UNIVERSITY
OF OSLO

Master's thesis

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

or: How I Learned to Stop Worrying and Love the Bomb

Magnus Dalen Kvalevåg

60 study points

Department of Informatics
The Faculty of Mathematics and Natural Sciences

Abstract

To be, or not to be, that is the question: Whether 'tis nobler in the mind to suffer The slings and arrows of outrageous fortune, Or to take Arms against a Sea of troubles, And by opposing end them: to die, to sleep No more; and by a sleep, to say we end The heart-ache, and the thousand natural shocks That Flesh is heir to? 'Tis a consummation Devoutly to be wished. To die, to sleep, To sleep, perchance to Dream; aye, there's the rub, For in that sleep of death, what dreams may come, When we have shuffled off this mortal coil, Must give us pause. There's the respect That makes Calamity of so long life: For who would bear the Whips and Scorns of time, The Oppressor's wrong, the proud man's Contumely, The pangs of dispised Love, the Law's delay, The insolence of Office, and the spurns That patient merit of th'unworthy takes, When he himself might his Quietus make With a bare Bodkin? Who would Fardels bear, [F: these Fardels] To grunt and sweat under a weary life, But that the dread of something after death, The undiscovered country, from whose bourn No traveller returns, puzzles the will, And makes us rather bear those ills we have, Than fly to others that we know not of? Thus conscience does make cowards of us all, And thus the native hue of Resolution Is sicklied o'er, with the pale cast of Thought, And enterprises of great pitch and moment, [F: pith] With this regard their Currents turn awry, [F: away] And lose the name of Action. Soft you now, The fair Ophelia? Nymph, in thy Orisons Be all my sins remember'd.

Contents

List of Figures

List of Tables

Preface

Part I

Introduction

Chapter 1

Introduction

1.1 Motivation

Cardiovascular disease is the number one cause of death globally, taking an estimated 17.9 million lives each year [6]. It is important to make a timely diagnosis so that patients may receive early treatment or for risk factor management. 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 not 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 cheap and 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 [31], and the images can be difficult to interpret due to ultrasound-specific artifacts. Despite this, it is still preferable in many cases because of the aforementioned reasons.

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 using ED and ES.

There are multiple ways of finding the ED and ES frames in a cardiac cycle [28]:

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.

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 [28]. Acquiring an ECG signal also requires applying electrodes to the patient, which is not ideal in emergency settings.

The other two methods, examining the aortic valve closure and finding the frame of maximum and minimum left ventricle volume, are both visual tasks and can be very time-consuming and laborious if done manually. A recent study has reported that the average time taken for manually annotating ED and ES frames from a video of 1 to 3 heartbeats is 26 seconds, with a standard deviation of ± 11 seconds [26]. 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. For these reasons automating ED/ES frame detection is desirable because it reduces time and creates 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, most recent methods revolve around the use of 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 has 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 [34] or becoming among the 0.2% best players in the world in the video game Starcraft II [37]. However, RL can do more than just play games, and many medical imaging applications also show promising potential [44].

1.2 Goal and Research Question

The goal of this Master's project is to explore the use of RL for automatically detecting the ED and ES frames from an ultrasound video. From a healthcare perspective it is interesting 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. 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 Limitations of the Work

1.4 Thesis Structure

What are in each chapter…

Chapter 2

Background

2.1 The Cardiac Cycle

(Taken from Wikipedia, are there any good citations I could use, or is it even needed for something as fundamental as this?)

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

The heart consists of two halves, the left heart and the right heart. The left heart pumps newly oxinated 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, is where the blood first enters the heart, and the lower chambers, the ventricles is where the blood exits the heart. Each chamber also have valves which are opened and closed during a cardiac cycle to help keep the blood flowing in one direction.

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 coming 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. As the ventricle pressure decreases and the pressure in the aorta increases, the valve going out of the ventricle is closed. Blood flows into the atria before the cycle starts over.

- Phases of the heart
  - Wiggers diagram
- Definition of ED/ES
  - Using this, clinicians can make measuremetns ....

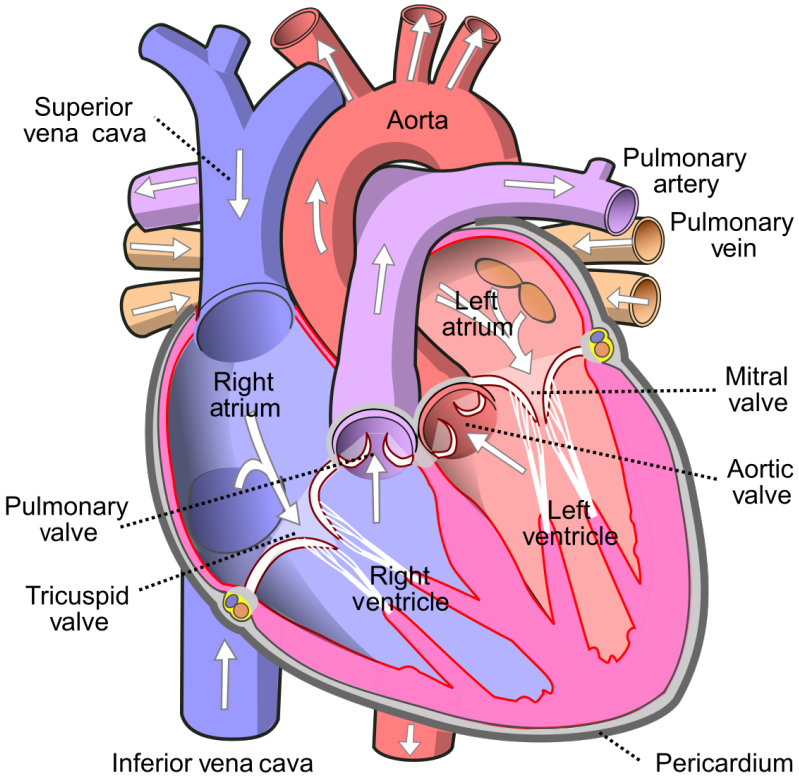

Figure 2.1: An illustration of the heart. The heart has two sides, each side having two chambers. Source: https://en.wikipedia.org/wiki/Atrium_(heart)

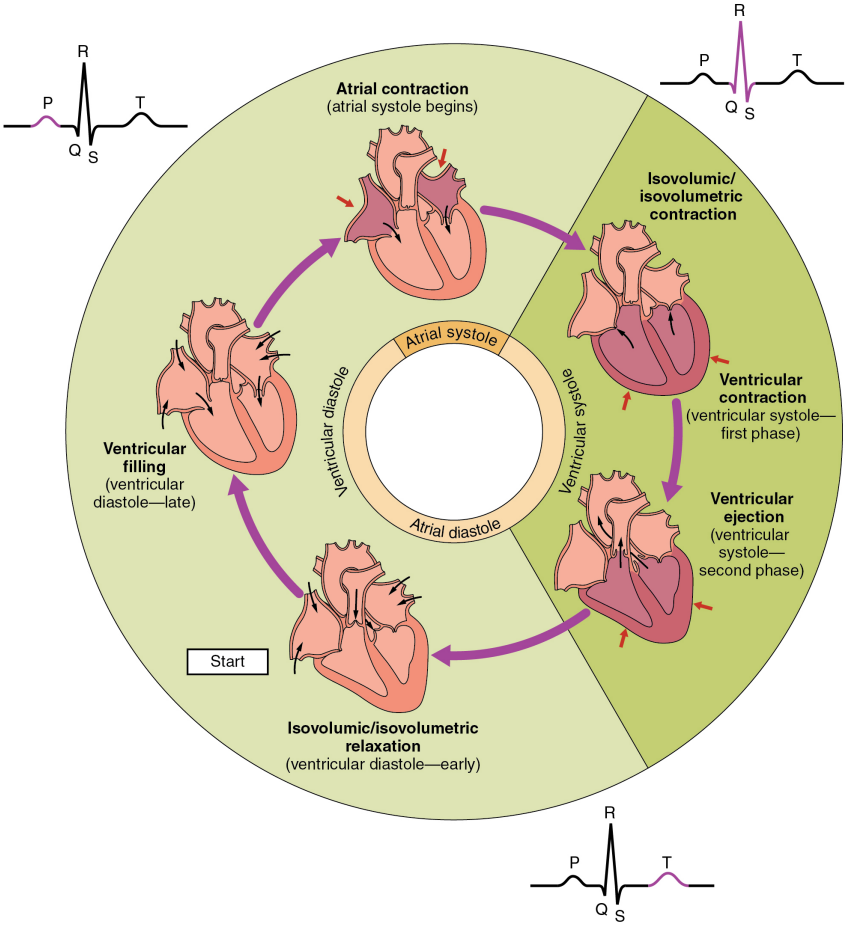

Figure 2.2: The cardiac cycle illustrated with the direction of blood flow and pressure from and into the atria and ventricles. Source: https://en.wikipedia.org/wiki/Heart

2.2 Echocardiography

Seeing is the act of sensing photons with our eyes. We can see the color of a curtain because photons that hit it and are reflected back into our eyes. A heart, being inside a body and all, can not be seen because light does not penetrate that deep, and thus is not reflected back into our eyes. To see the heart we need a signal that can penetrate the body just enough to reach the heart, but not so much as to completely penetrate the heart without sending back a reflection. As luck would have it — sound is such a signal.

