both in structure and functionality. [15]

The average human heart beats about 100,000 times a day, meaning it is our most worked muscle. Each heartbeat consists of two parts diastole and systole. In the diastole phase the ventricles relax, while the atria contracts, transferring the blood between the upper and lower chambers. On the other hand, in the systole phase the ventricles contract and pump blood out from the heart, while the atria relaxes and fills up with blood again.

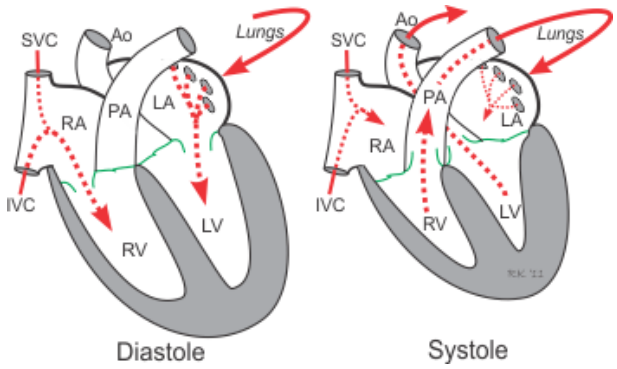

Figure 2.2: Diastole and systole phase of a cardiac cycle. [36]

The frequency of this chain of motion in resting state can range from around 60 to 90 beats per minute (BPM), mainly depending on genetics and the cardiovascular fitness of the person. This rate, however, can go up to 200 BPM as well on demanding physical activities. Generally athletes have lower BPMs in both resting and stressful environments as just like our skeletal muscles the myocardium adjusts overtime increasing in volume to boost efficiency. This growth of muscle mass in response to overload is called eccentric hypertrophy or "athlete's heart" and usually is considered healthy. There is an another type of ventricular hypertrophy called concentric hypertrophy which results from an increase in pressure the ventricles experience. At first the two types would seem similar, but while eccentric hypertrophy occurs due to a need in better throughput performance, concentric hypertrophy is a defensive mechanism to protect the ventricles from increased pressure by abnormally enlarging the ventricle wall. This defensive reaction can be triggered by many factors, such as hypertension, general weakening and/or defects in myocardium (hypertrophic cardiomyopathy referred as HCM) and can be a side effect of other serious underlying diseases such as amyloidosis and Fabry disease. Concentric hypertrophy is considered more dangerous and life threatening, but in abnormal cases eccentric hypertrophy can cause just as many complications. The excessive increase in the ventricle wall thickness can cause conduction abnormalities, and it is a predictor of serious cardiovascular events that can lead to stroke, congestive heart failure and even sudden cardiac arrest. Ventricular hypertrophy can occur on both left and right regions of the heart, but the

project's focus was specifically on left ventricular hypertrophy as it is the most common of the two. [14] [17] [25]

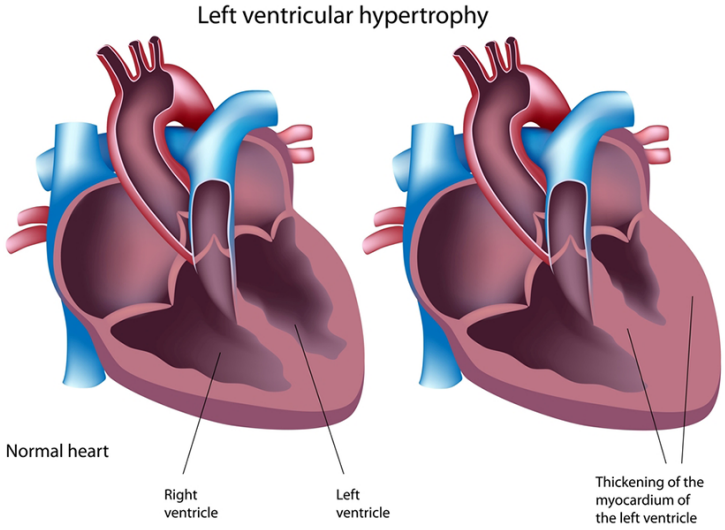

Figure 2.3: Heart with left ventricular hypertrophy compared to normal. [32]

2.3 Magnetic resonance imaging

Based on symptoms and diathesis of the patient, doctors may suggest to perform either computerized tomography (CT), echocardiography or magnetic resonance imaging (MRI) tests. From these methods MRI is the preferred method in identifying cardiovascular diseases as it gives the most detailed and clear results on soft tissue and is able to visualize multiple planes of the observed anatomy. It is a non-invasive technology which uses strong magnetic fields and radio waves to stimulate protons (usually hydrogen nuclei, as most of our bodies are made out of water) in the body. The magnetic field pulls and aligns the protons, while the radio waves disrupt and tip their balance. The disrupted protons eventually relax (realign) after the external radio frequency (RF) energy ceases. This relaxation process emits RF signals, which are detected by sensors (RF coils) then converted and mapped to gray-scale images. Each tissue has unique relaxation properties and thus varying the sequences of applied and measured RF provides different images. [5] [43] [13]

The three main parameters that are considered when describing MRI sequences are echo time (TE), repetition time (TR) and flip angle. TR is the time between consecutive pulses, TE is the time between the initial pulse and the received echo signal and flip angle is the amount of rotation induced by the applied RF pulses. The two basic types of sequences are T1-weighted and T2-weighted scans. T1 is the time it takes for a flipped nuclei to turn back to it's aligned state, while T2 is the time it takes for it to lose magnetization along the transverse plane. T1-weighting usually involves short TE and TR times, while T2-weighting longer. [28] [47] [43]

Many times paramagnetic contrast agents are introduced to the blood stream (or sometimes orally) to increase the clarity and detail of diagnostic images. They work by altering the T1 relaxation time of tissues and thus are used in combination with T1-weighted sequences. The most commonly used agents are Gadolinium (Gad) based which have not

been found to create harmful side-effects and have been approved by the FDA. It is however advised that the usage of these agents should be limited as they can build up in the body overtime. Gad based agents usually shorten T1 times which means that on T1-weighted images the agent appears brighter than it's environment. For this reason Gad enhanced images are useful in identifying abnormal biological structures such as tumors, inflammations and scarring. [43] [34] [13]

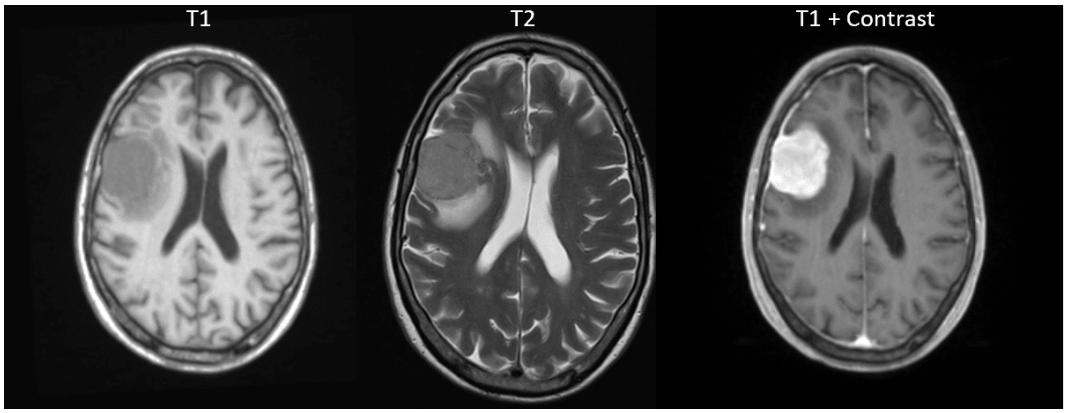

Figure 2.4: MRI images of a brain with meningeal tumor on T1-weighted, T2-weighted and Gad enhanced T1-weighted sequences. [31]

Chapter 3

Technological background

3.1 Basics of deep learning

Deep learning is a subclass of machine learning (and thus artificial intelligence as well), which uses multilayered neural networks to extract and learn features from large, complex datasets. It has become a very popular concept recently as every industry tries to utilize it's powers in some way. The surge in popularity is mainly thanks to the large amounts of amassed data, advancements in hardware technology and the growing number of promising, many times openly available model architectures. To properly understand how it works and why it's so successful we have to go back in time to the roots of the concept.

3.1.1 Artificial neuron

In 1943 Warren McCulloch and Walter Pitts [39] described the first artificial neuron, giving birth to the "ancestor" of the ones used in neural networks today. They based their concept on the biological model of the neurons in our brains. Approached in a simplified way biological neurons can be separated to three parts:

- The dendrites, which can be considered as the input points of the nerve cell that receive and weight the incoming signals.
- The Soma, which acts as the summation function of the neuron, that processes the input signals in a biased manner and forwards the result.
- The Axon, which decides based on the result signal whether the neuron should fire and transmit a pulse to the following neurons connected by synapses.

McCulloch and Pitts' artificial neuron mapped this structure the following way. They said that dendrites can be seen as multiple inputs to a summation function (Soma), which output is passed to a threshold function that produces the output of the neuron (Axon). The model was only capable of processing binary inputs and used a simple Threshold Step activation function meaning it was only capable of producing binary outputs as well. The two main restrictions were however that it didn't really have weights and a bias (an offset or starting input). The inputs were only prefixed as excitatory or inhibitory. The excitatory inputs increased the likelihood of the neuron "firing" (outputting 1), while the inhibitory inputs had an absolute negating power, meaning even if only a single inhibitory input was on the neuron had to output 0. [12] [38]

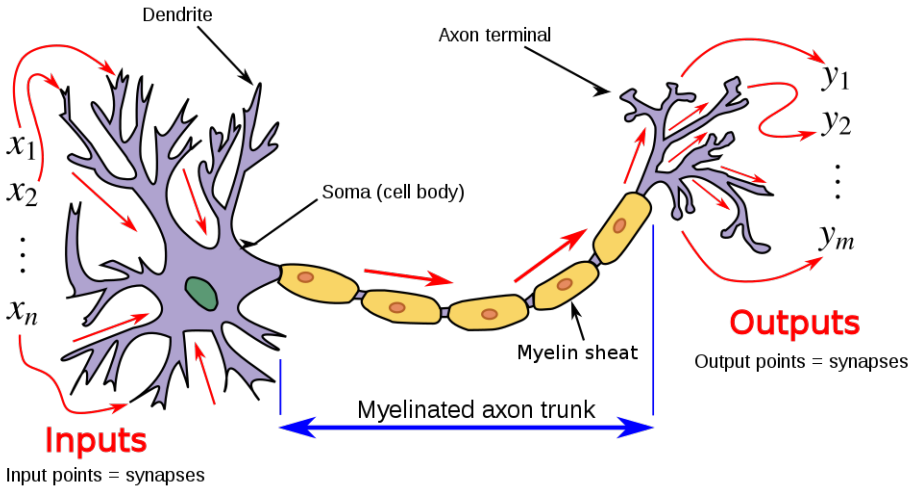

Figure 3.1: Biological structure of a neuron. [48]

3.1.2 Perceptron

Developed by Frank Rosenblatt [46] in 1957, the perceptron is the building block of neural networks used today. It is basically an upgrade of the McCulloch-Pitts model that's able to handle non-binary, real inputs and has adjustable, non-unit weights, meaning that not every input has the same amount of influence on the output. The weights have inhibitory and excitatory effects, but are never absolute, and the output is also influenced by a bias. [38]

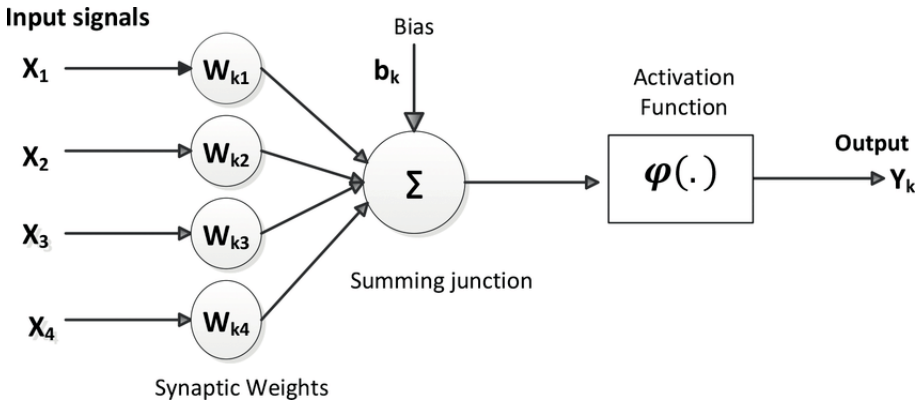

Figure 3.2: The computational model of a perceptron. [29]

Rosenblatt also came up with a learning algorithm that was able to converge and thus learn binary classification problems on linearly separable data. This so called perceptron rule was a very simple algorithm, which updated the weights in the following way:

- It calculated the error by subtracting the predicted output from the expected and multiplying the result with a constant called learning rate.
- Then for each input it adjusted the weights by adding the error multiplied by the input value to the initial weight.

The modern variants used in networks, like multi-layer perceptrons no longer use step activation functions and instead rely on continuous and many times continuously (sigmoid, tanh) or at least partially (ReLu) differentiable activation functions.

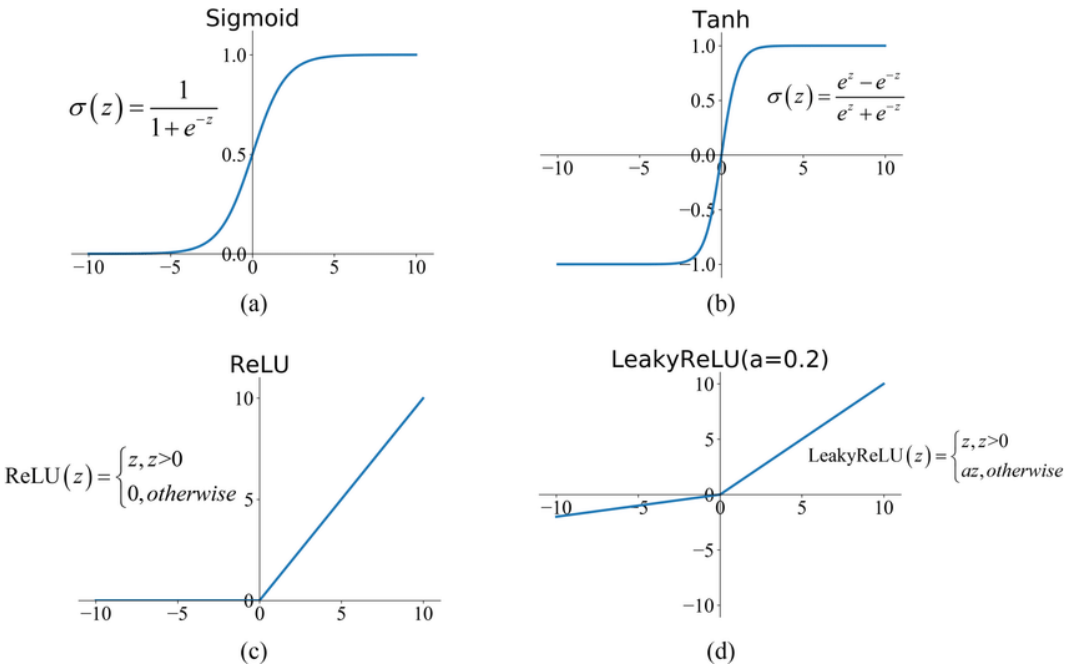

Figure 3.3: Some of the most common activation functions. [26]

These functions provide a more continuous and detailed mapping of the sum and open up gradient-based learning algorithms, which rely on calculating the derivative of the error function and applying the chain rule to find the respective gradients for each weight, this process is called backpropagation. In the most basic algorithm, gradient descent, these gradients are accumulated for all input data and then finally used to adjust the weights, converging the error to the local minima. This process is quite slow however, as the adjustment happens only after each batch of data, for this reason it is many times referred to as batch gradient descent. A faster converging alternative is the so called stochastic gradient descent (SGD), which updates the weights after each data. There are many modern optimization algorithms which use different or upgraded approaches, like et al. Adam[35] (discussed later at 6.3.1) which uses previous gradients (moving average) to determine ”momentum” and adjust the learning rates on the go.

