UNIVERSITY
OF OSLO

Master's thesis

Exploring Reinforcement Learning for End-Diastolic and End-Systolic Frame Detection

Magnus Dalen Kvalevåg

60 study points

Department of Informatics
The Faculty of Mathematics and Natural Sciences

Abstract

The thesis explores ways of formulating the problem of detecting the key cardiac phases from ultrasound videos, i.e., the end diastolic (ED) and end systolic (ES) phases, as a reinforcement learning (RL) problem, and whether there are any benefits in doing so. Of particular interest is the design of the RL reward function. Three reward functions are explored: one based on a generalization of the performance metric of average absolute frame difference (aaFD) that is only given to the agent at the end of an episode, and two based on per-frame phase classification given at every step. Additionally, two formulations of the RL environment are explored: binary classification environment (BCE), designed to be a direct reformulation of a supervised binary classification task, and m-mode binary classification environment (MMBCE), designed to provide the agent with the ability to explore the environment using synthetic m-mode imaging. Because of time constraints, MMBCE was only preliminary explored, yet the results indicate that the problem is too complex for the current setup and requires more work before we can draw any conclusions on its feasibility.

Experiments show that an RL agent is able to learn to perform phase detection even when the reward signal is very sparse. However, the less sparse reward functions perform better on nearly all metrics. The best agent predicts the correct number of ED and ES events in $\sim$ 80% of the videos on the test set, on which it yields an aaFD score of 1.69. It is concluded that there are multiple ways of formulating the problem of phase detection as a reinforcement learning problem, but not all formulations perform equally well. Reward sparsity and environment complexity contribute negatively to performance overall. There are also indications that lower values of the ϵ -greedy exploration hyperparameter ϵ have a regularizing effect on the model, prompting further research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter presents the motivation, goal, and structure of the thesis. Section 1.1 introduces the problem that the thesis aims to solve and gives motivation for why it needs to be solved. Section 1.2 states the goal and research questions of the thesis explicitly. Lastly, section 1.3 gives an overview of the structure of the rest of the thesis.

1.1 Motivation

Cardiovascular disease is the number one cause of death globally, taking an estimated 17.9 million lives each year [11]. It is important to make a timely diagnosis so that patients receive early treatment risk assessment. One standard tool used for diagnosis is cardiac imaging; non-invasive imaging of the heart.

In order to obtain images of the heart, clinicians use tools such as magnetic resonance imaging (MRI), computerized tomography (CT) scans, or ultrasound. MRI and CT are less routinely used due to being expensive, having limited availability and a prolonged acquisition time, and using radiation for CT scans. Furthermore, both MRI and CT scans can not be performed if the patient has any metal in their body, such as a pacemaker or metal implants. Ultrasound, on the other hand, is comparatively inexpensive. It is also more flexible; there even exists handheld devices that can be carried by hand and brought on-site. Ultrasound does have a lower imaging quality compared to, for example, MRI [52], and the images can be challenging to interpret due to ultrasound-specific artifacts. Despite this, it is still preferable in many cases because of the reasons mentioned above.

Many heart measurements depend on two key events in the cardiac cycle: end-diastole (ED) and end-systole (ES). Roughly speaking, ED is when the heart is the most relaxed, and ES is when it is the most contracted. Left ventricular ejection fraction is an example of an important measurement that is calculated from ED and ES frames of the cardiac cycle.

A recent study has reported that the average time taken for manually annotating ED and ES frames from visual cues from a video of 1 to 3 heartbeats is 26 seconds, with a standard deviation of ± 11 seconds [42].

Furthermore, because there is not much movement around these frames, the predicted ED and ES frames may differ between different operators. It may even differ for the same operator predicting on the same video at different times. Automating ED/ES frame detection is desirable because it can help reduce annotation time and create a more robust and deterministic result.

Machine learning methods show promising results on several tasks within medical imaging, as is explored in the following chapter. For ED/ES frame detection, the most recent methods utilize supervised deep learning, a family of methods in which a computer program is shown examples of correct predictions and, over time, learns to make the correct predictions itself. Reinforcement learning (RL) is another family of methods that have as of yet not been explored for the problem of ED/ES frame detection. RL is able to outperform humans in complex tasks, such as mastering the board game Go in 2016 [58] or becoming among the 0.2% best players in the world in the video game Starcraft II [64]. However, RL can do more than just play games, and many medical imaging applications also show promising potential [73].

1.2 Goal and Research Question

The goal of this thesis is to explore the use of RL for automatically detecting the ED and ES frames from an ultrasound video. It is interesting from a healthcare perspective because it may open the doors for better automated tools. Yet, it is arguably more interesting from a research perspective because RL is not an obvious choice for this task. RL is built for tasks that require strategic reasoning, but ED/ES frame detection is fundamentally a classification problem.

We pose the following research questions:

- **Is it possible to use reinforcement learning for the task of ED-/ES-frame detection?**
- **How does formulating the problem as reinforcement learning affect the performance of the model?**

Of importance to all types of machine learning is formulating the problem in a way that makes it easier to learn for the computer. That is, optimizing the *inductive bias* by incorporating human knowledge into the algorithm itself. Using RL for ED/ES frame detection may open up possibilities of seeing the problem from a new perspective, allowing us the add the right set of inductive bias.

1.3 Thesis Structure

The thesis is organized into seven chapters. The first chapter, *Introduction*, (which you are currently reading) states the motivation, goal, and research

questions of the thesis. Chapter 2, *Background*, gives an overiew of the concepts and technologies used throughout the thesis, as well as a summarizing previous work on ED-\/ES-frame detection and the use of RL in medical imaging. Chapter 3, *The Dataset*, gives an overview of the data used to train the models, and how it is pre-processed. Chapter 4, *Methodology*, steps through the methods used in the thesis, the decisions taken, and the reasoning behind them. Chapter 5, *Experiments and Results*, reports the results, and the impact of various hyperparameters on the results. Chapter 6, *Discussion*, discusses the results, methodology, and weaknesses of the study. Lastly, chapter 7, *Conclusion*, answers the research questions asked in chapter 1, and gives potential future paths on which to continue research.

Chapter 2

Background

This chapter aims to give an intuition and overview of the concepts and techonologies used throughout the thesis. Section 2.1 describes the function and anatomy of the heart, and its stages through the cardiac cycle. Section 2.2 gives a brief introduction to the physics involved in ultrasound. Section 2.3 builds upon the physics of the previous section and describes how one can use ultrasound to generate images. It also describes two of the imaging modes for echocardiography: b-mode and m-mode. Section 2.4 describes the basics of machine learning and the different families of machine learning. Finally, section 2.5 is a survey of previous work on ED- and ES-frame detection and previous use of reinforcement learning in medical imaging.

2.1 The Cardiac Cycle

The human heart is situated in the middle compartment of the chest, between the lungs, and is responsible for keeping the blood flowing by acting as a pump. Blood is used for transporting oxygen and essential nutrients throughout the body and carries metabolic waste such as carbon dioxide to the lungs.

The heart consists of two halves, the left heart and the right heart, as illustrated in figure 2.1. The left heart pumps newly oxygenated blood from the lungs out to the rest of the body, and the right heart pumps oxygen-depleted blood back to the lungs. Each side has two chambers, the atrium and the ventricle, for a total of four chambers. The upper chambers, the atria, are where the blood first enters the heart, and the lower chambers, the ventricles, are where the blood exits the heart. Each chamber also has valves that are opened and closed during a cardiac cycle to help keep the blood flowing in one direction [34].

The stages of the cardiac cycle is illustrated in figure 2.2. During a cardiac cycle, the different chambers are filled at different times. At the start of a new cycle, the left and right ventricles relax and are filled with blood from their respective atria. As the ventricles are filled with blood, the pressure increases, which causes the valves from the atria to close. After this, the ventricles start contracting, pushing blood out from the heart. This

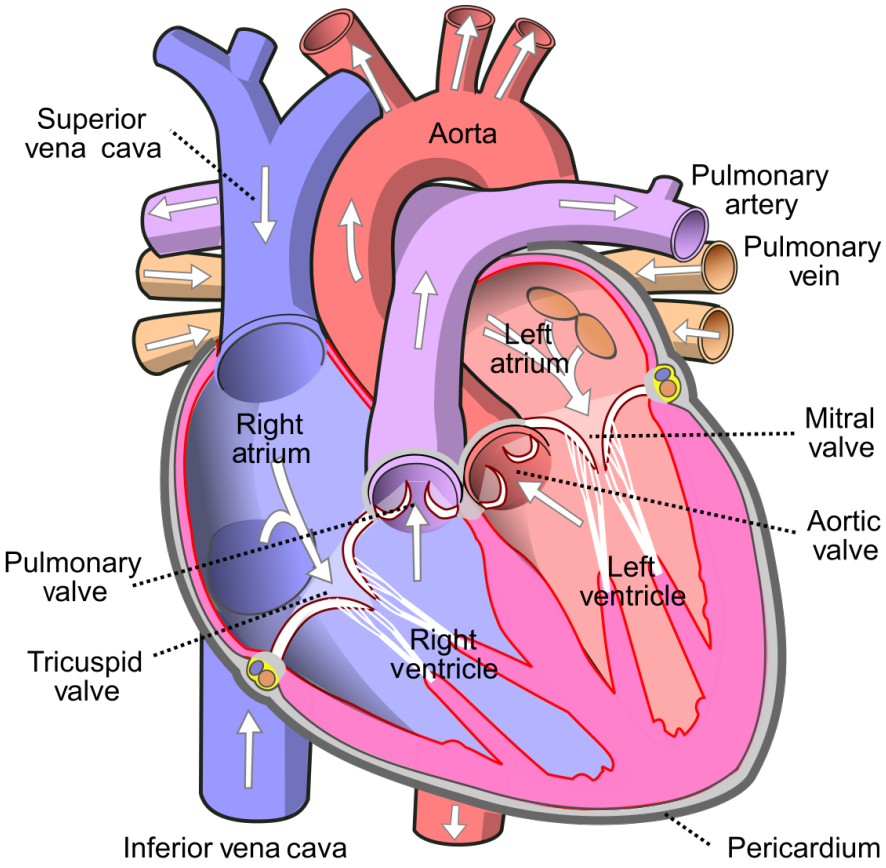

Figure 2.1: An illustration of the heart. The heart has two sides, each side having two chambers. Image reproduced from [4], License: CC BY-SA 3.0, User: Eric Pierce (Wapcaplet).

causes the ventricle pressure to decrease and the aorta pressure to increase, and the valve going out of the ventricle is closed. Finally, blood flows into the atria before the cycle starts over.

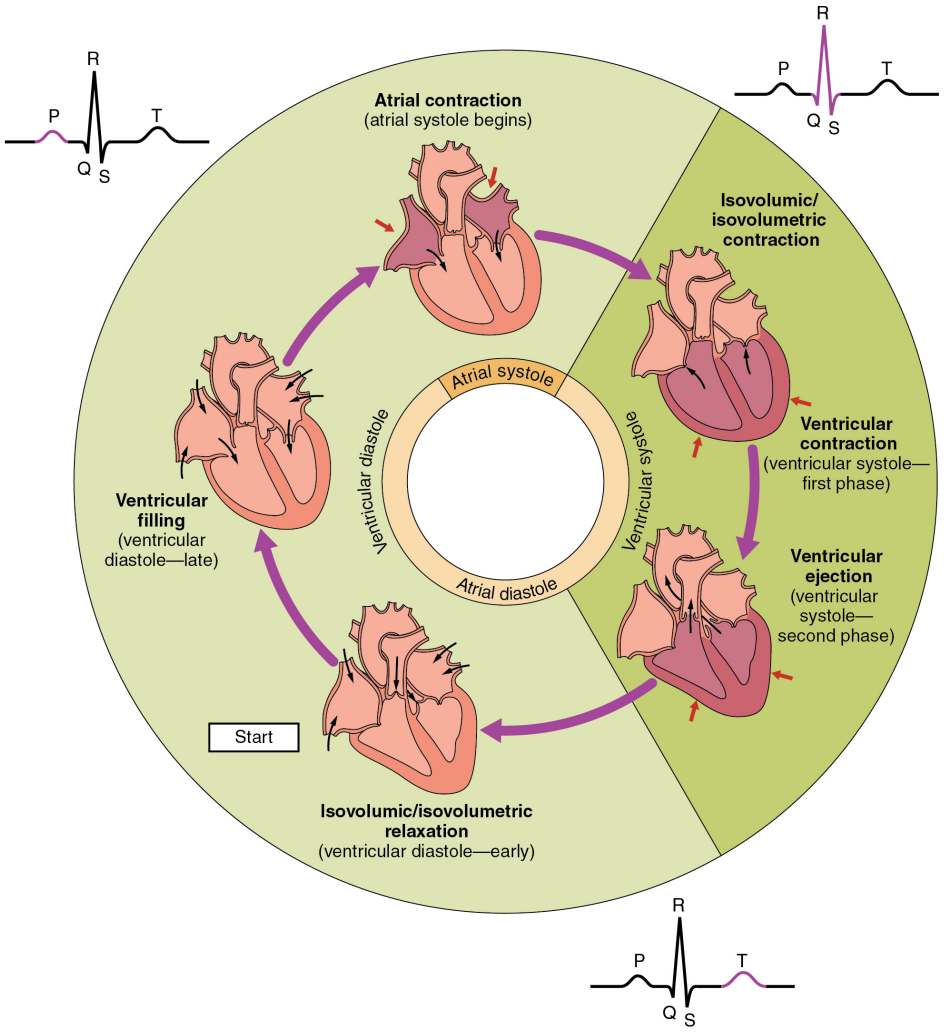

Figure 2.2: The cardiac cycle is illustrated with the direction of blood flow and pressure from and into the atria and ventricles. Image reproduced from [26], License: CC BY 3.0, User: OpenStax College.

There are multiple ways of finding the ED and ES frames in a cardiac cycle [45]:

1. Finding the frame with the maximum left ventricle volume (for ED) and the frame with the minimum left ventricle volume (for ES).
2. Finding the first frame following the closure of the mitral valve (for ED) and the first frame following the closure of the aortic valve (for ES).
3. Analyzing a simultaneously acquired electrocardiogram (ECG) signal.

These methods can be visualized in the Wiggers diagram [49], as seen in figure 2.3, which plots several key events in the cardiac cycle and the corresponding values of various measurements.

Out of these three, using the ECG signal is the least preferable. This is because the methods for detecting the ED and ES frame may become unreliable when given an unconventional ECG signal, such as from patients with cardiomyopathy or regional wall motion abnormalities [45]. Acquiring an ECG signal also requires applying electrodes to the patient, which is not ideal in emergency settings.

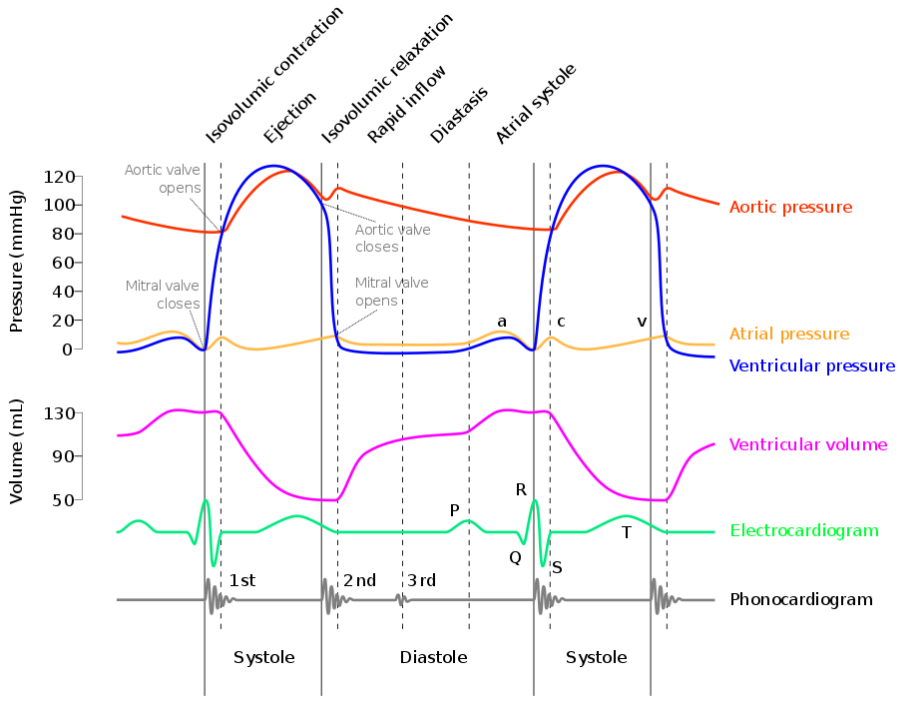

Figure 2.3: The Wiggers diagram describes the different phases of the cardiac cycle and what they represent in different measurements. Image reproduced from [67], License: CC BY-SA 4.0, User: adh30 revised work by DanielChangMD who revised original work of DestinyQx; Redrawn as SVG by xavax.

2.2 What is Ultrasound?

In physics, sound can be defined as a phenomenon where energy propagates through a medium — such as gases, liquids, or solids — by the mean of mechanical waves. In the special case of ultrasound, the waves we refer to are longitudinal pressure waves that are, by definition, slightly above the hearable range of humans (above 20 kHz) [61].

Sound waves push particles together, creating an increase in pressure. Particles in an area of high pressure move to areas of lower pressure, which

creates a chain reaction where a pressure field moves through particles. This is called wave propagation and is informally illustrated in figure 2.4. Sound is simply waves of pressure propagating through a medium.

Today, ultrasound form the basis of several advanced technology such as medical imaging probes, sonar, non destructive testing, and more. This thesis only covers echocardiography, a technology for imaging the heart using ultrasound waves.

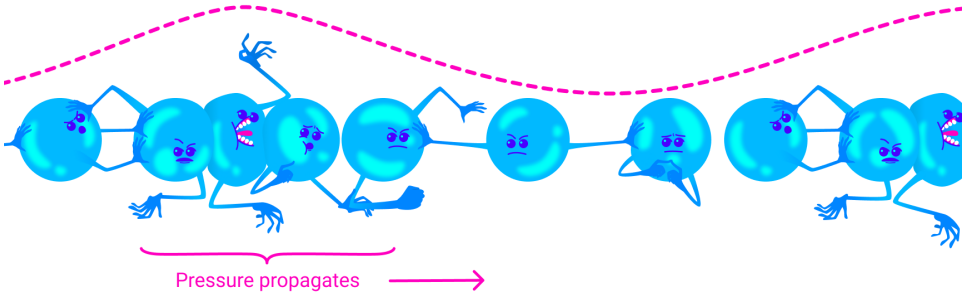

Figure 2.4: A pressure wave moves through a medium by pushing particles in a medium close together. The particles push back as the pressure increases, moving the pressure field. Warning: This image is just a representation of how particles interact — real particles do not look like this.

2.2.1 Attributes of a Sine Wave

A basic wave has three attributes: frequency, how fast it vibrates, amplitude, by how much it vibrates, and phase, where in its cycle a wave is at a given time [48], as visualized in figure 2.5. Our bodies have evolved to sense these properties, where frequency determines the pitch of a sound and amplitude determines the loudness. Sensing phase is a bit more subtle but aid us e.g. in determining the position of the source, relative to us. The relative phase between multiple sounds also affect the resulting sound, as they interfere with each other differently depending on the relative phase.

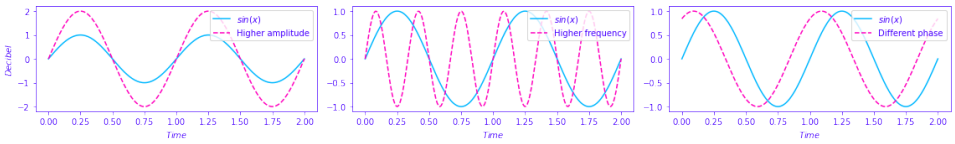

Figure 2.5: The left-most plot shows two basic waves where one has twice the amplitude. The middle plot shows two basic waves where one has a higher frequency. The right-most plot shows two basic waves that have different phases.

A basic wave means a sine wave in this context. Every sound can be represented as a sum of sine waves, and every sound can be transformed into its frequency spectrum through the use of the Fourier transformation [48]. As seen in figure 2.6, the frequency spectrum of a sine wave is just

a single spike. Because of the linear property of the Fourier transform, adding together two sounds has the same effect as adding their frequency spectrums.

Real-world sounds are often more complex than the narrow band sound presented previously. In the nature many acoustic phenomenons can be described by a broadband spectrum, which is a weighted sum of many sine waves. When we hear a piano and a clarinet play the same note, the frequencies with the highest amplitudes are generally the same for both sounds, but the frequency spectrum is much more complex. Musicians speak of overtones — it is the overtones that are different for different instruments playing the same notes. They are referring to the additional frequencies that can be seen in the frequency spectrum.

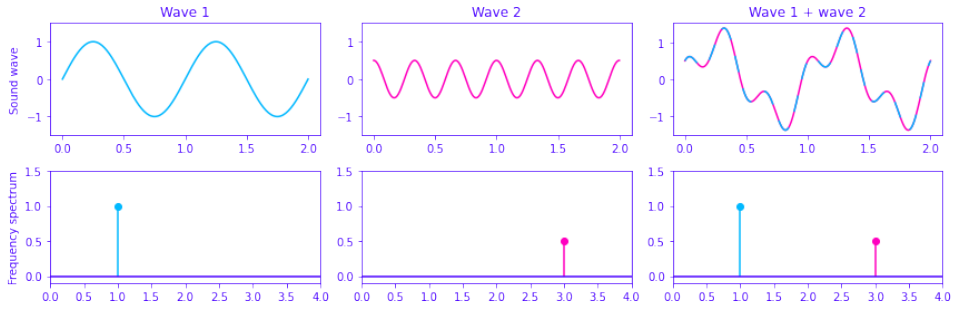

Figure 2.6: Adding two sounds together also adds their frequency spectrums together.

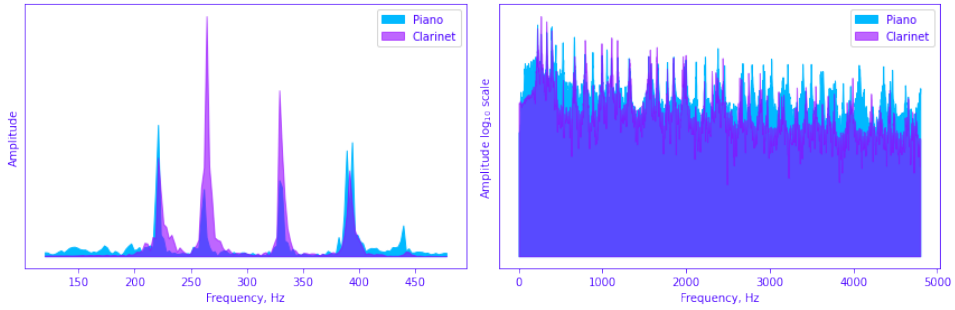

Figure 2.7: The overtones make two instruments sound different, even when playing the same notes. Left: frequency spectrum of a piano and a clarinet from 150 to 450 hertz. Right: the same frequency spectrum from 0 to 5000 hertz, in $\log_{10}$ scale. Both instruments are playing the Am7 chord, which consists of four notes. These four notes can be seen clearly in the left image, all having relatively high amplitudes for both instruments.

2.2.2 Attributes of the Medium

Another important aspect of sound is the medium through which it travels. Properties such as the speed of sound, density, attenuation, and

nonlinearity affect how a sound wave propagates through its medium [36]. Speed of sound is how fast a wave propagates through the medium. Assuming that the frequency stays the same throughout (which is not always true), the wavelength will be smaller if the sound speed is lower, as visualized in figure 2.8. Density is how tightly packed the particles are in the medium when at rest. Acoustic absorption is an energy loss caused by the viscosity of the propagating medium. The wave energy is then converted into heat at a molecular level. Attenutation is the reduction of the energy signal caused by either absorption or scattering. Nonlinearity is the property where the speed of sound at a point depends on the pressure at that point. In water, pressure waves propagate faster at higher pressure. The increased pressure may be caused by the wave itself, in which case the shape of the wave may change, as visualized in figure 2.9.

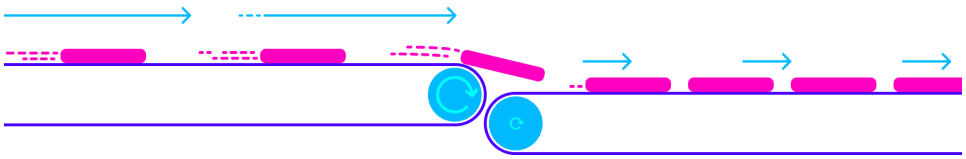

Figure 2.8: Even though the rate of packages per second stays the same, the distance between packages decreases when arriving on a slower conveyor belt. This is analogous to a sound wave propagating through a medium where the speed of sound changes. Even though the frequency is the same, the wavelength (the length between each top) decreases when it encounters a lower speed of sound.

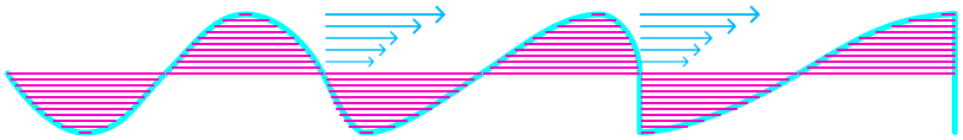

Figure 2.9: In a medium with nonlinearity, higher-pressure parts of a wave propagate faster than lower-pressure parts. Over time, the higher-pressure parts will "catch up" to the lower-pressure parts, and what started as a sine wave will start to resemble a sawtooth wave.

An important concept is "acoustic impedance," which measures how much resistance the wave encounters while propagating through the medium [61]. Acoustic impedance is a function of the speed of sound and density. When a wave propagates out of one medium and into another medium with a different acoustic impedance, a fraction of the energy is reflected. So when one hears a sound being reflected from a wall, it is because the air that the wave travels through and the wall has different acoustic impedance. Equation 2.1 shows the relationship between acoustic impedance, density, and speed of sound, where Z is the acoustic impedance, and ρ and c are the density and speed of sound of the medium, respectively. Equation 2.2 is the reflection factor. It determines how much of the energy is reflected, where Z_1 is the acoustic impedance of the original

medium, and Z_2 is the acoustic impedance of the second medium. When Z_1 and Z_2 are equal, no sound is reflected.

$$Z = \rho c \tag{2.1}$$

$$RF = \frac{Z_2 - Z_1}{Z_2 + Z_1} \tag{2.2}$$

2.3 Echocardiography

Light is a signal that does not penetrate very far into the body, which is why we cannot simply gaze into each other's hearts. We could, however, imagine a universe where light penetrates all the way, giving off no reflections at all. In this universe, we would not be able to see the heart either; in fact, we would not be able to see any body at all! To be able to look *inside* something based on reflections alone requires a sweet spot where the signal can penetrate tissue with enough energy while at the same time being reflected with enough energy so that we can measure it. Arguably, we are quite lucky with our universe, at least in terms of cardiac imaging, because sound is such a signal.

Table 2.1: Values of the acoustic wave velocity c and acoustic impedance Z of some substances from [59].

Substance	c (m/s)	Z=ρ c ($10^{6\text{kg}}/\text{m}^{2\text{s}}$)
Air (25°)	346	0.000410
Fat	1450	1.38
Water (25°)	1493	1.48
Soft tissue	1530	1.63
Liver	1550	1.64
Blood (37°)	1570	1.67
Bone	4000	3.8 to 7.4
Aluminium	6320	17.0

Table 2.1 lists the speed of sound and acoustic impedance Z of some substances. Notice how there is a large contrast in acoustic impedance between air and soft tissue. If there is air between the sound wave transmitter and the body, most of the energy will be reflected by the skin. To reduce this effect, ultrasound gel, which has a similar acoustic impedance to soft tissue, is applied between the body and the sound wave transmitter. Notice also the difference in acoustic impedance between bone and soft tissue. This has consequences for what we can image in the body, as bones such as the ribcage act as shields to the sound waves.

How can we use sound reflections to create images? We can send out a sound signal and measure the time it takes for a reflection to come back. The delay between sending and receiving gives information about the relative distance to various reflectors in the medium from the sound source, as visualized in figure 2.10. Suppose we know the speed of sound,

and assume that the speed of sound is homogeneous in the medium. In that case, we can approximate the distance that the wave has traveled by multiplying the delay between sending and receiving by the speed of sound (equation 2.3). This assumes that waves always travel in straight lines, which is not always true, but the effect is often negligible in medical ultrasound use cases.

$$\text{distance} = \text{delay} \times c \tag{2.3}$$

Likewise, suppose we want to know the reflected signal for a given distance away from the transmitter and receiver. In that case, we can calculate the corresponding delay of a signal traveling that distance and back by dividing the total distance by the speed of sound (equation 2.4). When we know the corresponding delay, we can simply look up its value in the signal through interpolation. To create a whole image, we repeat this process for every point in the image.

$$\text{delay} = \frac{\text{distance}}{c} \tag{2.4}$$

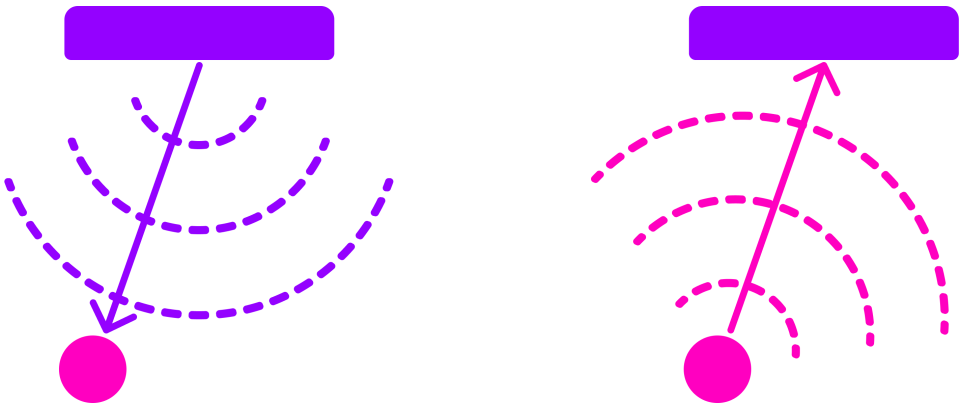

Figure 2.10: By measuring the time between sending a signal and receiving it back from a reflector, we can approximate how far away the reflector is — given that we know the approximate speed of sound.