2.2.1 What is Sound?

What we as humans perceive as sound are simply vibrations of the particles that surrounds us. When particles are disturbed, such as what happens to the air particles when we clap our hands together, they interact by pushing into eachother. As the atoms and molecules that make up the air bump into eachother, they also repell eachother, creating an increase in pressure and causing a chain reaction where the pertubation moves from particle to particle. This is called wave propagation. Sound is simply waves of pressure propagating through a medium.

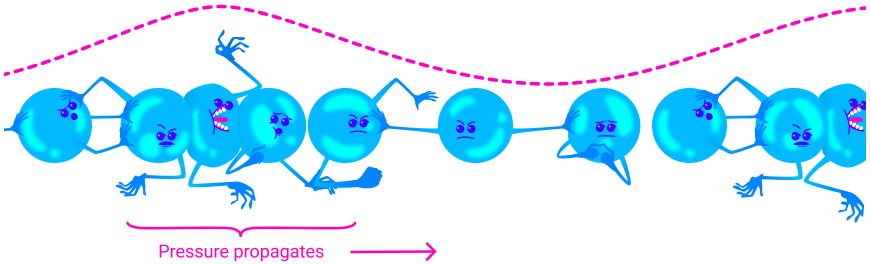

Figure 2.3: A pressure wave moves through a medium by pushing particles in a medium close together. The particles pushes back as the pressure increases, making the pressure field move further on. Warning: this image is just a representation of how particles interact — real particles don't look like this.

Attributes of a Sine Wave

A basic wave has three important 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. Our ears have evolved to sense frequency and amplitude, where frequency determines the pitch of a sound and amplitude determines the loudness. Phase can not be sensed by human ears on its own, but can affect the sound in relation with other sound waves.

A basic wave means a sine wave in this context. Every sound can be represented as a sum of sine waves, and every sound has a unique frequency spectrum. Finding the frequency spectrum is the same as decomposing a sound into its sine waves. We can also take the frequency spectrum and convert it back to its original sound. These operations

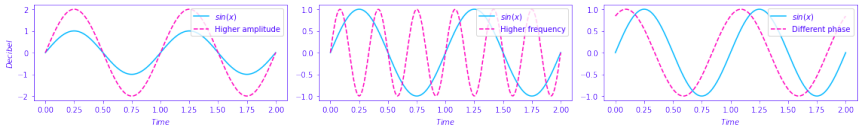

Figure 2.4: The left-most plot shows two basic waves where one has twice the amplitude. The middle plot shows two basic waves where one has a higher frequency. The right-most plot shows two basic waves that have different phases.

are called the Fourier Transform and the inverse Fourier Transform, respectively. As seen in 2.5, the frequency spectrum after adding two sine waves together is fairly simple as well, but real world sounds often have much more complex frequency spectrums, as many more sine waves are needed to represent it. When a piano and a clarinet plays the same note, what we are really saying is that the frequencies with the highest amplitudes are generally the same for both sounds. Musicians speak of overtones — it's the overtones that are different for different instruments playing the same notes. What they are referring to are the additional frequencies that can be seen in the frequency spectrum.

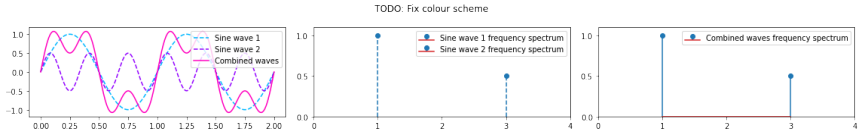

Figure 2.5: Adding two sounds together means that their frequency spectrums are also added together.

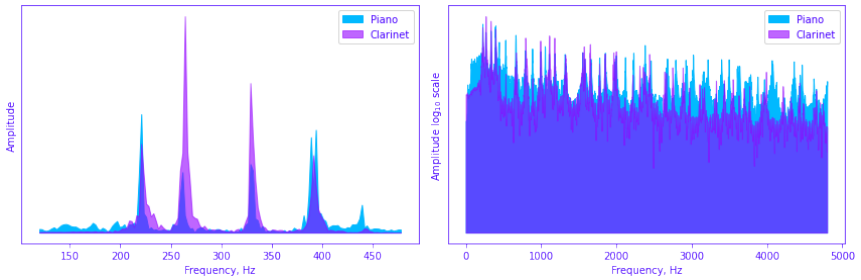

Figure 2.6: It's the overtones that makes two instruments sound different, even while they are playing the same notes. To the left is the frequency spectrum of a piano and a clarinet from 150 to 450 hertz. To the right is the same frequency spectrum from 0 to 5000 hertz, in $\log_{10}$ scale. Both instruments are playing the Am7 chord which consists of four notes. You can see the notes clearly in the left image, all having relatively high amplitudes for both instruments.

Attributes of the Medium

The other important thing about sound is the medium in which it travels through. Medium properties such as speed of sound, density, attenuation and non-linearity affect how the wave propagates through it. Speed of sound is how fast a wave propagates through the medium. Because the frequency will stay the same, if the speed of sound is lower then the wavelength will be smaller. Density is how tightly backed the particles are in the medium when at rest. Attenuation is a fancy word for absorption, how much energy the wave loses as it propagates through the medium. Non-linearity is the property where the speed of sound at a point depends on the pressure at that point. For example, in water, waves propagate faster the higher the pressure — pressure for example caused by the wave itself.

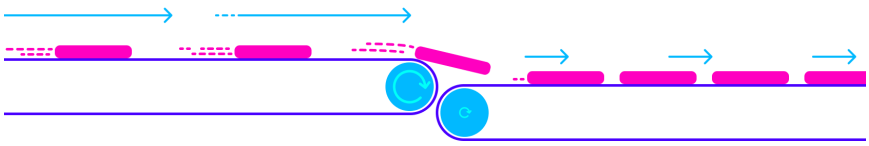

Figure 2.7: Even though the rate of packages per second stays the same, the distance between each package decreases when arriving on a slower conveyor belt. This is analogous to a sound wave propagating through a medium where the speed of sound changes. Even though the frequency is the same, the wavelength (the length between each top) decreases when it encounters a lower speed of sound.

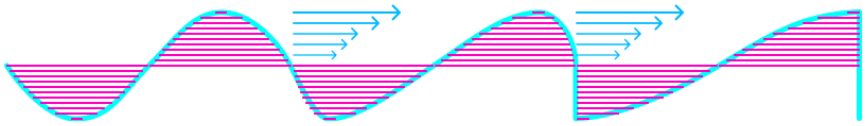

Figure 2.8: In a medium with nonlinearity the higher-pressure parts of a wave propagates faster than lower-pressure parts. Over time, the higher-pressure parts will "catch up" to the lower-pressure parts, and what started as a sine wave will start to resemble a sawtooth wave.

An important concept is "acoustic impedance" which is a measure of how much resistance the wave encounters while propagating through the medium, and is a function of the speed of sound and density. When a wave goes from one medium and into another medium that has a different acoustic impedance a part of the energy is reflected back, the amplitude being reduced for both resulting waves. So when one hears a sound being reflected back 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, while ρ and c are the density and speed of sound of the medium, respectively. Equation 2.2 is the reflection factor and determines how much of the energy is reflected back, where Z_1 is the acoustic impedance of the original medium and Z_2 is the acoustic

impedance of the second medium. When the Z_1 and Z_2 are equal, no sound is reflected back, which is what we expect — after all we usually don't hear an echo while speaking when there is only air in front of us. However, when there is a difference it doesn't matter which medium has the highest or lowest acoustic impedance — the same amount of energy is reflected either way. The only thing that changes is the sign og the reflection factor, but the magnitude of the wave stays the same whether $Z_1 > Z_2$ or $Z_1 < Z_2$. This means that the amount of echo would be the same if you were talking in the second medium, into the first one, or the other way around.