3.1.3 Artificial neural networks

Marvin Minsky and Seymour Papert[40] in 1969 stated that since the perceptron is not capable of implementing logical functions like XOR, which can handle non-linearly separable data, multiple perceptrons concatenated into a network wouldn't either. Arguably this statement in a form of a book led to an AI winter in the 1970s. Over the years however many multi-layered implementations have been successful in this regard and in 1989 George Cybenko[22] proved that using multiple perceptrons with sigmoid activation functions on a single hidden layer can actually approximate any function. This was later generalized for neural networks independently from the activation function used. [12]

For many years it was believed that since shallow (1 layered) networks are capable of estimating any function, adding more layers wouldn't provide any benefits. It was soon (as

we now know) however found out that it can actually perform better with less parameters. Researches in neuroscience suggested that the neurons in our brains are purposefully ordered in a hierarchical way so that the first "layers" of neurons learn how to extract useful information and the following neurons learn how to interpret them. It turned out that this is what's happening in multi-layered or deep neural networks as well, the network learns which features of the input data are most important and uses them specifically. This feature extraction property of deep networks is the basis of deep learning and what makes it so powerful in handling large amounts of complex and unstructured data as the network can learn which features of the data are important for a specific problem.

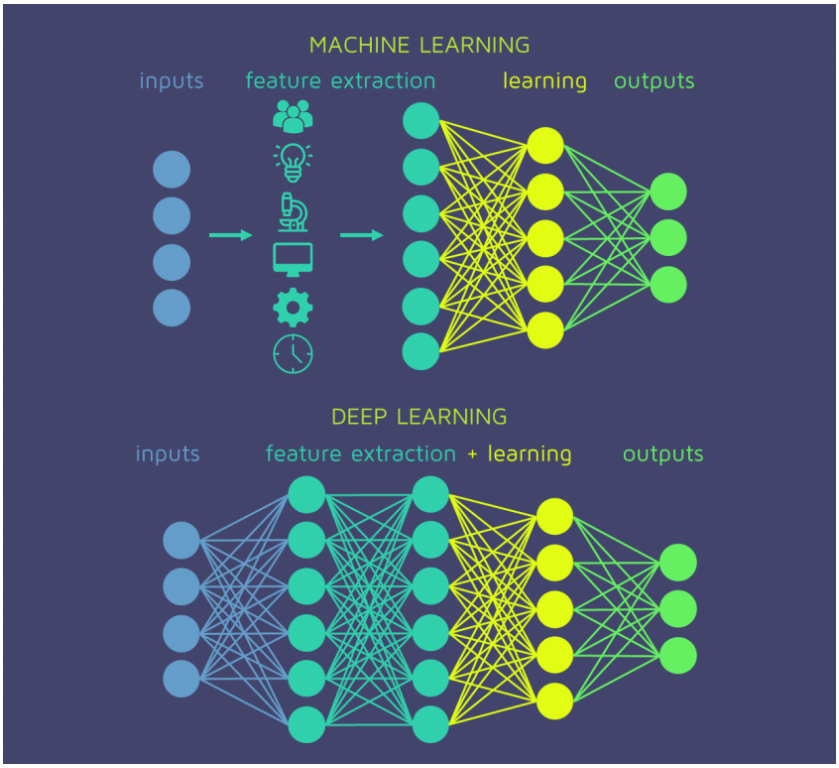

Figure 3.4: Shallow versus deep fully connected neural network, the basis of deep learning. [8]

3.1.4 Convolutional neural networks

Traditional fully connected neural networks or multilayer perceptrons (MLP) had huge successes on data where the input features' relative location to each other did not matter, but didn't perform well on tasks where spatial information was important, like in case of image processing and classification. This is mainly because they interpret each data as a flattened one dimensional vector of input features, so in case of an image, which can be represented as a matrix, it loses the relative location based information between the rows and thus are not spatially invariant. Another huge problem is that for large inputs like RGB images, which can be 256x256x3 matrices, there are just way too much weights to be trained if all input feature has it's own separate weight.

There were many solutions for these problems, the one that we utilize in most modern image processing tasks is called convolutional neural network (CNN), which was proposed by Yann LeCun et al. [37] in 1998, based on the Neocognitron[27] model of Kunihiko Fukushima. CNNs, and the Neocognitron, were inspired by the visual cortex of mammals.

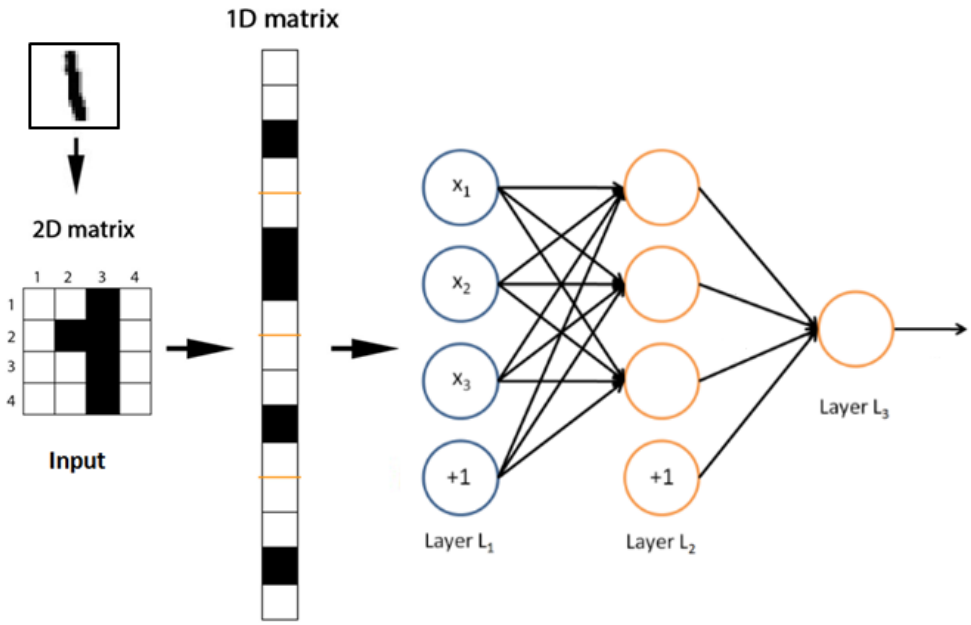

Figure 3.5: Processing of input image in an MLP. [42]

Researches in neuroscience showed that simple cells in our visual system (true for auditory and somatosensory as well) each have an individual restricted region or range where they pick up and respond to stimuli. This region is called the receptive field. These fields overlap and they are used to detect and identify complex relations like edges in visual data. Since the cells are ordered in a hierarchical way, the receptive field can consist of the initial receptors or lower neurons as well. There are so called complex cells on top of the hierarchy that process the interpreted stimulus of simple cells. In case of visual data they use the relative location and orientation between edges to identify shapes and objects independently of their location in the combined receptive field of it's connected simple cells. This means that complex cells can be considered spatially invariant. [11] Convolutional layers similarly use different, most commonly 3x3 or 5x5, sized filters/kernels which can be seen as a matrix of weights. The kernel shifts across the input matrix from left to right and top to bottom with a fixed step length called the stride. At each step the overlapped part of the input is multiplied by the corresponding kernel and the values of the resulting matrix are summed with an additional adjustable bias. These sums create a new higher level feature map that contains information about specific spatially sensitive features in each sampled submatrix of the input.

The size of the feature map is determined by the input matrix size, the additional zero padding added, kernel size, the dilation rate of the kernel and the stride. Dilation rate is the amount of space between the sampling units of the kernel and zero padding is just the amount of rows and columns that are added to the sides of the input matrix. There is usually more than one kernel used and each learn to recognize a different feature of the input by having their own weights that can be tuned. The feature maps created by kernels are concatenated as multiple different channels of the output and finally pushed through an activation function. Usually a pooling layer is introduced to reduce the size of the matrix and for example in case of max pooling keep only the outliers and most impactful values of the map.

The biggest benefit of these layers other than being able to learn spatial features is that, since we only assign weights to each kernel, we have significantly less parameters to adjust.

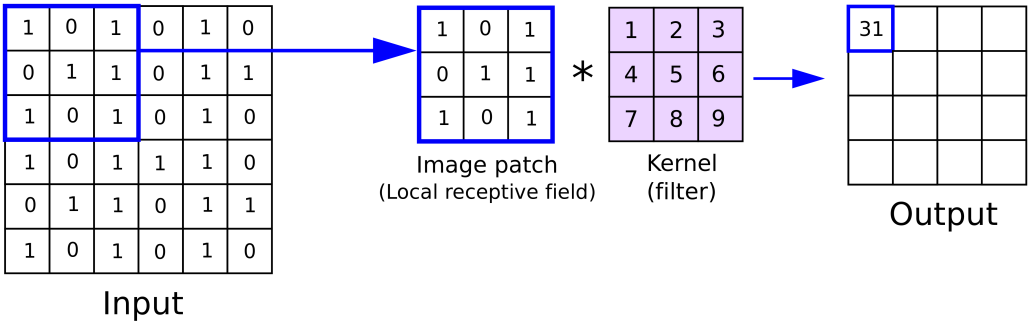

Figure 3.6: A simple example of 2D convolution. [45]

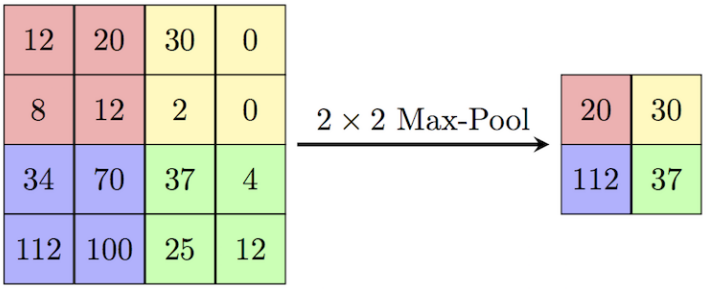

Figure 3.7: Example of max pooling with 2x2 kernel size and a stride of 2. [3]

The layers can also be used hierarchically similarly how simple and complex cells work to achieve the same feature extraction effect that makes deep neural networks so powerful. In a deep CNN for image classification the lower layers for example learn how to detect edges while the higher ones the shapes that these edges make. The final feature maps can then be flattened and used in a fully connected layer to perform classification based on the abstracted features.

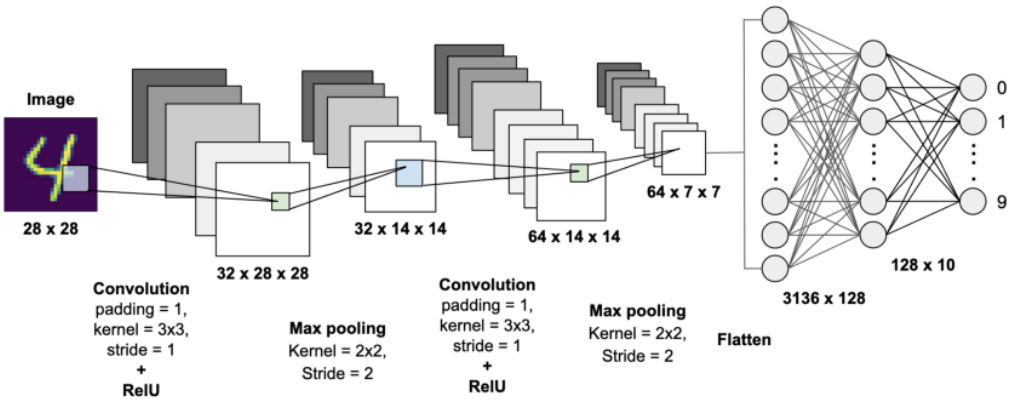

Figure 3.8: Example of max pooling with 2x2 kernel size and a stride of 2. [18]

3.1.5 Residual neural network

Deep neural networks had great results on large and complex datasets, but they introduced two main problems. The deeper a network was the more likely it was to experience the so called vanishing gradient problem. The gradient calculated by the backpropogation got so small for lower layers due to the depth of the network that training these layers was impossible or at least incredibly slow. The other problem was that after reaching a certain perfect depth for a network on specific dataset adding more layers resulted in a problem called degradation. This problem most likely occurs due to the network struggling to learn an identity function for the extra parameters and in the process introduces a lot of unnecessary noise. Kaiming He et al. [30] was the first to showcase this problem and came up with a solution called residual learning. The concept was to add skip connections or extra identity mapping between layers. These connections did nothing more than adding the outputs of a previous layer to the output of the next layer basically skipping the mapping of the layer between. In case of differing dimensions (like in CNNs) it used a simple projection to make the outputs match. This was a rather easy but game changing addition to deep networks, as it solved both the vanishing gradient and degradation problem by giving the network clear non weighted passages to reach the early layers and also a way to skip excess parameters. The introduction of residual blocks gave birth to very deep networks like ResNet152 (where 152 stands for the number of layers) which was created by Kaiming He and the rest of the Microsoft research team for the ImageNet (ILSVRC) competition and won with an astonishing top-5 error rate of 3.57%.

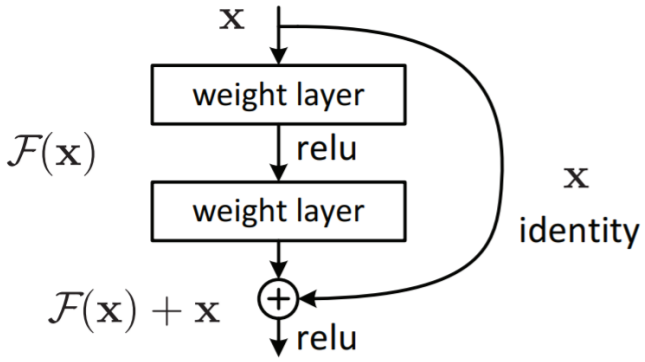

Figure 3.9: The residual block proposed by Kaiming et al. in his paper Deep Residual Learning for Image Recognition. [30]

3.2 State of deep learning based approaches in medical imaging

There are fortunately many promising results using deep learning methods in medicine. Some of the most impressive and inspiring ones for me were the following:

- Zhang et al. [50] successfully used different CNN models to build a fully automated analysis pipeline for echocardiograms. They created a classification model to determine the view from 23 different viewpoints and a segmentation model for cardiac chambers in all five main views. The segmentation models output was used to quantify chamber volumes and left ventricular mass and determine longitudinal strain

through speckle tracking. Finally they tested different CNN models for a three class disease classification, separating HCM, amyloidosis and pulmanory arterial hypertension (PAH). The axis classifier model accurately identified the views in general, for parasternal long axis it managed to reach a 96% accuracy. The disease classifier reached C statistics of 0.93 for HCM, 0.87 for amyloidosis and 0.85 for PAH. Comprehensive validation across 11 internal consistency metrics also showed that automated measurements (segmentation and measurement of longitudinal strain) achieved similar and in some cases superior results than manual measurements.

- Mortazi et al. [41] created a multi-planar CNN model with adaptive fusion strategy utilizing complementary information from different planes of 3D scans like MRI and CT to precisely segment substructures of the heart. They used three different, but architecturally same CNN for three planes. The models were trained for voxel-wise labeling for seven cardiac structures, myocardium of left ventricle, left atrium, left ventricle, right atrium, right ventricle, ascending aorta and main pulmonary artery. They achieved a precision and dice index of 0.93 and 0.90, and 0.87 and 0.85 for CT and MR images respectively, using the MM-WHS dataset containing 120 cardiac images.