When we only have a single receiver that measures the reflected sound waves, we can not know the exact location of a given reflector, only the distance. By utilizing more receivers spread over some area, we get more information about where the signal originated from, as there will be a correlation between signals across receivers at the reflecting object.

By utilizing multiple sender elements that can send sound waves independently of each other, we can shape the wavefront as we wish. For example, this lets us focus the energy of the sound wave in a specific area or shape the wavefront to be planar. The Huygens-Fresnel principle states that every point of a wavefront is the source of a new spherical wavefront. We can simulate the Huygens-Fresnel [36] principle by imagining a desired wavefront passing through the sender elements, activating each element when the wave hits it. Each sender element on its own creates a spherical

wavefront, but together they make up the desired imagined wavefront. Time delays are to sound waves like a lens is to a magnifying glass [61]. An example of this has been visualized in figure 2.11.

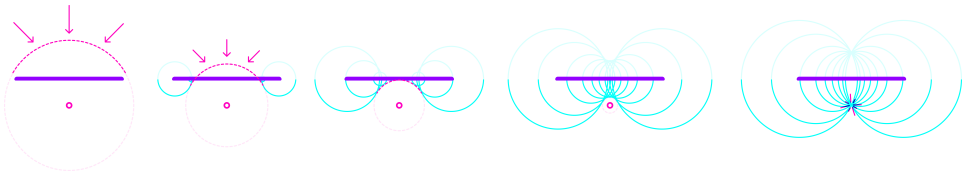

Figure 2.11: Because of the Huygens-Fresnel principle, we can create a desired wavefront by creating spherical waves at each sender element when the imagined wavefront hits it. The dashed, pink curve represents the imagined desired wavefront as it approaches the sender elements marked by the purple rectangle. Each sender element is activated when the imagined wavefront passes through it, creating new spherical waves, represented by the cyan semi-circles. The generated spherical waves converge on the same point as the imagined wavefront.

In reality, an ultrasound probe consists of many elements acting both as transmitters and receivers. The elements are made out of piezoelectric material. Piezoelectric materials produce vibrations when given an electric current and, vice-versa, produce an electric current when exposed to vibrations. With a transducer, we can independently apply an electric current to each element to create sound waves with given wavefront characteristics and read off the electric current generated by reflected pressure waves [61].

There are multiple modes of ultrasound imaging. The two most important modes for this thesis are B-mode imaging and M-mode imaging [61].

In B-mode (as in "Brightness"-mode) imaging, an image is created by visualizing the amplitude of the reflected signal as the brightness for a given point. This imaging mode often sends out individual, focused transmits in multiple directions, creating a sector scan — a fan-like image, as seen in figure 2.12. Another method is to transmit unfocused plane waves. A single transmit creates an unfocused image of the scatterers in the medium. However, multiple transmits in different directions may be compounded to create an image of comparable quality to those of focused transmits [51].

B-mode imaging provides images of the whole area of interest, but because they require multiple transmits, they also take longer to acquire, as we have to fire each transmit after the other. In extreme cases, this could pose a problem, given that the heart is an organ that moves quite rapidly. If we are transmitting too slow, then the heart may have a noticeably different phase on one side of the sector scan compared to the other. This is not a significant problem for 2D images as even multiple transmits can be made and received back in a short period of time, but it does have consequences for the temporal resolution.

In M-mode (as in "Motion"-mode) imaging, only one direction is

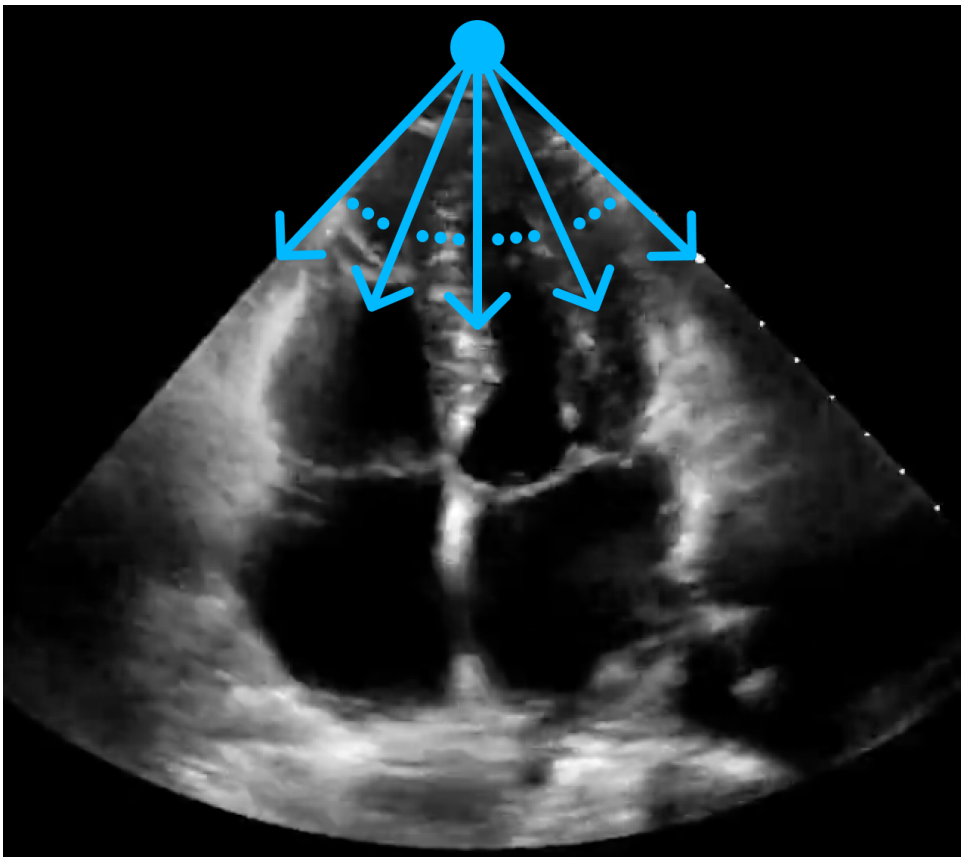

Figure 2.12: Imaging along different angles from a common starting point creates a sector scan.

imaged over time instead of a whole sector. This means that it only requires one transmit per frame, giving it a higher temporal resolution compared to B-mode imaging, but at the cost of only focusing in a single direction. Each transmit can be concatenated into an image where the y-axis represents the amplitudes at different depths, and the x-axis represents time, as seen in figure 2.13. M-mode imaging lets us see the motion of a focused part of the heart in a single image.

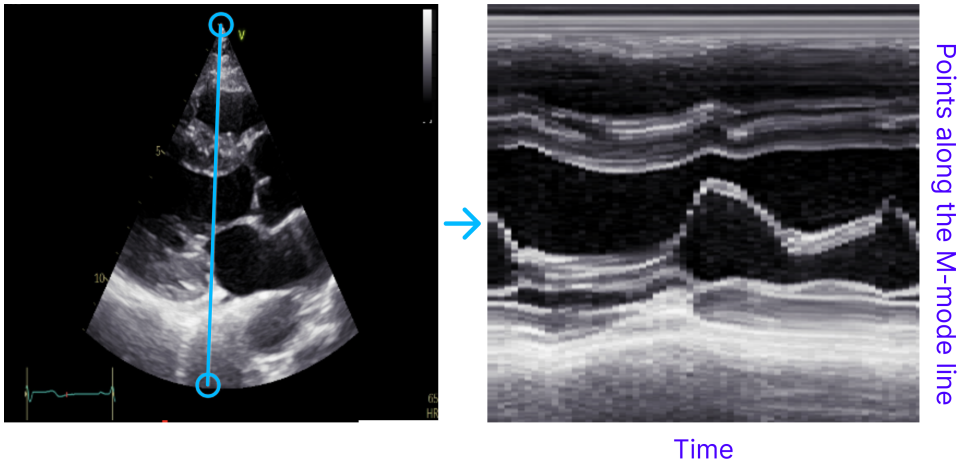

Figure 2.13: Left: a still of a sector scan. Right: the corresponding M-mode image of the video for the indicated blue line.

2.4 Deep Learning

2.4.1 Gradient Descent

The most significant deep learning innovations of the last decade have used a gradient descent technique. Gradient descent takes advantage of the fact that even if we do not know the true nature of some function, if it is differentiable, then we can calculate its slope at a given point. This gives us information about how to update its parameters to maximize or minimize the result. This is easily visualized when we have a differentiable function that takes a single parameter x, as seen in figure 2.14. Even though we may now know the true shape of the function, as represented by the dashed line, we can calculate its slope. If we nudge x in the opposite direction of the slope, i.e., reduce x if the slope tends upwards and vice-versa, and repeat this multiple times, we will eventually reach a minimum where the slope becomes 0. This iterative process of calculating the gradient at a point and updating the parameters a small step in the opposite direction is what is called gradient descent [23].

Gradient descent scales to an arbitrary number of parameters, allowing us to optimize big models. One example could be a model that performs some operation on an image. Suppose we want to process each pixel individually in some parameterizable way at least once. In that case, the

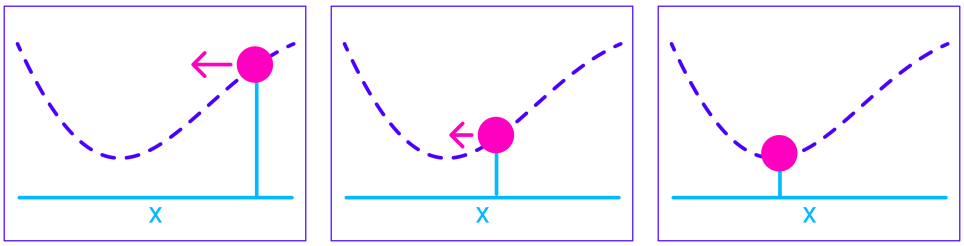

Figure 2.14: Visualization of gradient descent of a function that takes a single parameter x. Nudging x in the opposite direction of the gradient at the current point minimizes the result of the function.

number of parameters is at least equal to the number of pixels in the image. If the image is 100-by-100 pixels big, the model will take at least 10 000 parameters. It is no longer possible to visualize this high-dimensional parameter space as we did in figure 2.14, but the principles still hold, and gradient descent still works the same way.

The function that we optimize using gradient descent consists of two parts: a model and a loss function. The job of the model is to perform the task at hand, and the job of the loss function is to quantify the error of the model so that we can minimize it. As long as the model and the loss function are differentiable, we can optimize it using gradient descent. Not all models and not all loss functions are equally good, however. Some models may better represent the problem at hand than others, and some loss functions may produce gradients that are easier to optimize for than others. One important aspect is the shape of the gradient, whether it contains a lot of local minima and how steep it is at regions [71].

We may want to optimize some parameters working on a set of images, for example, when training a model to classify pictures as those of cats or dogs. These images define a distribution that we want the model to be able to represent. Because of either memory or computational constraints, there may be too many pictures in the dataset for the model to try to optimize for at once. It is common to apply gradient descent on just a subset of the whole dataset at once, chosen randomly at each iteration. This is called stochastic gradient descent (SGD), and, perhaps surprisingly, it is often better at generalizing on the dataset than using gradient descent on the whole dataset at once [38].

Deep learning algorithms risk simply memorizing the training data if it has the capacity to do so. Therefore, the performance of a model on the data it has been trained on is not representative of how it would perform on unseen data. It is common to split the data into three splits: training (train), validation (val), and testing (test), where the val and test split are used for performance evaluation only. When the model performs better on the training split at the cost of performance on the others, the model is said to be overfitting. The validation data can be used to monitor the amount of overfitting over time such that we can select the model and hyperparameters that generalize best on the unseen data. This runs the risk of introducing bias towards the validation split, which is why we have

the third split, test, used to report the performance of the final model [63].

Another aspect of great importance is to instill inductive bias into the model; that is, implicit knowledge about the task at hand. Some models capture implicit knowledge about the problem at hand better than others, and some relevant models are explored in the following section.

2.4.2 Deep Neural Networks

Neural networks are data-processing models inspired by a simple view of how our brain's neurons interoperate. On a high level, neural networks are often abstracted into a set of layers. When there are multiple such layers in a neural network, it is often called a "deep" neural network. This section presents some of the neural network layers referenced throughout the thesis.

Fully connected layers process the inputs linearly, i.e., each input x_i is multiplied by some weight w_{ij} and added with some bias b_{ij}, and produce a given number of outputs y_j. This may be written as matrix multiplication, such that $\hat{y} = wx + b$. The shape of the matrix w determines the number of output neurons it produces.

Multiple fully connected layers can be stacked to create a more complex network. However, because they are linear operations, they can only represent linear relationships no matter how many of them are stacked. To allow the network to represent more complex, non-linear relationships, we need to add non-linearity to the network. This is usually done using *activation functions*. Examples of activation functions are the sigmoid function, seen in equation 2.5, or ReLU, seen in 2.6.

$$S(X) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

$$ReLU = max(0, x) \tag{2.6}$$

Using just two fully connected layers separated by non-linear activation functions, one could represent any arbitrary function, given that one includes enough neurons [32]. That does mean that they are the right tool for every job.

Fully connected layers combine every input with every output. This does, for example, not take advantage of the spatial locality of images. A given pixel is often more related to pixels that lie closer to it in an image. *Convolutional layers* take advantage of this by applying filters to an image, with each filter only processing a small part of the image at a time. The filters are often small matrices that are applied everywhere in an image.

Convolutional layers consist of a set of filters with a width and height. Other important hyperparameters are the stride and dilation, which (among other hyperparameters) affect how the filters are applied to the image. Stride affects the distance in pixels between subsequent applications of the filters [71]. Dilation affects the spacing between the parameters in each filter [46]. Stride and dilation is best explained through visualization, and are illustrated in figures 2.15 and 2.16, respectively.

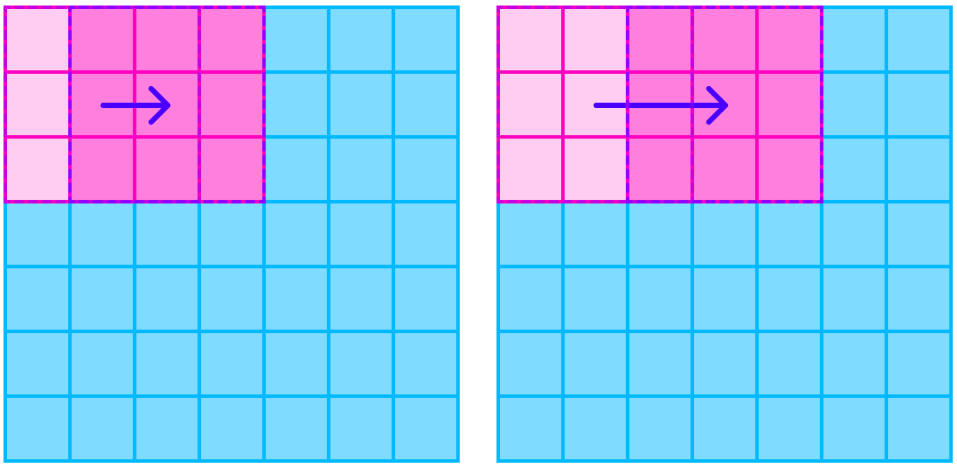

Figure 2.15: Stride affects the distance between subsequent applications of a filter, visualized here in pink. Left: A stride of 1 moves the filter one pixel at each application. Right: A stride of 2 moves the filter by two pixels at each application.

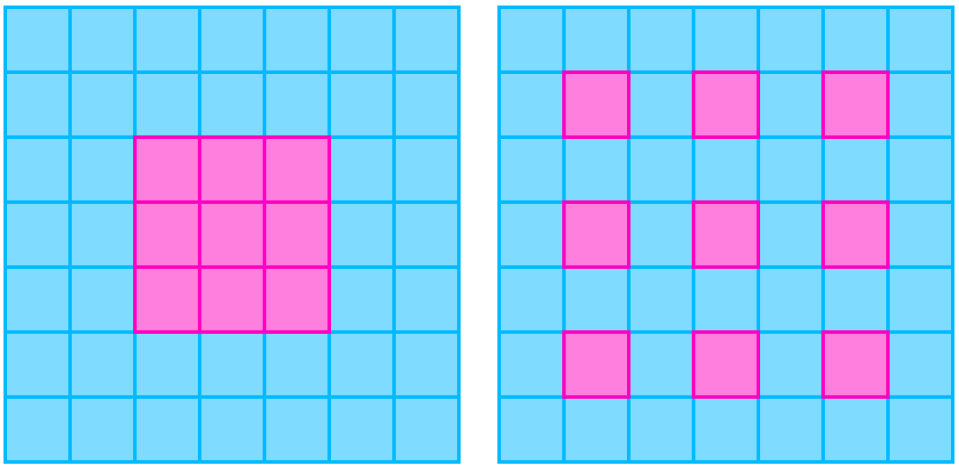

Figure 2.16: Dilation affects the spacing between the parameters in each filter. Left: A dilation of 1 means that each parameter is spaced apart by 1 pixel. Right: A dilation of 2 means that each parameter is spaced apart by 2 pixels.

Another popular layer is the *recurrent layer*. The recurrent layer is designed to be efficient at processing sequential data [71]. Recurrent layers are not part of the methodology of this thesis but have been used in related work. In short, like convolutional layers apply the same filter at every position in an image, recurrent layers apply the same computation at every item in a sequence. Furthermore, it can capture temporal information by also outputting an internal state that is included in the computation for the next item in the sequence. Figure 2.17 illustrates this computation. Some popular implementations of a recurrent layer are LSTM [30] and GRU [13].

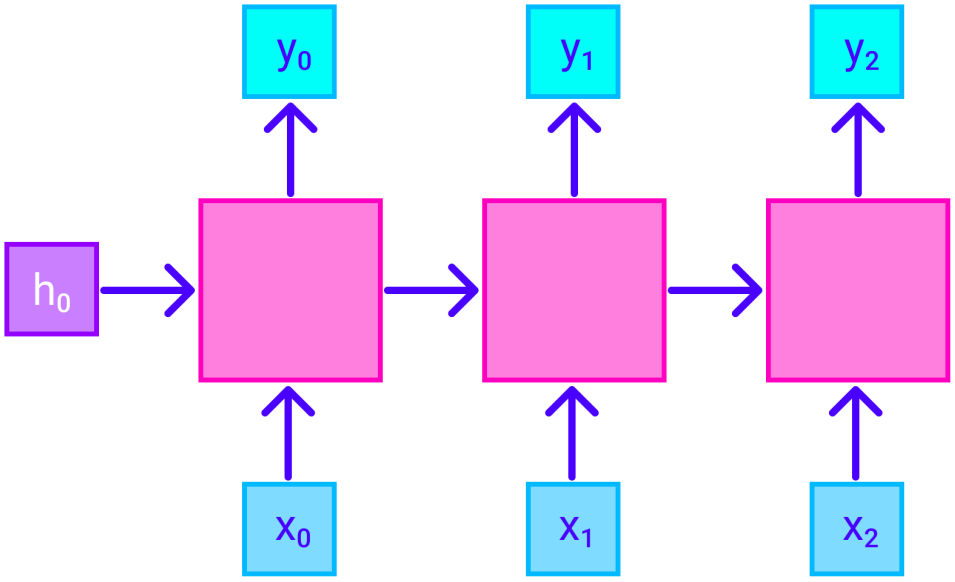

Figure 2.17: A visualization of a basic recurrent layer. Each pink square represents the same computation that takes an input item, x, and a hidden state h and outputs y.

Like the fully-connected layers, both convolutional and recurrent layers by itself are linear operations and we need to add non-linearity for them to be able to approximate arbitrary functions.

2.4.3 Optimization Process

For each iteration of gradient descent, we update the model's parameters a small step in the opposite direction to minimize it. It is crucial only to update them in a small step each time; otherwise, they may overshoot and, in the worst case, cause the model's performance to diverge. For standard SGD, we choose how much to update the parameters using the hyperparameter α, often a low number between 0 and 1.

Using a low α means that we do not update the parameters too much when the gradient is steep, but it also means that the parameters are updated little when the gradient is not very steep. In addition, we may encounter flat regions of the gradient, which can be hard to move past regardless of the chosen value of α. For these reasons, more advanced

optimizers have been developed, such as ADAM [39].

Machine learning aims to train models that generalize to data samples outside of the training set. When we optimize a model on given data distribution, we risk making the model specialize too much on that specific distribution, earlier defined as overfitting. *Regularizers* are tools that attempt to reduce the chance of overfitting [71].

2.4.4 Supervised and Unsupervised Learning

One way of designing the loss function is to define it as the difference between the predicted values from the model and ground truths labeled beforehand. Learning methods that use these kinds of loss functions are generally called supervised learning, as if a "supervisor" tells the model what the correct answer ought to have been.

When we do not have access to the data's ground truths, we must define the loss in other ways. This is called unsupervised learning. An example of unsupervised learning is manifold learning. Manifold learning tries to ensure that similar points in the high-dimensional space are projected close together in the low-dimensional space [72].

2.4.5 Reinforcement Learning

RL allows an agent to learn a strategy, called a *policy*, that maximizes the total reward received through interacting with an environment. RL can leverage time in a way that neither supervised nor unsupervised learning is able to because it takes future decisions into account when deciding on the next action. An RL agent can make a decision now that has no immediate benefit but will lead to a better result in the future.

At the core of RL are markov decision processes (MDP) [60], which can be described using four elements:

- The state space S
- The action space A
- The transition function $P(s_{t+1}|s_t, a_t)$
- The reward function $R(s_t, a_t)$

An RL agent is faced with a sequence of decisions. At each step, it is presented with the current state $s_t \in S$ of the environment and must take an action $a_t \in A$. In an episodic task, the agent's goal is to maximize the total reward r it receives during its lifetime, called an episode. The environment may change after the agent takes an action in a given state, and how it changes, i.e., what the next state s_{t+1} will be, is determined by the transition function $P(s_{t+1}|s_t, a_t)$. How much reward the agent receives after taking an action in a given state is determined by the reward function $R(s_t, a_t)$. The goal of RL is to find a policy π, a strategy that, if followed, will yield the most amount of total reward during the lifetime of the agent.

In practice, the policy is simply a function that takes in the current state s_t and returns the probability of taking an action a_t: $\pi(a|s) \in [0, 1]$.

The agent's goal is not to maximize the immediate reward r but rather the expected return. The return is denoted as G_t and is in its simplest form a sum of all the future rewards, as seen in equation 2.7. T marks the timestep where the episode ends.

$$G_t = r_{t+1} + r_{t+1} + r_{t+2} + \ldots + r_T \tag{2.7}$$

However, some tasks are not episodic, which means that they may run forever. The returns G becomes infinite for environments with limitless rewards, making the optimization problem intractable. To solve this problem we include *discounting* to the returns, as seen in equation 2.8. γ is the *discount rate* and is a number in the range $[0, 1]$. If $\gamma < 1$, then future rewards count for less in the full returns, and as the number of steps into the future approaches infinity, the corresponding rewards approach 0. Discounting guarantees that non-episodic tasks converge to optimal solutions while also giving a mechanism for preferring more immediate rewards compared to future rewards.

$$\begin{aligned} G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \ldots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \\ &= r_t + \gamma G_{t+1} \end{aligned} \tag{2.8}$$

One way to select an action is to predict the following state's value after taking that action. For this we could use the *state value function* $V_\pi(S_t)$ which estimates the expected return G_t of being in state s_t, while following the policy π. Alternatively, we could use the *state-action value function* $Q_\pi(s_t, a_t)$ which estimates the expected return of taking action a_t in state s_t, while following the policy π. Both value functions depend on the policy being followed because the policy decides what actions to take in the future, which again has consequences for what rewards the agent expects to receive at subsequent steps. For this setup, the "learning" part of RL could be considered to be updating a value function towards the "optimal value function," defined as the value function that uses the optimal policy when estimating returns. The optimal policy π^* is one *(of the possibly many policies)* that yields the maximum amount of total reward if followed.

Another important aspect of RL is the exploration-exploitation trade-off: How often should the agent explore the environment versus exploit its current assumptions about the environment? An agent that never performs exploration, i.e., always acts greedily, may never discover the optimal policy. An agent that only ever explores, i.e., only takes random actions, may end up revisiting the same low-potential states over and over. There is a balance to be made. One way to induce exploration for an agent is to force it take a random action a given percentage of the time. This is called an ϵ -greedy policy, where ϵ is the hyperparameter that decides how often the agent takes a random action instead of what it considers to be the best

one. E.g., a value of $\epsilon = 0.1$ means that the agent takes a random action 10% of the time.

One algorithm for updating the state value function is called temporal difference learning (TD). In TD, the state value function $V(s_t)$ is updated after every step, by comparing the value it expected to see, with a value that takes the newly observed reward r_{t+1} into consideration, as seen in equation 2.9. $(r_{t+1} + \gamma V(s_{t+1}))$ is called the TD-target, and because it incorporates the actual observed reward r_{t+1}, it can be considered as a more up-to-date version of the state value function. $(r_{t+1} + \gamma V(s_{t+1})) - V(s_t)$ is called the TD-error. The lower the TD-error is, the better the RL agent is able to reason the value of states, and as such, we want to minimize it. We do this by updating the state value by nudging it slightly towards the TD-target. How far it is nudged at each update is determined by α.

$$V(s_t) \leftarrow V(s_t) + \alpha[(r_{t+1} + \gamma V(s_{t+1})) - V(s_t)] \tag{2.9}$$

To be able to use $V(s)$ for making a decision, the agent needs knowledge about the transition function $P(s_{t+1}|s_t, a_t)$. This is because it needs to know what the next state will be to select the best action to take. $Q(s, a)$ does not need knowledge about the transition function because it directly learns the value of taking an action for a given state. TD can be modified to use the state-action value function instead of the state value function, in which case it is called Q-learning. In equation 2.10, the target (Q-target), is defined as the immediate reward of taking action a_t, plus the discounted value of taking the best action in the following state.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[(r_{t+1} + \gamma max_a Q(s_{t+1}, a)) - Q(s_t, a_t)] \tag{2.10}$$

In TD-learning, the agent must associate each state with its corresponding value as it explores the environment. The same is true for Q-learning, but it also has to take state-action pairs into account, meaning that it has to store up to $\|S\| \times \|A\|$ entries. That is fine when the state- and action-space are small but becomes infeasible when they are too big.

The described way of storing and updating the values is called tabular methods because we treat the states, or state-action pairs, as entries in a table. Tabular methods break down when the state space or the action space becomes very large or even continuous. Creating RL algorithms that can handle very large or continuous action spaces is challenging [73]. However, methods exist that can scale RL to handle very large or continuous state spaces.

Deep Reinforcement Learning

A modified Q-learning algorithm has been shown to be able to play Atari games simply by looking at the raw pixel values [50]. The state-space thus consists of the pixel values of the current game screen. A simple Atari game has $210 \times 160 = 33600$ pixels, and each pixel can be one of 128 colors [50]. In theory there are $128^{33600} \approx 10^{70803}$ different states. If a computer were

able to process 1 000 000 000 such states every second, it would still take more than 10^{70785} years to process all of them.

We assume that there exists a way to approximate the value of states in a much more compressed way. This can be done through function approximation [60], where instead of storing and updating the value estimates in a table, such as with tabular methods, they are approximated using a neural network. This may also allow the agent to generalize state value or state-action value functions to new not-before-seen states.

Much of today's research into RL goes into scaling it up to larger state-spaces. Methods that scale RL by modifying the Q-learning algorithm are called "action-value methods," but they are not the only ones to do so. Policy gradient is another popular set of methods that can learn a parameterized policy directly, without consulting a value function [60]. Policy gradient methods may more naturally model continuous action spaces as it outputs a distribution of action probabilities instead of the values of a discrete set of actions. As seen in later chapters, the RL formulations used in this thesis all use discrete action spaces, and only action-value methods are considered for this thesis.

Deep Q-Network

The modified Q-learning algorithm was termed deep q-network [50] (DQN) for its ability to take advantage of recent deep learning advances and deep neural networks.

The original DQN algorithm takes the raw pixel values from an Atari game as input, followed by three convolutional layers and two fully connected layers. The final fully connected layer outputs one value for each possible action, approximating the expected value of taking each action given the state, i.e., $Q(s, a)$. An ϵ -greedy policy then chooses either the action with the highest approximated value with probability $1 - \epsilon$ or a random action with probability ϵ.

The authors showed how the network is able to reduce the state space by applying a technique called "t-SNE" to the DQNs' internal state representation. t-SNE is an unsupervised learning algorithm that maps high-dimensional data to points in a 2D or 3D map [44]. As expected, the t-SNE algorithm tends to map the DQN representation of perceptually similar states to nearby points. Interestingly, it also maps representations that are perceptually dissimilar, yet are close in terms of expected rewards, to nearby points. This indicates that the network is able to learn a higher-level, but lower-dimensional, representation of the states in terms of expected reward. This is visualized in figure 2.18.