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

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

2.2.2 Creating Images From Sound

```
THE FOLLOWING ARE NOTES FOR THIS SECTION

Delay signal
one receiver only knows distance, not actual position.
multiple receivers, distance to object varies
summed together to make an image
more advanced techniques are not covered in this thesis.

To increase image quality and resolution multiple transmits can be made.
Common is to focus the sound wave in a direction, creating sector scan.
- Can take a long time because we have to wait for one transmit do finish
  before sending another one.
This is called B-mode imaging

Can also send a focused beam in just one line to get very high temporal
  resolution image.
This is called M-mode imaging

The heart is in the rib cage so it is hidden behind bones which reflect a
  lot of energy
There are some standard views, usually between the rib cage bones, but also
  down throat and from within
2-chamber, 4-chamber, etc...

Weakness of ultrasound
- Shadowing, phantoms(?), attenuation, bones in the way of the heart, etc...
```

How can we use the physical properties of acoustic waves to our advantage? The body consists of tissues of varying acoustic impedance which causes sound waves to be reflected when it hits them. Using this, we can send out a sound wave and listen for the reflections. By measuring how

much time it took from the sound signal was sent out until its reflection is received back, and given that we know the approximate speed of sound, we can calculate the distance that the sound wave travelled until it hit the reflector. See equation 2.3. Likewise, if we want to create a full image from the received sound signals, where we know which positions to image beforehand, we can lookup the received signal based on the distance to a given position. If we want to image position $(3, 4)$, and assuming that the sound was sent from position $(0, 0)$, we know that the distance would have had to travel 5 units. By using equation 2.4 we can find which part of the received signal corresponds to that position.

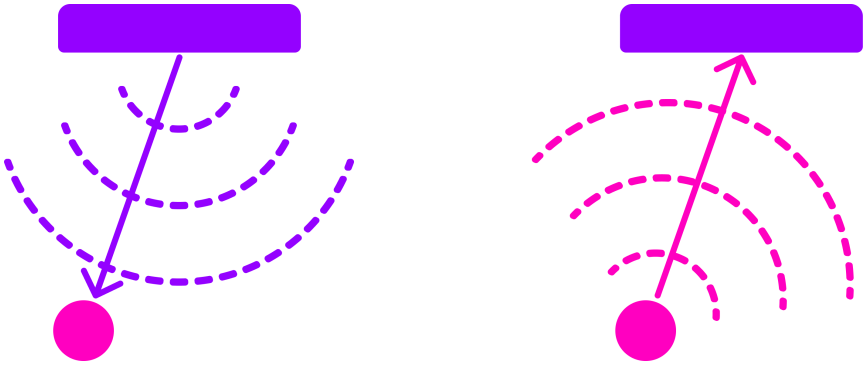

Figure 2.9: By measuring the time between sending a signal and receiving it back from a reflector we can approximate how far away the reflector is — given that we know the approximate speed of sound.

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

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

An ultrasound transducer consists of many different receivers [1] and thus we can repeat the process of delay the signal for each receiver depending on the distance from the point we want to image to the position of the receiver. If we sum all the resulting images we get a final image of different reflectors in the are. This algorithm is called Delay-And-Sum.

Limitations

- attentuaion, resolution, speckles, shadowing, side-lobes

2.3 Deep Learning

2.3.1 Gradient Descent

Based on gradient descent.

[1] An ultrasound transducer also consists of many different individual transmitter elements, but that is irrelevant for now.

- What is gradient descent... How can we build models...
- Models and Loss

Try to build models that makes the problem easy to learn. inductive bias

2.3.2 Deep Neural Networks

Activation functions Fully connected layers Convolutional layers Batch normalization (++ more used in Mobilenet?)

2.3.3 Optimization Process

- SGD
- Optimizers like ADAM
- Overfitting and regularizers.

2.3.4 Supervised and Unsupervised Learning

Basically two families of loss functions

2.4 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 can reason about future decisions. An RL agent can make a decision now that has no immediate benefit, but that will lead to a better result in the future.

At the core of RL are Markov Decision Processes (MDP) [35], 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 amount of 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 st and returns the probability of taking an action at: $\pi(a|s) \in [0, 1]$.

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:

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

where T marks the timestep where the episode ends. However, some tasks are not episodic, which means they can, in theory, run forever. For this reason, we apply discounting to the return, giving greater weight to more immediate rewards and less weight to rewards in the far future:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = r_t + \gamma G_{t+1}$$

where γ is the discounting factor. Discounting ensures that the return, which we are trying to maximize, can not be infinite, even when, in theory, the agent could go on forever.

To select a good next action, the policy needs to know the value of states and actions. 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. 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 is one of the possibly many policies that yield the maximum amount of total reward if followed.

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:

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

$(r_{t+1} + \gamma V(s_{t+1}))$ 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 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.

To be able to use $V(s)$ for making a decision, the agent needs knowledge about the transition function. This is because it needs to know what the

next state will be in order to select the best action to take. $Q(s, a)$ does not need knowledge about the transition function because it learns the value of taking an action in a state directly. TD can be modified to use the state-action value function instead of the value function, in which case it is called Q-learning:

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

Here the target, the 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.

In TD-learning, as the agent explores the environment and encounters new states, it has to store those states and their associated values. 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 a number of $\|S\| \times \|A\|$ entries. That is fine when the state space and the action space are small but become 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 [44]. However, there exist methods that can scale RL to handle very large or continuous state spaces.

2.4.1 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[30]. 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 [30]. 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. In practice, the vast majority of pixel permutations are not used, so we could ignore them, but the number of possible states would still be too high to explore exhaustively.

The values of even such a large state space can be represented in much less data without losing much relevant information. This can be done through function approximation [35], 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 allows the agent to generalize state value or state-action value functions to new not-before-seen states.

A lot of today's research into RL goes into scaling it up to a larger state space. 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 is able to learn a parameterized policy directly, without consulting a value function [35].

Only action-value methods are covered in this essay, but policy gradient methods will be considered for the final thesis.

Deep Q-Network

The modified Q-learning algorithm was termed Deep Q-Network [30] (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 [27]. 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, but that 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 2.10.

Using function approximation does have its problems. Naively training the network by inputting state and returns pairs as they are generated by the agent can result in the algorithm becoming unstable. There is a strong correlation between consecutive samples, which leads to variance in the network updates. 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 y_j , possibly leading to oscillations or divergence of the policy. These problems are mitigated by using experience replay and by using a separate network for generating the targets y_j 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 y_j in the Q-learning update adds a delay between the time an update to Q is made and the time it affects the targets y_j, making the algorithm more stable and reducing the chance of oscillations or divergence.

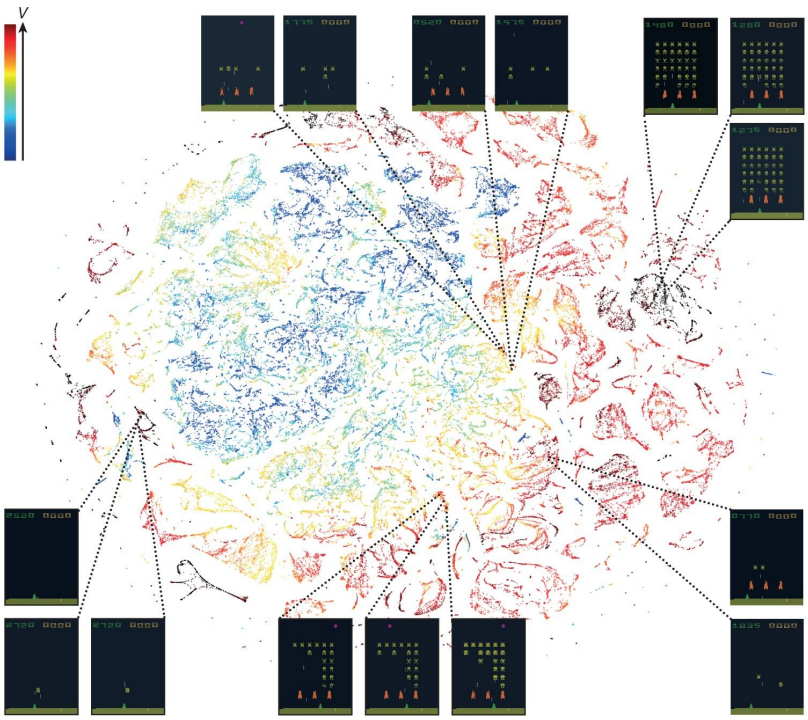

Figure 2.10: From [30]: "Two-dimensional t-SNE embedding of the representations in the last hidden layer assigned by DQN to game states experienced while playing Space Invaders. The plot was generated by letting the DQN agent play for 2 h of real game time and running the t-SNE algorithm on the last hidden layer representations assigned by DQN to each experienced game state. The points are coloured according to the state values (V, maximum expected reward of a state) predicted by DQN for the corresponding game states (ranging from dark red (highest V) to dark blue (lowest V)). The screenshots corresponding to a selected number of points are shown. The DQN agent predicts high state values for both full (top right screenshots) and nearly complete screens (bottom left screenshots) because it has learned that completing a screen leads to a new screen full of enemy ships. Partially completed screens (bottom screenshots) are assigned lower state values because less immediate reward is available. The screens shown on the bottom right and top left and middle are less perceptually similar than the other examples but are still mapped to nearby representations and similar values because the orange bunkers do not carry great significance near the end of a level. With permission from Square Enix Limited."