- In 2018 Fauw et al. [24] (DeepMind) created and trained a network on 14884 optical coherence tomopgraphy (OCT) scans to detect 9 different pathologies. The model achieved an accuracy score of 94% in detecting the most urgent condition, which based on statistics matches that of leading eye experts.

- Very recently Aggarwal et al. [16] published a meta-analysis on the state of deep learning based methods in pathology identification. They identified 9484 individual studies in this field from which they selected 503 based on some abstract and title filtering. 82 of the studies was in ophthalmology, 82 in breast diseases, 115 in respiratory diseases and 224 in other mixed specialities. They observed the following regarding the use of DL methods in these fields. In ophthalmology diabetic retinopathy, age-related macular degeneration and glaucoma can be identified with high sensitivity, specificity and AUC on both OCT scans and retinal fundus photographs. In respiratory medicine high sensitivity, specificity and AUC for chest pathologies can be achieved on CT scans and chest x-ray. In breast cancer high diagnostic accuracy can be achieved in identifying breast cancer on mammograms, ultrasound and digital breast tomosynthesis.

Aggarwal et al. and Davenport [23] both came to the conclusion that while AI and specifically deep learning based approaches have potential in healthcare, there are still a few problems that make the application of AI in this particular field rather complicated. Aggarwal et al. stated that one of the main problems is that the evaluation of these methods is not standardized properly and in both paper they highlighted ethical questions being a huge obstacle as well. At the time of writing, there are currently 29 FDA approved [20] AI based medical technologies in use from which 13 utilize some kind of deep learning methods. Only one of these [2] focuses on cardiac MRI analysis.

Chapter 4

Planning the model

4.1 Brief introduction to the dataset

To come up with a model we have to first understand the data that is available to us. The dataset was provided by the Városmajor Heart and Vascular Centre and contained a total of 890 patients MRI scans, pathologies and for some patients contour data for the short axis view of left ventricle. Most of the patients had short, long and transverse (or axial) plane view non LGE (late gadolinium enhancement) scans. These usually used the BTFE-BH (Balanced Turbo Field Echo-Breath Hold), meaning these images had fairly similar properties compared to each other. A smaller portion of the patients had LGE scans available as well, which used IR (Inversion Recovery) gradient echo sequences with three separate echo times used. These LGE IR images had slightly different characteristics compared to the non-LGE BTFE-BH ones and most importantly they highlighted the gadolinium buildups around scar tissue. I will discuss more about the structure and processing of the dataset in the following chapter.

4.2 The model concept

The projects goal was to create a classifier model that can diagnose the different underlying causes of left ventricular hypertrophy in a patient as accurately as possible. Determining the different diseases solely based on myocardial mass is almost impossible in most cases, a better way of approaching the problem is to take the different scarring schemes of the left ventricle wall into account as well. This is where LGE scans come in handy as contrast agents usually build up around the location of scars due to infracted blood vessels. These agent build ups can be highlighted with a proper T1 weighted sequences like the IR Fast Gradient Echo used in the dataset.

As it can bee seen on 4.1 based on the relative location and size of the scar tissue we can give a very good estimate of the underlying disease. This means that creating an accurate scar segmentation model would most likely increase the overall performance of the classifier. There are a few ways to achieve this, a basic approach would be to use manually segmented data to train an automatic segmentation network. There were only endocardium and epicardium contour data available for this dataset and I wanted to create a model that required no manual assistance. The idea relied on the fact that unlike LGE, non-LGE scans have no visible scar tissue. This means that taking the exact same image

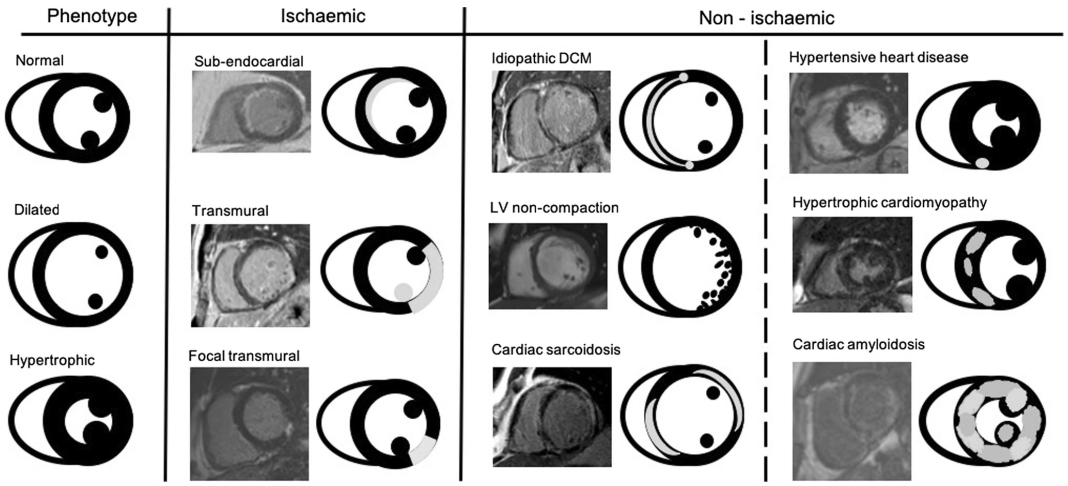

Figure 4.1: LGE patterns of different diseases with MRI image examples. [21]

of a specific frame of the heart in both types of scans and subtracting the non LGE scan from the LGE results in a mask of the scar tissue.

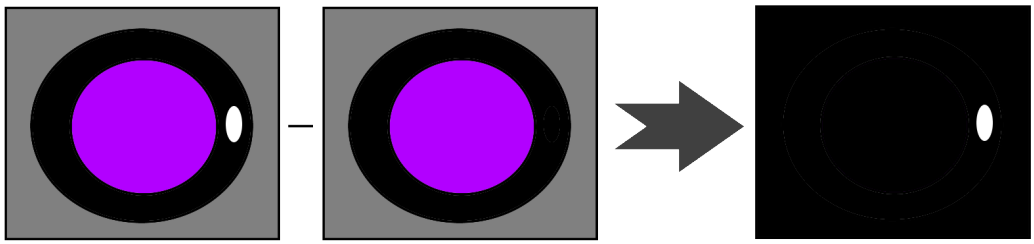

Figure 4.2: Simple demonstration of scar tissue masking by subtraction of LGE and non-LGE scans.

The concept was good, but for it to work the same image had to be found in both scan types. A better alternative was to just create a non-LGE image from an LGE one. An autoencoder was a perfect solution to achieve this. Autoencoders are a type of unsupervised learning models, which learn the latent space representation of the input data. The name unsupervised come from the fact that the input data is not labeled by humans, these models learn to extract the most important features of the input by having a so called bottle neck in the model. This bottle neck like architecture reduces the dimension of the input data on one side and then on the other side it tries to recreate it from the reduced representation. Autoencoders are widely used for image reconstruction, generation and noise reduction because of these properties. A good autoencoder model trained on non-LGE images would learn how to reconstruct these images fairly accurately. Feeding the pretrained autoencoder afterwards with LGE images would presumably clear all previously not seen features (noises), which means the scar tissue would be wiped from the image. This is exactly what we need as we can then extract the output of the autoencoder from the input to get the masks. The created masks can then be used as the output of the classifier. One of the main benefits of MRI is the possibility to create multi-view images of an organ. As previously mentioned, for most patients both short and long axis scans were available. The most important and mainly used view for diagnosis is the short axis, however the three different views (two chamber or vertical long axis, three chamber and four chamber or horizontal long axis) of the long axis can have complementary and some-

times even decisive information. To achieve the best possible results the model should take into account all four of these views. This can be achieved by creating an ensemble model, which simply means the combining of multiple models into one. These models almost always achieve better performance than their solo counterparts, as in the worst case scenario the non important planes get ignored. The only problem with ensembles is that if the submodels are trained in parallel they can require large amounts of computing power and memory. Many times, partially for this reason, the submodels are pretrained separately and only their final layers are trained in combination. The model concept as a whole was the following, build and train an autoencoder model on non-LGE images, then create an ensemble model with four different, but architecturally same, submodules for each main view of the heart. The ensemble model is then trained on scar tissue masks created by subtracting the output of the pretrained autoencoder from the original input LGE image. The model is demonstrated visually for better understanding in the diagram below (4.3).

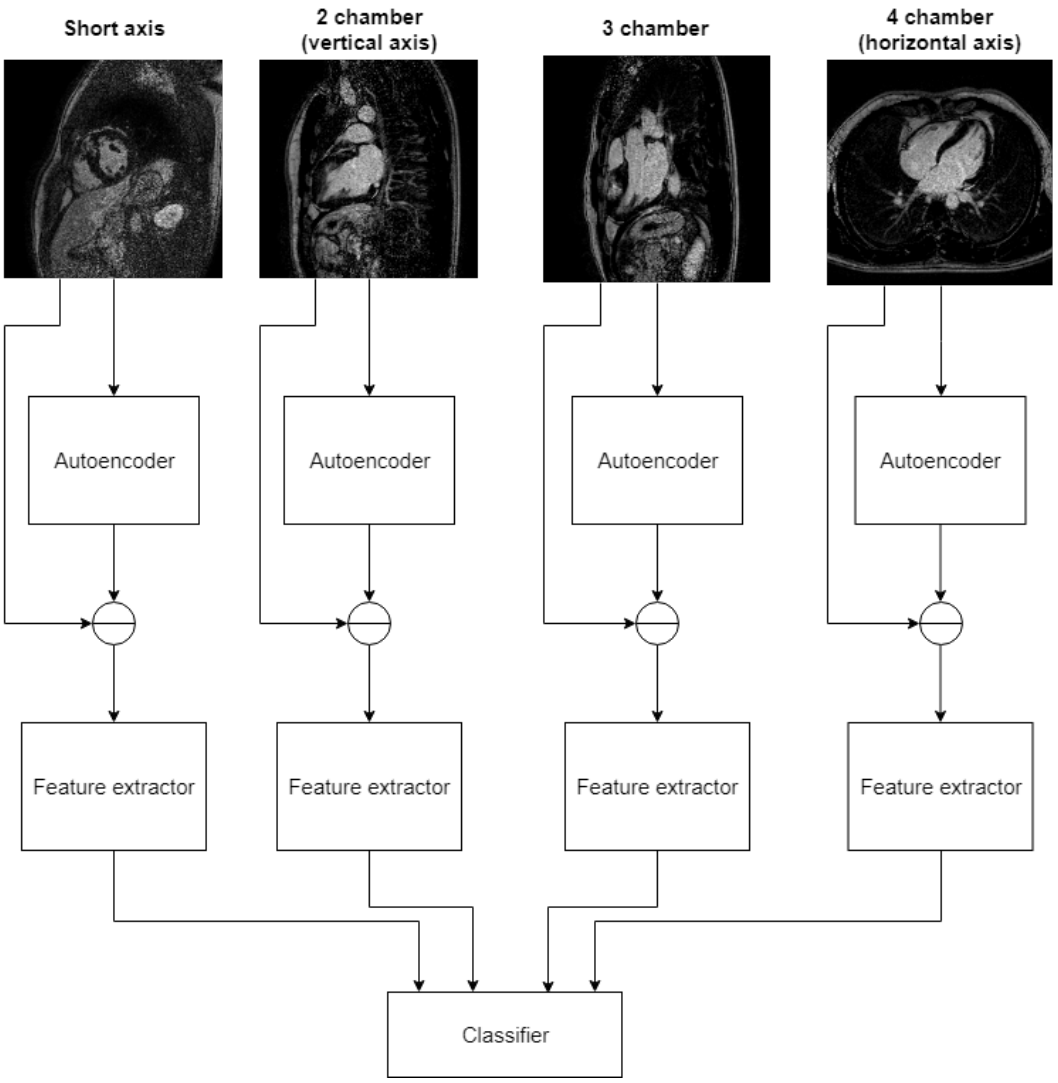

Figure 4.3: Architecture of the proposed model concept.

Chapter 5

Preprocessing the data

5.1 Structure of the dataset

As mentioned in the previous chapter the whole dataset consists of 890 patients' data. For each patient there is a separate folder which name is the same as the patients id. Inside the folder we can find a meta.txt file which contains the pathology of the patient, and one or more subfolders. Each subfolder corresponds to a view and scan sequence. The MRI images (frames) are inside these subfolders in DICOM (dcm) formats, each containing all the available information of the patient and technical details about the frame, sequence and the system used. The folder structure can be slightly different for some non-LGE short axis scans where contour data is available. In this case there is a contour.con file and an images folder inside the short axis subfolder.

DICOM [1] (Digital Imaging and Communications in Medicine) is a widely used international standard (ISO 12052) for storing and exchanging information in medical imaging. Most medical imaging devices use this standard to precouple and save image data with equipment related information.

Each patients data was about 100-150 MB of information, which means overall there was around 90 GB of data available. This is fairly large and most of it is not useful for the project so it had to be processed. The dataset was located on a remote server not directly accessible by me, instead I was granted a small sample, consisting of 5 selected patients' data, that had all the characteristics found in the main set. I had to create script that selects and separates important views, filters and collects a subset of the frames, then normalizes them and finally saves them in a serialization object for easy access. I would then receive the output of this script from the server and use that (after processing into a loadable dataset) for training and evaluating the model.

5.2 Tools used for processing

5.2.1 MicroDicom

DICOM data is usually accessed by healthcare experts through DICOM viewer software, which provides an easy access to the tagged medical data, tools for image analysis and a browser functionality to easily traverse through the data. I used the MicroDicom[2]

[1] Official website of DICOM: `https://www.dicomstandard.org/`
[2] Official website of MicroDicom: `https://www.microdicom.com/`

DICOM viewer during the initial inspection of the data and planning, which is a free non-commercial software for Windows equipped with most of the common image manipulation tools and a very transparent interface.

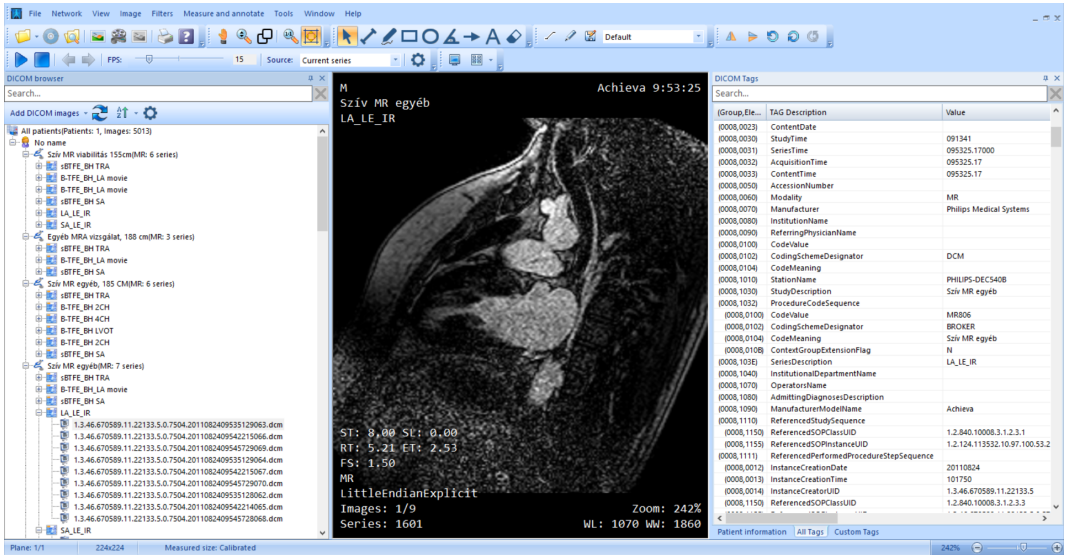

Figure 5.1: Sneak peek of the dataset in MicroDicom.