Using function approximation does have its problems. Naively training the network by inputting state and returns pairs as the agent generates them can make the algorithm unstable. There is a strong correlation between consecutive samples, and if a neural network receives a batch of very similar input, it might overwrite previously learned knowledge. Furthermore, an update that increases $Q(s, a)$ often also increases $Q(s + 1, a)$ and therefore also increases the target value, possibly leading to

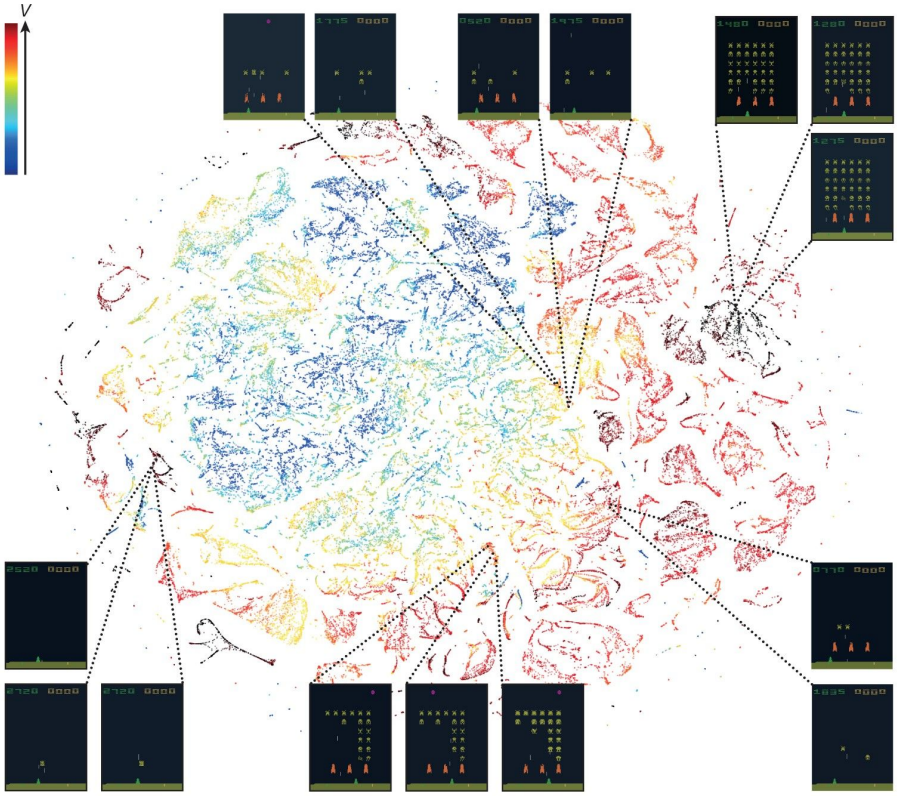

Figure 2.18: A figure from [50] that shows a two-dimensional t-SNE embedding of the representations in the last hidden layer assigned by DQN to game states experienced while playing Space Invaders. The points are colored according to the state values predicted by DQN for the corresponding game states. The states rendered in the top right, which are of almost full of enemy ships, and the states rendered in the bottom left, which are nearly empty, have similar predicted state values even though they are visually dissimilar, because the agent has learned that completing a screen leads to a new screen full of enemy ships.

oscillations or divergence of the policy. These problems are mitigated by using experience replay and by using a separate network to generate the targets in the Q-learning update.

In experience replay, the agent's experiences over multiple episodes are stored in a data set called the replay memory. Each experience item is a tuple consisting of the previous state, selected action, returned reward, and new state: (s_t, a_t, r_t, s_{t+1}). During training, randomly sampled batches from the replay memory are used to train the Q-network.

Using a separate network for generating the targets in the Q-learning update adds a delay between the time an update to Q is made and the time it affects the targets, making the algorithm more stable and reducing the chance of oscillations or divergence.

Double Deep Q-Network

Several improvements have been made to DQN over the years. Q-learning has been shown to produce overly optimistic action values as a result of using the maximum action value as an approximation for the maximum expected action value [24]. Double Q-learning attempts to reduce this overestimation by decomposing the target into an action selector and an action value estimator. The regular Q-learning target is written as:

$$r_{t+1} + \gamma max_a Q(s_{t+1}, a)$$

This can be rewritten as:

$$r_{t+1} + \gamma Q^A(s_{t+1}, argmax_a Q^B(s_{t+1}, a)) \tag{2.11}$$

Where Q^A acts as an action value estimator and Q^B acts as an action selector. If $Q^A = Q^B$, then this is just the regular Q-learning target. If we only update the action selector at each update and randomly choose which of the two Q-functions should be used as the action selector at each update, the overestimation is reduced. This also applies to DQN, and it has been shown that using a double DQN results in better policies than using a regular DQN [25].

Prioritized Replay

By using experience replay, agents are not forced to process transitions in the exact order that they are experienced. However, because we are sampling the transitions uniformly from the replay memory, all transitions are given equal priority. We might benefit from prioritizing transitions that have a high TD-error magnitude, which acts as a proxy measure of how "surprising" a transition is to the agent [56].

Prioritizing experience by the magnitude of the TD-error may introduce a lack of diversity. One of the reasons for this is that an experience that initially had a low TD-error, but that later becomes large as the network is trained, will continue to be down prioritized because the TD-error is only updated when the transition is revisited — and because

of its low prioritization, the probability that it will be revisited soon is low. A stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling is introduced to overcome this challenge.

Another problem with prioritized experience replay is that DQN minimizes the expected TD-error squared with respect to the network parameters θ, assuming that the samples in the replay buffer correspond to the same distribution as seen while exploring. Prioritized experience replay breaks this assumption, introducing a bias in the calculated gradient. This is fixed by using importance sampling, such that the less-sampled experiences are compensated for in the gradient. As the unbiased nature of the updates is most important near convergence at the end of the training, the importance sampling is gradually added towards the end, with less importance sampling included at the start of training.

Prioritized replay is found to speed up an agent's ability to learn by a factor of 2.

Dual Deep Q-Network

In the dueling architecture, or Dual DQN, the network that approximates the Q-function is split into two parts: one for estimating the value of the current state and one for measuring the so-called advantage of taking an action in this state [66]. The combination of the state-value estimate and the advantage yields the Q values:

$$Q(s,a) = V(s) + A(s,a) \tag{2.12}$$

However, because the state value function $V(s)$ can be expressed in terms of the state-action value function $Q(s,a)$ by taking the mean of $Q(s,a)$ over all actions, then it means that the mean of the advantage function $A(s,a)$ over all actions equals zero. This is not necessarily the case because the networks are simply approximations. To fix this issue, the authors also subtract the mean advantage from the equation. This change loses the original semantics of $V(s)$ and $A(s,a)$ but results in a more stable algorithm.

$$Q(s,a) = V(s) + A(s,a) - \frac{\sum_a A(s,a)}{N_{actions}} \tag{2.13}$$

The dueling architecture lets the network train the state-value and advantage functions separately.

Multi-Step Learning

We look only one step ahead when constructing the target in the Q-learning update, but this is not a requirement. We could extend it to look N steps ahead if we wanted to, which is called N-step learning or multi-step learning [60].

To use multi-step learning we must look at N consecutive experiences for every update, and sum the appropriately discounted rewards and add

it to an appropriately discounted value estimation of the final state in the sequence. The N-step target for a given state s_t is given as:

$$\sum_{k=0}^{N-1} \gamma^k r_{t+k+1} + \gamma^N max_a(Q(s_{t+N}, a)) \tag{2.14}$$

If we set N to be 1, the algorithm would equal the standard Q-learning algorithm. As we increase N, the algorithm would become more and more similar to the Monte Carlo method, which looks ahead all the way until the agent hits a terminal state.

$$r_{t+1} + \gamma max_a Q(s_{t+1}, a) = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n max_a(Q(s_{t+n}, a)), \text{ iff n=1} \tag{2.15}$$

The best choice of N usually lies somewhere between 1 and the length of an episode. This is because bootstrapping works best when it is over a length of time in which a significant and recognizable state change has occurred. Another intuition for why multi-step learning improves performance is that when we look further ahead, we depend less on our estimates of the future.

Distributional Reinforcement Learning

The Q-function is an approximation of the *expected* returns, but it is also possible to approximate the *distribution* of returns instead [7]. It makes sense to think about the returns as a distribution, even when the environment has deterministic rewards, because stochasticity is still introduced while training through various sources. Firstly, state aliasing, the conflation of two or more states into one representation, may cause different amounts of rewards to be observed even though the agent "sees" the same state. Secondly, because of bootstrapping, target values are nonstationary while training, and the returns will take on different values over time. Lastly, approximation errors will make the returns seem stochastic because we only approximate the true Q-function.

Approximating the distribution of returns instead of the expected returns results in more stable learning targets.

Noisy Deep Q-Network

Exploration of the environment is often enabled by using an ϵ -greedy policy, where ϵ is gradually reduced. For particularly hard problems, like the Atari game "Montezuma's Revenge", this technique becomes insufficient for exploration [8]. ϵ -greedy policies explore with a fixed probability that is the same for every state. An alternative could be to let the network itself learn when it should explore, and for what states.

NoisyNet-DQN does this by applying learnable parameterized noise to the value network parameters [17]. This does not only enable it to change the amount of exploration itself, alleviating the need for hyperparameter

tuning, but also to apply different amounts of exploration to different states.

Rainbow Deep Q-Network

Many of the improvements that has been made to DQN may be complementary and could be combined into a single algorithm. The Rainbow [29] algorithm combines six such improvements:

1. Double DQN [25]
2. Prioritized replay [56]
3. Dual DQN [66]
4. Multi-step learning [60]
5. Distributional RL [7]
6. Noisy DQN [17]

The authors show that the combined algorithm performs much better than each extension alone in terms of both learning speed and overall performance.

They also performed an ablation study on the Rainbow algorithm to see how much each extension contributes to its overall performance. The study concludes that prioritized replay and multi-step learning contribute the most to the overall performance, as removing them from the algorithm reduces its performance the most. Distributional Q-learning ranked directly below, followed by Noisy DQN, and then Dual DQN. The benefit of using a Double DQN is not apparent, as removing it from the algorithm does not reduce its performance.

2.5 Related Work

2.5.1 ED-/ES-Detection

One early attempt for detecting the ED and ES frames took advantage of the rapid mitral valve opening during early diastole [37]. By measuring the mean intensity variation over time in a small region of interest, one could capture the mitral valve opening and define the frame corresponding to peak intensity as ES. This signal was, in some cases, disturbed by early longitudinal motion of the heart, which led to falsely labeling frames as ES. In the same paper, the authors introduced another method that took advantage of the left ventricle deformation during the cardiac cycle. With this method, ES was defined as the frame with the lowest correlation with the ED frame. The correlation curve would flatten out because of little movement around systole, making the predictions more uncertain. The best results were achieved when using a combination of both methods. For this, a small time window was selected around ES using the correlation

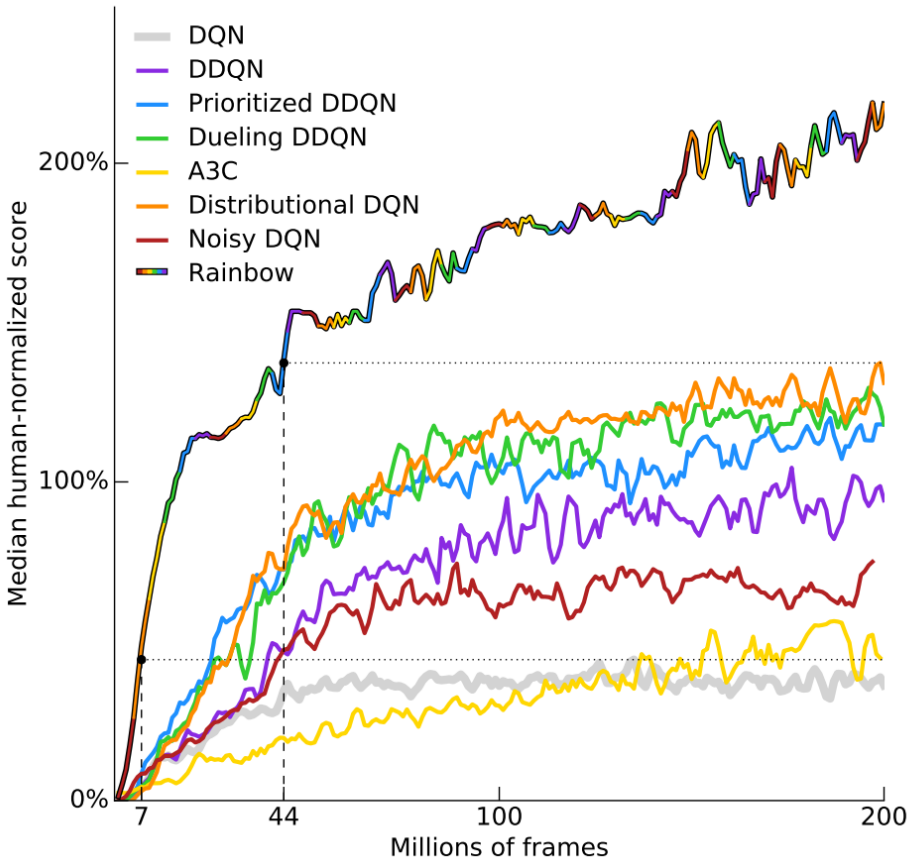

Figure 2.19: A figure from [29] showing the median performance of multiple modified DQN agents compared to human performance across 57 Atari games. After 200 million frames, all modifications show an improvement over regular DQN, but together (Rainbow), they perform significantly better than any one single improvement. Curves are smoothed with a moving average of 5 points.

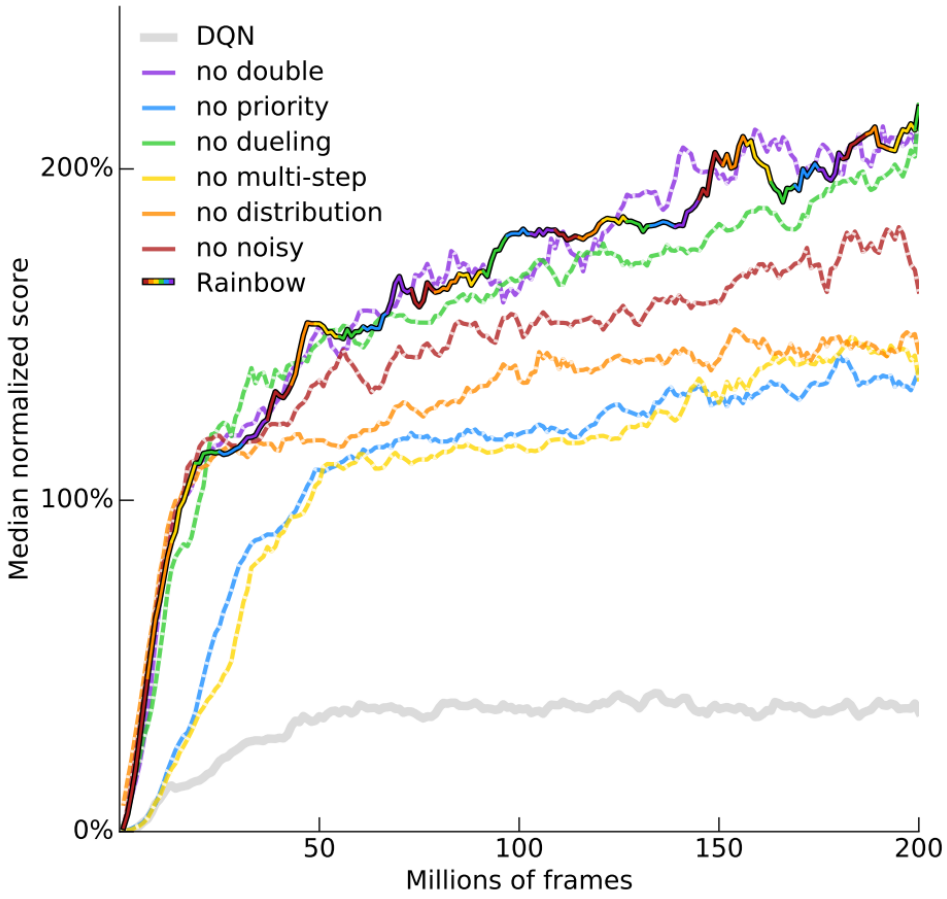

Figure 2.20: A figure from [29] visualizing an ablation study of the various DQN modifications (dashed lines). Dashed lines that are close to the rainbow line indicate that the corresponding DQN modification does not add much benefit to the overall agent or is overshadowed by other modifications. According to the ablation study, the three most important modifications are N-step bootstrapping (multi-step), distributional Q-learning, and prioritized replay.

method, and the mean intensity variation method was used to determine the final ES frame prediction.

The first method requires the clinician to select multiple landmarks to define the correct region of interest around the mitral valve. The second method assumed that the ED frame has already been found to compute the correlation between it and the other frames. The main disadvantage of this approach is that it is only semi-automated.

It has become more common to apply end-to-end Machine Learning (ML) for fully automating tasks like this in recent times. Gifani et al. (2010) employed manifold learning, an unsupervised learning algorithm used to map high-dimensional data onto a lower-dimensional manifold. The authors reduced the dimensionality of each frame down to two dimensions, followed by analyzing the density between the projected points to determine the ED and ES frames [21]. This method is based on the fact that there is no prominent change in ventricular volume during the three cardiac phases: isovolumetric contraction, isovolumetric relaxation, and reduced filling. Frames that lay close together, i.e., in dense regions, are considered part of one of these three phases. The projected points move very little in these dense regions, and the three points that had the least movement were selected as representative of three phases. The ED and ES frames were then found by finding the pair of said frames with the minimum correlation. The manifold learning algorithm that the authors used is called Locally Linear Embedding (LLE). In a follow-up paper, they used Isomap instead [22], which yielded better results. When using Isomap, they defined the ED and ES frames as the projected points with the greatest distance between them.

Non-negative Matrix Factorization (NMF) is another unsupervised learning method that has been employed to reduce the dimensionality of ultrasound videos [69]. In this work, rank-2 NMF was used to generate two end-members from a cardiac ultrasound video. The end-members turn out to be quite similar to the ED and ES frames, and the end-member coefficient peaks can be used to find ED and ES. NMF was found to give predictions with less error than LLE and Isomap manifold learning.

	ES Difference[1]		ED Difference[2]	
	mean	variance	mean	variance
NMF	0.93	1.72	0.93	1.17
LLE	2.76	5.92	2.22	8.79
ISOMAP	1.93	3.94	1.90	8.75

1. Difference between the frame number of extracted ES and Ground truth ES.
2. Difference between the frame number of extracted ED and Ground truth ED.

Figure 2.21: Comparison between NMF, LLE, and ISOMAP results for all 99 cases in the apical 4 view, taken from [69].

Other methods use either image segmentation or speckle tracking to track the changes to the left ventricle volume, taking advantage of the fact that it is most expanded during ED and most contracted during ES [6] [14] [1]. However, these methods are prone to significant errors due to noise

inherent in cardiac ultrasound or discontinuous edges.

A CNN and an RNN were combined to do spatial and temporal feature extraction to detect the ED and ES frames by Kong et al. in 2016 [40]. The combined network was trained on cardiac MRI data, and it used a Zeiler-Fergus model [70] for the CNN, and an LSTM [30] for the RNN. The problem was treated as a regression problem for a function that monotonically decreases during diastole and monotonically increases during systole. Thus, the function being regressed is a latent space representation of the left ventricle volume as it expands and contracts, and the ED and ES frames can be found by finding the highest peaks and lowest valleys of the model's output. This approach was later improved by swapping out the CNN with a ResNet [15], and then again by swapping it out for a DenseNet [62], while different choices for the RNN did not significantly improve the performance of the model.

Instead of treating the model's output as a function regression, it has also been treated as a binary classification of either ED or ES [16]. The authors of this paper argued that treating it as a regression problem forced the model to learn a function that was not present in the data because the regressed function does not represent the actual left ventricle volume. They argued further that, in some cases of pathology, such as in the event of post-systolic contraction, the volume might not be smallest at the time of ES. Their model also uses a 3D CNN with a sliding window that does both spatial and temporal feature extraction on the data before being passed into an LSTM. A similar architecture has been used for finding the ED frames in cardiac spectral Doppler imaging [35]. Spectral Doppler is a technique that outputs a spectrogram representing the blood velocity over time. It thus has one spatial dimension and one temporal dimension. A CNN with a sliding window was used to extract spatial and temporal features, followed by a bidirectional GRU that further connects said features temporally. For each patch in the sliding window, the model predicts whether it contains an ED frame and which frame in the patch it is.

The latest model iteration in this sequence of papers reverts to a regression-based approach, countering the anti-regression argument by stating that a simple binary classification ignores high-level spatial and temporally related markers [42]. The authors explore multiple architectures, but a ResNet50 followed by two layers of LSTM yielded the best results and is the current state-of-the-art. Lastly, they also provided a method for benchmarking different architectures by providing their patient dataset and models to the public and including performance reports on an independent external dataset.

RL has produced even better results than supervised learning methods for many tasks, including medical imaging tasks [73]. RL has not yet been applied to the problem of ED and ES detection, even though it has seen a similar increase in capabilities as supervising learning has in the last decade. The following section introduces examples of how RL has been applied to medical imaging.

2.5.2 Reinforcement Learning in Medical Imaging

RL has seen many medical imaging applications in the last decade, especially in the last five years [73]. One of the main challenges of applying RL is formulating the problem to fit into the RL framework of states, actions, and transition and reward function. The reward function is usually the most difficult to get right out of these four elements.

One way to formulate the problem is as a search through parameter space. Here, the actions are defined as taking a single step along one of the parameter dimensions. The reward function could be how much closer the agent got to the optimal solution after taking a step (the state and transition function definitions vary depending on the problem). This formulation has been applied to many different medical imaging problems, including that of landmark detection.

Landmark detection aims to find a point in an image that represents a medical landmark. In a 2D image, it can thus be defined by the parameters $[x, y]$, where the goal is to find the x and y values that correspond to a given landmark. The state presented to the RL agent will thus be defined in terms of these parameters, such as a smaller section of the image centered around the current point. The action space is defined as a change to the parameters, for example, by increasing or decreasing one of them by some value δ:

$$A = \pm\delta x, \pm\delta y$$

The reward signal could be to look at the change of distance to the ground truth landmark after taking an action, which incentivizes the agent to take steps that take it closer to the landmark:

$$R(s_t, s_{t-1}, a) = D(x_{t-1}, y_{t-1}) - D(x_t, y_t)$$

where $D(x, y)$ returns the distance from the point (x, y) to the ground truth landmark. If the distance were 10 in the previous state and 8 in the new current state, the reward would be $10 - 8 = 2$. If the distance were 4 in the previous state and 7 in the new current state, then the reward (or penalty, in this case) would be $4 - 7 = -3$.

This formulation was used for landmark detection in 2D and 3D CT images in a series of papers by Ghesu et al. [20] [19] [18]. Compared to other state-of-the-art methods at the time, which performed an exhaustive search across the input image, an RL agent only has to follow a simple path, which in the first paper of the series was reported to speed up the detection by 80 times for 2D data and 3100 times for 3D data [20].

The agent traverses the space by taking a step in one direction, up, down, left, right, forward, and back for 3D images, until it converges around a point that is then considered landmark prediction. Convergence occurs when the agent starts showing oscillating behavior. In the follow-up papers [19], and [18], a multi-scale approach was used, wherein the agent searches for the landmark at increasingly fine levels. The first and largest field of view ensures that the agent has access to sufficient global context. When the agent converges, the next scale level is used, and the

agent continues searching on this finer scale. A final prediction is made when the agent converges on the finest scale level.

Q-learning is used with a deep CNN as a function approximator, making it a DQN, similar to the model used in [50]. A different model is trained at each scale.

In addition to a strong speed-up and ability to detect landmarks perfectly from the authors' validation data, the agent can also detect when a landmark is outside of the present scan. In this case, the agent will attempt to leave the image space.

Different versions of DQN and landmark detection problem formulation have been explored. Inspired by the work by Ghesu et al., Alansary et al. explore using a DQN, a Double DQN, a Duel DQN, and a Double Dual DQN for landmark detection in 3D ultrasound and MRI [3]. The formulation of the problem into state, actions, and reward function remains mostly the same as in [19] and [18], except that the state also has a buffer of the last three previously visited states. Including a small history buffer of previous states increases stability and prevents the agent from getting stuck in repeating cycles. Both fixed and multi-scale searching strategies are compared, but the same DQN is shared across all levels in the multi-scale case. They conclude that a multi-scale search strategy improves the performance, especially for large or noisy images, while also speeding up the search process by 4-5 times, but that the choice of deep RL architecture depends on the environment.

A medical image may consist of multiple different landmarks. Vlontzos et al. extend the DQN to a collaborative model where multiple agents share a common CNN but look for different landmarks [65]. This is done using a shared CNN, followed by K different sets of fully connected layers, where K equals the number of agents. The fully connected layers learn to find their respective landmarks, while the CNN is trained on data from all the agents at once. This collaborative framework acts as an implicit form of layer regularization to the network and provides indirect knowledge transfer between agents.

The formulation for treating RL as a search through parameter space has been applied to other tasks as well, such as image registration [44] [41], object/lesion localization and detection [47], and more [73].

Image Registration is about aligning two or more images, transforming them into the same coordinate system, and allowing them to provide complementary information in combination. If the transformations can be assumed to be rigid, the set of parameters could consist of simply translation and rotation, making a total of 6 parameters, or 12 actions, for 3D images [44]. If the transformations have to be non-rigid, then free form deformations can be used on the image to be registered, such as in the work by Krebs et al. in 2017 [41]. In their paper, to reduce the number of actions, they use the first m modes of the PCA as the parameter vector, making a total of $m \times 2$ actions.

Object/lesion localization and detection apply object localization to medical imaging. The goal of the algorithm is to find a bounding box around particular objects in the image. For lesion detection in 3D breast

scans, Maicas et al. (2017) used a parameter space consisting of translation and scale [47]. The agent can take a step along any of the three spatial dimensions or change the scale of the bounding box, making a total of eight actions. Additionally, a ninth action was added that acted as a trigger for when the agent has found a lesion instead of relying on an agent's oscillating behavior around the target.

Not all problems fit into this formulation, however. Video summarization is the task of reducing the length of a video while keeping as much useful information as possible. Liu et al. (2020) use RL for summarizing 15 to 65 minutes long fetal ultrasound videos. It is difficult to formulate this problem as a search in parameter space, and therefore the aforementioned reward function based on distance can not be used. Instead, the authors design a reward function that tries to encapsulate what it means to have a good video summarization. The reward function is a sum of three parts:

- $\mathcal{R}_{det}$: the likelihood that a selected frame is of a standard diagnostic plane.
- $\mathcal{R}_{rep}$: the temporal cohesiveness of the selected frames, incentivizing selecting continuous video sections.
- $\mathcal{R}_{div}$: the diversity of the frames, incentivizing selecting frames that are different from each other such that the summarization will be more representative of the whole session.

The action space consists of only two actions: include the current frame or do not include the current frame in the video summary. By using this straightforward action-space formulation, and a set of high-level rewards, the agent is still able to achieve good performance. The agent's predicted summary scores 62.08 in precision and 64.54 in recall compared to a user annotated summary.

This work serves as inspiration for this thesis and helps guide our RL formulations for the task of ED-/ES-frame detection.

Chapter 3

The Dataset

Only one dataset was used in this thesis. This chapter gives an overview of the dataset used to train the models, how it is organized and how we pre-process it.

3.1 Echonet-Dynamic Dataset

The Echonet-Dynamic Dataset [53] is an openly available collection of 10,030, 112-by-112 pixels echocardiography videos for studying cardiac motion and chamber volumes. Each video has been cropped and masked to exclude text, ECG- and respirometer-information, and downsampled from its original size into 112-by-112 pixels using cubic interpolation. All videos are of the apical-4-chamber view, and each video is from unique individuals who underwent imaging between 2016 and 2018 as part of routine clinical care at Stanford University Hospital. Images were acquired by skilled sonographers using iE33, Sonos, Acuson SC2000, Epiq 5G, or Epiq 7C ultrasound machines. Each video has been labeled by a registered sonographer and verified by a level 3 echocardiographer in the standard clinical workflow.

The dataset consists of three parts: *FileList.csv* contains general information about each video, its variables are listed in table 3.1. *VolumeTracings.csv* contains the volume tracings and ED/ES frame index of each video, its variables are listed in table 3.2. And finally *Videos*, containing all the ultrasound videos in `.avi` format. Video frame samples can be seen in figure 3.1.

3.1.1 Getting ED/ES Frame Information

To get the ED and ES frames, we have to look at the volume tracings, whose variables are listed in table 3.2. The volume tracings list the line segments that define the heart's volume at a given frame. There are two sets of line segments for each video, one for ED and one for ES, but which one is which is not given explicitly. We can find this information by calculating the volume from the line segments for both frames and comparing them — the one with the largest volume is ED, and the other one is ES.

Table 3.1: Echonet video general information variables.