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 approximation for the maximum expected action value[17]. 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:

$$r_{t+1} + \gamma Q^A(s_{t+1}, argmax_a Q^B(s_{t+1}, a)) \tag{2.5}$$

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, then 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[18].

Prioritized Replay

Using experience replay, an agent isn't 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[33].

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 de-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 visited again soon is low. To overcome this challenge, a stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling is introduced.

Another problem with prioritized experience replay is that DQN optimizes for minimizing the expected TD-error squared, with respect to the network parameters θ, assuming that the samples in the replay buffer corresponds 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 training, the importance sampling is gradually added towards

the end of training, 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[39]. The combination of the state-value estimate and the advantage yields the Q values:

$$Q(s,a) = V(s) + A(s,a) \tag{2.6}$$

But 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 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.

$$Q(s,a) = V(s) + A(s,a) - \frac{\sum_a A(s,a)}{N_{actions}} \tag{2.7}$$

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

Multi-Step Learning

We look only one step ahead when constructing the target in the Q-learning update, but this isn't a requirement. We could extend it to look N steps ahead if we wanted to, in which it is called N-step learning, or multi-step learning[35].

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:

$$\sum_{k=0}^{N-1} \gamma^k r_{t+k+1} + \gamma^N max_a(Q(s_{t+N}, a)) \tag{2.8}$$

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

$$r_{t+1} + \gamma max_a Q(s_{t+1}, a) = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n max_a(Q(s_{t+n}, a)), \text{ iff n=1} \tag{2.9}$$

The best choice of N usually lies somewhere between 1 and the length of an episode. This is because bootstrapping works best if it is over a length of time in which a significant and recognizable state change has occurred. Another intuition for why it is better is that when we look further ahead into the future we depend less on our own 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[4]. 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 return will seem to take on different values over time. Lastly, because we are approximating the Q-function, approximation errors will make the returns seem stochastic.

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 become insufficient for exploration[5]. ϵ -greedy explores 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[11]. This does not only enable it to change the amount of exploration itself, alleviating the need for hyper parameter 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[19] algorithm combines six such extensions:

1. Double DQN[18]
2. Prioritized replay[33]
3. Dual DQN[39]
4. Multi-step learning[35]

5. Distributional RL[4]

6. Noisy DQN[11]

The authors are able to 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.

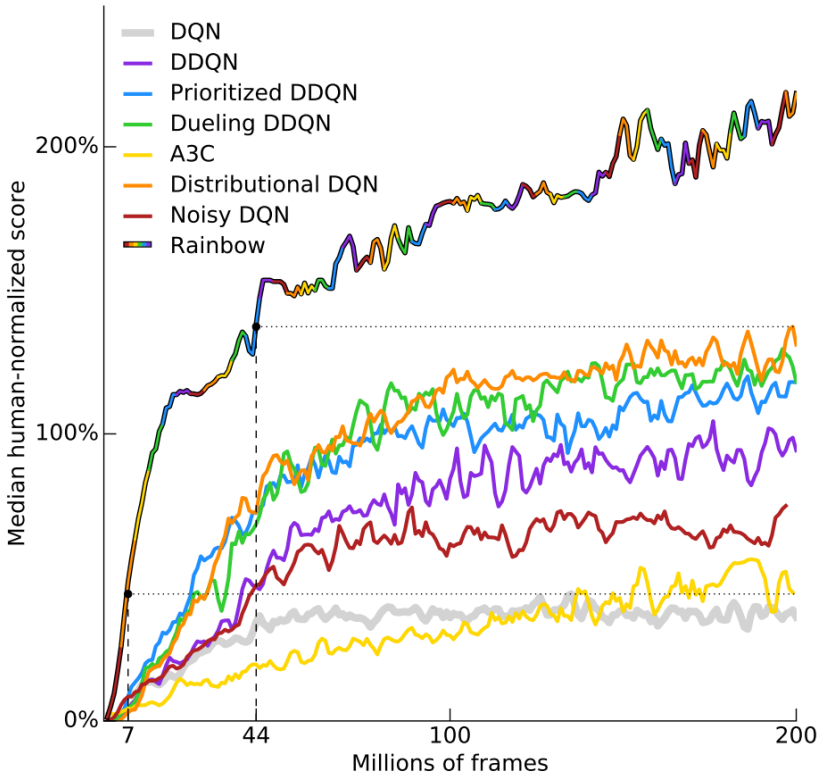

Figure 2.11: From [19], figure 1: Median human-normalized performance across 57 Atari games. We compare our integrated agent (rainbowcolored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final performance. Curves are smoothed with a moving average over 5 points.

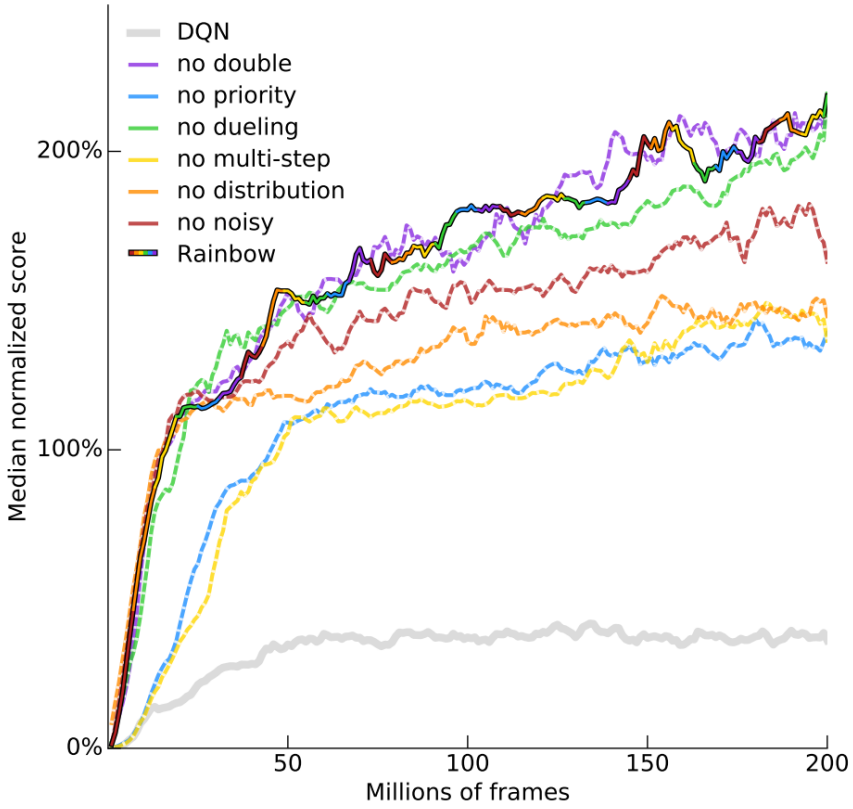

Figure 2.12: From [19], figure 3: Median human-normalized performance across 57 Atari games, as a function of time. We compare our integrated agent (rainbow-colored) to DQN (gray) and to six different ablations (dashed lines). Curves are smoothed with a moving average over 5 points.

2.5 Related Work

2.5.1 ED-/ES-Detection

TODO: This is copied from the essay.

- ☐ Read through to confirm that it still makes sense
- ☐ Re-add references
- ☐ Double-check for plagiarization

One early attempt for detecting the ED and ES frames took advantage of the rapid mitral valve opening during early diastole [23]. 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. The authors introduced another method in the same paper that took advantage of the left ventricle deformation during the cardiac cycle. With this method, ES was defined as the frame which had the lowest correlation with the ED frame. Because of little movement around systole, the correlation curve would flatten out, making the predictions more uncertain. The best results were achieved when using a combination of both methods. A small time window was selected around ES using the correlation method, and the mean intensity variation method was used to determine the final ES frame prediction.

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

It has become more common to apply end-to-end Machine Learning (ML) for fully automating tasks like this in recent times. ML is the study and development of algorithms that can learn from experience. If given enough examples, ML can approximate any mapping between input and output data [43]. ML is generally divided into three categories:

1. Supervised learning, where the algorithm learns a mapping between input and ground truth labels.
2. Unsupervised learning, where the algorithm learns to recognize patterns in the input data without any explicit ground truth labels.
3. Reinforcement learning, where the algorithm learns a strategy for solving a sequential task, given a reward signal.

Gifani et al. (2010) employed manifold learning, an unsupervised learning algorithm that is used to map high-dimensional data onto a lower-dimensional manifold. Manifold learning tries to ensure that points that are similar in the high-dimensional space are projected close together in

the low-dimensional space. 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 [15]. 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., frames that lay in dense regions, are considered to be 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 [16], 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 [41]. 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.