5.2.2 Pydicom

Pydicom[3] is a purely Python library that makes it easy to read, manipulate and save DICOM files. I used this library to parse the files in the dataset, extract the pixel arrays and orientational properties of the frames, and to apply modality look up table to the extracted pixel array to transform the device manufacturer dependent pixel values to independent, normalized ones.

5.2.3 Pickle

Pickle [4] is a python object serialization library. It uses binary protocols to serialize python object hierarchies into byte streams and vice versa. The serialization process is often referred to as ”pickling”, while the deserialization as ”unpickling”. It has some very handy features like backwards compatibility with python versions, being able to serialize and then restore user-defined class instances and it stores an internal reference graph to determine whether an object was pickled already or not. This prevents shared objects from being serialized multiple times.

5.2.4 NumPy

I used NumPy's [5] main module for manipulating arrays and it's linear algebra (*linalg*) submodule for some vector operations. NumPy is a very widely known and used library that provides the ability to create and handle highly optimized multi dimensional arrays.

[3] Official website of Pydicom: `https://pydicom.github.io/`
[4] Official documentation of the Pickle library: `https://docs.python.org/3/library/pickle.html`
[5] Official website of NumPy: `https://numpy.org/`

It also has a large library of implemented mathematical functions that can be used on these arrays.

5.3 Processing the data

The following things had to be done:

- Collect both non-LGE and LGE frames for the short axis and long axis, selecting evenly distributed images across the cardiac cycle.
- Separate long axis frames into the three main views.
- Collect the pathologies of the patients.

Collecting the pathologies was the easiest task as it only consisted of reading the value from the meta.txt file inside each patient directory.

5.3.1 Short axis images

Non-LGE short axis scans usually had around 14 different slices (13-16 based on the sample data) and 25 frames per slice. Each slice is a view of the heart in the direction of the short axis planes normal vector at different locations. At each location a cardiac cycle is recorded in 25 frames.

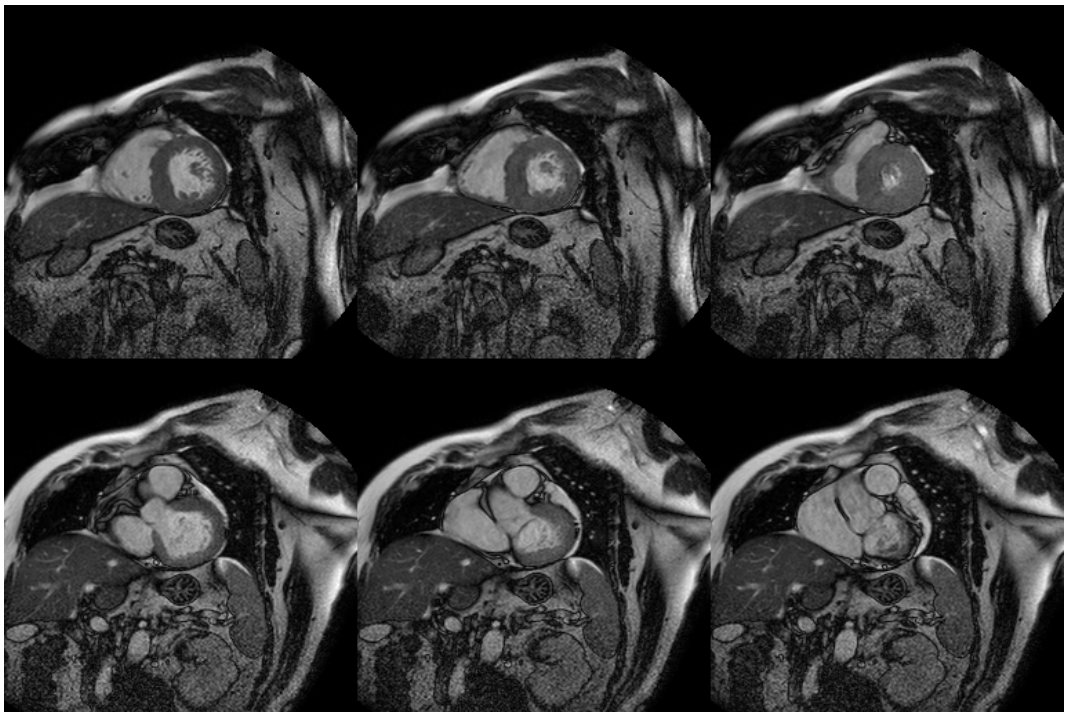

Figure 5.2: Two different slices of the same patients heart with 3 different frames (left to right) of the cardiac cycle. (Short axis view)

LGE scans are similar, but there are only three frames (sometimes less) per slice each captured with different echo time.

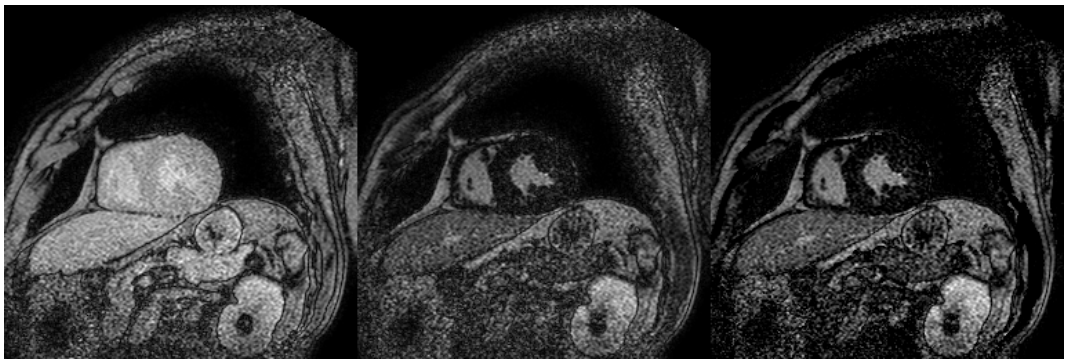

Figure 5.3: Captures (frames) of a slice with different echo times in an LGE short axis scan.

The short axis part of the script was rather simple. I created a class (by modifying a sample implementation granted by my advisor) which constructor takes a folder name then collects and orders all DICOM files found inside it. For each found DICOM file it extracts the frame's pixel array then applies a modality lookup table to normalize the image. The acquired frames are then grouped in different arrays based on their SliceLocation tags. Finally these arrays were grouped as well into a "slices" array. In the main script for each patient directory I checked for the existence of 'sa' (short axis) or 'sale' (short axis LGE) subdirectories and if found I initialized the previously introduced data collector class with it. The data objects were than passed to a function that was responsible for selecting a maximum of 5 evenly distributed slices and for every slice 3 evenly distributed frames. It seems like a large amount of data at first, but I wanted to make sure that I have enough to train the autoencoder properly. Less LGE images would probably have been sufficient, however having a bigger range of images to select from helps to compensate in case of future errors.

5.3.2 Long axis images

For the most part the collecting of long axis images was identical to that of the short axis ones. There was one major difference however that made it a bit more difficult. I had to figure out how to extinguish and collect the three different chamber views of the long axis. Similarly to short axis, the images were separated into slices and frames, and for the frames everything said in the previous section applied. The difference was that there were only 3 slices and each slice corresponded to one of the chamber views. Slice location is a relative value and it differs from scan to scan, so it could not be used to determine the view, fortunately there was another way of going about it using the value of a DICOM tag called ImageOrientationPatient [6]. This value contains the direction cosines of the first row and first column of the image with respect to the patient. This meant that by calculating the cross product of these direction cosines we can get the normal vector of the slice (or the direction of the view). The angle between this vector and the normal vector of the corresponding chamber view plane should be very low, thus by knowing the normal vectors of all view planes it is possible to tell which one of them the slice belongs to.

I collected all ImageOrientationPatient values from the sample data for both LGE and non-LGE long axis slices and manually separated them into the three view groups based

[6] Documentation entry for ImageOrientationPatient `https://dicom.innolitics.com/ciods/ct-image/image-plane/00200037`

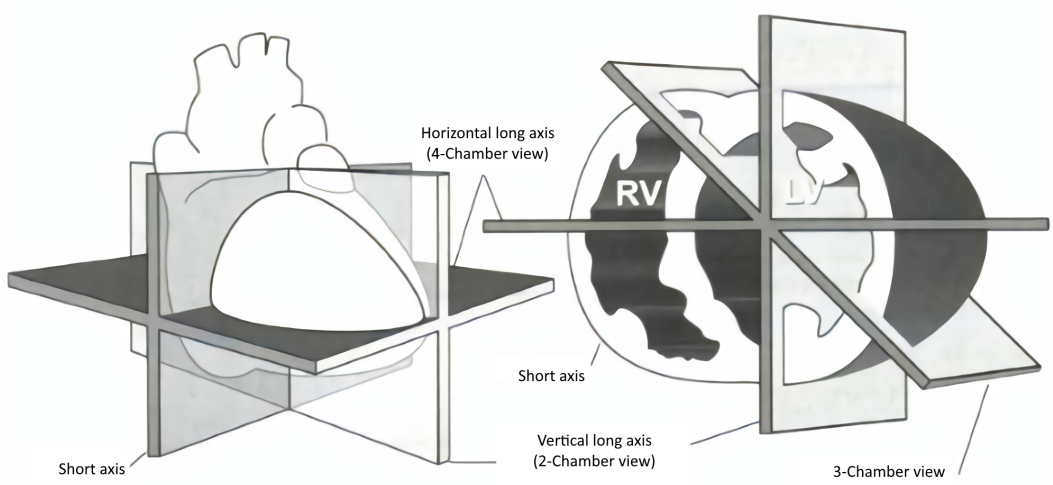

Figure 5.4: 4 main view planes of the heart. (Modified artwork of [49])

on the inspection of related images in MicroDicom. I calculated the normal vectors and their averages for each view group. Finally I built a data collector class that worked similarly to the previously introduced short axis one, but instead of creating an array of slices by grouping frames based on the SliceLocations, I assigned each frame to one of three chamber view arrays. The process to determine which view the frame belonged to was the following:

- Calculate the cross product of the direction cosines extracted from the ImageOrientationPatient tag to get the direction vector of the frame.
- For each chamber view, calculate the cosine of the angle between the resulting direction vector and the normal vector of the chamber view plane.
- Assign the frame to the chamber view for which the absolute value of the cosine was the largest.

After grouping the frames, the rest of the processing went the same way as for the short axis, but instead of multiple slices, there were only three chamber view arrays with each having usually 25 frames. I selected 15 evenly distributed frames for each view from the non-LGE scans and all three available frames for the LGE ones.

5.3.3 Filtering and serializing the processed data

I created a Patient class to hold all the data collected. Other than the images and the pathology I also saved the ID of the patient for easier tracking. Before serializing the Patient object I filtered the images to remove the corrupt (blank) ones. These were most likely produced by device failure on the initial scan and usually were compensated by a secondary scan, so in most cases the images could be replaced. I also rescaled the pixel data of images between 0-255 and converted them from float64 to uint8 data type to lower the resulting size of the processed dataset. After collecting, processing and assigning the data into data objects, I used the Pickle python library to serialize them into a pickle file that could then be transferred to my PC from the server and easily deserialized for later use.

5.3.4 Introduction to the collected data

There were 10 different pathology groups for patients. There was a large amount of imbalance among the groups, and some of them had very few samples. I only used the data of patients that had images for all four views, which also filtered out a considerable amount. For this reason I merged some of the groups later on during the training and testing of the model to compensate for it. I did not include two of the groups, U18_m (Under 18 male) and U18_f (Under 18 female), in the classification. I merged the groups in two ways creating different classification problems:

- Normal, Sport (adult_f_sport, adult_m_sport), HCM and Other (Fabry, EMF, Aortastenosis, Amyloidosis)
- Normal (Normal, Sport), Abnormal (HCM, Fabry, EMF, Aortastenosis, Amyloidosis)

	All	4 LGE scan
adult_f_sport	61	30
adult_m_sport	129	84
Amyloidosis	41	26
Aortastenosis	8	5
EMF	14	11
Fabry	9	6
HCM	258	218
Normal	218	138
U18_f	34	9
U18_m	118	30

Table 5.1: Number of patients for each pathology group overall and with all four view LGE scans available.

Chapter 6

Implementing the model concept

6.1 Tools and environments used

6.1.1 PyTorch

PyTorch[1] is an open source machine learning library developed by Facebook's AI Research. The library provides optimized tensor computing for deep learning solutions with both CPU and GPU support, and has many useful submodules that give access to implemented building blocks (layers and activation functions) of networks (*torch.nn*), optimization algorithms (*torch.optim*), efficient data loading (*torch.utils.data*) and much more. Some of the biggest strengths of the library are that it is tightly integrated with the Python language, it feels like a native python library and works seamlessly with most frequently used libraries like NumPy for example. It provides data parallelism to make use of multiple GPUs and has support for both ROCm and CUDA platforms, Cloud computing and TPU's as well. Pytorch also uses dynamic computational graphs meaning the networks can be easily changed and debugged programatically during runtime.

6.1.2 Torchvision

Torchvision[2] is a separate library inside the PyTorch ecosystem that provides access to popular datasets, implemented model architectures and common image transformations.

6.1.3 Scikitlearn

Scikitlearn[3] is an open source machine learning and data analysis library. It features a large range of optimized learning algorithms and metrics. It is primarily built on top of NumPy and SciPy meaning it can be easily used in combination with other ML and scientific libraries. I used the library's metrics module to help with calculating and measuring the performance of the classifier.

[1] Official website of PyTorch: `https://pytorch.org/`
[2] Official documentation of Torchvision `https://pytorch.org/vision/stable/index.html`
[3] Official website of Scikitlearn: `https://scikit-learn.org/stable/`

6.1.4 Matplotlib and Seaborn

Both Matplotlib[4] and Seaborn[5] are data visualization libraries for python. Matplotlib is a plotting library that can process NumPy arrays to produce a wide range of graphs. Seaborn is built on top of Matplotlib that makes it easier to create complex graphs by providing a simpler syntax and many good looking default themes. I used matplotlib to plot line graphs and visualize the change of model performance metrics over time, and seaborn to create matrix visualizations of the classifier model's outputs.

6.1.5 Google Colab

Google Colab[6] is a free cloud-based python development environment created and maintained by Google Research. It essentially enables the execution of python scripts on remote servers and grants access to GPU and TPU resources. I used this environment to train and evaluate the classifier model as it required a considerable amount of VRAM and the GPU's provided by Colab usually had between 12GB-16GB available. There was only one drawback of relying on this environment, I unfortunately could not run longer sessions due to usage limits so doing time heavy tasks like hyperparameter tuning were very complicated and sometimes even impossible.

6.2 Used loss functions

6.2.1 MSE

MSE or Mean square error is a very common and simple loss function. As it's name suggests, it calculates the loss by squaring the difference between each predicted and actual values (pixels in case of images), then summing the squares and finally dividing the sum by the number of values. Pytorch implementation used: *torch.nn.MSELoss*

6.2.2 Cross-entropy loss

Cross-entropy loss is probably the most commonly used loss function for classification problems. Cross-entropy is basically the difference between two probability distributions. In simple terms, for classification, cross-entropy loss compares the predicted probability of each class to their respective target values (0 or 1) and calculates the loss based on how big the difference was between these values. The loss is calculated logarithmically meaning it penalizes large differences significantly more than small ones.