Variable	Description
FileName	Hashed file name used to link videos, labels, and annotations
EF	Ejection fraction calculated by the ratio of ESV and EDV
ESV	End systolic volume calculated by the method of discs
EDV	End diastolic volume calculated by the method of discs
FrameHeight	Video Height
FrameWidth	Video Width
FPS	Frames Per Second
NumberOfFrames	Number of Frames in the whole video
Split	Classification of train/validation/test sets used for benchmarking

Table 3.2: Echonet video volume tracing variables

Variable	Description
FileName	Hashed file name used to link videos, labels, and annotations
X1	X coordinate of the left-most point of line segment
Y1	Y coordinate of the left-most point of line segment
X2	X coordinate of the right-most point of line segment
Y2	Y coordinate of the right-most point of line segment
Frame	Frame number of video on which tracing was performed

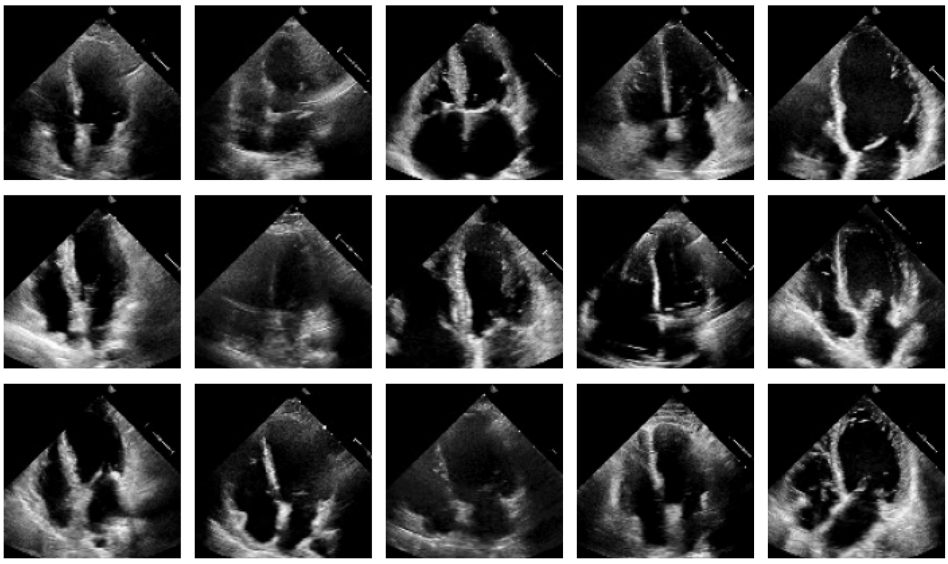

Figure 3.1: The first frames of 15 randomly sampled videos from the Echonet dataset.

3.1.2 Extrapolating Diastole and Systole Labels

As is explored in later chapters, we would also like to label the phase of each frame in the video, not just the frame that ends each phase. When we only have access to the end-frames of each phase, the first phase will only have one labeled frame. For example, if the ED frame comes first, then only the first frame will be labeled diastole as the rest will be systole, as visualized in figure 3.2.

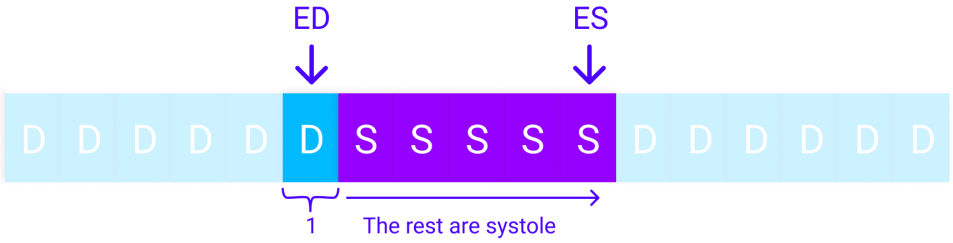

Figure 3.2: Class imbalance: only the first frame is marked with the phase of the first end-event (either ED or ES). All others are marked with the other phase.

We can extract more frames before and after the labeled frames by exploiting the periodicity of the cardiac cycle. As the heart goes from one phase-end to the other, the difference between the current frame and the first phase-end differs more and more. When the opposite end-phase is reached, the frames will start to differ less. For example, the next frame with the biggest difference from the ED frame is likely close to the ES frame. This periodic effect can be seen if we plot the absolute difference between a frame and the rest of the video, as seen in figure 3.3.

An optimistic approach would be to label all the frames until the previous or next peak difference. For example, if the first event is ED, we could label all previous frames until the next peak difference as diastole. Likewise, if the final event is ES, we could label all following frames until the next peak difference as diastole. The peak can be found by finding the first frame whose difference is less than the one preceding it, i.e., when the difference is no longer increasing. This risks labeling too few frames if there is a local peak due to noise, but this problem can be mitigated by smoothing the summed absolute difference values. A gaussian blur with a kernel standard deviation of 5 was used to smooth the values.

We also risk labeling too many frames, adding wrongly labeled frames, because there are no guarantees that the peaks directly coincide with the change of phase. This problem can be mitigated by only including a certain percentage of frames leading up to the peak. We elect to include 75% of the frames leading up to the peaks.

An example of a smoothed absolute-difference curve with 75% of extrapolated frames highlighted is plotted in figure 3.4.

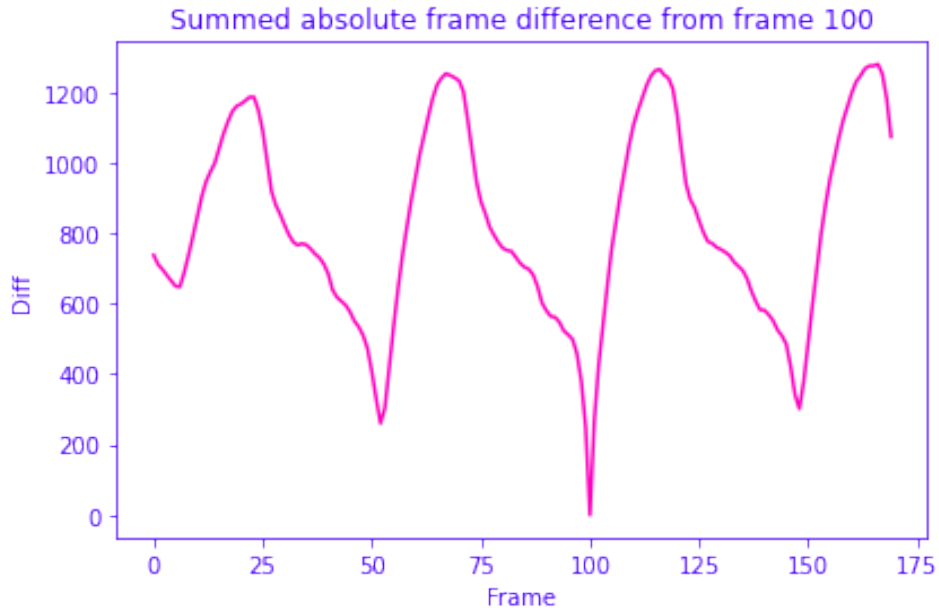

Figure 3.3: The absolute frame difference of all frames in a video compared to frame 100. Notice that the difference for frame 100 is 0 as it (of course) equals itself.

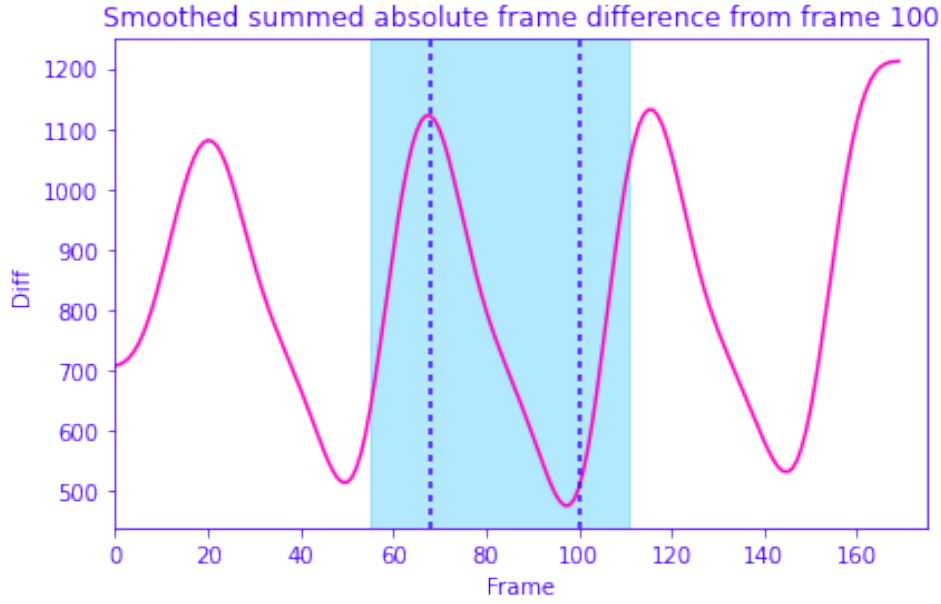

Figure 3.4: The same summed absolute frame difference plot as in figure 3.3, but smoothed using a gaussian blur with a kernel standard deviation of 5. The dashed lines represent phase-end events, and the frames in the light blue area are frames that have their phase labeled. Notice how the labeled frames' perimeter only extends 75% towards the peak on the right side. Also note that the gaussian blur causes the summed absolute frame difference for frame 100 to no longer be 0.

3.1.3 Normalizing and Removing Invalid Videos

When labeling the frames, an assumption is that both events occur within the same cardiac cycle, though this is not always the case in the dataset. To filter out videos where the annotated end-phase events go beyond a single cycle, we again analyze the periodicity using a similar method to the one used in the previous section.

The summed absolute frame difference should at most have one peak if the frames are from the same cardiac cycle. If it has two or more peaks, it suggests that the labeled video contains more than one heartbeat and thus can not be adequately labeled. There are 19 such videos in total, and these are filtered out. A set of good and bad video label examples are visualized in figure 3.5.

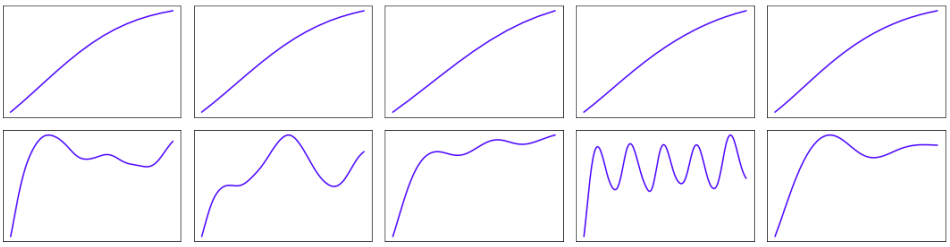

Figure 3.5: The summed absolute frame difference between the first end-phase event and the frames until the next end-phase event. This should only be a half cardiac cycle, so there should be at most one peak. The upper plots show videos where the end-phase labels only cover one half cardiac cycle, while the bottom plots show videos with more than one cardiac cycle and thus have incorrect labels.

The videos already have the same size of 112-by-112, but the frames-per-seconds (FPS) differs. Luckily, most videos in the dataset have the same FPS — almost 80% of the videos have exactly 50 FPS. The smallest FPS is 18, and the highest FPS is 138. See figure 3.6 for a histogram (logarithmic scale on the y-axis) of the different FPS values.

To normalize the videos with a much smaller FPS than 50, we would have to add information to them by inserting new frames. However, this may add unwanted bias to the data, and it is not obvious how to label the interpolated frames when the video goes from one phase to another. We would have to remove frames to normalize the videos with a much higher FPS. Unless the FPS is a multiple of 50, we risk introducing varying FPS to the video, which may confuse the model. For example, if a video has 75 FPS, we could opt to remove every third frame to make it 50 FPS, but this would make it seem like the heart moves slightly faster every third frame.

Because the Echonet dataset is so large, we opt to simply filter out all videos that have an FPS other than 50. Thus, we filter out another 2071 videos, leaving us with 7946 videos.

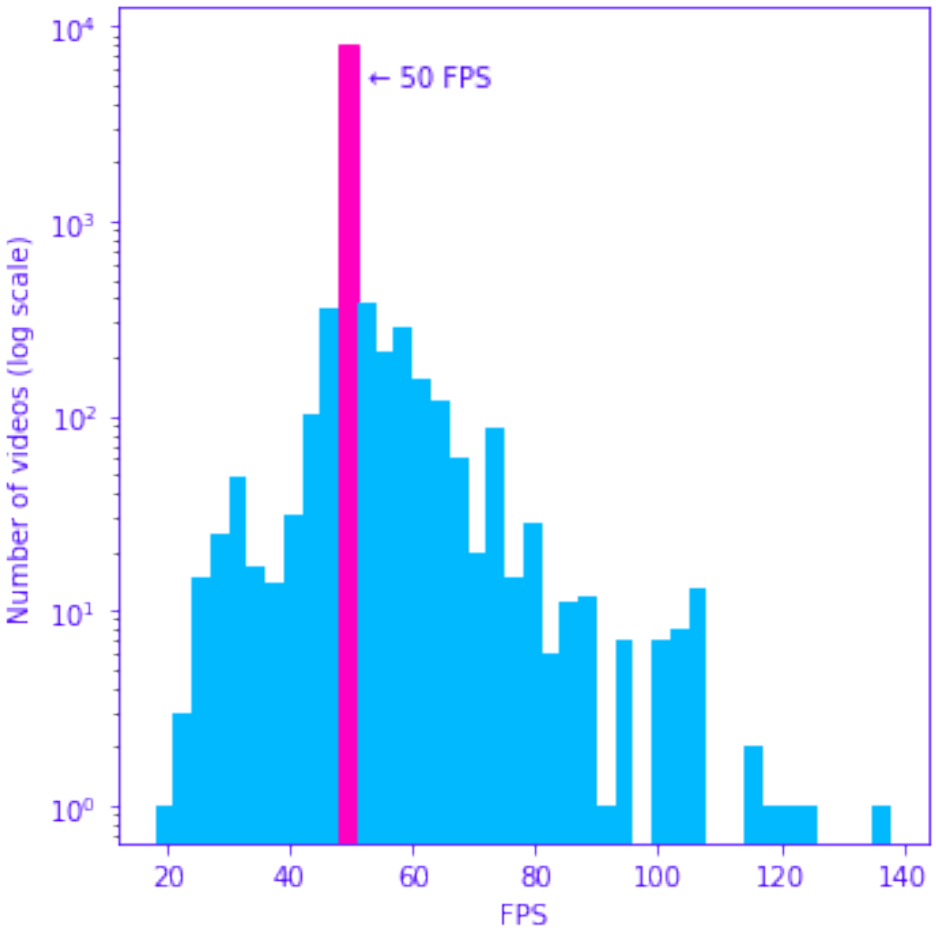

Figure 3.6: A histogram of the different FPS rates of the videos in the Echonet dataset. Note that the y-axis is on a logarithmic scale — in fact, almost 80% of the videos have precisely 50 FPS.

3.1.4 Training, Validation, Test Split

The dataset has already been split into three parts: one part for training the algorithm, one part for validation, and one for testing (i.e., presenting results). The percentage split is approximately 75% for training, 12.5% for validation, and 12.5% for testing. After filtering out videos as explained in the previous two sections, the split ratios remain approximately the same. We opt to continue using this split in this thesis.

A full Echonet-Dynamic dataset pipeline is visualized in figure 3.7.

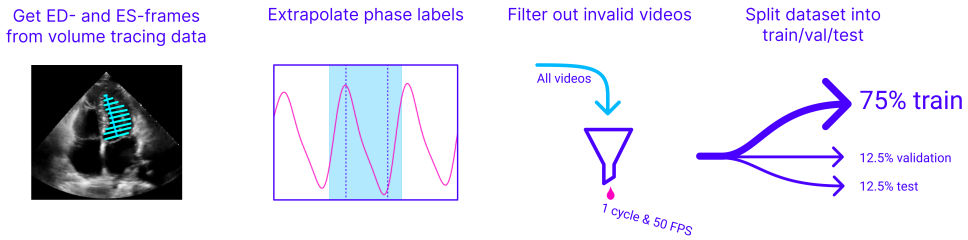

Figure 3.7: A visualization of the data processing pipeline for the Echonet-Dynamic dataset, as described in the previous subsections. First, the ED- and ES-frames from the video are extracted from the volume tracings data. The frame with the biggest volume is ED; the other is ES. Next, more frame labels are extrapolated by looking at the absolute pixel differences between the ED- or ES-frame and the other frames of the video. Then, videos are filtered such that not more than one cardiac cycle is included in the labeled frames and all videos have 50 FPS. Finally, the videos are split randomly into three subsets: training, validation, and testing.

Chapter 4

Methodology

This chapter steps through the methods used in the thesis, the decisions taken, and the reasoning behind them. Section 4.1 introduces and describes the binary classification environment (BCE), and the design of the three different reward functions R_{GaaFD}, R_{simple}, and $R_{proximity}$. Section 4.2 gives an overview of the software technologies used to train the models. Section 4.3 describes the architecture of the agent and neural network, and how it was optimized and trained. Section 4.4 describes how we evaluate the models in the Experiments and Results chapter. Section 4.5 describes the hyperparameters used in the experiments and why they were chosen. Section 4.6 explores different ways of incorporating search into the environment. Finally, section 4.7 describes the m-mode binary classification environment (MMBCE), a version of BCE that incorporates search using a synthetic m-mode image.

4.1 Environment Formulation

As described in section 2.4.5, a markov decision process (MDP), which is at the core of RL, can be described using four elements: the state space, the action space, the transition function, and the reward function. The states and actions dictate what information the agent receives from the environment and how it can, in turn, interact with the environment. The transition function defines the effect of actions on the environment. The reward function defines the goal of the agent.

4.1.1 Binary Classification Environment

BCE is visualized in figure 4.1. After observing the current and adjacent frames, the agent takes an action predicting that the current frame is either in the diastole or systole phase and receives a reward dependent on its prediction before the environment moves the current frame one frame forward.

More formally, the observation o_t at time t is the current frame in the video prepended by the N previous frames and the N next frames. The shape of an observation is thus $(W, H, 2N + 1)$. The agent takes the

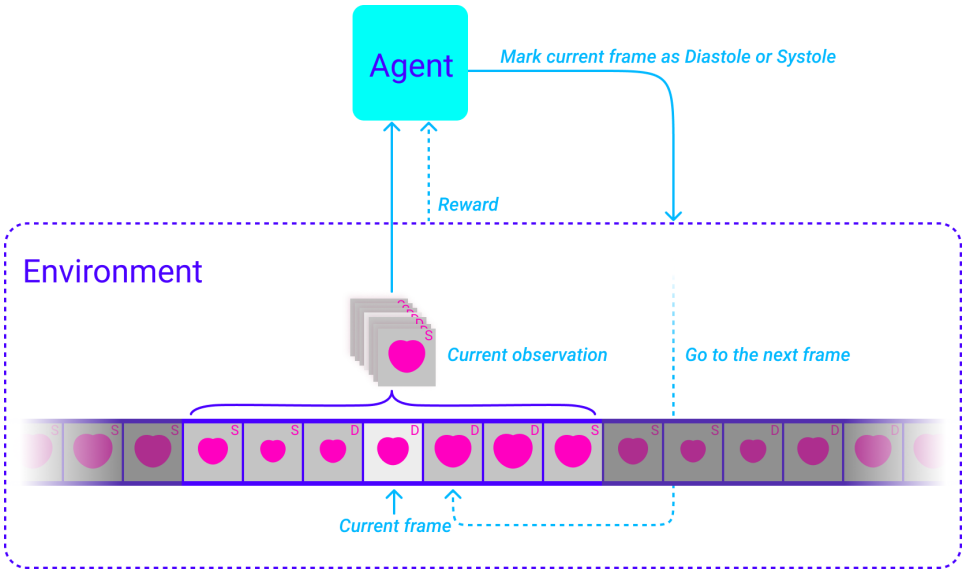

Figure 4.1: Visualization of the Binary Classification Environment loop. An agent sees the observation from the current frame and takes an action, either marking it as diastole or as systole, and gets back the reward and the observation for the next frame from the environment.

observation as-is and takes one of two actions: *Mark current frame as diastole* or *Mark current frame as systole*. After taking an action a_t, the agent receives a reward r_{t+1} and is presented with the next observation o_{t+1}. The current frame is moved one frame forwards after each action is taken, and the episode ends when there are no more labeled frames left.

Given that videos from the dataset are 112-by-112, the only two hyperparameters for this setup are N and the choice of reward function. Increasing N means that the agent has access to more temporal information but at the cost of increased computational and memory requirements and a decrease in the number of videos with enough adjacent frames on either side. The number of valid videos for a given N and the change in the number of valid videos is plotted in figure 4.2. As a starting point, N was selected[1] to be 3. This means that an observation has the shape $(112, 112, 7)$, having $2 \times 3 + 1 = 7$ channels.

4.1.2 Reward Function Design

The standard metric for this task is the average absolute frame difference (aaFD), as defined in equation 4.1. aaFD measures the precision and accuracy of predictions by measuring the frame difference between each ground truth event y_t and the corresponding prediction $\hat{y}_t$ generated by the model — a lower aaFD meaning that the model is making fewer errors. t is the index of a specific event, of which there are N in total.

[1] Perhaps a bit arbitrarily selected.

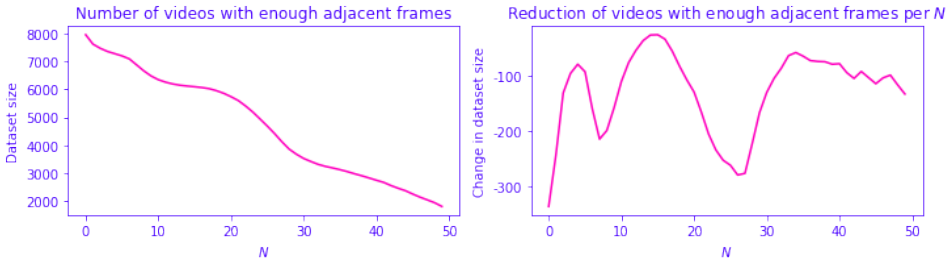

Figure 4.2: The effect of N on the size of the dataset. Left: the number of valid videos (videos with at least N adjacent frames on either side) for the whole dataset. Right: the change in the number of valid videos per N for the whole dataset.

$$aaFD = \frac{1}{N}\sum_{t=1}^{N}|y_t - \hat{y}_t| \tag{4.1}$$

One weakness of aaFD is that it is only defined when there are an equal number of predicted events as there are ground truth events. This is not always the case, as an imperfect model may predict more or fewer events. A generalized aaFD ($GaaFD_1$) was considered for a metric instead, calculated as the average frame difference between each predicted event and its nearest ground truth event as in equation 4.2, having the property that it converges towards the true aaFD as the model improves. In equation 4.2 $\hat{N}$ is the number of predicted events and $\mathcal{C}(y, \hat{y})$ is the frame difference between the predicted event to the *closest* ground truth event of the same type. For cases where there are more predicted events than there are ground truth events, $GaaFD_1$ would, as is rational, give a worse score. However, for cases with fewer predicted events than ground truth events, $GaaFD_1$ would give a score that does not reflect its inability to predict all events.

$$GaaFD_1 = \frac{1}{\hat{N}}\sum_{t=1}^{\hat{N}}|\mathcal{C}(y, \hat{y}_t) - \hat{y}_t| \tag{4.2}$$

Similarly, we could base it on the ground truth events and take the distance to the nearest predicted event, $GaaFD_2$, as in equation 4.3, we get the opposite problem — too many predicted events are not reflected negatively in the score.

$$GaaFD_2 = \frac{1}{N}\sum_{t=1}^{N}|y_t - \mathcal{C}(y_t, \hat{y})| \tag{4.3}$$

By combining $GaaFD_1$ and $GaaFD_2$ as in equation 4.4 we mitigate these problems while maintaining the convergence property.

$$GaaFD = \frac{1}{N+\hat{N}}\left(\sum_{t=1}^{N}|y_t - \mathcal{C}(y_t, \hat{y})| + \sum_{t=1}^{\hat{N}}|\mathcal{C}(y, \hat{y}_t) - \hat{y}_t|\right) \tag{4.4}$$

Using negative GaaFD (negative because we wish to minimize it) as a reward function for RL means optimizing the agent directly for our main metric aaFD. However, it has one final flaw: it is only defined on whole episodes. This means that the agent has to run an entire episode before getting a reward, making the reward signal sparse.

Instead, we could frame the problem as a simple classification problem where the agent must classify individual frames as either ED, ES, or neither. This allows us to give a reward at each step depending on whether the prediction was correct or not. One problem with this approach is that there is a heavy class imbalance because most frames are neither ED nor ES. A solution to this is to instead predict the phase, either diastole or systole, as it is trivial to find ED and ES from the phase by finding the frames where it transitions from one to the other.

From this, we can define a simple reward function R_{simple} that gives a reward of 1 if the predicted phase was correct and -1 if it was incorrect, as seen in equation 4.5. The information that the agent receives from the reward signal R_{simple} is slightly different from the one defined through GaaFD, as GaaFD penalizes predictions that are more wrong heavier than those that are close to the ground truth.

$$R_{simple}(s,a) \triangleq \left\{ \begin{array}{ll} 1 & \text{if phase}(s) = a \\ -1 & \text{if phase}(s) \neq a \end{array} \right\} \tag{4.5}$$

We can make the reward signal more similar to GaaFD by defining it in terms of the distance to the nearest predicted phase, as seen in equation 4.6, where $d(s,a)$ is the distance in frames from the current state s to the nearest frame that has the predicted phase a.

$$R_{proximity}(s,a) \triangleq -d(s,a) \tag{4.6}$$

4.2 Frameworks and Libraries

The code to train and run the agent is written in Python because of its ML and data-processing ecosystem. The main framework for data-processing is JAX [9]. Other frameworks considered were Tensorflow [2] and PyTorch [54]. A list of the most important ones can be found in table 4.1.

4.3 Agent Architecture

Deep Q-Network was selected for the RL agent architecture. DQN is a well-established method for scaling up RL by approximating the expected returns of an action in a given state using a (deep) neural network. It is also simple to train distributedly as it is off-policy, enabling us to separate the algorithm into a learner and multiple agents, as explained in a following section.

We take advantage of a few additions to the original DQN algorithm: Prioritized Replay, N-step returns, and Double Q-Learning. An ϵ -greedy policy is used for facilitating exploration.

Table 4.1: A collection of the most important libraries used in the project.

Library	Description
jax	Main data-processing framework. Provides autodifferentiation, vectorization, Just-In-Time (JIT) compilation, and more [9]
gym	An interface for defining RL environments [10]
dm-haiku	A neural network library for JAX [27]
optax	A gradient processing and optimization library for JAX [28]
rlax	Building blocks for building RL agents [5]
dm-acme	Distributed RL agent implementations and building blocks [31]
dm-reverb	A database for storing and sampling experience replay [12]
dm-launchpad	A library for defining and creating distributed systems [68]
Scikit-learn	A collection of machine learning algorithms. In this project it is mostly used for calculating metric [55]

4.3.1 Neural Network

The neural network that approximates the Q-function is inspired by the original Atari DQN paper [50]. It has two convolutional layers and two fully connected layers. A ReLU activation layer follows each layer except for the last one. The first convolutional layer has 16 output channels, a kernel size of 8-by-8, and a stride of 4. The second has 32 output channels, a kernel size of 4-by-4, and a stride of 2. The data is flattened before being passed to the fully connected layers. The first fully connected layer has an output size of 256. The final layer has two outputs, each representing the estimated value of taking one of the actions, given the input state. In total there are 1 621 810 parameters. The network is visualized in figure 4.3.

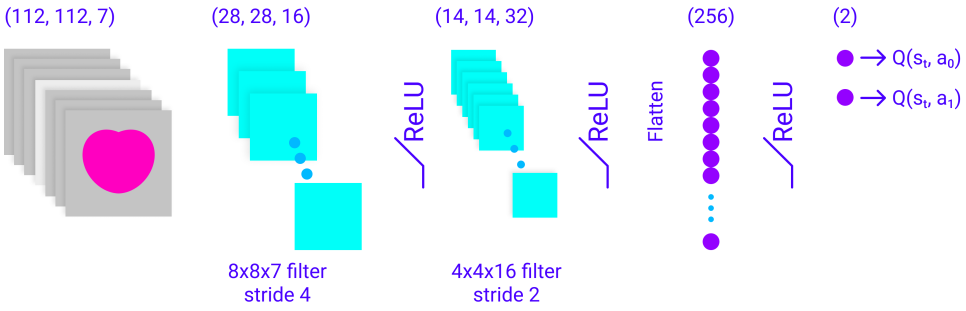

Figure 4.3: A visualization of the simple DQN-Atari-paper-inspired CNN.

4.3.2 Loss Function and Optimizer

The loss function is the Double Q-Learning loss where the TD-error is calculated with respect to another Q-network. Because of this, we have to keep track of two sets of network parameters: one for the selector Q-network and one for the estimator Q-network. Huber loss [33] is applied to

the TD-error such that the L2 loss becomes linear after a certain threshold. In addition, the loss is weighted with respect to the prioritized replay importance weights.

The Adam optimizer [39] is used to update the selector parameters, and the target network parameters are updated to equal the selector parameters every 100 gradient descent steps.

4.3.3 Distributed Training

As mentioned, DQN lends itself nicely to distributed training. In this project, this is achieved through a library called Acme [31]. At the center of Acme is another library called Reverb [12]. Reverb is a database for storing experience replay samples that lets us insert and sample experiences independently. If we separate the learning step and the acting step of the algorithm, Reverb can be used as the communication point between the two. One or more actors, possibly on different machines, can generate experience samples and insert them into the Reverb experience replay database. A learner, also possibly on a different machine, can sample from it to perform gradient descent. The actors and the learner do not need to know about each other, except when an actor needs to update its parameters, in which case it needs to query the learner for the latest trained parameters. It is also trivial to add one or more evaluators that can run in parallel and that only need to query the learner for the latest trained parameters. Inter-process communication is facilitated by a third library called Launchpad [68].