TODO: Re-add image from essay! Caption: Comparison between NMF, LLE and ISOMAP results for all 99 cases in the apical 4 view, taken from [41].

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 [3] [8] [1]. However, these methods are prone to significant errors due to noise inherent in cardiac ultrasound or discontinuous edges.

The most successful approaches to the task of ED and ES frame detection so far have been to use supervised learning methods. A 2D video consists of a sequence of 2D images and thus has two spatial dimensions and one temporal dimension. Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) are both supervised learning models that can extract spatial and temporal features, respectively. A basic CNN consists of one or more convolutional layers that each consists of a set of filters. The filters act as pattern-matchers and are applied to every part of the input image, and each subsequent convolutional layer can capture more high-level features of the image. A basic RNN consists of one or more processing units that are repeatedly applied to the items in the input sequence and can build up a memory of previous items.

Due to the increase in computing power in the form of GPUs, it is possible to train CNNs with many convolutional layers, or RNNs with stacked processing units, creating deep networks that can learn increasingly complex image features. This architecture gives rise to the term "Deep Learning" and is what has made some supervised learning methods so successful.

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 [24]. The combined network was trained on cardiac MRI data, and it used a Zeiler-Fergus model [42] for the CNN and an LSTM [20] for the RNN. The problem was treated as a regression problem for a monotonically decreasing function during diastole and monotonically increasing 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 maximum peaks and minimum valleys of the DL model's output. This approach was later improved by swapping out the CNN with a ResNet [9], and then again by swapping it out for a DenseNet [36], 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 [10]. 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 [22]. 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 back 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 [26]. The authors explore multiple different 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 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. RL has produced even better results than supervised learning methods for many tasks, including medical imaging tasks [44]. The next section will introduce RL, and it is followed by some examples of how it 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 [44]. 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. Out of these four elements, the reward function is usually the most difficult to get right.

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.

The goal of landmark detection is 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 at the new current state, then 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. [14] [13] [12]. Compared to other state-of-the-art methods at the time, which performed an exhaustive search across the input image, an RL agent only have 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 [14].

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 [13] and [12], 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 [30]. 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 [2]. The formulation of the problem into state, actions, and reward function remains mostly the same as in [13] and [12], 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 [38]. 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 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 [27] [25], object/lesion localization and detection [29], and more [44].

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 [27]. 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 [25]. 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 is the application of object localization to medical imaging. The goal of the algorithm is to find a bounding box around certain objects in the image. For lesion detection

in 3D breast scans, Maicas et al. (2017) used a parameter space consisting of translation and scale [29]. 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 very simple 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.

Part II

The Project

Chapter 3

Datasets

Overview of the chapter. Short description of the different datasets used.

3.1 Echonet-Dynamic Dataset

The Echonet-Dynamic Dataset[32] 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 their 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 is a list of line segments that together define the volume of the heart at a given frame. For each video there are two sets of line segments, 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 biggest 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 ratio of ESV and EDV
ESV	End systolic volume calculated by method of discs
EDV	End diastolic volume calculated by method of discs
FrameHeight	Video Height
FrameWidth	Video Width
FPS	Frames Per Second
NumberOfFrames	Number of Frames in 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 left most point of line segment
Y1	Y coordinate of left most point of line segment
X2	X coordinate of right most point of line segment
Y2	Y coordinate of right most point of line segment
Frame	Frame number of video on which tracing was performed

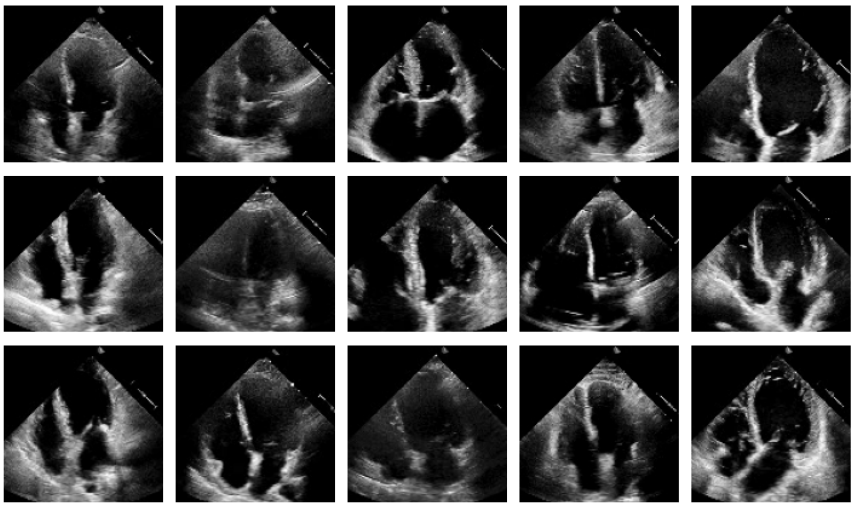

Figure 3.1: The first frames of 15 randomly sampled videos from the Echonet dataset.

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 which ends each phase. When we only have access to the end-frames of each phase, on of them 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.

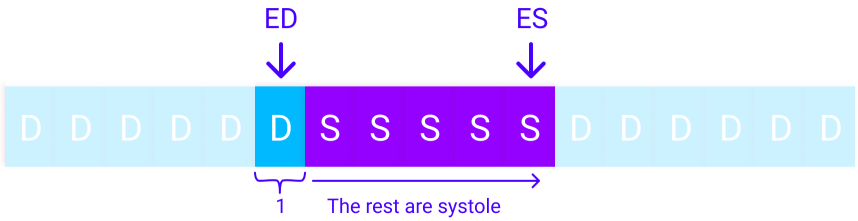

Figure 3.2: Class imbalance: only the first frame is marked with the phase of the first end-event (either ED or ES), all others are marked with the other phase.

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 another the image gradually gets more different, until it starts going back towards the starting position again. For example, the next frame with the biggest difference from the ED frame is likely to be close to the ES frame.

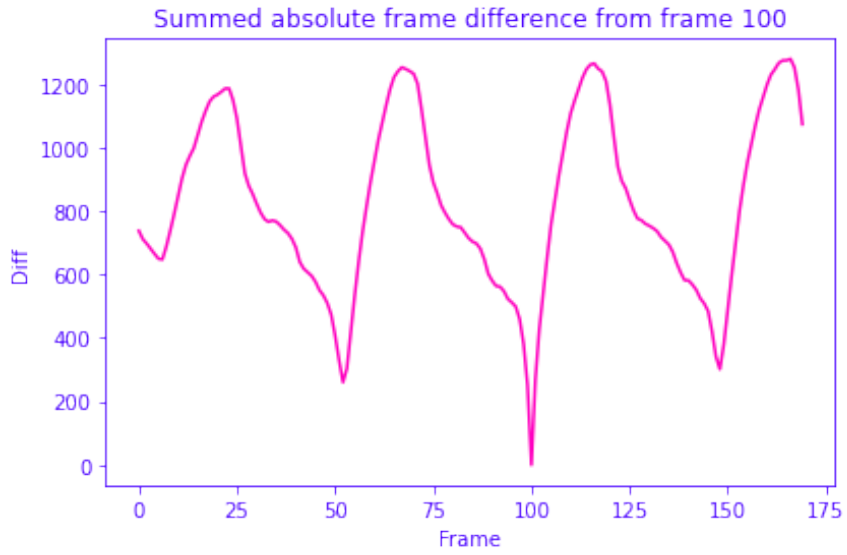

Figure 3.3: The absolute frame difference of all frames in a video compared to frame 100. Notice that the difference for frame 100 is 0 as it (of course) equals itself.

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 then we could label all previous frames up until the next peak difference as diastole. Likewise, if the final event is ES then we could label all following frames up 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.

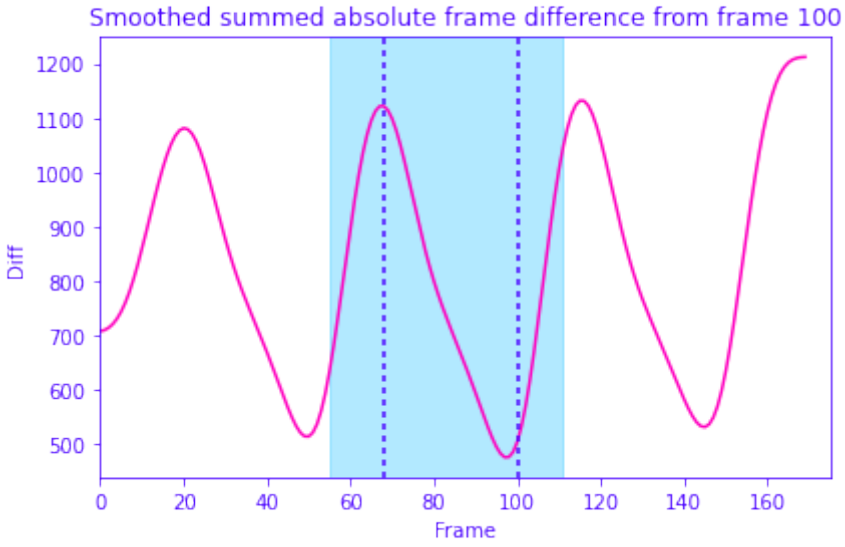

Figure 3.4: The same summed absolute frame difference plot as in figure 3.3, but smoothed using a gaussian blur with a kernel standard deviation of 5. The dashed lines represent phase-end events and the frames in the light blue area are frames with labeled phase. Notice how the labeled frames area only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

3.1.3 Removing Invalid Videos

An assumption made when labeling the frames 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 goes 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 then it suggests that the video contains more than one heartbeat and thus can not be properly labeled. There are 19 of such videos in total, and these are filtered out.