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i)$$

Figure 6.1: Formula for Cross-entropy loss, where n is the number of classes, t_i is the truth/target value and p_i is the predicted probability for i^th class.

In case of multi-class (single label) classification, there is only one target class, which essentially reduces the formula to 6.2, this is called Categorical Cross-entropy.

[4]Official website of Matplotlib: `https://matplotlib.org/`
[5]Official website of Seaborn: `https://seaborn.pydata.org/`
[6]Access page of Google Colab: `https://colab.research.google.com/notebooks/intro.ipynb`

$$L_{CE} = -log(p)$$

Figure 6.2: Formula for Categorical Cross-entropy loss, where p is the predicted probability for the target class.

These formulas only make sense if the prediction probabilities make up for a valid probability distribution. This means that each probability have to be between 0 and 1 and their sum has to equal 1. The actual output of a network usual does not have a valid distribution, which is why softmax function is necessary. Softmax applies the standard exponential function to every predicted value then divides each one of them with the sum of resulting values. The exponential function handles negative values and the division normalizes each prediction between 0 and 1.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Figure 6.3: Formula for softmax function, where z is the input vector and K is the number of classes. The function calculates the normalized value for i^th element in the vector.

Pytorch implementation used: *torch.nn.CrossEntropyLoss*

6.3 Used optimization algorithms

6.3.1 Adam

Adam is an abbreviation for Adaptive Moment Estimation, where adaptive means that the algorithm adjusts the initial learning rate (by which the gradients are multiplied on adjusting the weights) based on previous gradients. Adam does this by dividing the learning rate by the exponentially decaying average of previous squared gradients and multiplying that with the exponentially decaying average of previous gradients. This results in a momentum like behavior, where the adjustment rate speeds up or exponentially decelerates depending on previous gradients. Pytorch implementation used: *torch.optim.Adam*

6.3.2 AMSGrad

Amsgrad is a very similar algorithm to Adam, which aims to solve a specific problem that has been observed and discussed in many papers in the recent years. In some specific scenarios the algorithm failed to generalize well compared to simpler ones like SGD with momentum. Redi et al. [44] showed that Adam fails to utilize large gradient changes introduced by small important parts of the data to converge out of suboptimal solutions. They suggested that the issue is the exponential averaging used by the algorithm, which diminishes the influence of sudden gradient changes and thus makes it less capable of finding a good generalization for datasets with important outliers. They proposed a fix for this problem with the introduction of Amsgrad which used the maximum of previous squared gradients instead of their exponential average and showed that it performed better on some smaller datasets. [7] Pytorch implementation used: *torch.optim.Adam, Amsgrad variant flag set to true*

6.4 Implementing the AutoEncoder

6.4.1 Creating the used custom dataset

The first task was to define the dataset that should be used and implement the process of selecting and feeding the input to the network. Fortunately, PyTorch provides a DataLoader class in the *torch.utils.data* submodule that handles the selecting and loading part. This class has many options to customize the process, from which the most important one is batch size. This value represents the number of samples (or the size of the mini-batch) that should be collected and propagated through the network at a time. Finding a good batch size is an important part of optimizing a model as it affects the estimation of the gradient indirectly. With bigger batch size, more errors are accumulated and used to calculate the gradient and more memory is taken by the results stored intermediate calculations. In case of using a GPU the batch have to first be transferred to the devices VRAM to be used. Another commonly used option of the DataLoader is shuffle, which if set true the loader uses a random sampling strategy to select data in a different order for each iteration (epoch). We can also assign a custom sampler to the DataLoader if a simple random pattern is not sufficient. In order to use the DataLoader class I had to create a custom PyTorch Dataset class that has two API requirement. It has to have a ___getitem___ method that given an index returns the associated sample in form of a tensor and a ___len___ method that returns the number of samples in the dataset. To train the AutoEncoder I needed all the available non-LGE images per view. I had around 10.000-11.000 images available for each view, which I separated into a training and validation set in a 80%-20% ratio so that I could test the performance of the model trained on the training set on data that it had not encountered before. I created a simple script that deserialized the collected patient data files and saved them into a folder with the name of the view it belonged to and a subfolder of either "validation" or "training". Afterwards I created the custom Dataset class, which constructor required a folder path and on initialization it indexed all the images found under the given folder, creating a map of index and absolute path pairs. The ___len___ method simply gave back the number of keys in the map. The ___getitem___ function loaded the image from the absolute path corresponding to the given index and applied a defined set of transformations one of which was to transform the image into a tensor that can be used by the model.

6.4.1.1 Applied transformations

I applied some transformations to the input images to improve the overall performance of the model:

- I cropped the images from 224x224 to 168x168 in the center using *Torchvision.transforms.CenterCrop*. After manually inspecting some of the images in each view, this seemed like a safe crop size for reducing some of the biological noise without removing any important information.
- I calculated the mean pixel intensity and standard deviation for the whole LGE dataset, then used *Torchvision.transforms.Normalize* to normalize each input image using Z-score normalization. This normalization technique simply consists of subtracting the mean from the pixel's intensity (value) then dividing the results by the standard deviation. This is useful as it lowers the impact of outliers in the data, but doesn't remove it entirely (as in case of min-max normalization).

- Due to the nature of the IR sequences, which were used for the LGE scans, these images were fairly noisy compared to the non-LGE ones. The network thus trained on data that had significantly less noise compared to the images it later had to reconstruct, which in turn introduced a strong denoising effect for the LGE images. This was a problem because upon subtracting the output from the input, to create the mask, the resulting image contained all the noise that was removed from the original image. To solve this I added Gaussian noise to the input images and trained the model to try to reproduce the noised images. I used *torch.randn_like* to create a similar sized tensor to each batch, multiplied the resulting tensor with 0.09 (standard deviation), added it to the input batch and finally used *torch.clamp* to restrict the pixel values between 0 and 1.
- Another idea was to randomly erase a rectangle shaped are of the image with white pixels and train the model to reproduce the original from that. This way the model would learn to denoise parts of the image where the pixel intensity should not be high and thus remove the scar tissue on LGE scans. I used the *RandomErasing* transform from the Torchvision library, which was originally proposed by Zhong et al. [51], for erasing the pixels of the input.

6.4.2 Implementation process

I have experimented with a few different architectures, in all cases I used MSE for loss function and Adam for optimizer. With most architectures the reconstructed images of the trained model were fairly accurate. The problem was, however, that in most cases the encoder did not denoise larger scar buildups. The spatial dimension reduction, by the bottlenecking of the network, caused the edges on the images to be smoothened as it can be seen on 6.4. This essentially made the model a reverse edge detector as subtracting the output from the input resulted in an image that only contained the edges found in the input. This worked good for the cases when the scar tissue was located near the epicardium and was relatively small, because the pixel intensities changed drastically only in a small pixel frame.

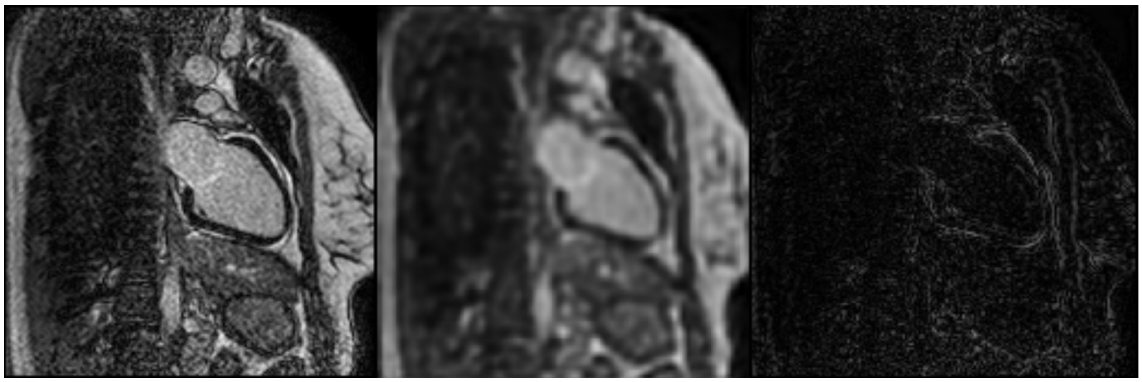

Figure 6.4: Reconstruction of a 2CH LGE image using 6.7 architecture.
Original - Left, Reconstruction - Middle, Pixel-wise difference - Right

I tried slightly larger kernel sizes for the first and last layers, reducing the amount of kernels for each layer and using a single average pooling (instead of two max pooling). The idea was that it would be easier to reconstruct the original image without loss, resulting in less

noise, but at the same time it would have to take into account a larger pixel area at a time and would have less capacity to work with making it generalize better. It worked for some extent, but sadly most of the times it created fake scars on the mask as it can be seen on 6.5.

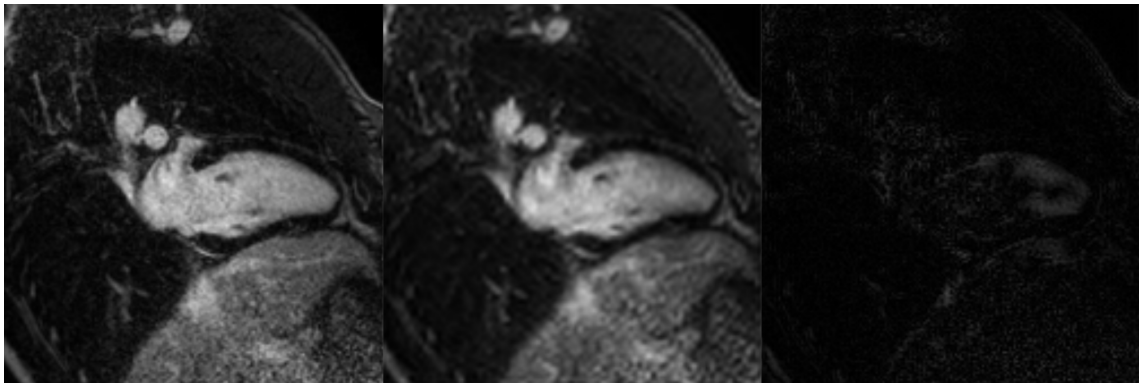

Figure 6.5: Reconstruction of a 2CH LGE image using 6.8 architecture.
Original - Left, Reconstruction - Middle, Pixel-wise difference - Right

After many trials and errors, the best model that I used later for classification, was an extension of both previous models. It used a larger initial kernel size as 6.5, roughly the same kernel numbers, but the rest of the architecture was the same as used for 6.4 with an extra bottleneck layer added. The masks created from the input of the model kept the edge detecting property, but highlighted the scar tissues slightly better. The edge detecting while at first seemed as unnecessary noise, which it was for some extent, I realized that by this the mask preserved the contours of the epicardium and endocardium as well. This meant that it was still possible to take the wall thickness into account, which is the first property to consider in determining the existence of hypertrophy.

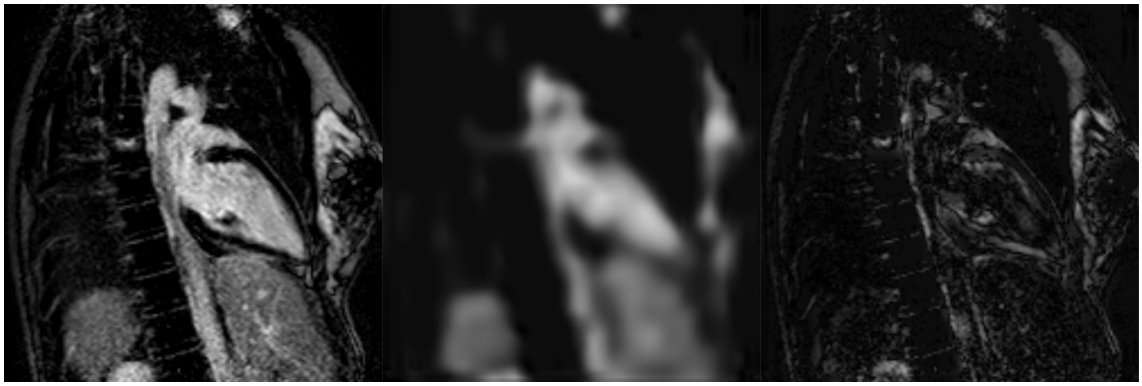

Figure 6.6: Reconstruction of a 2CH LGE image using 6.9 architecture.
Original - Left, Reconstruction - Middle, Pixel-wise difference - Right

6.4.3 Main architectures

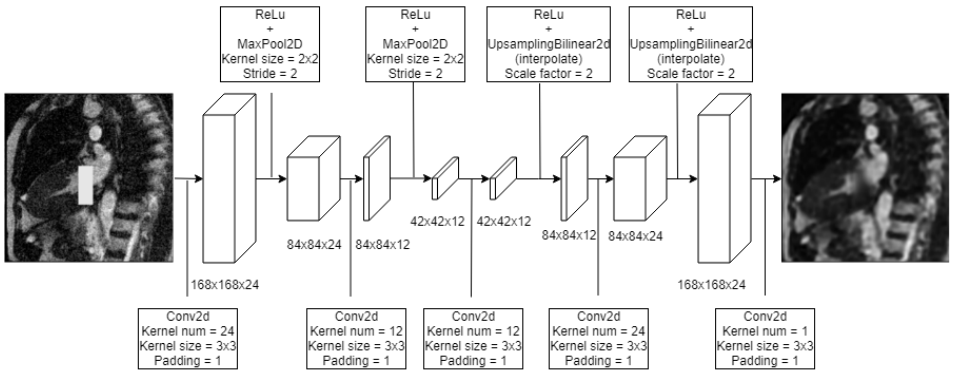

Figure 6.7: Autoencoder architecture *A*.

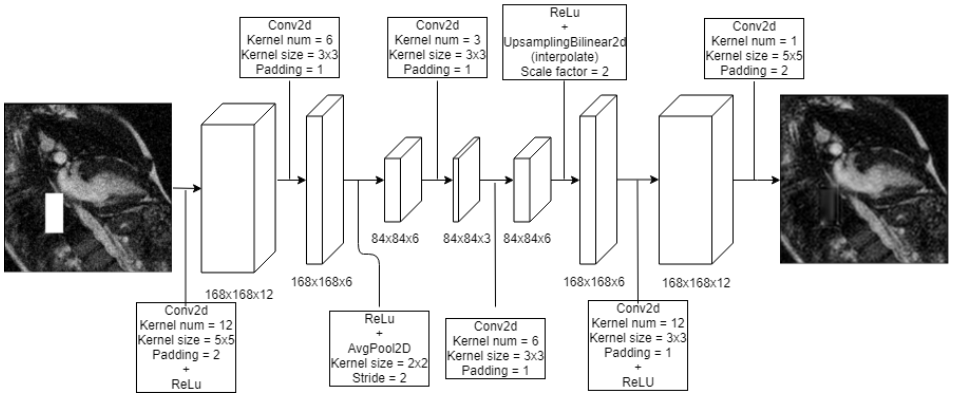

Figure 6.8: Autoencoder architecture *B*.