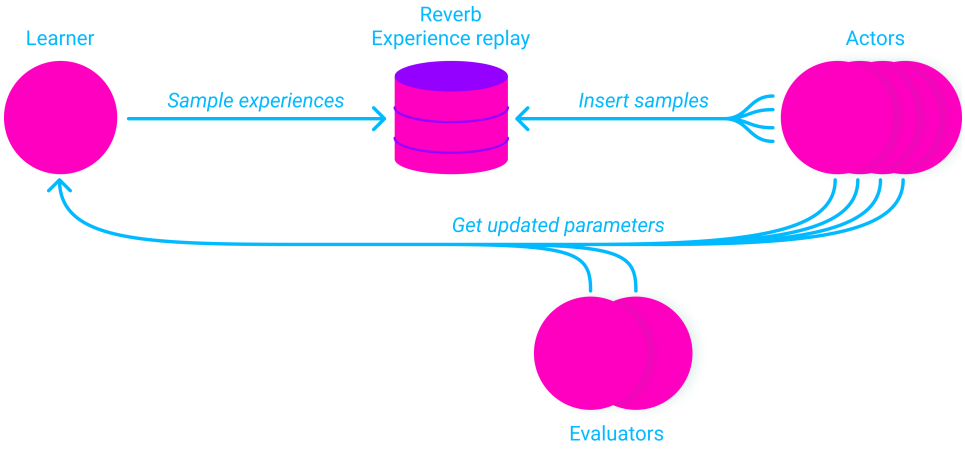

Figure 4.4: An illustration of the distributed RL training system. Each pink node runs in a separate Python process, and each blue arrow is an inter-process function call facilitated by Launchpad.

There is a balance between how fast experience samples should be added to the experience replay and how fast the learner should sample them. If the learner samples faster than the actors can generate new samples, the network will be trained using trajectories generated from outdated policies. If the actors generate new samples much faster than the

learner can sample, then we are wasting computer resources.

Reverb helps maintain this balance through rate limiters. We use a rate limiter that tries to maintain a specified ratio between insertions and samples, blocking either the actors from inserting new samples or the learner from sampling if the ratio differs too much. For example, using a samples-per-insert ratio of 2 means that, on average, each insertion made by an actor will be sampled twice. A ratio of 0.5 means that, on average, each insertion will be sampled half a time — i.e., there are twice as many insertions as there are samples.

4.4 Evaluation

During training, the updated parameters of the model are continuously evaluated using GaaFD on 50 videos, randomly selected each time, from the validation set. Smoothing is applied to the learning curves using a Gaussian filter with a kernel standard deviation of 10 to compensate for the low sample size for each point. The best parameters are selected by finding the parameters that produce the lowest GaaFD during training for the smoothed GaaFD learning curve.

The primary evaluation metric for the trained model is aaFD. However, some videos may not receive the same number of predicted events as there are ground truth events, so aaFD is undefined. Because of this, aaFD is only reported for videos where it is defined. Additionally, the percentage of videos with a defined aaFD is reported. The corresponding ground truth event to each predicted event is chosen to be the closest one, and we can therefore use GaaFD, as defined in equation 4.4, for calculating aaFD.

It may also be interesting to see the density plots of GaaFD for all videos and compare the performance of the agent on ED- and ES-frames individually. The density plots used are an approximation of the continuous distribution of GaaFD. A histogram may also be used, but density plots were found to be easier to compare using density plots. They are created using gaussian kernel estimation (KDE) [57]. The kernel bandwidth is automatically selected using Scott's rule, the default selection method for SciPy's KDE implementation.

Because the RL problem formulation is similar to a regular binary classification problem, accuracy and balanced accuracy are also reported. Accuracy and balanced accuracy are defined on frame phase predictions instead of end-phase events. Accuracy is simply the percentage of correctly labeled frames, as defined in equation 4.7, where $1(y = \hat{y})$ is the indicator function. Given that there is a class imbalance between diastole and systole frames, balanced accuracy gives a more representative score of the actual model performance. Balanced accuracy weights systole frames accuracy higher than diastole frames and is defined in equation 4.8. TP, FP, TN, and FN stand for "true positives", "false positives", "true negatives", and "false negatives", respectively. It is also defined as the average between the sensitivity and the specificity. The balanced accuracy score is also rescaled such that it gives a score in the range $[-1, 1]$, where 0 means that the

model's predictions are random, and -1 and 1 mean that the predictions are all incorrect or all correct, respectively.

$$\text{accuracy}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} 1(\hat{y}_i = y_i) \tag{4.7}$$

$$\text{balanced-accuracy}(y, \hat{y}) = \frac{1}{2}\left(\frac{TP(y, \hat{y})}{TP(y, \hat{y}) + FN(y, \hat{y})} + \frac{TN(y, \hat{y})}{TN(y, \hat{y}) + FP(y, \hat{y}}\right. \tag{4.8}$$

Models are also evaluated on their inference time — how long it takes to make predictions for a video. To use a trained model, one can use the Q-network directly, without instantiating a gym environment or using an ϵ -greedy policy. The Q-network outputs the expected returns of taking either action, so picking the action with the highest output is the same as following a greedy policy. The Q-network can be evaluated on individual frames or on the video as a whole, where all the frames are combined into a single batch. Evaluating each frame individually enables incorporating the model into a pipeline of streaming frames, of which one step is predicting the current cardiac phase. Evaluating the whole video as a batch is generally faster as it gets away with less IO overhead of sending data back and forth between the CPU and the GPU.

Batching the frames of a video may require more JIT compilation with JAX. This is because, to speed the network up significantly, it is JIT-compiled to XLA, but JIT-compiled functions require that the shape of the data remain the same. If the shape of the data is not the same, e.g., if we are evaluating two videos with a different number of frames as two different batches, the function will be recompiled, adding overhead. This could be solved by fixing the batch size to a constant number. For videos with fewer frames than the batch size or with a number of frames that can not be split into equal chunks of the batch size, frames filled with zeros can be added. These extra frames create needless work on the GPU but do not require recompilation.

Inference time is evaluated using single frame-inference and batched-frames inference with a batch size of 128 on the CPU and GPU. Additionally, IO overhead is reported by comparing the average processing time when sending the data to GPU for each call versus pre-placing the data on the GPU. The average run time is calculated by taking the elapsed time, averaged over 1000 calls.

Finally, models are evaluated on how long it took to train them in clock time and the number of SGD steps performed.

4.5 Selection of Hyperparameters

4.5.1 Generalized Average Absolute Frame Difference Reward Function

Using GaaFD directly as the reward function has the benefit that we are directly optimizing the agent for the primary performance metric aaFD, as

defined in equation 4.1. However, as discussed in section 4.1.2, a weakness is that it is only defined at the end of an episode, making the reward signal very sparse. The agent will only get a reward at the last step of an episode, which, on average, lasts for 50 steps.

To solve for reward-sparsity, we use multistep bootstrapping with a value of $N = 200$. An episode is automatically terminated once it reaches 200 steps [2] so this will, in practice, mean that the agent is trained using the Monte Carlo method.

We also set the discount value $\gamma = 1.0$, which means that an agent tries to maximize all future rewards. A value of $\gamma < 1.0$ means that the calculated returns will be noisier and harder to predict because the discounted returns calculated for steps earlier in an episode would have a lower value than those calculated closer to the end.

The expected returns are assumed to be very sensitive to the current policy as a correctly selected next action's returns may be jeopardized by future wrongly selected actions. This would make it hard for the agent to extract information about which actions were wrong and which were correct. Because of this, we opt to use very low values of the exploration parameter ϵ, as higher values of ϵ means that actions will be selected at random more often, making the expected returns harder to predict. Three values are tested for the exploration hyperparameter ϵ: $\epsilon = 0.0$, $\epsilon = 0.01$, and $\epsilon = 0.1$. The agents were allowed to train until they visually reached a plateau. A full list of the hyperparameters used is listed in table 4.5.1 (most relevant ones are highlighted).

Hyperparameter	Value
Epsilon	$\{0.0, 0.01, 0.1\}$
Discount	1.0
N (N-step bootstrapping)	∞
Target update period	100
Importance sampling exponent	0.2
Priority exponent	0.6
Number of actors	8
Min replay size	10 000
Max replay size	250 000
Samples per insert ratio	0.5
Optimizer	Adam with default parameters
Huber loss parameter	1.0
Learning rate	1^{-4}
Gradient descent steps	$\{100\,000, 150\,000, 200\,000\}$
Batch-size	128

[2] Though in the case of BCE, this will never happen because no video has this many frames.

4.5.2 Simple- and Proximity-Based Reward Functions

Using reward functions based on each phase prediction gets around the reward sparsity problem of using GaaFD as the reward function. Two more reward functions are explored: a simple reward function R_{simple}, as defined in equation 4.5, and an proximity-based reward function $R_{proximity}$, as defined in equation 4.6. This makes it quite similar to a supervised regression problem where we want to learn the Q-values given an observation and an action. The returns only depend on the current action and not on all the actions in an episode, as we saw with the GaaFD reward function. As a result, it is assumed that the optimal discounting factor is $\gamma = 0.0$, meaning that the returns are calculated using only the immediate reward. A discount value of $\gamma > 0.0$ would make expected future returns predictions depend more on the current policy, adding noise to the target values until the policy converges.

Unless discounting is not zero, there will be no need for bootstrapping, and we can ignore N-step bootstrapping for these reward functions by setting $N = 1$.

Since an action does not affect future states, exploration is not as important. Instead, we can view the exploration variable ϵ as affecting how input/label pairs are sampled. An exploration value of $\epsilon = 1.0$ means that actions are sampled uniformly, and a value of $\epsilon = 0.0$ means that actions are sampled based on how good it is assumed to be. Three values are tested for the exploration hyperparameter ϵ: $\epsilon = 0.1$, $\epsilon = 0.5$, and $\epsilon = 1.0$. The agents were trained for 200 000 SGD steps. A full list of the hyperparameters used for experiments with reward functions R_{simple} and $R_{proximity}$ is listed in table 4.5.2 (most relevant ones are highlighted).

Hyperparameter	Value
Epsilon	$\{0.1, 0.5, 1.0\}$
Discount	0.0
N (N-step bootstrapping)	1
Target update period	100
Importance sampling exponent	0.2
Priority exponent	0.6
Number of actors	8
Min replay size	10 000
Max replay size	250 000
Samples per insert ratio	0.5
Optimizer	Adam with default parameters
Huber loss parameter	1.0
Learning rate	1^{-4}
Gradient descent steps	200 000
Batch-size	128

4.6 Incorporating Search

Although RL is designed to be able to perform a search through an unknown state space, in the previous setup, there is no exploration as previous actions do not affect future actions. Therefore, there is no reason to believe that RL will outperform a carefully designed supervised learning approach. By transforming the problem to one that requires search, we will have a problem not trivially solved by supervised learning but where RL can shine. Though this may seem like straightening a screw to make it work with a hammer, there may be unforeseen benefits. Of great importance to ML is to represent the problem space such that it is easy for an algorithm to learn from it. Perhaps there is an optimal representation of the problem of ED-/ES-detection that also happens to require search?

4.6.1 Temporal Search

We could formulate the problem as a search in time where the agent must learn to move the current frame towards the end-phase event. The agent sees the current frame and some number of previous and following frames and can either move the current frame backward or forwards. The agent can be rewarded with 1 if it moves a step closer to the nearest end-phase frame and -1 if it moves away from it.

There are a handful of issues with this approach. **Issue 1**: we would have to train two different agents: one for ED and one for ES. **Issue 2**: there is no terminal state, and the episodes can run forever. **Issue 3**: there will be ambiguity in what frame the agent truly predicts as the end-phase because it will likely show oscillating behavior around the predicted frame. **Issue 4**: we would have to run multiple agents at different points in the video to find all end-phase events, and it is not obvious how to do so.

Issue 2 and **issue 3** can be partially solved by including a third action for marking the current frame and ending the episode, though this may still lead to the agent getting stuck in an endless loop of going back and forth. We could also keep just the two actions but terminate the episode once the agent starts showing oscillating behavior, as in [3], as this indicates that it has found the predicted frame. The problem with this is that the final predicted frame would be ambiguous as we do not know which of the two frames the agent oscillates between is the actual predicted frame. However, using DQN, we could peek at the Q-values and pick the frame where the expected reward of taking the action with the maximum expected reward is the lowest. **Issue 4** may be solved by starting an agent from each frame, though this would increase the computational requirements of the algorithm.

4.6.2 Spatial Search

Instead of searching through the video frames, we could let the agent search spatially in the video. In this formulation, the agent only has access to a part of the images while predicting the phase of frames. Like landmark

detection tasks, it can move its focus around in the image, the hope being that it can discover parts of the video, which makes it easier to identify the correct phase. This can be seen as reducing the space and memory requirements at the cost of speed, as the agent has to process a smaller part of the image but may explore multiple steps before making a prediction.

One option is to look at a region of interest (ROI) around a point that the agent can move. Building upon the simple binary classification environment described in previous sections, this would add four new actions: move up, move down, move left, and move right. This is visualized in figure 4.5. Because we reduce the size of the observations, we could either trade it for reduced memory usage or for including more temporal information in terms of included adjacent frames.

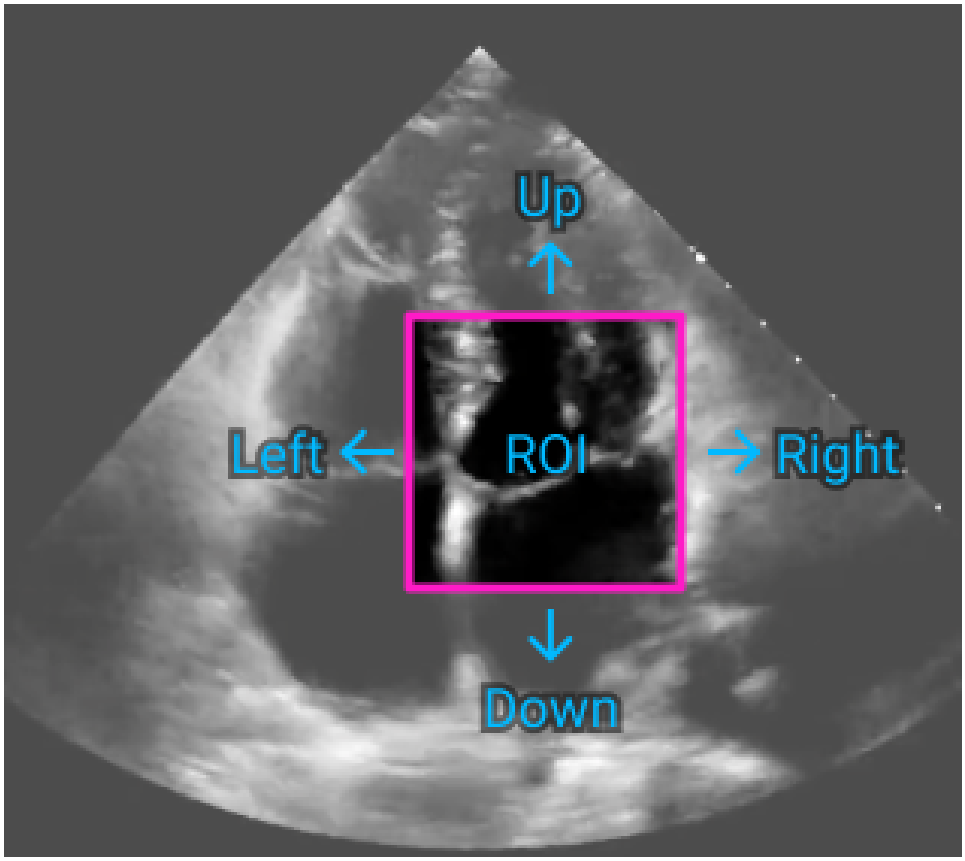

Figure 4.5: A region of interest (ROI) is given to the agent, which it can then move around to explore.

Another option is to take inspiration from m-mode imaging used in ultrasound. We can define a synthetic m-mode image in terms of a line in the video, where it shows how the pixels along this line change over time. A video can be seen as a 3D data cube consisting of width, height, and time. When using the synthetic m-mode technique, width and height are replaced by the line, effectively removing one spatial dimension while keeping the temporal dimension intact. The m-mode can be seen as taking a 2D slice of the video, as seen in figure 4.6. This synthetic m-mode

exploration formulation adds six new actions: move up, move down, move left, move right, rotate left, and rotate right. M-mode imaging is also a well-established imaging mode in clinical settings, so this is the method that we want to explore further.

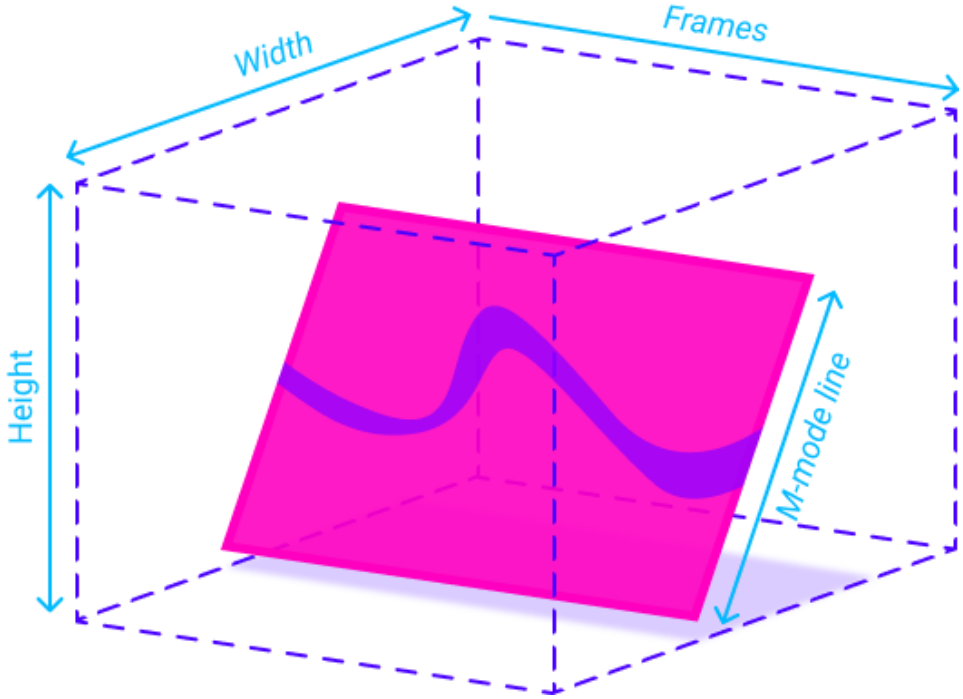

Figure 4.6: An m-mode image is an intersecting plane in 3D "video space".

Moving the synthetic m-mode line up, down, left, or right is done relative to its rotation. We call this local translation, different from global translation, where the movement is independent of the rotation of the line. Local and global translation is visualized in figure 4.7. Using local translation is presumed to add some rotational invariance, as the m-mode line can counteract the rotation of the video itself without changing the perceived m-mode effects of translation. This also makes the effects of the up- and down-translations trivial, as seen in figure 4.8 — independent of rotation, it simply shifts the m-mode image down or up, respectively.

4.7 M-Mode Binary Classification Environment

We formulate the m-mode binary classification environment using a synthetic m-mode search space scheme. The agent can make one of 8 actions, as listed in table 4.2. The movement magnitude is 1 pixel, and the rotation is 0.1 radians.

In addition to the current synthetic m-mode image, we also want to give the agent information about what it would look like if it moved or rotated the line and a history of the latest actions. An observation thus consists of the synthetic m-mode image for three different rotations (rotated left,

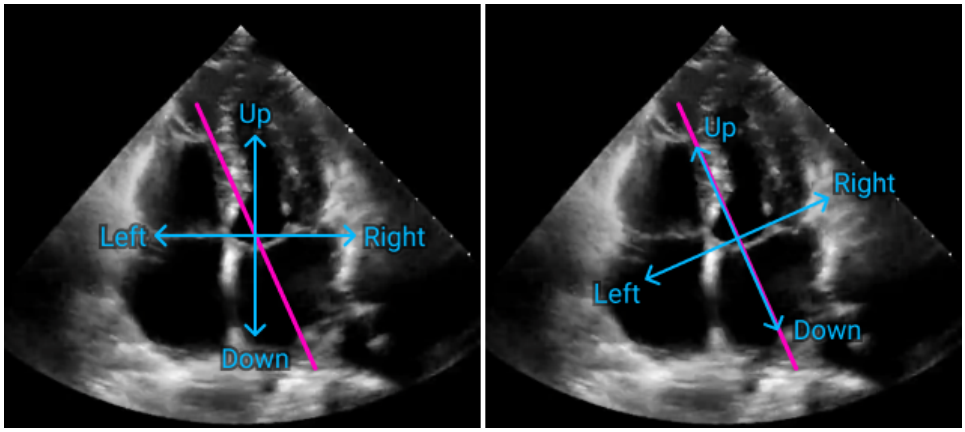

Figure 4.7: Global (to the left) versus local (to the right) translation. Local translation means that the movement depends on the direction of the m-mode line.

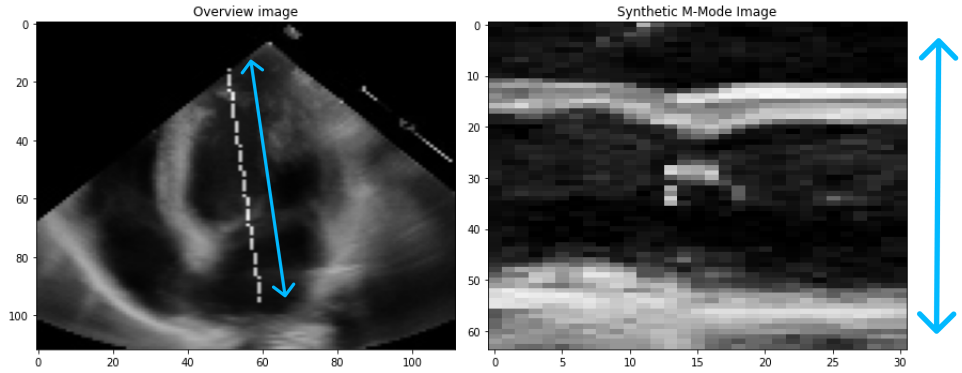

Figure 4.8: Moving the synthetic m-mode line up or down using local translation changes the resulting image very little — it simply translates it up or down, as indicated by the blue arrows. To the left: an overview image of a video with the line added on top. To the right: the resulting synthetic m-mode image.

Table 4.2: The actions that an agent can take in the MMBCE formulation.

Name	Description
Diastole	Mark current frame as diastole
Systole	Mark current frame as systole
Up	Move the line in its current direction
Down	Move the line in the opposite of its current direction
Left	Move the line in the negative direction perpendicular to itself
Right	Move the line in the positive direction perpendicular to itself
Rotate left	Rotate the line to the left
Rotate right	Rotate the line to the right

not rotated at all, and rotated right) and for three different perpendicular movements (moved to the left, not moved at all, moved to the right), for a total of 9 synthetic m-mode images. Up and down line movements are not included as additional channels because they do not provide as much information to the agent, as seen in figure 4.8. The synthetic m-mode image is created by interpolating the line across the video using nearest-neighbor interpolation. An overview image consisting of the average of the first 50 frames and the current position of the synthetic m-mode line is also included in the observation. Lastly, we include the last five actions taken as a one-hot encoded array of shape $(5, 8)$ — 8 being the number of possible actions. Observations are thus a tuple of:

1. An "overview" image of shape $(W, H, 2)$
2. A synthetic m-mode image of shape $(T, L, 9)$
3. An action history array of shape $(5, 8)$

W and H are the width and height of the video, respectively, and T and L are the number of frames (amount of temporal information) and length of the line, respectively.

At the start of an episode, the line is placed randomly within a bounding box. This is to force the agent to learn to explore instead of learning to predict the phase from a common starting position. First, the line is centered, facing upwards. Then it is translated in the direction it is facing by a random amount, sampled uniformly from the interval $[-0.1H, 0.1H]$. Then it is translated perpendicular to the direction it is facing by a random amount, sampled uniformly from the interval $[-0.1W, 0.1W]$. This ensures that the line's center is never more than $0.1H$ units away in the y-direction or $0.1W$ units away in the x-direction from the image's center. Lastly, it is rotated by an angle sampled uniformly from the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$ radians. If a line is somehow generated outside of the video bounds, a new line is generated.

The random starting positions were verified to be reasonable through visual inspection of a sample of 1 000 lines, as seen in figure 4.9. An episode ends once the agent has predicted the phase of every frame or for a maximum of 200 steps. We have to cut the episode off at 200 steps because the agent may now move indefinitely. Given that the average video length is 50 frames, 200 steps should give the agent ample time to find the best synthetic m-mode line position in most cases.

The reward function is the same as in the simple binary classification environment but with some modifications. With MMBCE, the agent may move the line out of the bounds of the video. It may also get stuck in an infinite loop of actions. The agent is determined to be stuck in a loop if the line ends up in a previously visited position and there are no phase predictions since then. If the agent moves the line out of bounds or gets stuck in a loop, the line is moved to a new random position, and the agent is given a reward of -1.

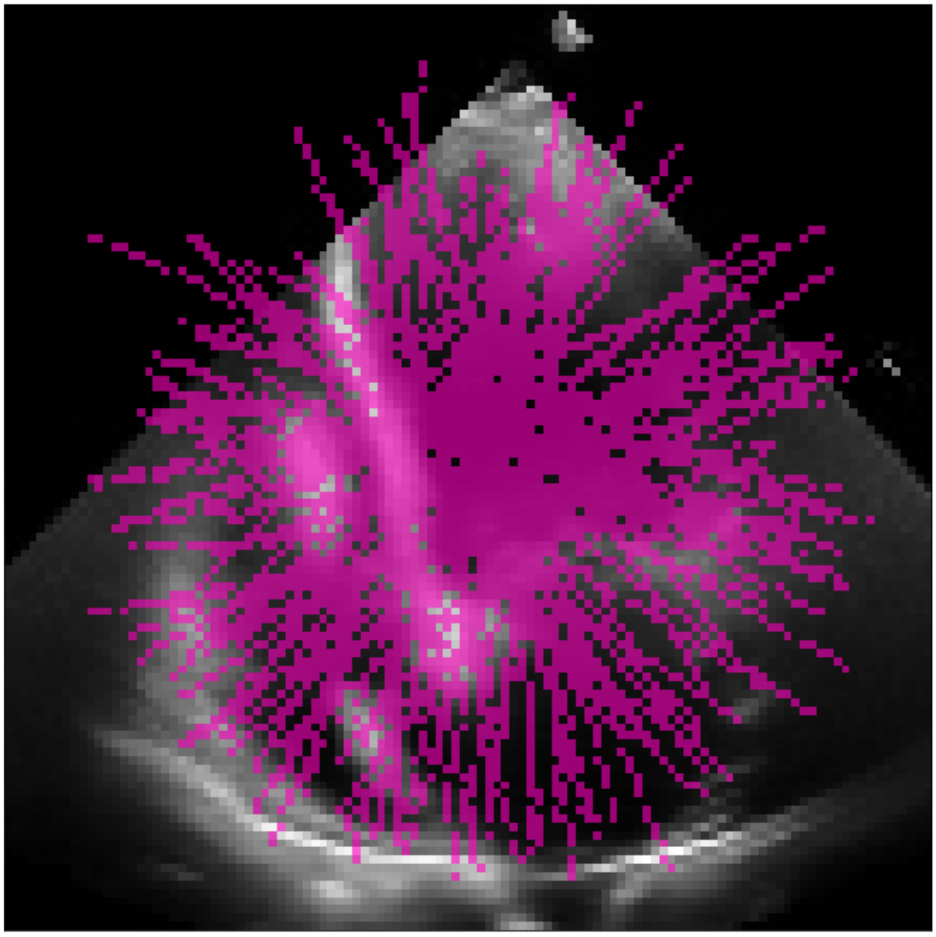

Figure 4.9: The union of 100 randomly sampled m-mode lines.

4.7.1 Agent Architecture

We keep the same base architectures as in the simple binary classification environment, but we also need to accommodate the overview image and action history array. This is done by concatenating the result of passing all three arrays through their corresponding neural networks. The synthetic m-mode and overview images are passed through the Atari DQN-paper-inspired CNN that is visualized in figure 4.3, but with the final output layer removed in order to accommodate concatenation. The action-history array is flattened before being passed into a fully connected layer with 32 outputs, followed by a ReLU activation layer. After concatenating the three results, they are passed through yet another fully connected layer of 64 outputs and a ReLU activation layer before being passed through a final fully connected layer with two outputs. The network is visualized in figure 4.10.