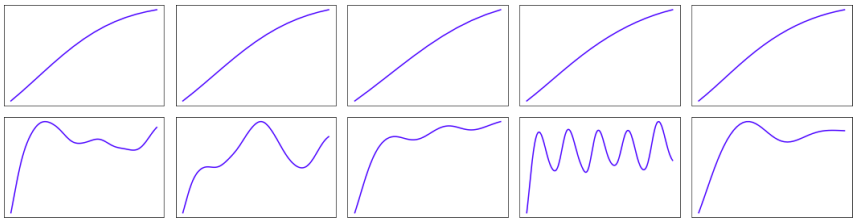

Figure 3.5: The summed absolute frame difference between first end-phase event and the frames up ti the next end-phase event. This should only be a half cardiac cycle, so there should not be any peaks (or at most one peak). The upper plots show videos where the end-phase labels only cover one half cardiac cycle, while the bottom plots show videos with more than one cardiac cycle, and thus have incorrect labels.

3.1.4 Normalizing Videos

The videos all already have the same size of 112-by-112, but the FPS differ. 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 higest 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. This may add unwanted bias to the data however, and it is not obvious how to label the interpolated frames when the video goes from one phase to another. To normalize the videos with a much higher FPS we would have to remove frames from them. 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 a total of 7946 videos.

3.1.5 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 the validation, and 12.5% for testing. This split remains after filtering out

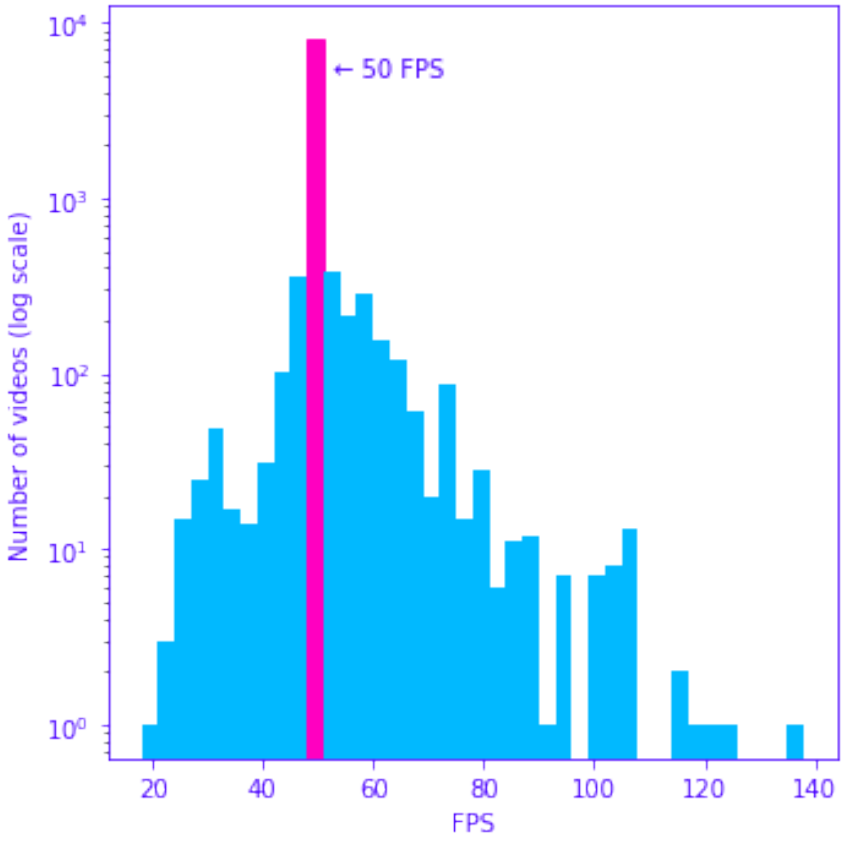

Figure 3.6: A histogram of the different FPS rates of the videos in the Echonet dataset. Note that the y-axis is in logarithmic scale — in fact, almost 80% of the videos have exactly 50 FPS.

videos as explained in the previous two sections. We opt to also use this split in this project.

3.2 Dataset 2

TOOD: Dataset by Elizabeth Lane.

Chapter 4

Methodology

4.1 Ed-/ES-Detection as a Reinforcement Learning Problem

The goal of this thesis is to explore the potential of using deep RL in the task of ED-/ES-Detection. Thus, the main methods revolve around exploring different formulations of the problem as a RL problem and comparing them.

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.

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

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 becomes better. In equation 4.2 $\hat{N}$ is the number of predicted events and $\mathcal{C}(\dagger, \hat{(\dagger)})$ 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. But for cases where there are fewer predicted events than there are ground truth events $GaaFD_1$ would give a score that does not reflect its inability to predict all events.

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

If we instead calculate the average frame difference between each ground truth event and its nearest predicted event, $GaaFD_2$, as in equation 4.3, we get the opposite problem — too many predicted events are not reflected 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.

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

Using GaaFD, or rather its inverse since we want to minimize the error, as a reward function for RL means that we are optimizing the agent directly for our main metric aaFD. It does have one final flaw, however: 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.

We could instead 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_1, as seen in equation 4.5. The information that the agent receives from the reward signal R_1 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. 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 from the current phase s to the nearest predicted phase a.

$$R_1(s, a) \triangleq \left\{ \begin{array}{ll} 1 & \text{if } s = a \\ 0 & \text{if } s \neq a \end{array} \right\} \tag{4.5}$$

$$R_2(s, a) \triangleq -d(s, a) \tag{4.6}$$

For this thesis, we will be exploring the reward functions R_1 and R_2.

4.1.1 Simple Binary Classification Environment

A simple baseline environment is defined as such: The agent, after observing the current and adjacent frames, takes an action predicting that the current frame is either of Diastole or Systole phase, and receives a reward dependent on its prediction before the environment moves the

current frame one frame forwards. This environment is visualized in figure 4.1.

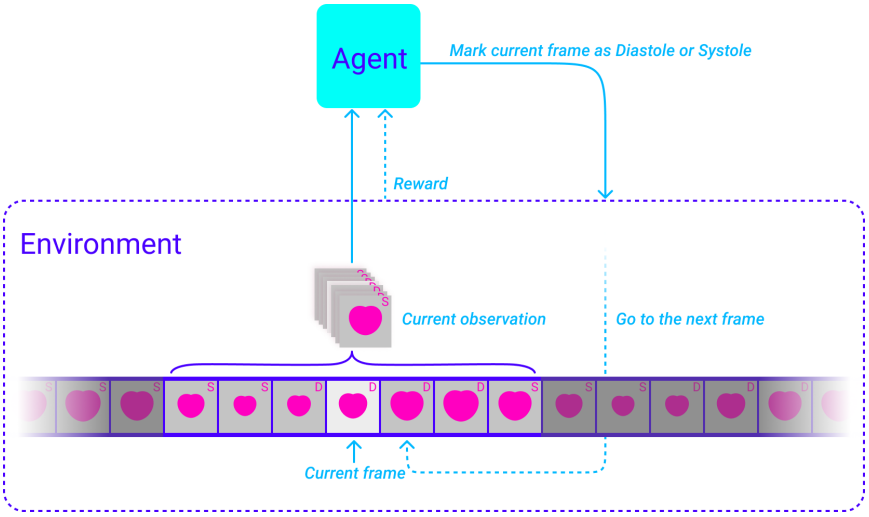

Figure 4.1: Visualization of the Binary Classification Environment loop. An agent sees the observation from the current frame and takes an action, either marking it as Diastole or as Systole, and gets back the reward and the observation for the next frame from the environment.

More formally, the observation o_t at time t is an array of $(2N+1, W, H)$ grayscale pixel values normalized to be between 0 and 1, where N is the number of adjacent frames on either side of the current frame and W and H are width and height, respectively. As a tentative first attempt N is set to 3, making the number of channels of the observations be $3+1+3=7$, and the final shape of the observation be $(7, 112, 112)$, given that we set all videos to be 112-by-112 pixels. The number of adjacent frames is assumed to be an important hyper parameter because it is what gives the agent temporal information.