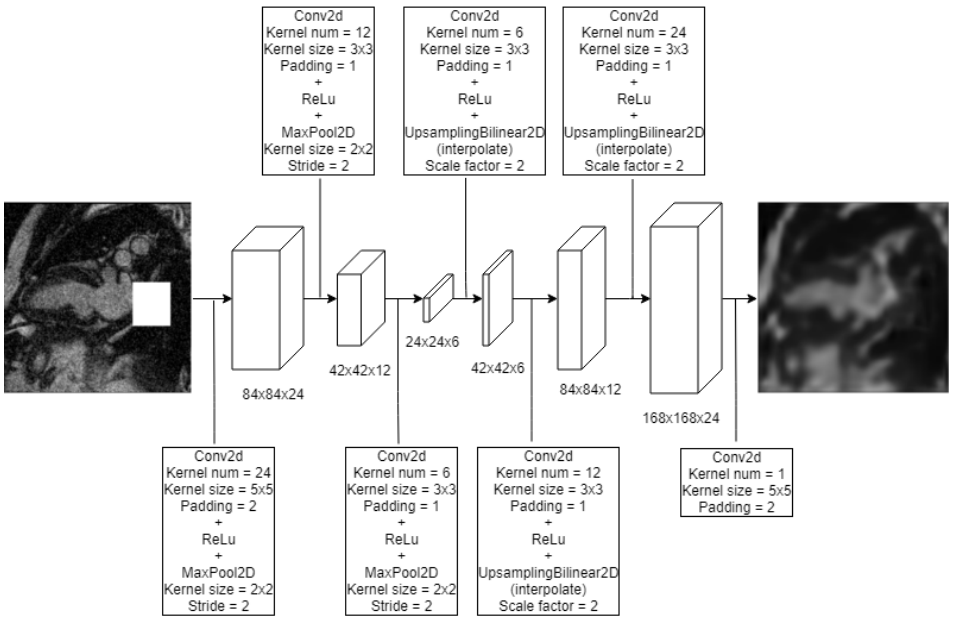

Figure 6.9: Autoencoder architecture *C*. (The final model.)

6.4.4 Further optimizations

By analyzing the results of the model and the original images, I noticed that from the three different echo timed LGE images the one being on the middle had the most distinguishable scar tissue and overall detail. To provide the best inputs for the classifier, I wrote a script that calculated the mean pixel intensity for each view's LGE images among all the patients, then selected the image with the closest mean to that value for each respective view inside every patient data. The result was the same patient class object as before, but for every view there was only one image. An extra benefit of this solution was that in case of short axis view, which had images from multiple slices, it also selected the slice in which the myocardium was best visible. This happened because the myocardium is very dark on high echo timed images and compared to that the epicardium is bright, so slices that were outside of the heart for example had a relatively high mean intensity and thus were further from the overall mean of the LGE dataset.

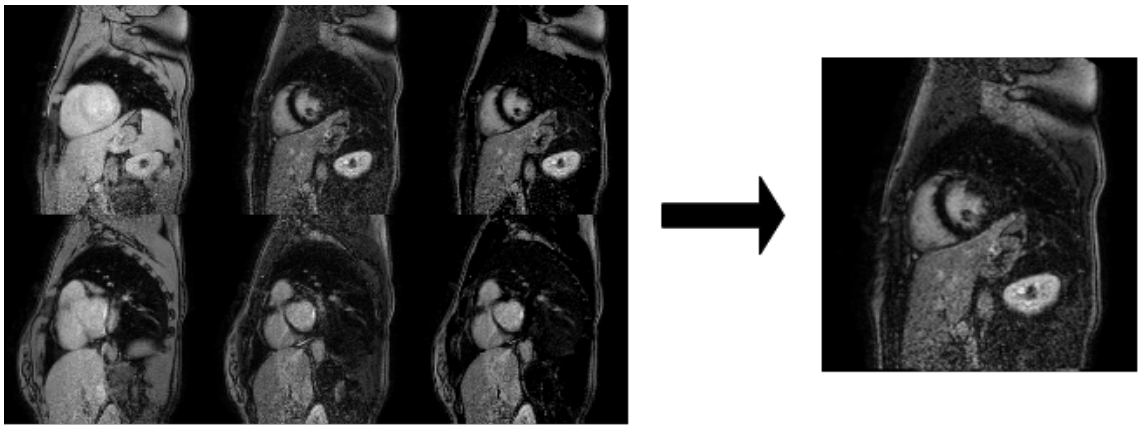

Figure 6.10: Selecting the most informative LGE image.

6.5 Implementing the ensemble classifier

6.5.1 Creating the used custom dataset

The input data for an image classification model consists of two parts, the images and their respective labels. There are many ways to structure these two parts together, for example a common approach is to hold the absolute path and label pair for each image in a csv file. I wanted to use a more visually accessible method, and so for each patient I created a folder, which name contained the label and the patients id, and had a subdirectory for each view in which the single image selected after 6.4.4 was found. I then created a dataset class that was able to parse this structure, map the image absolute paths and assign a class index for each patient data. This index served as the one-hot target vector for the input as it determined which element of the softmaxed predicted output vector should be considered and passed to the loss function.

6.5.1.1 Data augmentation

Data augmentation is a method for increasing the size and diversity of the dataset by applying (many times) random transformations to the data. Well-chosen transformation can help the model to generalize better and become more robust. However, not

all transformations are suitable for every dataset and classification problem as if done wrong they can generate misleading outliers and schemes that in turn make it harder for the model to fit for real data. Medical images are in general sensitive to heavier transformations as in many cases the diagnosis can depend on very miniature details and proportional differences. This is especially the case for LGE images as the position and volume of scar tissue can be the lead determining factor for diagnosing the underlying disease. (4.1) For this reason I only applied a random rotation (*torchvision.transforms.RandomRotation*) between -10° and 10° degrees and a random perspective transformation (*torchvision.transforms.RandomPerspective*) with a distortion scale of 0.1. As the autoencoders were trained on center cropped 168x168 images, I also applied the same amount of center cropping to the input images here as well. I separated the dataset into three parts, training, validation and test with 70%-15%-15% ratios. The classes, and in case of merges the subclasses, were divided with the same ratio as well to keep the exact class balance for each set.

6.5.2 Performance metrics

In order to track the performance of the model we have to first define a way to measure it. While the cross-entropy loss can be a good measure that takes into account the confidence of the network, it is model and dataset specific, the value only gives a relative overview for the performance of the model and cannot be interpreted generally. There are however metrics which provide a real, absolute measurement for the model's performance and make it possible to compare it to other ones in a similar problem domain. For a classification task the most important metrics like this are precision, recall, accuracy and F1-score. All of these rely on a one-hot binary interpretation of the predictions, where the element corresponding to the class with the highest predicted probability is considered as 1 and the rest as 0. This vector is than compared to the target one-hot vector and based on that for each class (element of the vector) it is labeled as one of four outcomes:

- True positive (TP) if both the target and predicted value is 1
- True negative (TN) if both the target and predicted value is 0
- False positive (FP) if the target value is 0 and the predicted is 1
- False negative (FN) if the target value is 1 and the predicted is 0

A good way to visualize these values for each class is to create a so called confusion matrix. It is a NxN matrix, where N is the number of classes, in which each row represents the target and each column the predicted class. The value inside the cells where the row and the column class matches is the number of TPs for that class.

6.5.2.1 Accuracy

Accuracy is the most straightforward metric of the four, as it tells how much of the predictions were right overall.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{6.1}$$

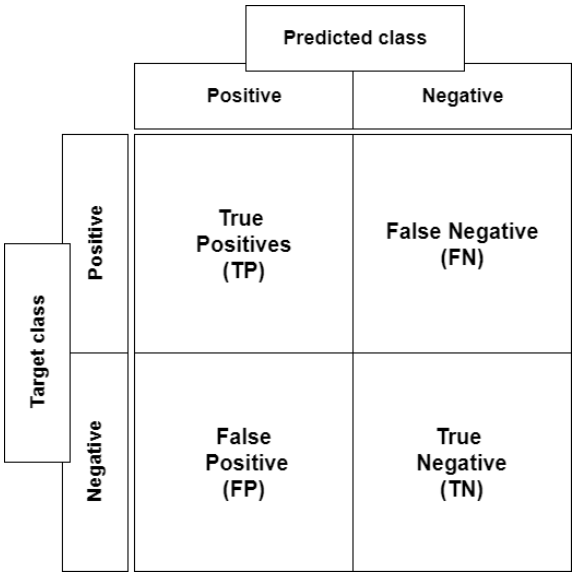

Figure 6.11: Confusion matrix in case of binary classification.

The problem with accuracy is that for example in case of heavy class imbalance it is easy to achieve high accuracy by always predicting or at least favoring the largest class for each input.

6.5.2.2 Precision and Recall

Precision is calculated as:

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

It tells us how many of the predicted classes, or in our case diagnosises were on point.

Recall (or sensitivity) is calculated as:

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

Which tells us how many of the occurrences of each class was diagnosed compared to the total.

To understand the importance of these two metrics, by projecting them to a binary medical diagnosis problem, we can say that precision penalizes diagnosing healthy patients with a disease, while on the other hand recall penalizes not diagnosing an otherwise ill one. For which one is more important, well it depends on the specific case, but usually recall is considered as the superior one for CADs as it is better to diagnose a healthy patient ill than to not recognize an underlying disease.

6.5.2.3 F1-score

F1-score aims to find the balance between these previous two metrics and therefore it is defined as their harmonic mean:

$$Recall = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{6.4}$$

While accuracy emphasizes TP and FP, F1-score penalizes mistakes more (FP, FN) and thus it is a much more suited metric for datasets with imbalanced class distributions where smaller classes have huge importance. For this reason F1-score is usually more important than accuracy, especially in case of medical diagnosis, so I adjusted my model based on this metric.

6.5.2.4 Applying the metrics for a multi-class problem

Precision, Recall and F1-score are all metrics initially defined for a binary classification problem. To make them work with multiple classes these scores are calculated for each class separately and then the results are used to determine the respective metrics. I used *sklearn.metrics.precision_recall_fscore_support* to calculate these, which provided three different approaches for combining the binary results:

- *micro*, which calculated total TP, FN and FP across all classes and used that to calculate the scores (this results in accuracy and F1-score being equal)
- *weighted*, which uses the weighted average of the individual scores where classes with more samples have larger weights. This is the exact opposite of what I wanted to achieve as it made the impact of class imbalance even bigger.
- *macro*, which uses the unweighted mean of each classes scores. This was the one that I used as in this case all classes had the same impact on the score.

6.5.3 Architecture

Based on previous experiences with the dataset and research, I decided to use ResNet-18 as the architecture for all four submodules of the ensemble. I have already briefly introduced the basics of this architecture at 3.1.5, the main reason I chose this one is that compared to it's size it has achieved a very good top 1 accuracy on ImageNet[7] (based on pytorches benchmarks 69.75%) and has relatively low computational requirement. Architectures like DenseNet[33] achieved better results, but usually are more memory and computational power hungry.

6.5.3.1 Transfer learning

I have experimented with some simpler custom made architectures, but they took a long time to converge and didn't get near the performance of ResNet-18. Other than performing better with random weights the implementation that I used from the *torchvision.models* library had pretrained weights available. This provided the opportunity to perform a process called transfer learning on the model. Transfer learning is really popular nowadays,

[7] Official website of ImageNet: `https://www.image-net.org/`

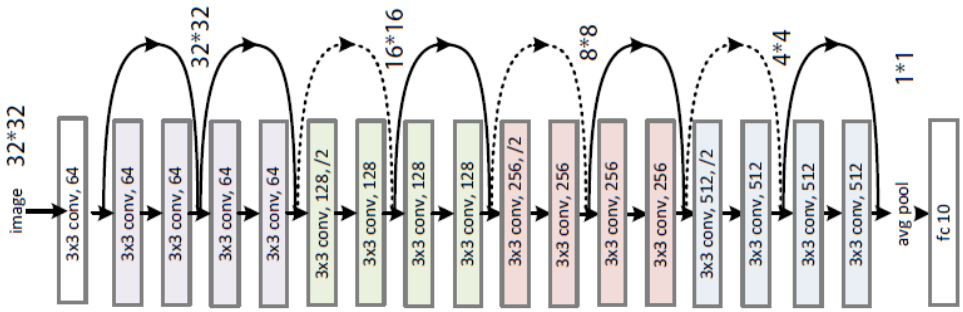

Figure 6.12: Architecture of ResNet18. [6]

it relies on the idea that similar, but different tasks require the model to essentially learn the same low level features. For example in case of image classification the first few layers of the network learn to recognize edges and shapes on the image and only the last layers learn how to interpret this information to classify the image. This means that the first layers will have similar weights on an identical model that is trained on a different image classification task. Utilizing this we can significantly speed up the learning process of the network and converge better on smaller datasets. I used a technique called fine tuning, which means that after loading the weights I enabled to train all the parameters of the network for each of the 4 submodules.

6.5.3.2 Creating the ensemble

The ensemble model contained 8 submodules, for each view a ResNet-18 module and as it's input a pretrained autoencoder. I freezed the weights of the autoencoders so that they were not adjusted during the learning process, the reason behind this was that the model already had a slightly large capacity for the dataset and thus letting the autoencoders train unnecessarily increased the overfitting (discussed later at 6.5.4.2) phenomenon further. Following the original idea I subtracted the autoencoders output from the original image tensor and provided the resulting mask as input for the respective ResNet submodule. ResNet originally takes inputs with 3 channels (RGB) so in order to use the grayscale masks created by the autoencoder I copied the single channel image tensor into all three channels of a new tensor. Each submodel had 1000 features as output (ImageNet has 1000 classes) so I concatenated them and used the result as the input for the final classifier inside the main module. The classifier consisted of one layer with an input size of 3000 and output size of 256 and a final layer with an input size of 256 and output size of the number of classes in question. The model thus followed the architectural concept given at 4.3.

6.5.4 Training and improving the model

At each epoch I ran two iterations, one training in which the parameters were adjusted and one validation for evaluating the freezed model after the training phase. I saved the model state corresponding to the best F1-score achieved on validation for later evaluation on the test set. I implemented a simple early stopping with a patience of 8 epochs on the sum of validation accuracy and F1-score to terminate the learning process if it doesn't converge any more. The first classification problem that I worked on and on which I mainly adjusted the model contained 4 classes, Normal, HCM, Sport and Other. This seemed like a good initial challenge for the model as it had a fairly big class imbalance

and diversity. The idea was that if the model can handle this problem well it will be able to perform good on fewer classes as well.

Normal	Sport	HCM	Other
138	114	218	48

Table 6.1: Distribution of the dataset as a 4 class problem. Other(EMF, Fabry, Aortastenosis, Amyloidosis)

6.5.4.1 Initial performance

First I tried the network with a very common setup using Adam with a $1e^{-4}$ learning rate, simple cross-entropy loss and a batch size of 32. The results were okay considering the imbalance of classes and the amount of overall data available, but definitely not ideal. The model showcased a considerable amount of overfitting (6.13), as it reached almost 100% accuracy for the training in the first few epochs, while achieving much less on validation. I tried to combat this issue with a few best practices, which worked to a certain amount, but unfortunately didn't solve the main issue originating from the size and distribution of the dataset. The initial benchmarks of the model without addressing the overfitting and class imbalance:

	Precision	Recall	F1-score	Accuracy
Training	98.64%	99.34%	98.98%	99.07%
Validation	70.29%	66.17%	67.37%	70.59%

Table 6.2: Benchmarks of the model without any optimization using the best found weights (based on F1-score) for the validation set.

Figure 6.13: Change of F1-score over training epochs for the model without optimization.

6.5.4.2 Addressing overfitting

Overfitting as it can be seen on 6.13 is a concept which means that the data has fitted to the training set perfectly, while failing to generalize well for other data outside of that. This is obviously bad as the best case scenario would be to find the best generalization for that problem domain and not to a specific dataset. Overfitting usually occurs due to the combination of too much model capacity and small training dataset. The last issue could not be solved, but there were a few things that could be done to improve the situation.