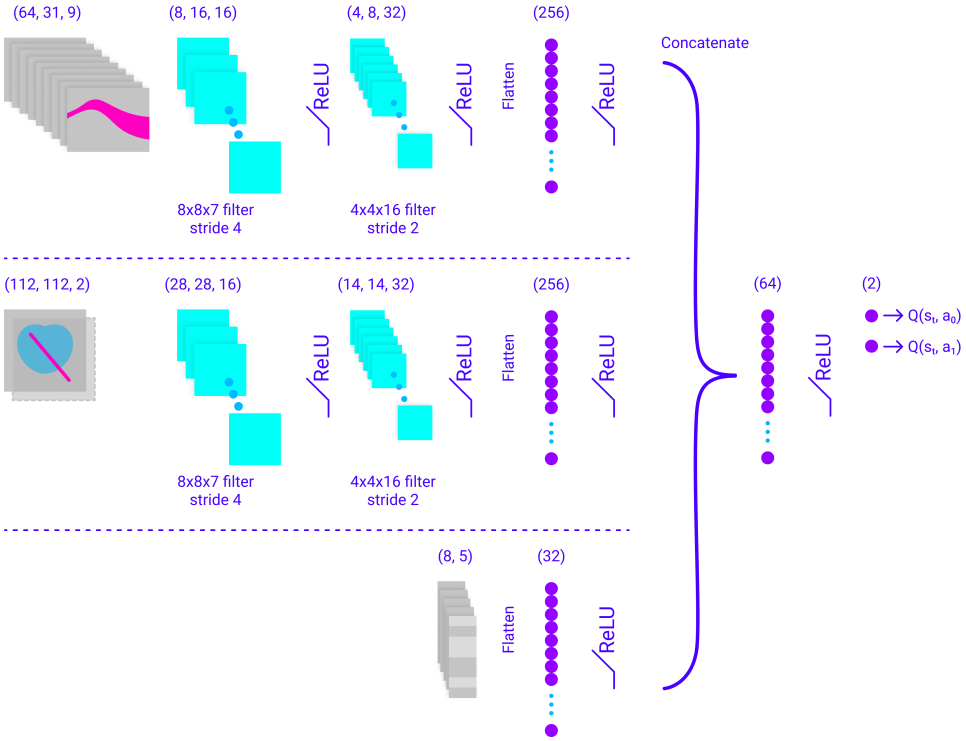

Figure 4.10: The network architecture of the m-mode agent. An observation consists of three parts. Each part is processed independently by a neural network before being concatenated and used to produce the approximated Q-values.

Chapter 5

Experiments and Results

This chapter reports the results and the impact of various hyperparameters. Section 5.1 gives an overview of the performance metrics for all the trained models in the experiments. Section 5.2 compares the generalized average absolute frame difference (GaaFD) performance for models trained using different values of the exploration hyperparameter ϵ. Section 5.3 compares the accuracy metric performance for models trained using different reward functions and values of ϵ. Section 5.4 report the learning- and loss curves of the different models. Section 5.5 dives into the Q-values produced by the different models and compare them. Section 5.6 reports how long it took to just-in-time compile and run the BCE neural network on the CPU and GPU. Finally, 5.7 reports the results of the single MMBCE experiment that was run.

5.1 Performance Metrics — An Overview

The performance metrics is reported in the following four tables. Tables 5.1, 5.2 and 5.3 presents the performance of agents traines using R_{GaaFD}, R_{simple} and $R_{proximity}$, respectively. Lastly, table 5.4 presents a comparison between the best models for each reward function.

Out of all the trained models, the model trained using R_{simple} with a value of $\epsilon = 0.5$ performed the best.

Table 5.1: Performance of agents trained using GaaFD as the reward function on the test dataset.

	$\epsilon = 0.1$	$\epsilon = 0.01$	$\epsilon = 0.0$
Best model SGD step	167 336	136 996	95 616
GaaFD	5.84	4.59	3.68
GaaFD ED	5.69	4.84	3.74
GaaFD ES	5.83	4.20	3.50
% valid aaFD	63.71%	70.33%	76.63%
aaFD	3.51	2.71	2.43
Accuracy	0.82	0.86	0.88
Accuracy diastole	0.90	0.91	0.94
Accuracy systole	0.69	0.75	0.77
Balanced accuracy	0.58	0.67	0.70

Table 5.2: Performance of agents trained using R_{simple} as the reward function on the test dataset.

	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1$
Best model SGD step	6 220	10 960	8 964
GaaFD	2.52	2.46	2.57
GaaFD ED	2.48	2.43	2.52
GaaFD ES	2.47	2.41	2.55
% valid aaFD	79.30%	80.26%	76.95%
aaFD	1.71	1.69	1.69
Accuracy	0.91	0.91	0.91
Accuracy diastole	0.93	0.93	0.93
Accuracy systole	0.87	0.88	0.88
Balanced accuracy	0.80	0.81	0.81

Table 5.3: Performance of agents trained using $R_{proximity}$ as the reward function on the test dataset.

	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1$
Best model SGD step	107 936	6 880	7 096
GaaFD	2.55	2.56	2.63
GaaFD ED	2.52	2.56	2.67
GaaFD ES	2.50	2.48	2.52
% valid aaFD	78.87%	79.40%	76.95%
aaFD	1.74	1.80	1.71
Accuracy	0.91	0.91	0.91
Accuracy diastole	0.94	0.93	0.93
Accuracy systole	0.86	0.87	0.88
Balanced accuracy	0.80	0.80	0.81

Table 5.4: Performance of the best agent for each explored reward function on the test dataset. The best agent was selected by the best GaaFD score.

	R_{GaaFD}	R_{simple}	$R_{proximity}$
ϵ	$\epsilon = 0.0$	$\epsilon = 0.5$	$\epsilon = 0.1$
Best model SGD step	95 616	**10 960**	107 936
GaaFD	3.68	**2.46**	2.55
GaaFD ED	3.74	**2.43**	2.52
GaaFD ES	3.50	**2.41**	2.50
% valid aaFD	76.63%	**80.26%**	78.87%
aaFD	2.43	**1.69**	1.74
Accuracy	0.88	**0.91**	**0.91**
Accuracy diastole	**0.94**	0.93	**0.94**
Accuracy systole	0.77	**0.88**	0.86
Balanced accuracy	0.70	**0.81**	0.80

5.2 The Impact of Epsilon on Average Absolute Frame Difference

Lower values of ϵ yield a better GaaFD score when using R_{GaaFD} as the reward function. This is best seen in figure 5.1. There is no consistent difference between GaaFD on ED- or ES-frames individually, but there is less difference in performance on ES-frames between the training split and the test split, as seen in figure 5.2. Figure 5.3 shows that lower values of ϵ also reduce the mismatch between the number of predicted versus ground truth events. This was also clearly seen in table 5.1, where a value of $\epsilon = 0.0$ predicted the correct number of events 77% of the time, while $\epsilon = 0.1$ and $\epsilon = 0.01$ yielded 64% and 70%, respectively. We also see a significant "bump" when the difference between predicted and ground truth events is two.

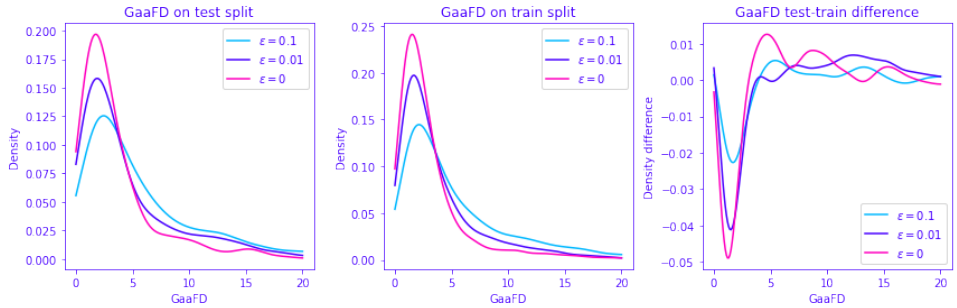

Figure 5.1: Gaussian KDE of the GaaFD-performance for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using GaaFD as the reward function. The left plot compares all three models on the test split. The middle plot compares all three models on the train split. The right plot shows the difference between the two as a means to visualize model overfitting.

The choice of ϵ did not matter as much for agents trained using R_{simple} or $R_{proximity}$, compared to those trained using R_{GaaFD}, as seen in figures 5.4 and 5.6. As with R_{GaaFD}, R_{simple} and $R_{proximity}$ showed little consistent difference in performance between gaaFD on ED- or ES-frame individually, as seen in figures 5.5 and 5.7. Figure 5.8 further shows that there is little difference between values of ϵ for predicting the correct number of events as the number of ground truth events, both for R_{simple} and $R_{proximity}$.

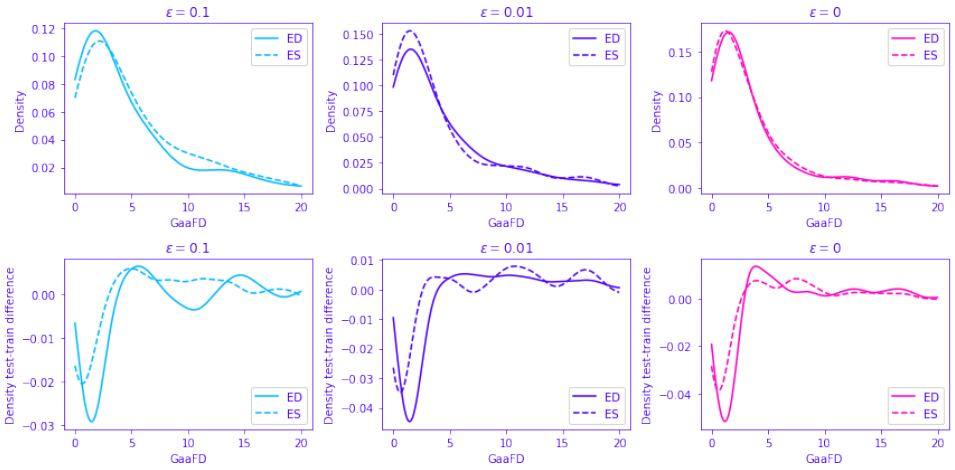

Figure 5.2: Gaussian KDE of the GaaFD-performance for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using GaaFD as the reward function, only accounting for either ED- or ES-events individually. The upper row compares the performance of ED and ES for each model. The bottom row shows the difference in GaaFD-density on the test-set versus the train-set as a means to visualize model overfitting.

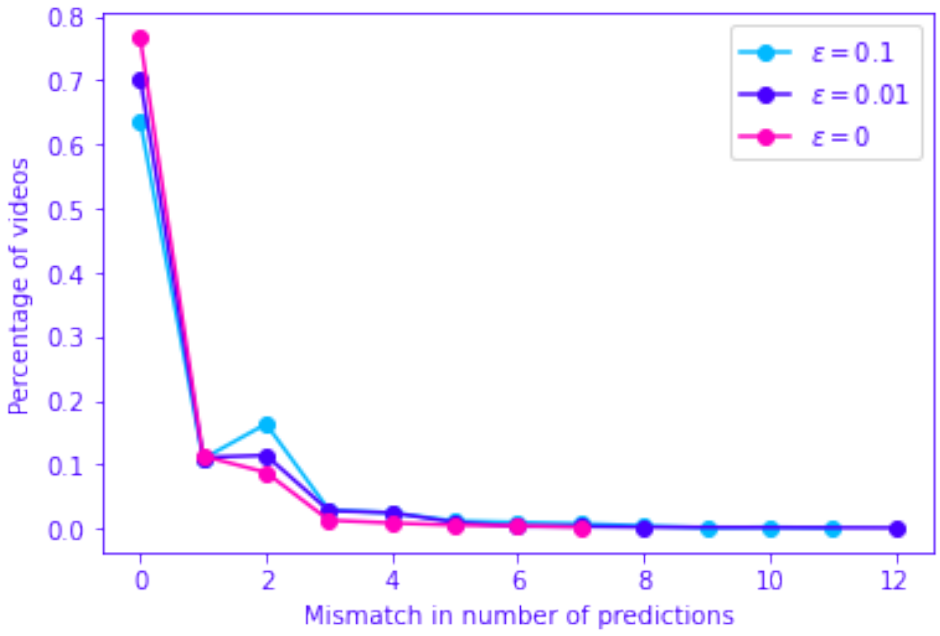

Figure 5.3: The difference between the number of predicted events and the number of ground truth events for each model when using GaaFD as the reward function. Most predictions produce the same number of predicted events as ground truth, e.g., the model with $\epsilon = 0$ produces the correct number of events 77% of the time, also shown in table 5.1.

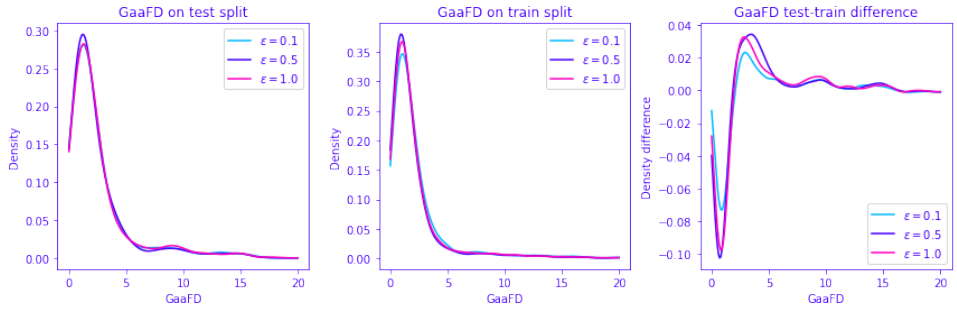

Figure 5.4: Gaussian KDE of the GaaFD-performance for each model ($\epsilon = 0.1$, $\epsilon = 0.5$, and $\epsilon = 1.0$) when using R_{simple} as the reward function. The left plot compares all three models on the test split. The middle plot compares all three models on the train split. The right plot shows the difference between the two as a means to visualize model overfitting.

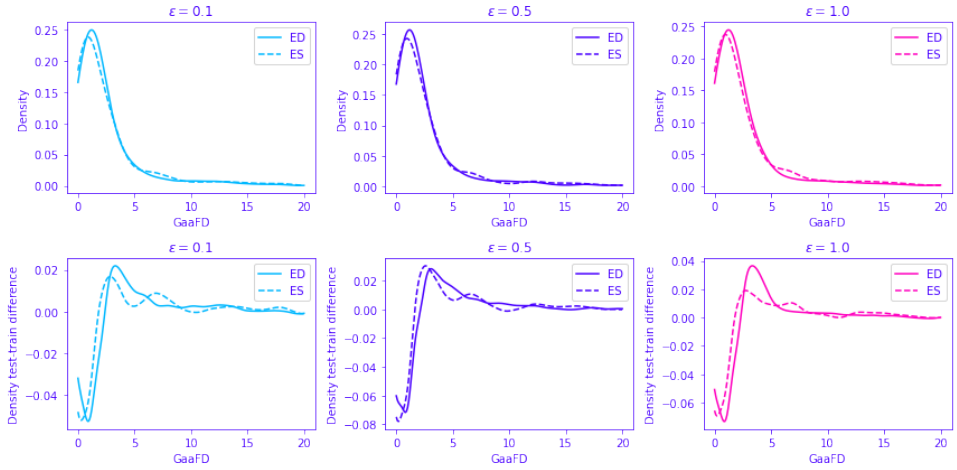

Figure 5.5: Gaussian KDE of the GaaFD-performance for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using R_{simple} as the reward function, only accounting for either ED- or ES-events individually. The upper row compares the performance of ED and ES for each model. The bottom row shows the difference in GaaFD-density on the test-set versus the train-set as a means to visualize model overfitting.

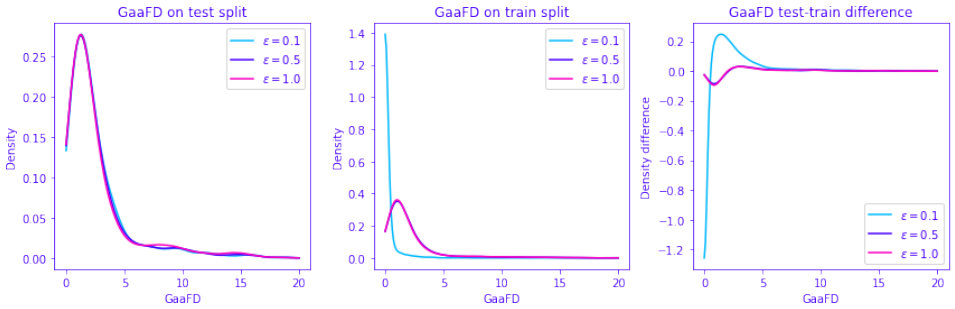

Figure 5.6: Gaussian KDE of the GaaFD-performance for each model ($\epsilon = 0.1$, $\epsilon = 0.5$, and $\epsilon = 1.0$) when using $R_{proximity}$ as the reward function. The left plot compares all three models on the test split. The middle plot compares all three models on the train split. The right plot shows the difference between the two as a means to visualize model overfitting.

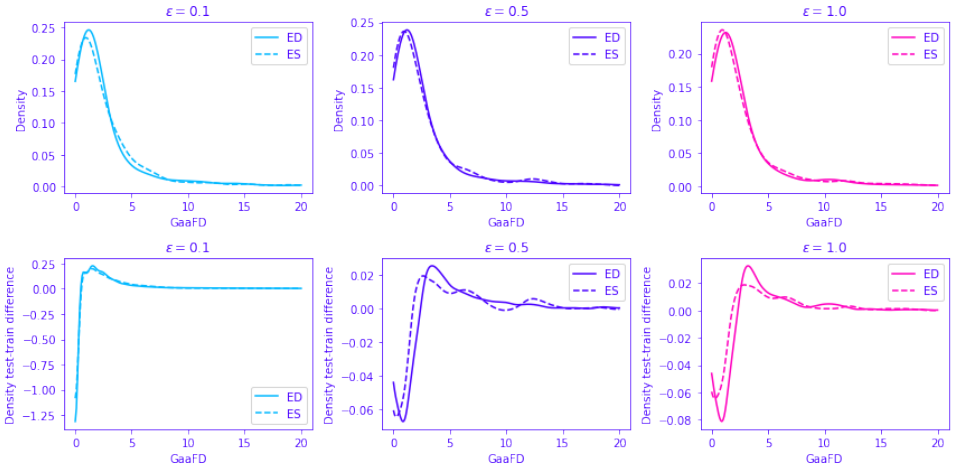

Figure 5.7: Gaussian KDE of the GaaFD-performance for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using $R_{proximity}$ as the reward function, only accounting for either ED- or ES-events individually. The upper row compares the performance of ED and ES for each model. The bottom row shows the difference in GaaFD-density on the test-set versus the train-set as a means to visualize model overfitting.

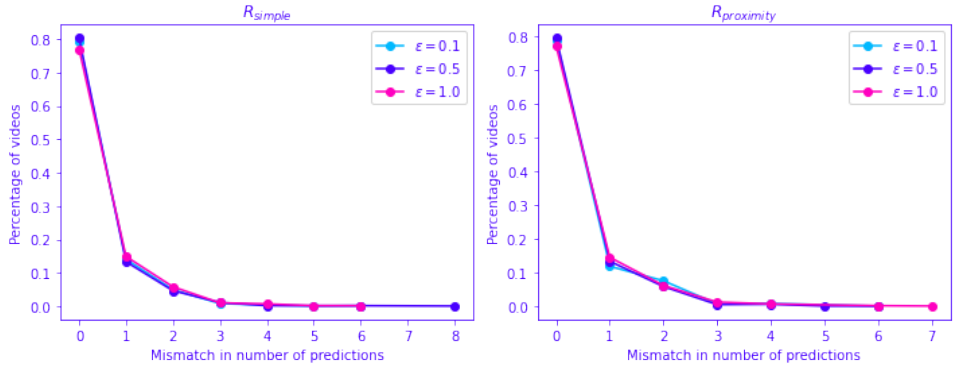

Figure 5.8: The difference between the number of predicted events and the number of ground truth events for each model when using R_{simple} (left) and $R_{proximity}$ (right) as the reward function. Most predictions produce the same number of predicted events as ground truth, e.g., the model with $\epsilon = 0.5$ and R_{simple} as the reward function produces the correct number of events 80% of the time, which can also be seen in table 5.2.

5.3 The Impact of Reward Function and Epsilon on Accuracy

Accuracy and balanced accuracy are not the main metrics that we want to optimize for, but it is helpful to report them to understand better how the model performs.

The accuracy of the agents trained using R_{GaaFD} show a similar pattern as with GaaFD, as lower values of ϵ give better scores. This can be seen in figure 5.9 as well as in table 5.1. Figure 5.9 also shows this when accounting for class imbalance through balanced accuracy. All 3 models performs perform better at classifying diastole frames compared to systole frames, as visualized in figure 5.10.

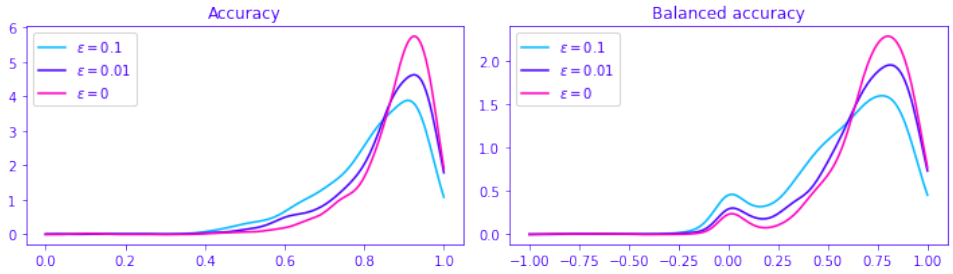

Figure 5.9: Gaussian KDE of the accuracy and balanced accuracy for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using GaaFD as the reward function. The left plot shows the accuracy. The right plot shows the balanced accuracy, which accounts more for class imbalance.

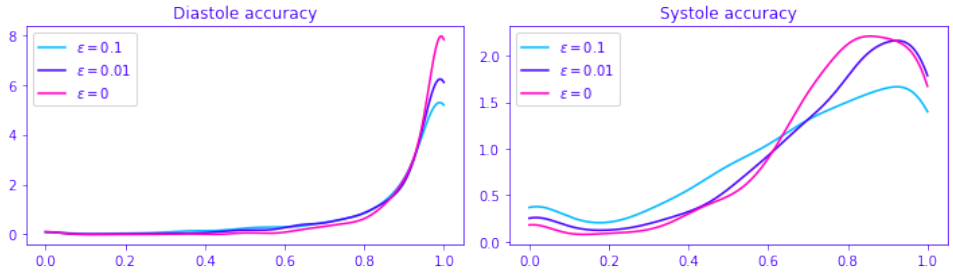

Figure 5.10: Gaussian KDE of the accuracy for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using GaaFD as the reward function for diastole or systole phase predictions individually. The left plot shows the accuracy for diastole frame predictions. The right plot shows the accuracy for systole frame predictions.

Again, the choice of ϵ did not matter as much for models trained using R_{simple} or $R_{proximity}$ with regards to accuracy and balanced accuracy, as seen in figures 5.11 and 5.13. These models also perform better at classifying diastole frames compared to systole frames.

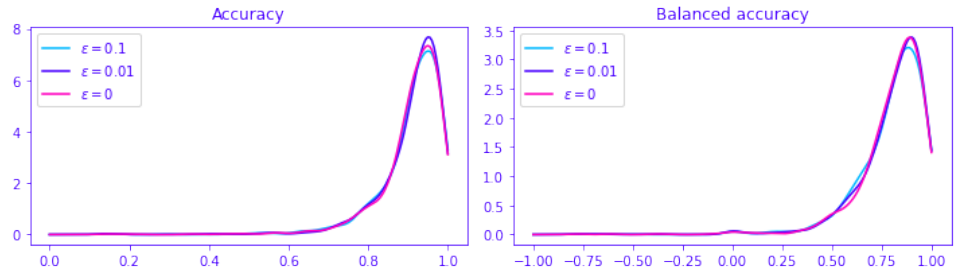

Figure 5.11: Gaussian KDE of the accuracy and balanced accuracy for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using R_{simple} as the reward function. The left plot shows the accuracy. The right plot shows the balanced accuracy, which accounts more for class imbalance.

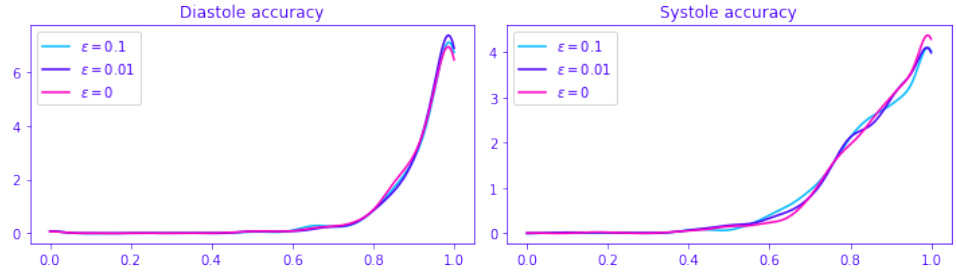

Figure 5.12: Gaussian KDE of the accuracy for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using R_{simple} as the reward function for diastole or systole phase predictions individually. The left plot shows the accuracy for diastole frame predictions. The right plot shows the accuracy for systole frame predictions.

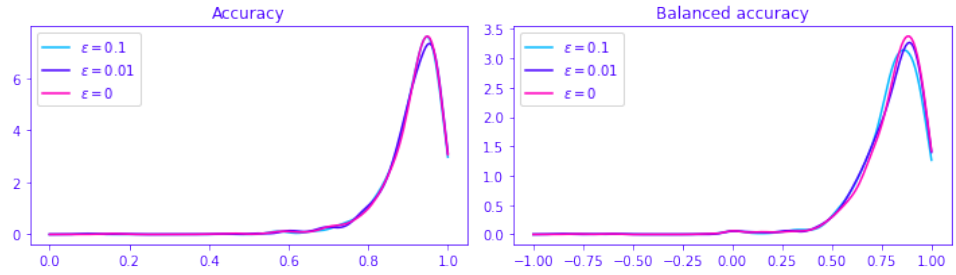

Figure 5.13: Gaussian KDE of the accuracy and balanced accuracy for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using $R_{proximity}$ as the reward function. The left plot shows the accuracy. The right plot shows the balanced accuracy, which accounts more for class imbalance.

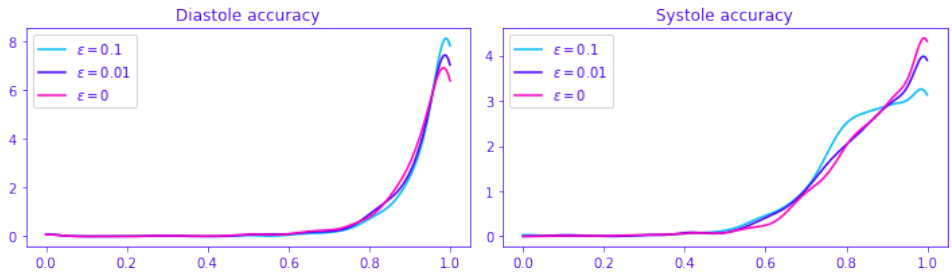

Figure 5.14: Gaussian KDE of the accuracy for each model ($\epsilon = 0.1$, $\epsilon = 0.01$, and $\epsilon = 0$) when using $R_{proximity}$ as the reward function for diastole or systole phase predictions individually. The left plot shows the accuracy for diastole frame predictions. The right plot shows the accuracy for systole frame predictions.

5.4 Learning Curves

For models trained using R_{GaaFD}, lower values of ϵ converge faster, as seen in the training curve in figure 5.15. There is no apparent degradation in performance over time that would indicate that the models start to overfit at some point, but the models trained using $\epsilon = 0.0$ or $\epsilon = 0.01$ perform slightly better on the training split than on the test split. This is not apparent for the model trained using $\epsilon = 0.1$. The loss curves for all models follow a peculiar pattern where it starts with sinking rapidly before increasing again, followed by a slight decrease until it (presumably) converges, as seen in figure 5.16. Interestingly, the model trained using a value of $\epsilon = 0.01$ reaches a higher loss than the other two at its peak following the rapid sinking at the start. Also interesting, the model trained using the highest value of ϵ has a loss curve sitting between the other two.

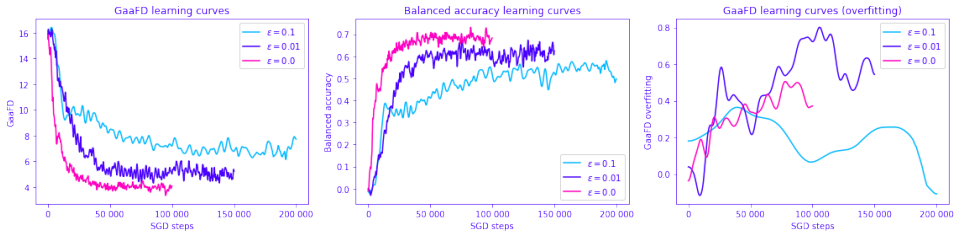

Figure 5.15: The learning curves of using GaaFD as the reward function for different values of the exploration parameter ϵ. Left: GaaFD over training time (gradient descent steps). Middle: Balanced accuracy over training time. Right: The difference in GaaFD between the validation set and the training set over training time, positive values indicating overfitting on the training set. Each point in the curve is calculated on 50 random videos in the validation (or training) set. The curves have been smoothed using a gaussian filter with a kernel standard deviation of 4 to reduce noise due to the low sample size of each data point. The overfitting (right) plot has also been smoothed using a gaussian filter with a kernel standard deviation of 50 to ensure that the overall trend is visible.

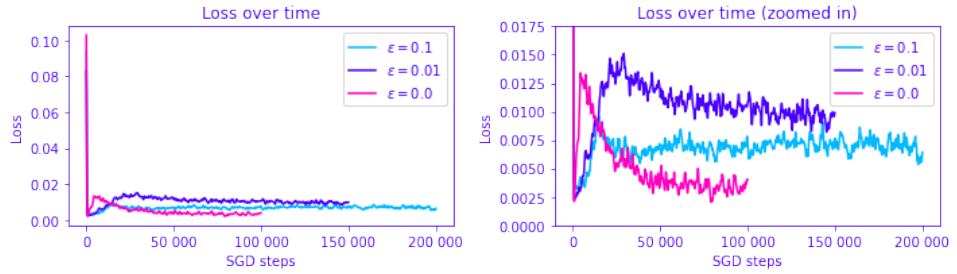

Figure 5.16: The training loss over time for different values of epsilon. The left plot shows the full y-axis, while the right plot shows the same plots but with a zoomed-in y-axis.