The agent has a policy $\pi(a|s)$, which, given a state s (which is just the observation in our case), returns probability of taking an action a. The goal of the agent is to find the policy π^* that, if followed, maximizes our reward function R.

4.1.2 Agent Architecture

The experiments in the next chapter all use a Deep Q-Network (DQN) RL architecture with Prioritized Replay, N-Step returns, and Double Q-Learning. It uses an ϵ greedy policy to facilitate exploration.

For the Q-network, two architectures are explored:

1. A simple CNN with few layers, inspired by the Atari DQN paper (TODO: citation).

2. MobileNet-v1: a bigger and more complex CNN that uses batch normalization.

4.1.3 Distributed Training

As mentioned, DQN lends itself nicely to distributed training. In this project, this is achieved through Deepmind's library Acme[21]. At the center of Acme is another library by Deepmind called Reverb[7]. 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 om the algorithm Reverb can be used as the communication point between the two. In this way one or more actors, possibly on different machines, can generate experience samples and insert them into the Reverb experience replay database and a learner, also possibly on a different machine, can sample from it to perform gradient descent. The actors and the learner doesn't 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, also by Deepmind, called Launchpad[40].

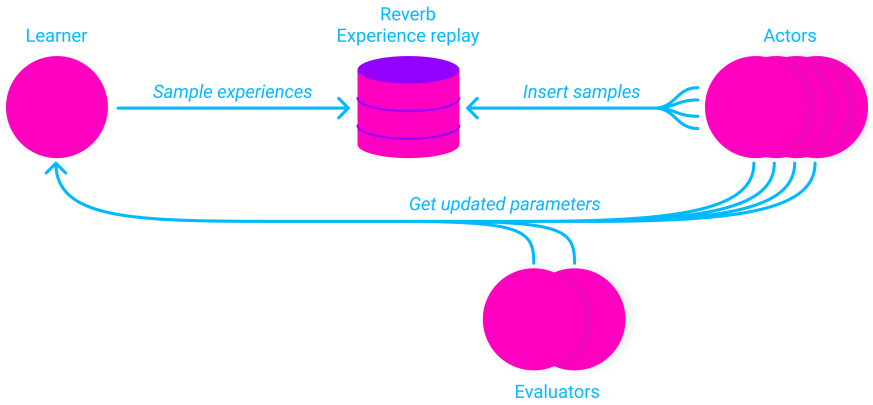

Figure 4.2: The distributed RL training system. Each pink node runs in a separate Python process, and each blue arrow is a inter-process function call facilitated by Launchpad.

There is a balance to be made between how fast experience samples should be added to the experience replay and how fast they should be sampled by the learner. If the learner samples faster than the actors are able to generate new samples then the network will be trained using trajectories generated from outdated policies. If the actors generate new samples much faster than the learner is able to sample then we are arguably 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 starts to differ too much. Using 6 actors was found to be suffient for generating new experience samples fast enough. TODO: For cases where we use MobileNet also (don't we need

less actors there since the learner takes more time)?

TODO: Write about why we chose DQN, what alternatives we considered, etc.

4.1.4 Discussion

Under the hood, the DQN algorithm is solving a regression problem. Given a state, the model predicts the expected future returns after taking a given action.

TODO: BCE is using RL for a job that asks for Supervised Learning. There is no exploration, but we still use exploration mechanisms like greedy-epsilon. Using epsilon of 1.0 (100% random decisions while training) is a sign that something is off. It is like an inefficient supervised learning training loop.

- How is this similar to regular supervised learning classification problem?
  - DQN predicts expected future returns of taking an action. We can set up a supervised learning regression problem that predicts the same thing
- We use epsilon=1 and discount=0 — implications?
- Write about how DQN is simply a regression problem
- Future work could be using Policy Gradient methods

4.2 Incorporating Search

RL is a tool meant for solving problems that require search, so in order to get any benefits from it we must transform the problem to one that requires search. This may sound like straightening a screw to make it work with a hammer, and the author sympathizes with this sentiment.

We could let the agent search through frames to find the ED or ES frames. In this case the action set could be to move to the previous, move to the next frame, marking the current frame as ED, or marking it as ES, 4 actions in total. One problem with this formulation is that once the agent has marked one frame as ED or ES, it must know that this state can be ignored and that it should start to look for other ED or ES frames. One work-around to this problem that enforces that the agent visits all frames at least once is simply by initializing it at every frame, but this setup is just a slower, less robus version of the Binary Classification Environment.

Another option is to perform exploration in space, taking inspiration from papers like (TODO: add paper of RL landmark detection). This can be done by looking at just a small region of interest in the video, which the agent can move around before taking an action. In this way, the agent loses some global context depending on how small the region of interest is,

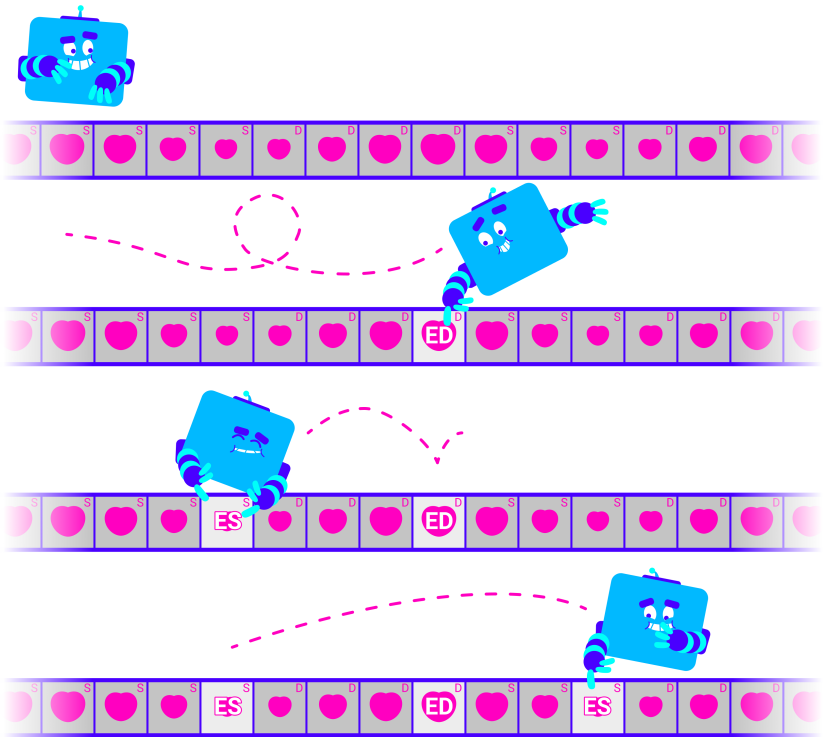

Figure 4.3: An agent moves to the previous or next frame and marks frames that it predicts to be ED os ES.

but the smaller input size makes the model less computationally intensive, enabling us to include more temporal information.

Another version of the space-exploration scheme is to let the agent create a synthetic m-mode image from the video. Here, a line can be translated and rotated by the agent, forming the bases of the m-mode image. The pixels along the line for the current frame and some number of previous and next frames are concatenated together into one image. A video can be seen as a 3D data cube, consisting of width, height, and time, but using the synthetic m-mode technique width and height are replaced by the line, effectively removing one spacial dimension while keeping the temporal dimension intact. Compared to the region of interest exploration scheme, synthetic m-mode exploration allows us to keep more temporal data. M-mode imaging is also a well established imaging mode in clinical settings, so this is the method that we want to explore further.