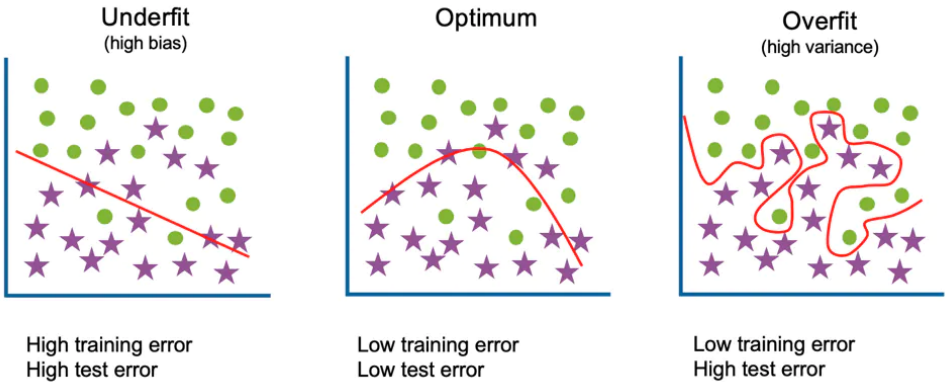

Figure 6.14: Underfitting, balanced and overfitting model showcased on bivariate data. [10]

Lowering the model capacity
The main thing that can be done against overfitting is to find the perfect model capacity that is just enough for generalizing on the training set. Initially I tried freezing specific layers of the network as a whole, but it was not easily adjustable and wasn't precise enough. A better alternative was that before passing the parameters of the network to the optimizer I iterated through them and freezed the first $X\%$ of them, which provided a lot more exact control over the capacity of the network.

Introducing dropout layers
Dropout is a simple technique in which we randomly ignore neurons of the network by setting their inputs to 0. This is useful as it forces the network to find more than one way to interpret data. It is most commonly used before dense layers as these are the most prone to co-adapting, creating strong interdependence and biasing towards specific inputs.

L1 and L2 regularization
There are two frequently used regularization methods that both aim to shrink the coefficients or the impact of each individual feature of the network minimalizing the complexity of the model. They both lower the impact of features by adjusting the corresponding weights of the network. L1 regularization mainly aims to zero out weights corresponding to features that provide no informative value to the model. On the other hand L2 regularization aims to lower the overall impact of individual weights by shrinking them towards zero, but not eliminating them entirely. They both achieve the regularization effect by adding penalties to the cost function. L1 penalty is calculated as the sum of absolute values of all weights, while L2 as the sum of squares of all weights. Both of these are

multiplied by a constant hyperparameter to control the magnitude of the regularization effect. In practice L1 performs a kind of feature selection where seemingly unimportant features are ignored, while L2 addresses multicollinearity by lowering the significance of the relationship between individual features. L2 regularization is implemented in most pytorch optimization algorithms. There is a parameter called *weight_decay* which is the constant by which the penalty is multiplied. L1 regularization is not implemented so I calculated the sum of absolute values for all weights of the network, multiplied it with a constant and added it to the final loss before backpropogation.

6.5.4.3 Addressing class imbalance

As discussed previously F1-score takes into account the performance of the model for each class equally. That is why the score is lower at 6.2 than the accuracy. To address the class imbalance and help the model achieve better results on smaller classes I tried two methods.

Oversampling
One of the issue with having a heavily imbalanced dataset is that for each batch there is only a slight chance that it will contain a sample from one of the smaller classes. This can cause problems, because if the first few batches contain only larger classes the network can quickly converge to a suboptimal point from which the loss of a single miss-predicted class may not be enough to revert from. To address this issue I created a function, that calculated the weights for each individual classes, assigning bigger probabilities for classes with less samples available. I used the resulting weights to create a *torch.utils.data.WeightedRandomSampler* and used that in the dataloader instead of the original random sampler. There was an option for the sampler to enable replacements which meant that if there was no more samples available to use in a batch it reused a previous one. This had to be turned on to fill all of the batches, but it unfortunately resulted in even more overfitting than before.

Using weighted loss
The other, better performing solution was to adjust the cross-entropy loss for each class by the weights calculated as mentioned in the previous paragraph. This way the penalty for miss-predicting a smaller class was proportionally bigger than for a larger class. This was also pretty simple to do as the pytorch implementation of Cross-Entropy loss (*torch.nn.CrossEntropyLoss*) had an option to set the weights by which the separate classes have to be multiplied.

6.5.4.4 Final performance

I tried many different combinations of the methods introduced in the previous section, and also experimented with using AMSGrad as optimizer instead of Adam. The final best results were achieved by using:

- batch size of 16
- weight decay (or L2 penalty) of $1e^{-5}$
- learning rate of $5e^{-5}$

- Adam as optimizer
- weighted Cross-Entropy loss
- a single dropout layer before the third layer of each submodule, with a probability of 0.2
- and finally freezing the first 20% of parameters for each submodule

Oversampling and L2 regularization in almost all cases provided worse results, AMSGrad in certain parameter combinations performed better than Adam, but overall achieved slightly worse scores. The intensity of overfitting lessened, as it can be seen on 6.15, the initial upwards curve of the train set smoothened compared to 6.13.

	Precision	Recall	F1-score	Accuracy
Training	99.48%	98.95%	99.20%	99.30%
Validation	73.75%	71.06%	71.80%	76.47%

Table 6.3: Benchmarks of the model, after addressing overfitting and class imbalance, using the best found weights (based on F1-score) for the validation set.

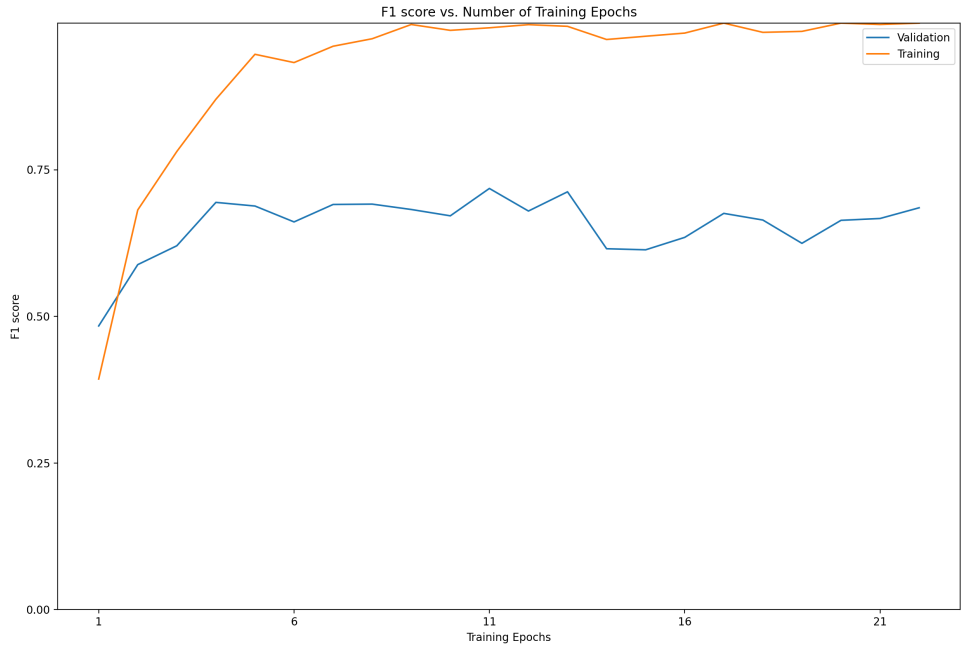

Figure 6.15: Change of F1-score over training epochs for the model after optimization.

Chapter 7

Evaluating the model

In this chapter I will showcase the test set results of the best performing model on validation for two separate classification problems. For comparison purposes I also adjusted, trained and evaluated the model using the original images without autoencoder based scar segmentation, and using the original short axis images only.

7.1 Normal-Sport-HCM-Other classification

As discussed in the previous chapter, this was the classification problem on which I adjusted and optimized the model. The class distribution can be seen at 6.1. The "Other" class contained the subclasses Amyloidosis, Aortastenosis, EMF and Fabry. The results for the three different approaches were the following:

	Precision	Recall	F1-score	Accuracy
4w/Ae-validation	73.75%	71.06%	71.80%	76.47%
4w/Ae-test	72.02%	69.48%	69.07%	70.27%
4w/oAe-validation	80.02%	68.01%	71.54%	75.00%
4w/oAe-test	82.73%	76.02%	77.74%	78.38%
sa-validation	55.77%	57.69%	55.90%	58.82%
sa-test	56.04%	57.84%	56.35%	59.46%

Table 7.1: Test set, and for reference, validation set results of the model for the Normal-Sport-HCM-Other classification problem.
4w/Ae = using all 4 views' images with autoencoder based scar segmentation
4w/oAe = using all 4 views' original images
sa = using only the original short axis images

Based on the results it was clear that using all four views of the heart improved the predictions significantly. On the other hand the model using the original images performed better on the test set than the one with scar masks. If we look at their confusion matrices 7.1 we can see that the most confusing class was Sport. Normal vs Sport was the hardest to predict which is not surprising as a heart with slight eccentric hypertrophy looks very close to a normal one. We can also see that the model using the original images and the one using the scar masks had very similar true predictions for all classes except Sport, where the former outperformed the latter. This is also an expected outcome as hearts with eccentric hypertrophy have no (or minimal) visible LGE, just like normal ones. Since

the scar mask model was based on utilizing gadolinium buildups as the main source of information, it had a much harder time differentiating between the two classes.

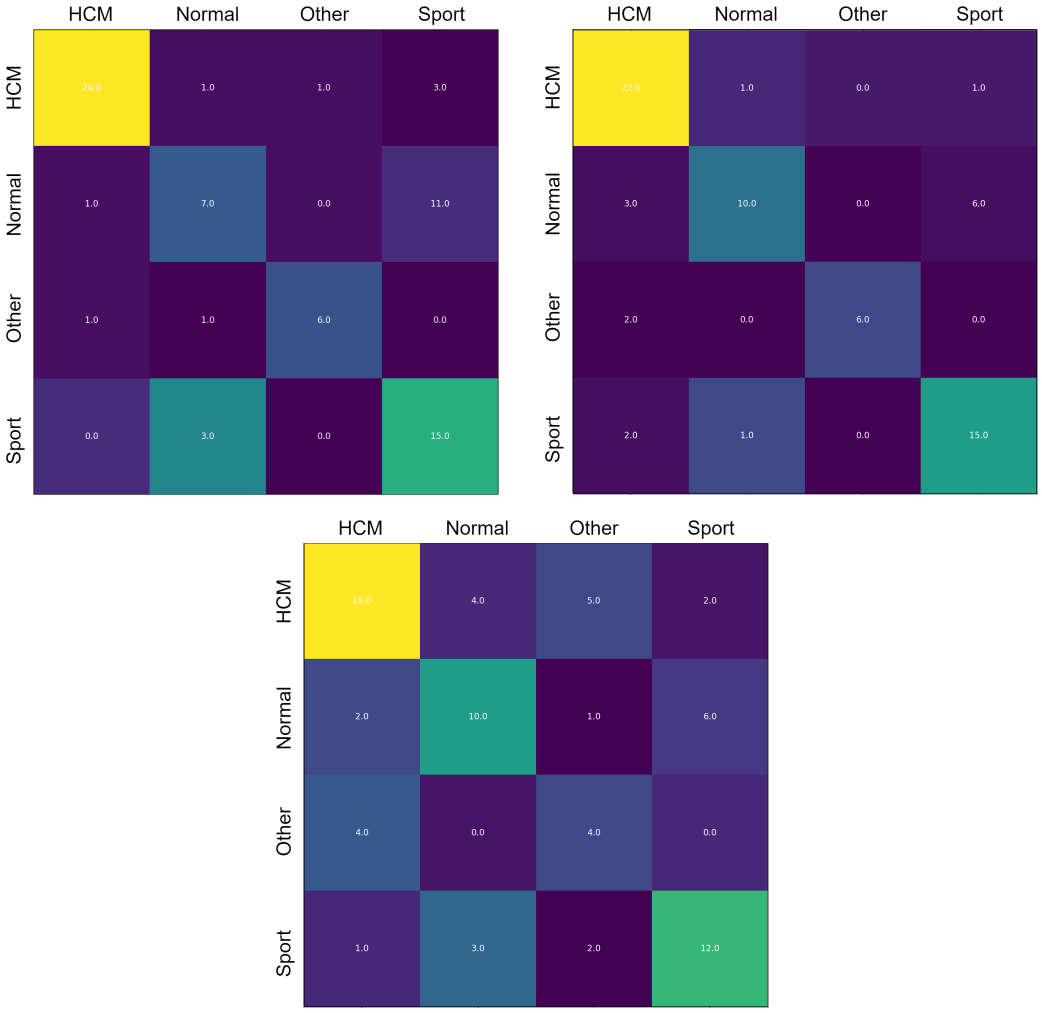

Figure 7.1: Confusion matrixes on test data for 4 class classification problem.
Top-left: 4 views + autoencoder based scar segmentation
Top-right: 4 views + original images
Bottom: Short axis images only

7.2 Normal-Abnormal classification

As previously discussed, athlete's or sport heart is not necessarily a dangerous condition. It is a natural adaptive response and to a certain degree it is considered healthy. A good classification task was to separate the dataset into a normal (or healthy) and abnormal class to determine whether the patients condition is serious or not. This separation of the dataset also eliminated most of the class imbalance, which in turn provided much better results.

Normal	Abnormal
252	266

Table 7.2: Distribution of the dataset as a binary class problem.
Normal (Sport, Normal), Abnormal (HCM, EMF, Fabry, Aortastenosis, Amyloidosis)

	Precision	Recall	F1-score	Accuracy
4w/Ae-validation	94.12%	94.12%	94.12%	94.12%
4w/Ae-test	90.57%	90.54%	90.54%	90.54%
4w/oAe-validation	95.95%	95.59%	95.58%	95.59%
4w/oAe-test	93.02%	91.89%	91.84%	91.89%
sa-validation	88.77%	88.24%	88.19%	88.24%
sa-test	85.16%	85.13%	85.13%	85.14%

Table 7.3: Test set, and for reference, validation set results of the model on Normal-Sport-HCM-Other classification problem.
4w/Ae = using all 4 views' images with autoencoder based scar segmentation
4w/oAe = using all 4 views' original images
sa = using only the original short axis images

The combination of lower class number, decreased confusing factor (Normal-Sport) and balanced dataset had achieved a significant increase in performance. Similarly to the 4 class problem, the model using the images of a single view scored the worst. Interestingly the model using the 4 original images yet again outperformed the one using the scars, although the difference was very slight this time. By looking at the confusion matrices 7.2 we can see that the difference between the two was only one prediction, the one using the scar masks predicted the abnormal classes, while the other one the normal classes better.