Lower values of ϵ yields less overfitting both for models trained using

R_{simple} and $R_{proximity}$, as seen in figures 5.17 and 5.18. In fact, the models are able to reach perfect accuracy on the training split, at the cost of worse performance on the test split, as seen in figure 5.19.

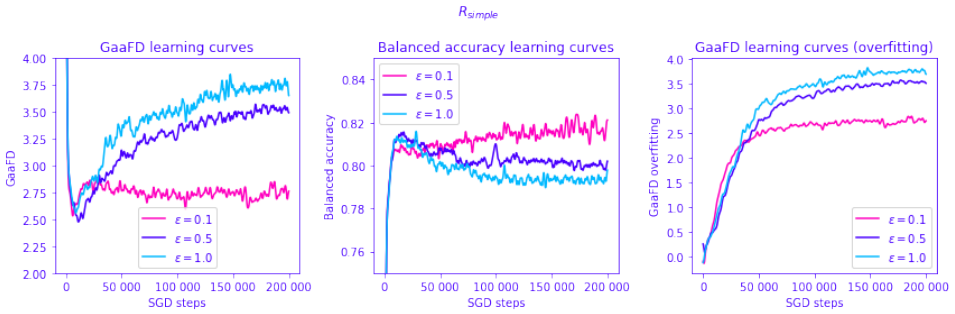

Figure 5.17: The training curves of using R_{simple} as the reward function for different values of the exploration parameter ϵ. Left: GaaFD over training time (gradient descent steps). Middle: Balanced accuracy over training time. Right: The difference in GaaFD between the validation set and the training set over training time, positive values indicating overfitting on the training set. Each point in the curve is calculated on 50 random videos in the validation (or training) set. The curves have been smoothed using a gaussian filter with a kernel standard deviation of 4 to reduce noise due to the low sample size of each data point. The overfitting (right) plot has also been smoothed using a gaussian filter with a kernel standard deviation of 50 to ensure that the overall trend is visible.

The loss curves for models trained using R_{simple} or $R_{proximity}$ do not show the same peculiar pattern as those trained using R_{GaaFD}. They all decrease quite rapidly at first before slowing down until convergence. The model trained with the lowest value of ϵ ($\epsilon = 0.1$) converges the fastest. The models trained using $R_{proximity}$ also converge faster overall.

Models trained using $R_{proximity}$ consistently perform better at the metric of GaaFD in later iterations of training. However, this is most apparent long after the models have already overfitted on the training split. The models trained using R_{simple} have indications of performing better at the metric of balanced accuracy, though the difference is most apparent in the models trained using a value of $\epsilon = 0.1$. Even though $R_{proximity}$ performs better at the important metric of GaaFD after overfitting occurs, the best model overall belongs to the model trained using R_{simple} with a value of $\epsilon = 0.5$.

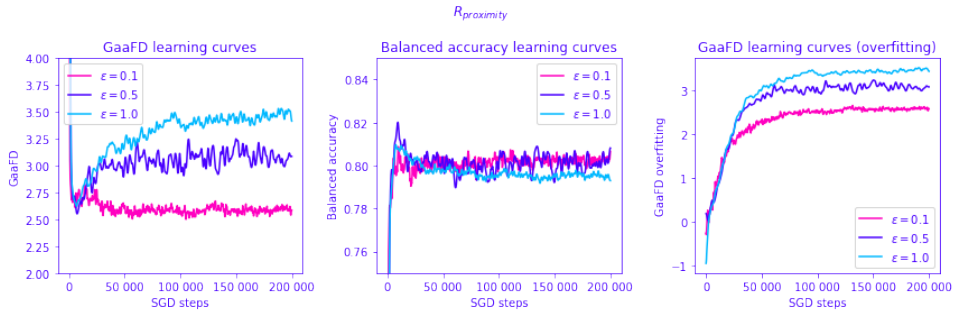

Figure 5.18: The training curves of using $R_{proximity}$ as the reward function for different values of the exploration parameter ϵ. Left: GaaFD over training time (gradient descent steps). Middle: Balanced accuracy over training time. Right: The difference in GaaFD between the validation set and the training set over training time, positive values indicating overfitting on the training set. Each point in the curve is calculated on 50 random videos in the validation (or training) set. The curves have been smoothed using a gaussian filter with a kernel standard deviation of 4 to reduce noise due to the low sample size of each data point. The overfitting (right) plot has also been smoothed using a gaussian filter with a kernel standard deviation of 50 to ensure that the overall trend is visible.

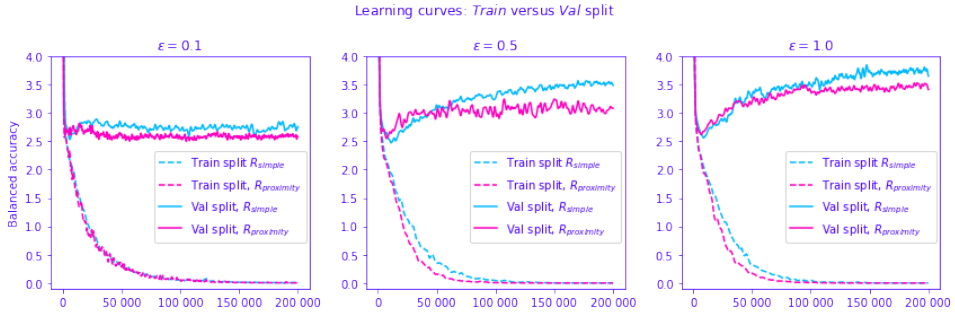

Figure 5.19: The GaaFD over training time (gradient descent steps) on the validation set (solid pink and blue line) and the training set (dashed pink and blue lines). The GaaFD on the training set reaches 0, meaning perfect predictions.

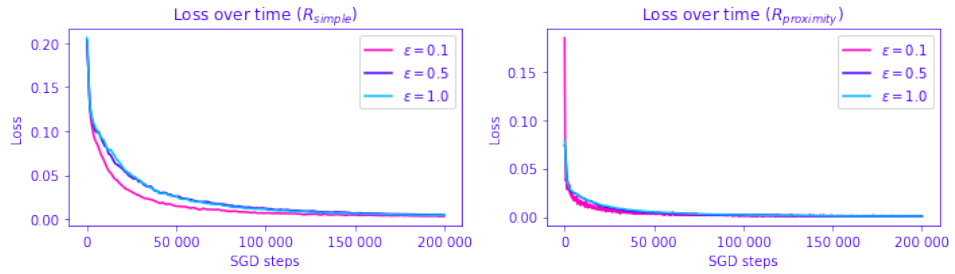

Figure 5.20: The training loss over time for different values of epsilon. Left: an agent trained using R_{simple}. Right: an agent trained using $R_{proximity}$.

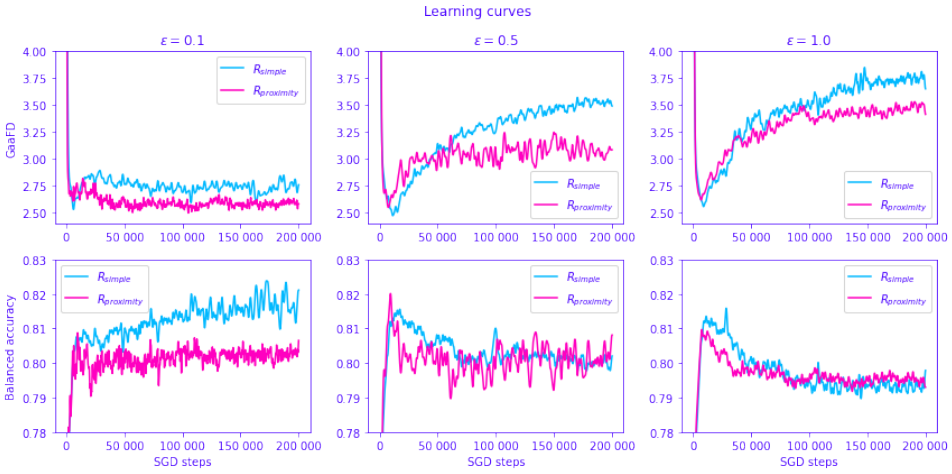

Figure 5.21: Comparison of the training curves using R_{simple} versus $R_{proximity}$ for different values of of the exploration parameter ϵ. The top row shows the GaaFD over training time (gradient descent steps). The bottom row shows the balanced accuracy over training time. Each column correspond to one of the agents, $\epsilon = 0.1$, $\epsilon = 0.5$, and $\epsilon = 1.0$, respectively.

5.5 The Impact of Reward Function and Epsilon on Q-Values

For DQN, the Q-values dictate what actions are taken. This section plots the Q-values for each reward function's best and worst-performing videos and the value of epsilon used for the reward function. These results are mainly qualitative but shed light on how the models "reason" about the frames in a video.

Figures 5.22, 5.23, and 5.24 plot the Q-values of each frame in the 3 best performing videos for that model for models trained with R_{GaaFD}, R_{simple}, and $R_{proximity}$, respectively, and for each value of ϵ. Likewise, Figures 5.22, 5.23, and 5.23 plots the same, but for the 3 worst performing videos.

The effect of a higher value of ϵ of agents trained with R_{GaaFD} is that the values of marking a frame as diastole or as systole grows closer, as seen in figure 5.22. There is also a noticeable positive spike for systole values in the middle of the diastole phase, mostly visible for lower values of ϵ.

The effect of using R_{simple} over $R_{proximity}$ seems to be that R_{simple} causes less noisy Q-values. Interestingly, the spikes in the middle of systole are also visible for agents trained with R_{simple} and $R_{proximity}$, though slightly less than for those trained with R_{GaaFD}.

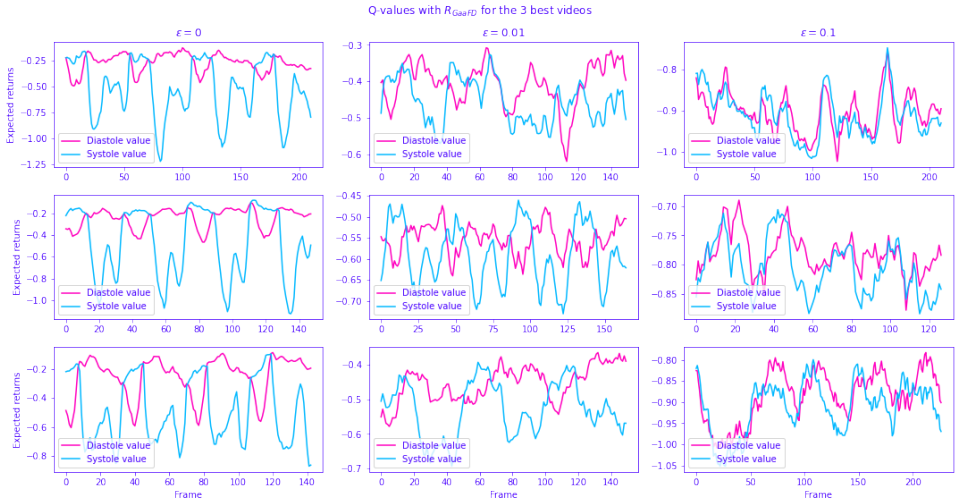

Figure 5.22: The Q-values for three of the best-predicted videos for each model trained using R_{GaaFD}. Each column is a different value of ϵ, each row is a different video. The x-axis represents time in the video.

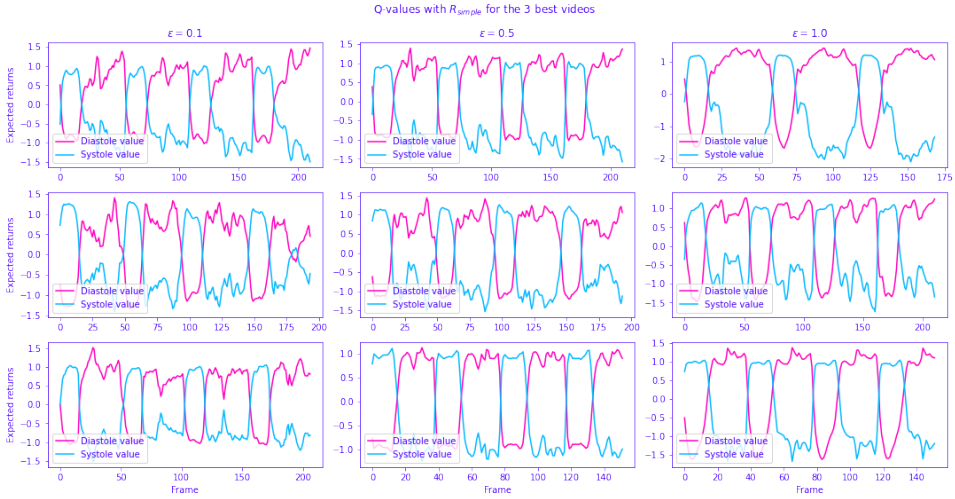

Figure 5.23: The Q-values for three of the best-predicted videos for each model trained using R_{simple}. Each column is a different value of ϵ, each row is a different video. The x-axis represents time in the video.

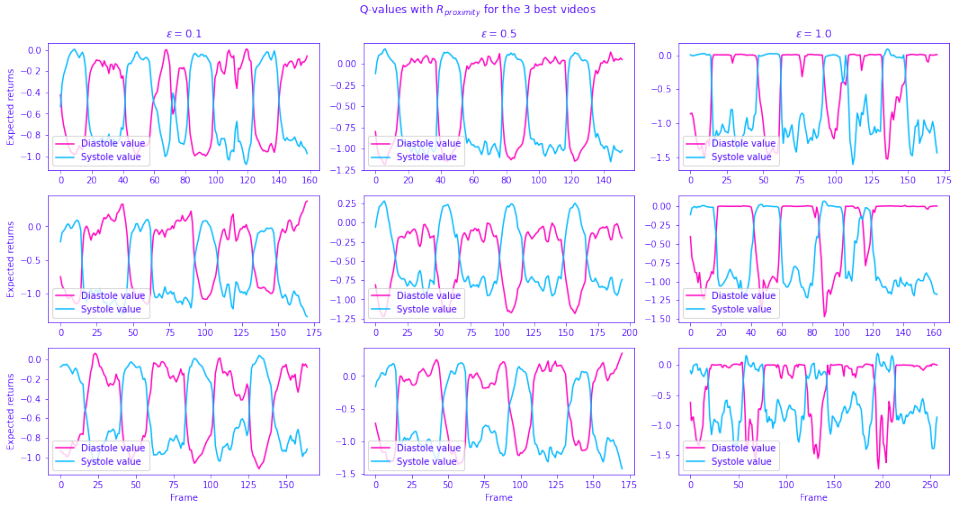

Figure 5.24: The Q-values for three of the best-predicted videos for each model trained using $R_{proximity}$. Each column is a different value of ϵ, each row is a different video. The x-axis represents time in the video.

Figure 5.25: The Q-values for three of the worst predicted videos for each model trained using R_{GaaFD}. Each column is a different value of ϵ, each row is a different video. The x-axis represents time in the video.

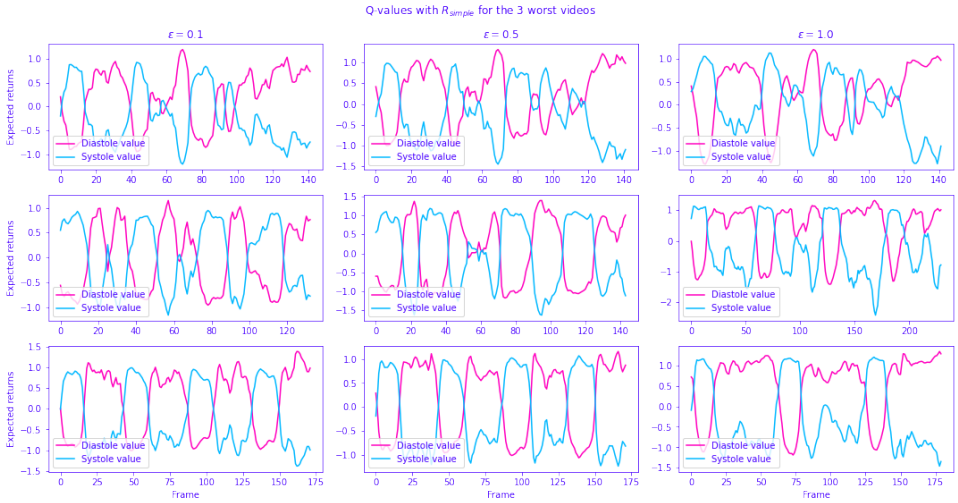

Figure 5.26: The Q-values for three of the worst predicted videos for each model trained using R_{simple}. Each column is a different value of ϵ, each row is a different video. The x-axis represents time in the video.

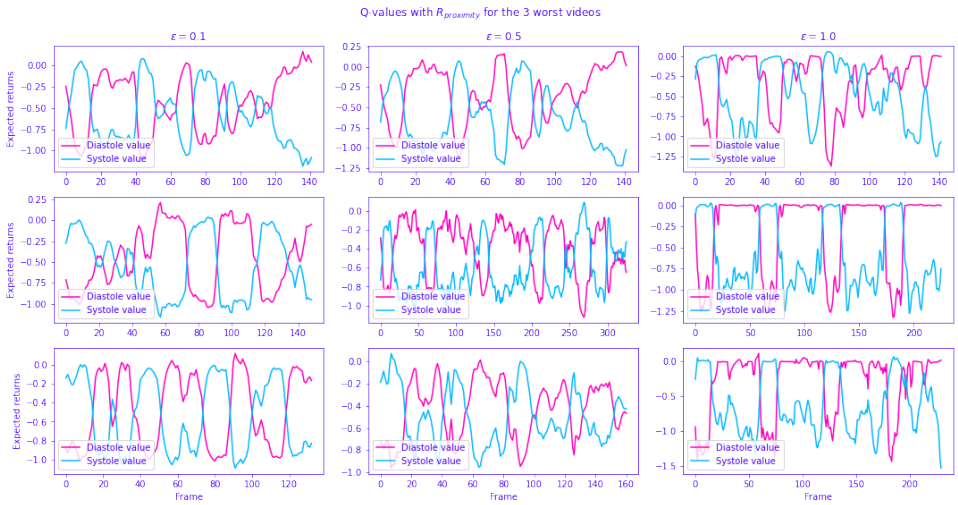

Figure 5.27: The Q-values for three of the worst predicted videos for each model trained using $R_{proximity}$. Each column is a different value of ϵ, each row is a different video. The x-axis represents time in the video.

5.6 Inference Speed

The inference time is faster on the CPU than the GPU when we include IO roundtrip time. However, ignoring IO, the network performs extremely fast on the GPU; processing a batch of 128 frames takes just a little over a millisecond, producing a framerate of over 125 000 FPS. However, this is merely considered a "fun fact." A realistic scenario would include IO roundtrip time. A single frame may be processed on the CPU in 0.80 milliseconds, meaning that it can likely be included as a step in a processing stream.

Table 5.5: The compilation time and average elapsed time over 1000 calls for the neural network, on the CPU and the GPU, with or without IO overhead.

Device	# frames	Compilation time	Average run-time
CPU	128 frames	273.95 ms	29.15 ms
	Single frame	205.93 ms	0.80 ms
GPU (including IO)	128 frames	2399.56 ms	34.43 ms
	Single frame	418.77 ms	2.88 ms
GPU (pre-placed data)	128 frames	251.10 ms	1.02 ms
	Single frame	285.56 ms	0.17 ms

5.7 M-Mode Binary Classification Environment Results

A single experiment was run using the m-mode binary classification environment (MMBCE), the result of which can be seen in table 5.6. The percentage of episodes where the agent actively explores its environment by moving the synthetic m-mode line is reported in addition to the key metrics. The agent is said to have explored if at least one of the actions in the episode moved or rotated the line. The GaaFD is also reported for episodes where the agent performs some exploration and where it performs no exploration individually.

The agent trained on the MMBCE performs worse than any agent trained on the BCE, as seen in table 5.6. It also performs very little exploration of the environment, where almost 35% of episodes contain no movement of the synthetic m-mode line at all. Figure 5.28 show the distribution of actions taken by the agent on the test split. Over 90% of actions were of marking the current frame as either diastole or systole.

Furthermore, in the episodes where the agent *did* perform any exploration, the agent performed worse than in the ones it did not move the synthetic m-mode line at all, as visualized in figure 5.29. Table 5.6 reports that the agent had an average GaaFD score of 5.47 for episodes where it performed exploration versus 3.13 for episodes where the line was still.

Moreover, the MMBCE agent is significantly slower at inference than the BCE agents, taking multiple seconds to evaluate a full video of 128

Table 5.6: Performance of agents trained on the m-mode binary classification environment.

Best model SGD step	23 080
GaaFD	4.66
GaaFD ED	4.85
GaaFD ES	4.37
% episodes with exploration	65.22%
% episodes without exploration	34.78%
GaaFD for episodes with exploration	5.47
GaaFD for episodes without exploration	3.13
% valid aaFD	59.29%
aaFD	2.22

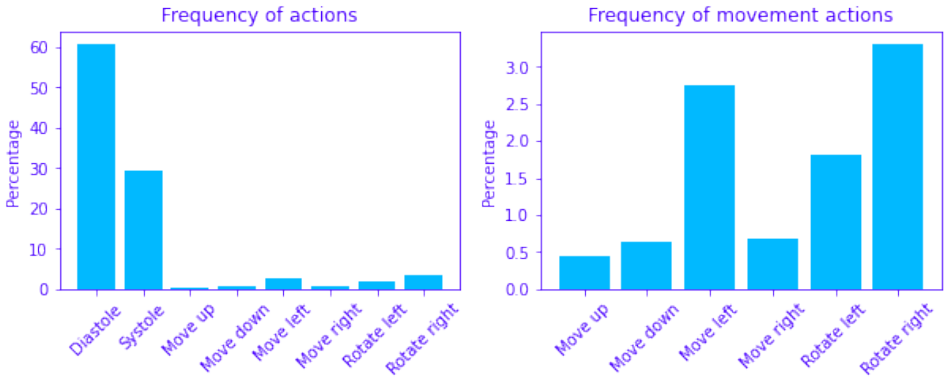

Figure 5.28: A bar chart showcasing the distribution of actions selected by the agent. The vast majority of actions are that of marking frames as diastole or systole. To the left are all actions, while to the right are only movement actions, i.e., marking a frame as diastole or systole not included.

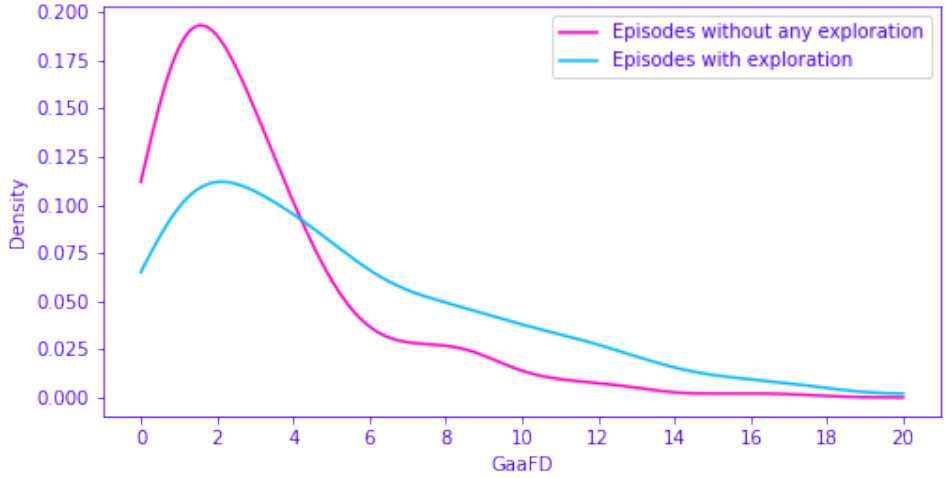

Figure 5.29: A density plot of GaaFD for episodes where the agent performed no other actions than marking frames as diastole or systole, i.e., no exploration, versus the density plot of GaaFD for episodes where the agent moved the synthetic m-mode line in any way at least once.

frames. It also takes more than three times as long when running the agent on the GPU.

Table 5.7: The average compilation and run time for predicting the phase of 128 frames in a video (including IO overhead).

Device	Compilation time	Average run time for 128 steps
CPU	389.70 ms	3169.83 ms
GPU	2384.04 ms	9738.50 ms

Chapter 6

Discussion

This chapter discusses the results, methodology, and weaknesses of the study. Section 6.1 discusses the results of the models trained using R_{GaaFD} and the reasoning behind them. Likewise, section 6.2 discusses the results of the models trained using R_{simple} or $R_{proximity}$. Section 6.3 discusses the results of agents trained one the MMBCE environment, and its shortcomings. Section 6.4 takes a critical look at the key performance metric for the task of ED-/ES-frame detection, aaFD. Section 6.5 discusses a weakness of the thesis in a lack of comparison to related work. Finally, section 6.6 gives an overview of pros and cons of using RL for tasks that can be solved using supervised learning.

6.1 On Generalized Average Absolute Frame Difference Reward Function

As seen in table 5.1, the best value of ϵ when using R_{GaaFD} was 0.0. This is likely because the learner has access to noisier signals the higher the value of ϵ. Recall that the agent only receives a reward at the very end of the episode, which on average lasts for 50 steps. Any mistake in those 50 steps will be penalized, and the agent has no way of knowing whether it was penalized for an action taken under its policy or an action taken randomly.

Further evidence of this can be found in the loss curves in figure 5.16, as generally, the models with $\epsilon \in \{0.01, 0.1\}$ have a greater loss at the end of training. A greater loss indicates that the model is more "surprised" by the data, which could be explained by the fact that when it makes a mistake through random exploration, the model will not know which action in the episode was the true culprit.

Another interesting feature of the loss curves is the valleys at the beginning of training. At the beginning of training, the model has no knowledge about the data, and any prediction will be random. As the actors learn which action to pick, the sample data distribution changes, reflecting the new policies. This in turn creates a change in loss as the learner "catches up" to the new policy. As the model approaches a reasonable estimate of the true Q-value Q^*, it will make fewer mistakes, and the loss will decrease.

Peeking inside the machinery of the DQN-agents trained using R_{GaaFD}, we see a potential cause of the performance discrepancy between the three models. For every frame, the agent predicts the future returns of marking a frame as diastole or systole. These predictions are plotted in figures 5.22 and 5.25. The model that uses $\epsilon = 0$ better differentiates the value of taking either action for a given state. The models trained with higher values of ϵ estimate that there is a smaller difference in values of marking the current frame as diastole or as systole.

The "spikes" in estimated systole in the middle of diastole returns seen in 5.22 indicate that these frames are ambiguous to the agent. What this ambiguity is is hard to tell, but it seems to be consistent. It may be due to the fact that the heart moves very little in the middle of the diastole phase. If that is the case, it may be beneficial to increase N, the number of adjacent frames, for the observations.

As is consistent with previous work on ED-/ES-frame detection, systole phase is trickier to predict than diastole phase. Not only are there naturally fewer systole examples, but the frames around ES also change very little.

In figure 5.3 we see a relative increase in videos that have a difference in the number of predicted and ground truth events equal to 2. This may be because the model sometimes predicts rogue frames with wrong labels, perhaps due to noise, which are quickly fixed in the following frames. This creates two events in rapid succession, as visualized in figure 6.1. Post-processing the predictions and removing noise will likely decrease the mismatch between the number of predicted and ground truth events.

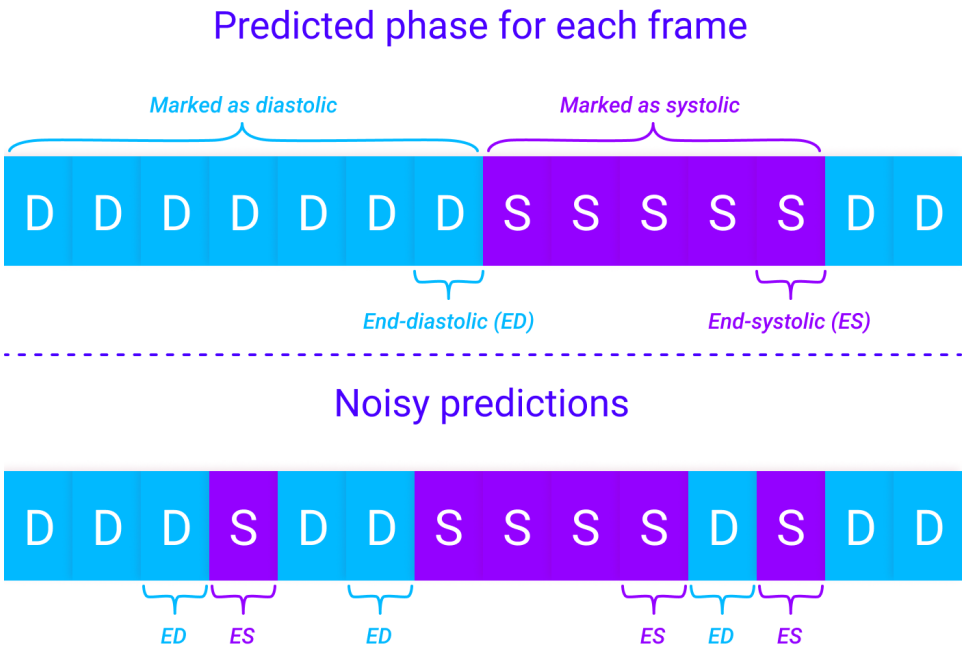

Figure 6.1: A single wrongly predicted phase that is corrected right after creates two incorrect events.

6.2 On Simple and Proximity Based Reward Functions

From the learning curves in figure 5.17 and figure 5.18 we see that the agent is able to learn to make correct predictions much faster than when using GaaFD as the reward function. This is very likely due to the GaaFD reward signal being much sparser than R_{simple} and $R_{proximity}$ — R_{simple} and $R_{proximity}$ simply provides information more efficiently, and with less noise, to the learner.

Compared to GaaFD, these models also reach their best performance much faster, though the performance on the validation set visibly degrades as the model starts to overfit. The exception to this is the model of the agent that has been trained using $\epsilon = 0.1$, i.e. the agent that most often greedily takes actions, which seemingly doesn't overfit that much.

To explain the fact that the least explorative agent is the one that overfits the least on the training set it is useful to think of exploration in this case as a means of sampling the training data. No action affects future states in this environment, so the agent is not exploring to discover long-term strategies. It instead provides the learner only with samples of which it believes are the best and this, perhaps surprisingly, seems to have a regularizing effect. However, despite the regularization, the agent is still able to achieve perfect accuracy and GaaFD on the training set, as seen in figure 5.19.

The effect of the reward function is also visible in the learning curves if we plot them for agents trained with R_{simple} and agents trained with $R_{proximity}$ together, as in figure 5.21. The blue curves are the performance of agents trained using R_{simple} and the pink curves are the performance of agents trained using $R_{proximity}$. Looking only on the right side of the plots, i.e. as the agent approaches 200 000 SGD steps, we see a clear pattern. The upper plots show GaaFD, in which lower values are better, and therefore $R_{proximity}$ is the better reward function, at least for agents with $\epsilon = 0.1$. The lower plots show balanced accuracy, in which higher values are better, and in this case R_{simple} is definitely the better reward function. So, it would seem that the reward function that was deliberately designed to be more similar to GaaFD turned out to get a better GaaFD score, even though it actually performs worse on a metric based on accuracy. This indicates that an agent trained using $R_{proximity}$ reward makes more errors compared to an agent trained using R_{simple}, but the errors are less often severe. However, this is only when looking at the right side of the plot, where the agent is already fitted to the training set. If we look at the curve at the point of the lowest GaaFD score the simple reward function R_{simple} outperforms $R_{proximity}$. This is likely due to $R_{proximity}$ being a more difficult function to estimate, as its values span multiple values, while R_{simple} can only be either 1 or -1.

Compared to the loss curves of agents trained using GaaFD as the reward, agents trained using R_{simple} and $R_{proximity}$ yield a more familiar-looking loss curve. The loss curves in figure 5.20 is seen dropping sharply in the beginning before slowly approaching some minimum. This is likely because in this case where discounting is 0.0 the value of taking an action

does not depend on the current policy, and thus the distribution of returns-estimations doesn't change as the policy changes as it did when using GaaFD as the reward function. Again, the agent with the lowest amount of exploration, plotted as the pink curve, stands out from the other models, its loss seeming to decrease faster in the beginning.

There is surprisingly little difference in the shape of the Q-values for the models trained using R_{simple} verus the models trained using $R_{proximity}$. This is further proof that $R_{proximity}$ does not add significant value to the agent, perhaps because of the increased complexity.

6.3 On M-Mode Binary Classification Environment

We were not able to create an agent that can explore the environment. There may still be advantages to this approach, however, given that we are able to make it work. The presented search can be viewed as a type of focus where the model only attend to a small part of the state-space at once. This may produce more efficient networks at the cost of possibly having to perform inference multiple times per phase prediction.

Its failure may be due to the exploration strategy. Using an ϵ -greedy policy may be too inefficient for exploring the combinatorically large search space of moving the synthetic m-mode line freely before predicting the phase.

6.4 Weaknesses of Using Average Absolute Frame Difference

aaFD is the key metric that we report but, as has been shown, it is only defined when the number of predicted events equal the number of ground truth events. To report an aaFD score for our models we opt to filter out all samples where the number of predicted events differ from the number of ground truth events. Presumably, these videos have wrongly predicted events because they are more difficult to predict on overall. Thus, filtering out these is a form of selection bias, where the aaFD is reported only on the easiest videos. The related papers don't report this problem, which, presuming this is the case, is why we must also consider the number of videos with the correct number of predicted events when comparing models.

Another weakness of aaFD is that it is sensitive to the FPS of the videos in the dataset. For videos with higher FPS, aaFD will also be proportionally higher. Unless this score is normalized, such as by dividing the score on a video by the video's FPS, it becomes hard to compare results.

6.5 Lack of Comparison Experiments

Only one other study was found to report their model's performance on the Echonet dataset [42]. This is considered a weakness of this project, and

getting access to multiple datasets early in the project should have been a priority. This makes it hard to gain precise insight into the performance of the methods versus supervised learning methods.

The authors report an average aaFD of 2.30 and 3.49 for ED and ES events in their paper. Our best model can report an average aaFD of 1.69 overall, but this is only for 80% of the videos, as 20% have an incorrect number of predicted events compared to ground truth events. The authors do not report how many, if any, of the videos had an incorrect number of predicted events.

We could have increased the number of valid results comparison by training and testing the agents on different datasets, such as MultiBeat dataset by Lane et al. [42], or the Camus echocardiographic image segmentation dataset [43].

6.6 Why Use Reinforcement Learning?

In the experiments that use the reward function R_{GaaFD}, we have seen that the RL agent can learn from a very sparse reward signal. This makes RL a very general tool that can be used when supervised learning methods are not applicable. However, there is no such thing as a free lunch, and bringing in the whole RL machinery for a classification task brings much complexity.

The methods that have been used in this thesis suffer from low data efficiency compared to a supervised learning approach. Each sample given to the learner is of only one of the phases, as only one phase is predicted at every step. We also applied additional data sampling constraints on the learner by enforcing that a data item should be sampled 0.5 on average, i.e., half of the data is discarded. For the formulations where the agent's actions do not affect future actions, such as in BCE with either phase classification reward function R_{simple} and $R_{proximity}$, the sampling should arguably be at least 1. The value of 0.5 for these experiments was due to running the experiments with R_{GaaFD} first, and it was overlooked in subsequent experiments.

The environment abstraction for sampling was also a significant performance bottleneck for the BCE environment. Every new sample had to be created by stepping through the environment on the CPU. For the BCU environment, whose next states were completely independent of the action taken by the actors, this abstraction limited us from batching the inference on the GPU. Because the environment abstraction is such an essential part of the RL ecosystem and libraries, we still opted to use it.

A synthetic m-mode version of BCE was included to give RL a fair shot at proving its usefulness. The search for a synthetic m-mode line to base future decisions on would be very challenging to solve using supervised learning, and here RL is assumed to be the best tool for the job. However, this was a tricky problem, presumably too complex for the current setup, and the benefits are not all apparent. Even if it did work, its inference would have been much slower than that of BCE since there would be no

way of batching forward passes through the neural network.

However, even though RL may not be the best tool for phase detection, it is still a promising technology, especially for problems that require exploration.

Chapter 7

Conclusion

This chapter concludes the work of this thesis. Section 7.1 attempts to answer the original research questions that was posed in section 1.2. Section 7.2 presents potential future work and research related to this thesis. Section 7.3 provides a link to the code used in this thesis.

7.1 Answers to the Research Questions

We are now ready to answer the original research questions:

Is it possible to use reinforcement learning for the task of ED-/ES-frame detection?
Yes, there are multiple ways, but some are more efficient than others.

We explored using the key metric of average absolute frame difference (aaFD) directly as the reward function, which we called R_{GaaFD}. We also explored formulating ED-/ES-frame detection as the phase classification task of marking each frame as either diastole or systole, and designing the reward function around that. The reward function R_{simple} gives a reward of 1 for correct frame phase predictions and -1 for incorrect ones. Likewise, the reward function $R_{proximity}$ gives a reward of 1 for correct predictions, but a negative reward proportional to how far the prediction was from being the correct phase. Due to reward sparsity, R_{GaaFD} performed the worst, while R_{simple} performed the best. $R_{proximity}$, despite being designed to have similar semantics to aaFD, performed worse than R_{simple}.

We also showed that it is possible to incorporate state-space search by letting the agent explore different synthetic m-mode images from the video. However, due to the additional complexity, this environment formulation made the agents perform worse than when using just a simple environment with no search.

RL is very flexible and allows us to model the problem in many different ways, both in terms of reward functions and environment dynamics.

How Does Formulating the Problem as Reinforcement Learning Affect the Performance of the Model?
We have shown that the design of the reward function matters a lot with re-

gards to how the agent learn from and represent the environment. Reward sparsity degrades performance of the agent. Designing the reward function to be more similar to the key metric makes the agent perform better on that metric in the long run. However, more complex reward functions are harder to learn, so the benefit may not be apparent until long into training, in which case overfitting may counteract any benefit.

Furthermore, we have shown that the exploration parameter ϵ is crucial for achieving a good performance when the reward signal is very sparse. For the sparse reward function R_{GaaFD}, using a value of $\epsilon = 0$, i.e. the agent follows a greedy policy, yielded the best results. For R_{simple} and $R_{proximity}$, the difference between different values of ϵ had little effect on the performance of the best model but seemed to have a regularizing effect. The model with the highest performance trained using $R_{proximity}$ as the reward function on the test split had almost perfect accuracy on the training split.

7.2 Future Work and Research

It would be interesting to dive deeper into why using lower amounts of exploration has a regularizing effect on the model. Perhaps there is something about the way that the data is sampled in RL that supervised learning can take advantage of.

In order to get a true comparison between the RL approach and a supervised learning approach for the problem of ED-/ES-frame detection we must train a neural network that is as similar as possible to the one we used as the Q-network. We have proved that it is possible to use RL for this task, but not whether there are any true benefits.

As a continuation of this work, it would also be interesting to see whether the performance improves for the RL agents when using more complex neural networks. Does the RL performance, as presented in this thesis, scale as the complexity of the Q-network increases?

The synthetic m-mode representation of the ultrasound videos are interesting because the temporal-to-spatial information ratio is very high, which is presumed to be advantageous for phase detection. Further research could be done to make MMBCE work. Policy-gradient methods may be a better approach than using a DQN because it more naturally model continuous action spaces.

7.3 Link to Code Repository

All the code used in this thesis can be found in the public Github repository https://github.com/magnusdk/edesdetect. The repository also contains videos of agents exploring the environment.

Bibliography

[1] Anas A. et al. 'Automatic Detection of the End-Diastolic and End-Systolic from 4D Echocardiographic Images'. In: *Journal of Computer Science* 11 (Jan. 2015), pp. 230–240. DOI: 10.3844/jcssp.2015.230.240.

[2] Martín Abadi et al. *TensorFlow, Large-scale machine learning on heterogeneous systems*. Nov. 2015. DOI: 10.5281/zenodo.4724125.

[3] Amir Alansary et al. 'Evaluating reinforcement learning agents for anatomical landmark detection'. en. In: *Medical Image Analysis* 53 (Apr. 2019), pp. 156–164. ISSN: 1361-8415. DOI: 10.1016/j.media.2019.02.007. URL: https://www.sciencedirect.com/science/article/pii/S1361841518306121 (visited on 11/05/2021).

[4] *Atrium (heart)*. en. Page Version ID: 1081739500. Apr. 2022. URL: https://en.wikipedia.org/w/index.php?title=Atrium_(heart)&oldid=1081739500 (visited on 15/05/2022).

[5] Igor Babuschkin et al. *The DeepMind JAX Ecosystem*. 2020. URL: http://github.com/deepmind.

[6] U. Barcaro, D. Moroni and O. Salvetti. 'Automatic computation of left ventricle ejection fraction from dynamic ultrasound images'. en. In: *Pattern Recognition and Image Analysis* 18.2 (June 2008), p. 351. ISSN: 1555-6212. DOI: 10.1134/S1054661808020247. URL: https://doi.org/10.1134/S1054661808020247 (visited on 31/05/2021).

[7] Marc G. Bellemare, Will Dabney and Rémi Munos. 'A Distributional Perspective on Reinforcement Learning'. In: *arXiv:1707.06887 [cs, stat]* (July 2017). arXiv: 1707.06887. URL: http://arxiv.org/abs/1707.06887 (visited on 18/05/2021).

[8] Marc G. Bellemare et al. 'Unifying Count-Based Exploration and Intrinsic Motivation'. In: *arXiv:1606.01868 [cs, stat]* (Nov. 2016). arXiv: 1606.01868. URL: http://arxiv.org/abs/1606.01868 (visited on 19/05/2021).

[9] James Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. 2018. URL: http://github.com/google/jax.

[10] Greg Brockman et al. 'OpenAI Gym'. In: *arXiv:1606.01540 [cs]* (June 2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540 (visited on 05/05/2022).

[11] *Cardiovascular diseases*. en. URL: https://www.who.int/westernpacific/health-topics/cardiovascular-diseases (visited on 12/05/2021).

[12] Albin Cassirer et al. 'Reverb: A Framework For Experience Replay'. In: *arXiv:2102.04736 [cs]* (Feb. 2021). arXiv: 2102.04736. URL: http://arxiv.org/abs/2102.04736 (visited on 29/03/2022).

[13] Kyunghyun Cho et al. 'Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation'. In: *arXiv:1406.1078 [cs, stat]* (Sept. 2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078 (visited on 17/05/2021).

[14] Saeed Darvishi et al. 'Measuring Left Ventricular Volumes in Two-Dimensional Echocardiography Image Sequence Using Level-set Method for Automatic Detection of End-Diastole and End-systole Frames'. In: *Research in Cardiovascular Medicine* 2.1 (Feb. 2013), pp. 39–45. ISSN: 2251-9572. DOI: 10.5812/cardiovascmed.6397. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4253755/ (visited on 31/05/2021).

[15] Fatemeh Taheri Dezaki et al. 'Deep Residual Recurrent Neural Networks for Characterisation of Cardiac Cycle Phase from Echocardiograms'. en. In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Ed. by M. Jorge Cardoso et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 100–108. ISBN: 978-3-319-67558-9. DOI: 10.1007/978-3-319-67558-9_12.

[16] Adrian Meidell Fiorito et al. 'Detection of Cardiac Events in Echocardiography Using 3D Convolutional Recurrent Neural Networks'. In: *2018 IEEE International Ultrasonics Symposium (IUS)*. ISSN: 1948-5727. Oct. 2018, pp. 1–4. DOI: 10.1109/ULTSYM.2018.8580137.

[17] Meire Fortunato et al. 'Noisy Networks for Exploration'. In: *arXiv:1706.10295 [cs, stat]* (July 2019). arXiv: 1706.10295. URL: http://arxiv.org/abs/1706.10295 (visited on 18/05/2021).

[18] Florin C. Ghesu et al. 'An Artificial Agent for Anatomical Landmark Detection in Medical Images'. en. In: *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2016*. Ed. by Sebastien Ourselin et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 229–237. ISBN: 978-3-319-46726-9. DOI: 10.1007/978-3-319-46726-9_27.

[19] Florin C. Ghesu et al. 'Robust Multi-scale Anatomical Landmark Detection in Incomplete 3D-CT Data'. en. In: *Medical Image Computing and Computer Assisted Intervention - MICCAI 2017*. Ed. by Maxime Descoteaux et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 194–202. ISBN: 978-3-319-66182-7. DOI: 10.1007/978-3-319-66182-7_23.

[20] Florin C. Ghesu et al. 'Towards intelligent robust detection of anatomical structures in incomplete volumetric data'. en. In: *Medical Image Analysis* 48 (Aug. 2018), pp. 203–213. ISSN: 1361-8415. DOI: 10.1016/j.media.2018.06.007. URL: https://www.sciencedirect.com/science/article/pii/S1361841518304092 (visited on 10/05/2021).

[21] Parisa Gifani et al. 'Automatic detection of end-diastole and end-systole from echocardiography images using manifold learning'. eng. In: *Physiological Measurement* 31.9 (Sept. 2010), pp. 1091–1103. ISSN: 1361-6579. DOI: 10.1088/0967-3334/31/9/002.

[22] Parisa Gifani et al. 'Noise reduction of echocardiography images using Isomap algorithm'. In: *2011 1st Middle East Conference on Biomedical Engineering*. ISSN: 1558-2531. Feb. 2011, pp. 150–153. DOI: 10.1109/MECBME.2011.5752087.

[23] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[24] H. P. van Hasselt (Hado) and Intelligent and autonomous systems. 'Double Q-learning'. en. In: *Advances in Neural Information Processing Systems*. The MIT Press, Dec. 2010. URL: https://ir.cwi.nl/pub/16889 (visited on 15/05/2021).

[25] Hado van Hasselt, Arthur Guez and David Silver. 'Deep Reinforcement Learning with Double Q-learning'. In: *arXiv:1509.06461 [cs]* (Dec. 2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461 (visited on 15/05/2021).

[26] *Heart*. en. Page Version ID: 1085543273. May 2022. URL: https://en.wikipedia.org/w/index.php?title=Heart&oldid=1085543273 (visited on 15/05/2022).

[27] Tom Hennigan et al. *Haiku: Sonnet for JAX*. 2020. URL: http://github.com/deepmind/dm-haiku.

[28] Matteo Hessel et al. *Optax: composable gradient transformation and optimisation, in JAX!* 2020. URL: http://github.com/deepmind/optax.

[29] Matteo Hessel et al. 'Rainbow: Combining Improvements in Deep Reinforcement Learning'. In: *arXiv:1710.02298 [cs]* (Oct. 2017). arXiv: 1710.02298. URL: http://arxiv.org/abs/1710.02298 (visited on 16/05/2021).

[30] Sepp Hochreiter and Jürgen Schmidhuber. 'Long Short-term Memory'. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[31] Matt Hoffman et al. 'Acme: A Research Framework for Distributed Reinforcement Learning'. In: *arXiv:2006.00979 [cs]* (June 2020). arXiv: 2006.00979. URL: http://arxiv.org/abs/2006.00979 (visited on 29/03/2022).

[32] Kurt Hornik, Maxwell Stinchcombe and Halbert White. 'Multilayer feedforward networks are universal approximators'. en. In: *Neural Networks* 2.5 (Jan. 1989), pp. 359–366. ISSN: 0893-6080. DOI: 10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208 (visited on 16/05/2022).

[33] Peter J. Huber. 'Robust Estimation of a Location Parameter'. In: *The Annals of Mathematical Statistics* 35.1 (1964). Publisher: Institute of Mathematical Statistics, pp. 73–101. ISSN: 0003-4851. URL: http://www.jstor.org/stable/2238020 (visited on 26/04/2022).

[34] Paul A Iaizzo. *Handbook of cardiac anatomy, physiology, and devices*. Springer Science & Business Media, 2010.

[35] Tollef Struksnes Jahren et al. 'Estimation of End-Diastole in Cardiac Spectral Doppler Using Deep Learning'. In: *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* 67.12 (Dec. 2020). Conference Name: IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, pp. 2605–2614. ISSN: 1525-8955. DOI: 10.1109/TUFFC.2020.2995118.

[36] Don H. Johnson and Dan E. Dudgeon. *Array Signal Processing: Concepts and Techniques*. Anglais. Facsimile édition. Englewood Cliffs, NJ: Prentice Hall, 1993. ISBN: 978-0-13-048513-7.

[37] Nadjia Kachenoura et al. 'Automatic detection of end systole within a sequence of left ventricular echocardiographic images using autocorrelation and mitral valve motion detection'. eng. In: *Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference* 2007 (2007), pp. 4504–4507. ISSN: 2375-7477. DOI: 10.1109/IEMBS.2007.4353340.

[38] Nitish Shirish Keskar et al. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. Tech. rep. arXiv:1609.04836. arXiv:1609.04836 [cs, math] type: article. arXiv, Feb. 2017. DOI: 10.48550/arXiv.1609.04836. URL: http://arxiv.org/abs/1609.04836 (visited on 16/05/2022).

[39] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 26/04/2022).

[40] Bin Kong et al. 'Recognizing End-Diastole and End-Systole Frames via Deep Temporal Regression Network'. en. In: *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2016*. Ed. by Sebastien Ourselin et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 264–272. ISBN: 978-3-319-46726-9. DOI: 10.1007/978-3-319-46726-9_31.

[41] Julian Krebs et al. 'Robust Non-rigid Registration Through Agent-Based Action Learning'. en. In: *Medical Image Computing and Computer Assisted Intervention - MICCAI 2017*. Ed. by Maxime Descoteaux et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 344–352. ISBN: 978-3-319-66182-7. DOI: 10.1007/978-3-319-66182-7_40.

[42] Elisabeth S. Lane et al. 'Multibeat echocardiographic phase detection using deep neural networks'. en. In: *Computers in Biology and Medicine* 133 (June 2021), p. 104373. ISSN: 0010-4825. DOI: 10.1016/j.compbiomed.2021.104373. URL: https://www.sciencedirect.com/science/article/pii/S0010482521001670 (visited on 12/05/2021).

[43] Sarah Leclerc et al. 'Deep Learning for Segmentation Using an Open Large-Scale Dataset in 2D Echocardiography'. eng. In: *IEEE transactions on medical imaging* 38.9 (Sept. 2019), pp. 2198–2210. ISSN: 1558-254X. DOI: 10.1109/TMI.2019.2900516.

[44] Rui Liao et al. 'An Artificial Agent for Robust Image Registration'. In: *arXiv:1611.10336 [cs]* (Nov. 2016). arXiv: 1611.10336. URL: http://arxiv.org/abs/1611.10336 (visited on 12/05/2021).

[45] Mada Razvan O. et al. 'How to Define End-Diastole and End-Systole?' In: *JACC: Cardiovascular Imaging* 8.2 (Feb. 2015). Publisher: American College of Cardiology Foundation, pp. 148–157. DOI: 10.1016/j.jcmg.2014.10.010. URL: https://www.jacc.org/doi/full/10.1016/j.jcmg.2014.10.010 (visited on 15/05/2021).

[46] Tanvir Mahmud, Md Awsafur Rahman and Shaikh Anowarul Fattah. 'CovXNet: A multi-dilation convolutional neural network for automatic COVID-19 and other pneumonia detection from chest X-ray images with transferable multi-receptive feature optimization'. en. In: *Computers in Biology and Medicine* 122 (July 2020), p. 103869. ISSN: 0010-4825. DOI: 10.1016/j.compbiomed.2020.103869. URL: https://www.sciencedirect.com/science/article/pii/S0010482520302250 (visited on 16/05/2022).

[47] Gabriel Maicas et al. 'Deep Reinforcement Learning for Active Breast Lesion Detection from DCE-MRI'. en. In: *Medical Image Computing and Computer Assisted Intervention - MICCAI 2017*. Ed. by Maxime Descoteaux et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 665–673. ISBN: 978-3-319-66179-7. DOI: 10.1007/978-3-319-66179-7_76.

[48] Dimitris G. Manolakis and Vinay K. Ingle. *Applied Digital Signal Processing: Theory and Practice*. Anglais. New York: Cambridge University Press, Nov. 2011. ISBN: 978-0-521-11002-0.

[49] Jamie R Mitchell and Jiun-Jr Wang. 'Expanding application of the Wiggers diagram to teach cardiovascular physiology'. In: *Advances in physiology education* 38.2 (2014). Publisher: American Physiological Society Bethesda, MD, pp. 170–175.

[50] Volodymyr Mnih et al. 'Human-level control through deep reinforcement learning'. en. In: *Nature* 518.7540 (Feb. 2015). Number: 7540 Publisher: Nature Publishing Group, pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236. URL: http://www.nature.com/articles/nature14236 (visited on 11/05/2021).

[51] Gabriel Montaldo et al. 'Coherent plane-wave compounding for very high frame rate ultrasonography and transient elastography'. In: *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* 56.3 (Mar. 2009). Conference Name: IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, pp. 489–506. ISSN: 1525-8955. DOI: 10.1109/TUFFC.2009.1067.

[52] Ify R Mordi et al. 'Efficacy of noninvasive cardiac imaging tests in diagnosis and management of stable coronary artery disease'. In: *Vascular Health and Risk Management* 13 (Nov. 2017), pp. 427–437. ISSN: 1176-6344. DOI: 10.2147/VHRM.S106838. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5701553/ (visited on 12/05/2021).

[53] David Ouyang et al. *EchoNet-Dynamic: a Large New Cardiac Motion Video Data Resource for Medical Machine Learning*. en. 2019. URL: https://www.semanticscholar.org/paper/EchoNet-Dynamic%3A-a-Large-New-Cardiac-Motion-Video-Ouyang-He/44bfcf2409c0826584c7c409b6a2fcf8c9910c88 (visited on 04/03/2022).

[54] Adam Paszke et al. 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[55] F. Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[56] Tom Schaul et al. 'Prioritized Experience Replay'. In: *arXiv:1511.05952 [cs]* (Feb. 2016). arXiv: 1511.05952. URL: http://arxiv.org/abs/1511.05952 (visited on 16/05/2021).

[57] David W Scott. *Multivariate density estimation : theory, practice, and visualization*. eng. ISBN: 0471547700 Place: New York Series: Wiley series in probability and mathematical statistics. Applied probability and statistics. 1992.

[58] David Silver et al. 'Mastering the game of Go with deep neural networks and tree search'. en. In: *Nature* 529.7587 (Jan. 2016). Number: 7587 Publisher: Nature Publishing Group, pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: http://www.nature.com/articles/nature16961 (visited on 21/05/2021).

[59] Paul Suetens. *Fundamentals of Medical Imaging*. 3rd ed. Cambridge: Cambridge University Press, 2017. ISBN: 978-1-107-15978-5. DOI: 10.1017/9781316671849. URL: https://www.cambridge.org/core/books/fundamentals-of-medical-imaging/E9D727DBE7EB6150768A74F655C07BAC (visited on 03/05/2022).

[60] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. en. Google-Books-ID: uWV0DwAAQBAJ. MIT Press, Nov. 2018. ISBN: 978-0-262-35270-3.

[61] Thomas L Szabo. *Diagnostic ultrasound imaging: inside out*. English. OCLC: 866931381. 2014. ISBN: 978-0-12-396542-4. URL: http://www.books24x7.com/marc.asp?bookid=58830 (visited on 22/11/2021).

[62] Fatemeh Taheri Dezaki et al. 'Cardiac Phase Detection in Echocardiograms With Densely Gated Recurrent Neural Networks and Global Extrema Loss'. In: *IEEE Transactions on Medical Imaging* 38.8 (Aug. 2019). Conference Name: IEEE Transactions on Medical Imaging, pp. 1821–1832. ISSN: 1558-254X. DOI: 10.1109/TMI.2018.2888807.

[63] Jimin Tan et al. *A critical look at the current train/test split in machine learning*. Tech. rep. arXiv:2106.04525. arXiv:2106.04525 [cs] type: article. arXiv, June 2021. DOI: 10.48550/arXiv.2106.04525. URL: http://arxiv.org/abs/2106.04525 (visited on 16/05/2022).

[64] Oriol Vinyals et al. 'Grandmaster level in StarCraft II using multi-agent reinforcement learning'. en. In: *Nature* 575.7782 (Nov. 2019). Number: 7782 Publisher: Nature Publishing Group, pp. 350–354. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z. URL: https://www.nature.com/articles/s41586-019-1724-z (visited on 21/05/2021).

[65] Athanasios Vlontzos et al. 'Multiple Landmark Detection using Multi-Agent Reinforcement Learning'. In: *arXiv:1907.00318 [cs]* (July 2019). arXiv: 1907.00318. URL: http://arxiv.org/abs/1907.00318 (visited on 11/05/2021).

[66] Ziyu Wang et al. 'Dueling Network Architectures for Deep Reinforcement Learning'. In: *arXiv:1511.06581 [cs]* (Apr. 2016). arXiv: 1511.06581. URL: http://arxiv.org/abs/1511.06581 (visited on 16/05/2021).

[67] *Wiggers diagram*. en. Page Version ID: 1029541282. June 2021. URL: https://en.wikipedia.org/w/index.php?title=Wiggers_diagram&oldid=1029541282 (visited on 15/05/2022).

[68] Fan Yang et al. 'Launchpad: A Programming Model for Distributed Machine Learning Research'. In: *arXiv:2106.04516 [cs]* (June 2021). arXiv: 2106.04516. URL: http://arxiv.org/abs/2106.04516 (visited on 29/03/2022).

[69] Baichuan Yuan et al. 'Machine learning for cardiac ultrasound time series data'. In: *Medical Imaging 2017: Biomedical Applications in Molecular, Structural, and Functional Imaging*. Vol. 10137. International Society for Optics and Photonics, Mar. 2017, p. 101372D. DOI: 10.1117/12.2254704. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10137/101372D/Machine-learning-for-cardiac-ultrasound-time-series-data/10.1117/12.2254704.short (visited on 31/05/2021).

[70] Matthew D. Zeiler and Rob Fergus. 'Visualizing and Understanding Convolutional Networks'. In: *arXiv:1311.2901 [cs]* (Nov. 2013). arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901 (visited on 31/05/2021).

[71] Aston Zhang et al. *Dive into Deep Learning*. 2020.

[72] Nanning Zheng and Jianru Xue. 'Manifold Learning'. en. In: *Statistical Learning and Pattern Analysis for Image and Video Processing*. Ed. by Nanning Zheng and Jianru Xue. Advances in Pattern Recognition. London: Springer, 2009, pp. 87–119. ISBN: 978-1-84882-312-9. DOI: 10.1007/978-1-84882-312-9_4. URL: https://doi.org/10.1007/978-1-84882-312-9_4 (visited on 16/05/2022).

[73] S. Kevin Zhou et al. 'Deep reinforcement learning in medical imaging: A literature review'. In: *arXiv:2103.05115 [cs, eess]* (Mar. 2021). arXiv: 2103.05115. URL: http://arxiv.org/abs/2103.05115 (visited on 10/05/2021).