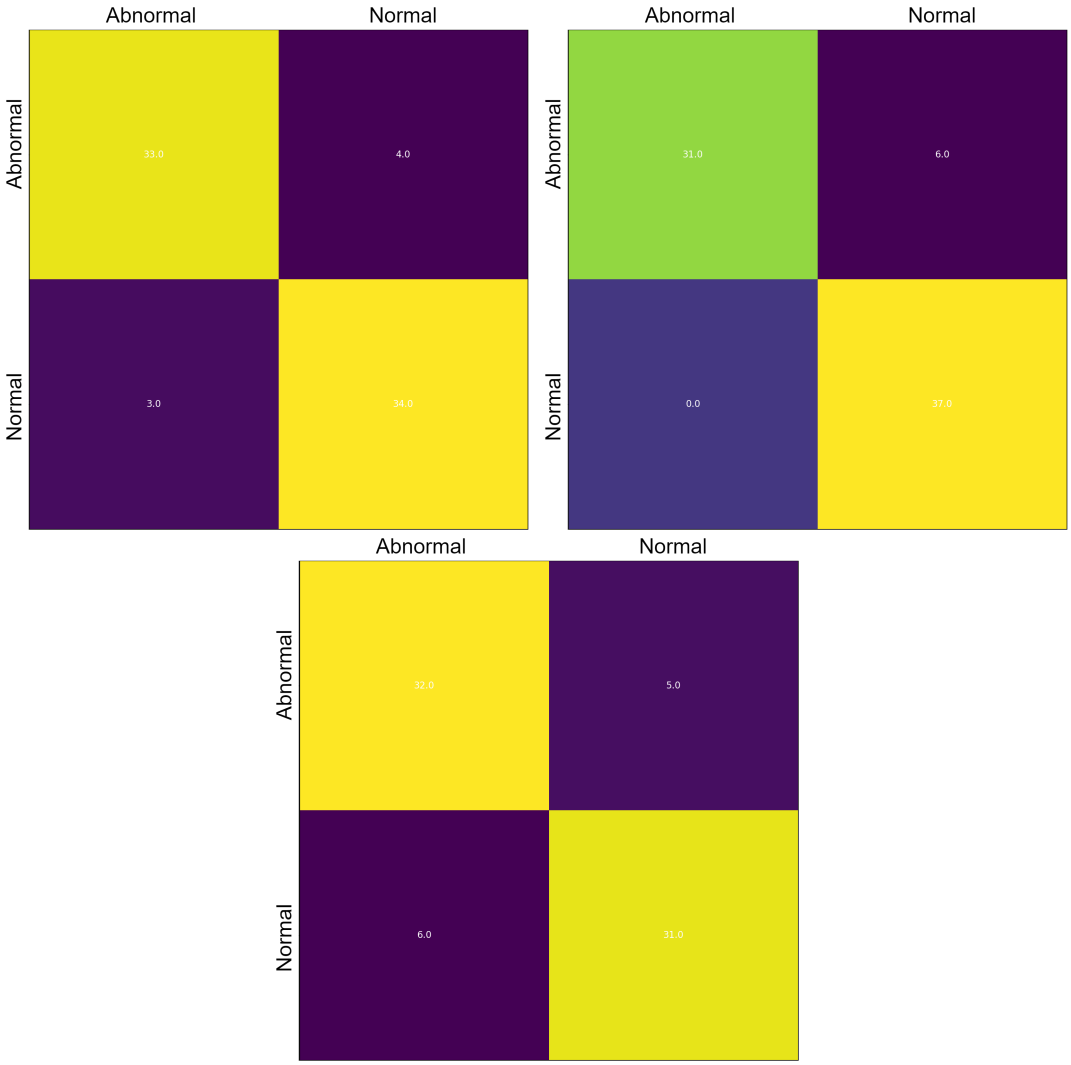

Figure 7.2: Confusion matrixes on test data for binary classification problem.
Top-left: 4 views + autoencoder based scar segmentation
Top-right: 4 views + original images
Bottom: Short axis images only

Chapter 8

Conclusion

The results showcased in the last chapter were a good demonstration of the power of ensemble models in cardiac MRI classification and in general. The use of multiple views provided a considerable amount of increase in performance in both examined classification tasks. In a previous project I experimented with similar concepts using short axis images only and I managed to reach better results than the ones used for comparison, however these were also slightly worse than what the models using all four views achieved. The scar segmentation approach unfortunately didn't provide the desired results. The autoencoder model used for scar segmentation can still definitely be improved on. The best approach would be to use the scar masks as a complementary information to the original images as the existence of LGE buildups in itself is not enough to differentiate between Sport, Normal and HCM images. The parameters of the model were mainly adjusted manually, most likely more optimization in this regard would have improved the results. A good option for this would be to use a hyperparameter optimization tool like RayTune[1] to find the best parameter combinations over multiple trials. Most of the training and evaluation happened on Google Colab, which has very strict limitations on longer run sessions so the possibility of such approaches were unfortunately very limited.

Overall I can safely say that the future of deep learning (and AI in general) is just as, if not more, promising in healthcare than in other fields. Deep learning based medical image analysis could provide a fast and robust companion tool for doctors to help in achieving better diagnosises. As Davenport and Ravi said[23] the biggest limitation for AI based approaches in this field is no longer the technology and success rate. The challenge that still lies ahead is the standardization and overall acceptance of these methods, which will take a considerable amount of time. Nevertheless, I expect and agree with Davenport and Ravi, that in the following years we will see a slow, but steady increase in the popularity and application of AI in medicine.

[1] Official website and documentation of Ray Tune: `https://docs.ray.io/en/latest/tune/`

Bibliography

[1] AlphaFold: Using AI for scientific discovery. *Deepmind*. URL /blog/article/AlphaFold-Using-AI-for-scientific-discovery.

[2] Medical Imaging Cloud AI. *Arterys*. URL https://arterys.com/clinicalApp/cardioapp.

[3] Max-pooling / pooling. *Computer Science Wiki*. URL https://computersciencewiki.org/images/8/8a/MaxpoolSample2.png.

[4] Thoughtco figure heart_interior-570555cf3df78c7d9e908901.jpg (900×596). URL https://www.thoughtco.com/thmb/2iyxbE0xfxRP7gE5KEY4exXyjzU=/900x0/filters:no_upscale():max_bytes(150000):strip_icc():format(webp)/heart_interior-570555cf3df78c7d9e908901.jpg.

[5] Magnetic Resonance Imaging (MRI). URL https://www.nibib.nih.gov/science-education/science-topics/magnetic-resonance-imaging-mri.

[6] ResNet-18 implements Cifar-10 image classification Pytorch - Programmer Sought. URL https://www.programmersought.com/article/68543552068/.

[7] An overview of gradient descent optimization algorithms. *Sebastian Ruder*, January 2016. URL https://ruder.io/optimizing-gradient-descent/.

[8] What is the difference between Deep Learning and Machine Learning? Quantdare. *Quantdare*, August 2019. URL https://quantdare.com/wp-content/uploads/2019/06/deep_learning-840x766.png. Section: Artificial Intelligence.

[9] The heart: Anatomy, how it works, and more. September 2020. URL https://www.medicalnewstoday.com/articles/320565.

[10] What is Overfitting? *IBM Cloud Education*, March 2021. URL https://www.ibm.com/cloud/learn/overfitting.

[11] Receptive field. *Wikipedia*, April 2021. URL https://en.wikipedia.org/w/index.php?title=Receptive_field&oldid=1016662663. Page Version ID: 1016662663.

[12] Universal approximation theorem. *Wikipedia*, April 2021. URL https://en.wikipedia.org/w/index.php?title=Universal_approximation_theorem&oldid=1018516631. Page Version ID: 1018516631.

[13] Cardiac magnetic resonance imaging. *Wikipedia*, April 2021. URL https://en.wikipedia.org/w/index.php?title=Cardiac_magnetic_resonance_imaging&oldid=1019970193. Page Version ID: 1019970193.

[14] Ventricular hypertrophy. *Wikipedia*, February 2021. URL https://en.wikipedia.org/w/index.php?title=Ventricular_hypertrophy&oldid=1009298552. Page Version ID: 1009298552.

[15] Muscle hypertrophy. *Wikipedia*, April 2021. URL https://en.wikipedia.org/w/index.php?title=Muscle_hypertrophy&oldid=1019755424. Page Version ID: 1019755424.

[16] Ravi Aggarwal, Viknesh Sounderajah, Guy Martin, Daniel S. W. Ting, Alan Karthikesalingam, Dominic King, Hutan Ashrafian, and Ara Darzi. Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis. *npj Digital Medicine*, 4(1):1–23, April 2021. ISSN 2398-6352. DOI: 10.1038/s41746-021-00438-z. URL https://www.nature.com/articles/s41746-021-00438-z. Number: 1 Publisher: Nature Publishing Group.

[17] S. M. Artham, C. J. Lavie, R. V. Milani, D. A. Patel, A. Verma, and H. O. Ventura. Clinical impact of left ventricular hypertrophy and implications for regression. *Prog Cardiovasc Dis*, 52(2):153–167, 2009.

[18] Meghna Asthana. Introduction to Convolutional Neural Networks. *Medium*, March 2020. URL https://medium.com/analytics-vidhya/introduction-to-convolutional-neural-networks-c50f41e3bc66.

[19] Regina Baily. The 3 Layers of the Heart Wall. *ThoughtCo*. URL https://www.thoughtco.com/the-heart-wall-4022792. Section: ThoughtCo.

[20] Stan Benjamens, Pranavsingh Dhunnoo, and Bertalan Meskó. The state of artificial intelligence-based FDA-approved medical devices and algorithms: an online database. *npj Digital Medicine*, 3(1):1–8, September 2020. ISSN 2398-6352. DOI: 10.1038/s41746-020-00324-0. URL https://www.nature.com/articles/s41746-020-00324-0. Number: 1 Publisher: Nature Publishing Group.

[21] Bostjan Berlot, Chiara Bucciarelli-Ducci, Alberto Palazzuoli, and Paolo Marino. Myocardial phenotypes and dysfunction in HFpEF and HFrEF assessed by echocardiography and cardiac magnetic resonance. *Heart Fail Rev*, 25(1):75–84, January 2020. ISSN 1573-7322. DOI: 10.1007/s10741-019-09880-4. URL https://doi.org/10.1007/s10741-019-09880-4. Figure 1.

[22] G Cybenko. Approximation by superpositions of a sigmoidal function. page 12, 1989. URL https://web.njit.edu/~usman/courses/cs675_fall18/10.1.1.441.7873.pdf.

[23] Thomas Davenport and Ravi Kalakota. The potential for artificial intelligence in healthcare. *Future Healthc J*, 6(2):94–98, June 2019. ISSN 2514-6645. DOI: 10.7861/futurehosp.6-2-94. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6616181/.

[24] Jeffrey De Fauw, Joseph R. Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, George van den Driessche, Balaji Lakshminarayanan, Clemens Meyer, Faith Mackinder, Simon Bouton, Kareem Ayoub, Reena Chopra, Dominic King, Alan Karthikesalingam, Cían O. Hughes, Rosalind Raine, Julian Hughes, Dawn A. Sim, Catherine Egan, Adnan Tufail, Hugh Montgomery, Demis Hassabis, Geraint Rees, Trevor Back, Peng T. Khaw, Mustafa Suleyman, Julien Cornebise,

Pearse A. Keane, and Olaf Ronneberger. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nat Med*, 24(9):1342–1350, September 2018. ISSN 1546-170X. DOI: 10.1038/s41591-018-0107-6.

[25] de Simone Giovanni. Concentric or Eccentric Hypertrophy: How Clinically Relevant Is the Difference? *Hypertension*, 43(4):714–715, April 2004. DOI: 10.1161/01.HYP.0000121363.08252.a7. URL https://www.ahajournals.org/doi/full/10.1161/01.HYP.0000121363.08252.a7. Publisher: American Heart Association.

[26] Junxi Feng, Xiaohai He, Qizhi Teng, Chao Ren, Honggang Chen, and Yang Li. Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. *Physical Review E*, 100, 09 2019. DOI: 10.1103/PhysRevE.100.033308. URL https://www.researchgate.net/figure/Commonly-used-activation-functions-a-Sigmoid-b-Tanh-c-ReLU-and-d-LReLU_fig3_335845675.

[27] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernetics*, 36 (4):193–202, April 1980. ISSN 1432-0770. DOI: 10.1007/BF00344251. URL https://doi.org/10.1007/BF00344251.

[28] Frank Gaillard. MRI sequences (overview) | Radiology Reference Article | Radiopaedia.org. *Radiopaedia*. URL https://radiopaedia.org/articles/mri-sequences-overview?lang=gb.

[29] Latifa Guesmi, H. Fathallah, and Mourad Menif. Figure 2. McCulloch-Pitts computational model of a neuron. *ResearchGate*. URL https://www.researchgate.net/figure/McCulloch-Pitts-computational-model-of-a-neuron_fig2_323465059.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL http://arxiv.org/abs/1512.03385. arXiv: 1512.03385.

[31] Ryan Hird. The Basics of MRI Interpretation, Figure 1. *Geeky Medics*. URL https://geekymedics.com/the-basics-of-mri-interpretation/.

[32] Vikram Hospital. Diagnosis, Causes & Treatment Options for Left Ventricular Hypertrophy(LVH), August 2018. URL https://www.vikramhospital.com/blog/diagnosis-causes-treatment-options-left-ventricular-hypertrophylvh/.

[33] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, January 2018. URL http://arxiv.org/abs/1608.06993. arXiv: 1608.06993.

[34] Michael A. Ibrahim, Bita Hazhirkarzar, and Arthur B. Dublin. *Gadolinium Magnetic Resonance Imaging*. StatPearls Publishing, Treasure Island (FL), 2021. URL http://www.ncbi.nlm.nih.gov/books/NBK482487/.

[35] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

[36] E. Richard Klabunde. CV Physiology | Cardiac Cycle. URL https://www.cvphysiology.com/Heart%20Disease/HD002.

[37] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object Recognition with Gradient-Based Learning. 1998. URL http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf.

[38] Jean-Christophe B. Loiseau. Rosenblatt's perceptron, the very first neural network. *Medium*, February 2021. URL https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a.

[39] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943. ISSN 1522-9602. DOI: 10.1007/BF02478259. URL https://link.springer.com/article/10.1007/BF02478259. Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 4 Publisher: Kluwer Academic Publishers.

[40] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computatuional Geometry*. 1969. URL https://mitpress.mit.edu/books/perceptrons. Publisher: The MIT Press.

[41] Aliasghar Mortazi, Jeremy Burt, and Ulas Bagci. Multi-Planar Deep Segmentation Networks for Cardiac Substructures from MRI and CT. pages 199–206, 2018. DOI: 10.1007/978-3-319-75541-0_21.

[42] Jeff Nyman. Demystifying Machine Learning, Part 5 – Stories from a Software Tester. URL http://testerstories.com/files/ai_and_ml/ml-mnist-2d-to-1d.png.

[43] David C. Preston. MRI Basics. URL https://case.edu/med/neurology/NR/MRI%20Basics.htm.

[44] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the Convergence of Adam and Beyond. February 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

[45] Anh H. Reynolds. Convolutional Neural Networks (CNNs). URL https://anhreynolds.com/img/cnn.png.

[46] Frank Rosenblatt. The Perceptron — A Perceiving and Recognizing Automaton. January 1957. URL https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf.

[47] Matt Skalski. MRI sequence parameters | Radiology Reference Article | Radiopaedia.org. *Radiopaedia*. URL https://radiopaedia.org/articles/mri-sequence-parameters.

[48] Prof. Loc Vu-Quoc. Artificial neuron. *Wikipedia*, March 2021. URL https://en.wikipedia.org/w/index.php?title=Artificial_neuron&oldid=1012965643. Page Version ID: 1012965643.

[49] Zezo. Cardiac Imaging Planes. *Thoracic Key*, June 2016. URL https://thoracickey.com/cardiac-imaging-planes/. FIGURE III2-1.

[50] Jeffrey Zhang, Sravani Gajjala, Pulkit Agrawal, Geoffrey H. Tison, Laura A. Hallock, Lauren Beussink-Nelson, Mats H. Lassen, Eugene Fan, Mandar A. Aras, ChaRandle Jordan, Kirsten E. Fleischmann, Michelle Melisko, Atif Qasim, Sanjiv J. Shah, Ruzena Bajcsy, and Rahul C. Deo. Fully Automated Echocardiogram Interpretation in Clinical Practice. *Circulation*, 138(16):1623–1635, October 2018. ISSN 1524-4539. DOI: 10.1161/CIRCULATIONAHA.118.034338.

[51] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random Erasing Data Augmentation. *arXiv:1708.04896 [cs]*, November 2017. URL `http://arxiv.org/abs/1708.04896`. arXiv: 1708.04896.