4.2.1 M-Mode Binary Classification Environment

The set of actions remain the same as in the binary classification environment, but an additional 6 actions are added: rotating the m-mode line *clockwise* and *anti-clockwise*, and translating the line *up*, *down*, *left*, or *right*. The rotation amount and step size are considered hyper parameters. Making them too big would make the line movement less precise, but making it too small would make both the training and inference slower, as well as fur-

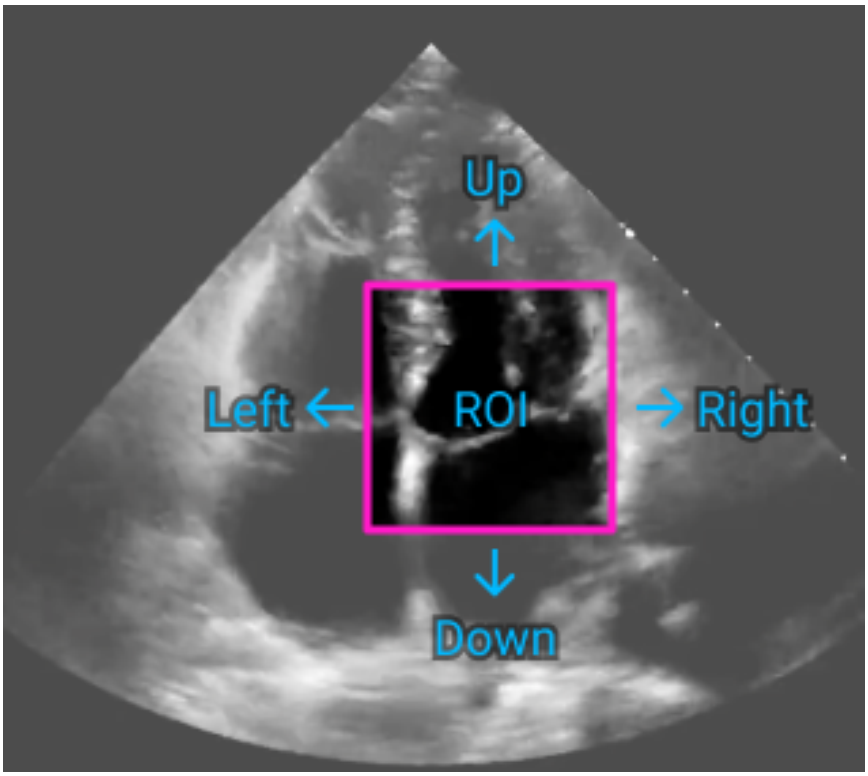

Figure 4.4: A Region Of Interest (ROI) is given to the agent which it can then move around in order to explore.

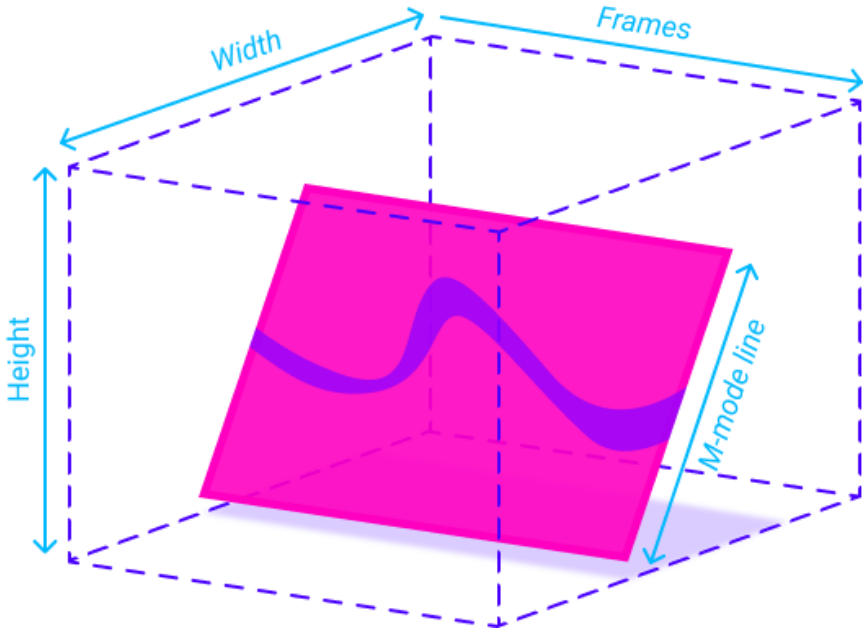

Figure 4.5: An m-mode image is an intersecting plane in 3D "video space".

ther sparsifying direct reward signals. A decision also has to be made for whether the translation should be global or local. Global translation means that the line moves in a direction relative to the video, while local translation means that the line moves in a direction relative to where it is pointing towards.

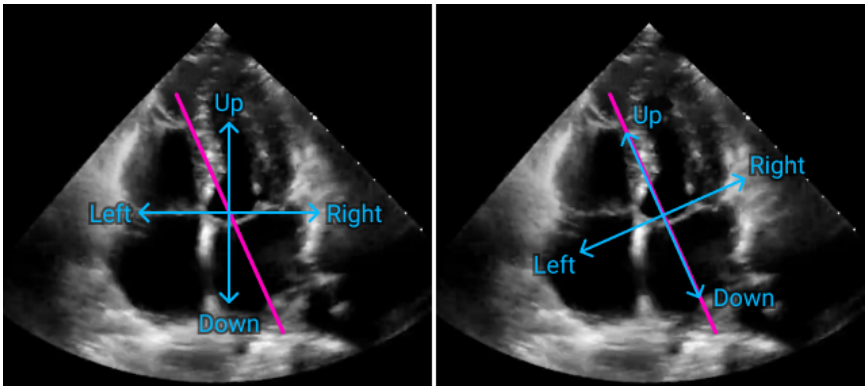

Figure 4.6: Global (to the left) versus local (to the right) translation. Local translation means that the movement depends on the direction of the m-mode line.

Using local translation is presumed to add some rotational invariance, as the rotation of the video itself can be counteracted by the m-mode line without changing the perceived m-mode effects of translation. This also makes the effects of the up- and down-translations trivial, independent of rotation — it simply shifts the m-mode image down or up, respectively.

TODO: Show how vertical translation (up or down) simply shifts the m-mode image.

At the beginning of an episode the m-mode line is placed in the center of the video, vertically. TODO: This should be at a random position/rotation in the image. The observations returned from the environment is the synthetic m-mode image from the current line, by looking 15 frames in the past and 15 frames in the future, for a total of 31 frames. 8 additional channels are included for the synthetic m-mode images that result from rotating the line to the left and to the right, and moving it to the left and to the right. Synthetic m-mode images resulting from moving the line up or down are not included as channels, as they add very little new information, given that we use a neural network architecture with translational invariance, like CNNs. To counteract the big loss of global context when using synthetic m-mode images, two other channels are included in the observations as well: an average of up to 50 frames around the current frames and an image with the location of the current line drawn onto it. The averaged video image and the current line image adds information about the current line position and provides the agent with some additional context.

TODO: Add image showing m-mode environment and example observations.

The same reward functions are explored as in the regular binary

classification environment. In practice, the agent will receive a sparser reward signal, since a reward may only be given when the agent selects to mark the current frame as diastole or systole, not when it only performs translation or rotation.

4.2.2 Reinforcement Learning Agent Architecture

TODO: Same as before, DQN, but network must be adjusted to fit a tuple of observations (m-mode + overview).

4.2.3 TODO Discussion

- Sparse reward signal may make the results worse. Can be counteracted by: what? n-step? Less dicsount (gamma closer to 1.0)? Using "advantage" for Q-function? Actor-critic network agent?

Chapter 5

Experiments and Results

```
Notes for this chapter


  Supervised Binary Classificaiton of Frames as a baseline
  - Using as many of the same hyper parameter / architecture choices as in the Simple B
    - Simple network with stride and big kernels (from original DQN paper)
    - MobileNet

  Simple Binary Classification Environment
  - Studying the effect of the RL framework on this problem.
    - Prioritized replay, epsilon, discount, N-step, reward spec

  M-Mode environment
  - How to make this work?


Sections below will be trashed and rewritten...
```

5.1 Supervised Binary Classification With MobileNet

5.2 Simple Binary Classification Environment

We do not expect the Binary Classification Environment (BCE) to perform any better than just a normal supervised learning classification task as this task does not require strategic planning. Regardless, and also to bring further evidence to this belief, we train the agent using this simple environment to see the results. Of interest is the effect of the exploration/exploitation ratio in the form of the ϵ hyper parameter, the amount of discounting, and the number N steps of bootstrapping.

A lower value of ϵ means that the agent will more often try to exploit its existing assumptions about the optimal policy. This can be good when the optimal policy consists of many steps and the agent can take

better advantage of exploration in areas that it already knows are fruitful. However, in the case of BCE the current step is completely independent of all previous steps, so the most efficient learning strategy is presumably to always explore, i.e. always take a random action, i.e. $\epsilon = 1$. Exploitation provides no additional value yet risk creating a sample imbalance where the presumed best step is taken over and over.

The discount hyper-parameter γ determined how much it values future rewards. A value of $\gamma = 0$ means that the agent will only try to maximize immediate rewards, and a value of $\gamma = 1$ means that the agent will try to maximize all future rewards, as well as, and as much as, the immediate reward. Again, as future steps are completely independent of the current step, the best value for γ is presumed to be 0 for BCE. If not, then the estimated return of taking an action will also depend on the returns of taking an action in the succeding state, and as the agent is still learning, this may only add noise to the value without giving any advantages.

N-step bootstrapping is a way to make to speed up the bootstrapping of value estimations. This is again not relevant in the case of BCE because the Q-values does not depend on future value estimations, and thus bootstrapping does not occur. A value of $N = 1$ is presumed to be best for BCE, which is the same as not performing N-step bootstrapping at all, just simply Q-learning.

To test these hypotheses, and to assure that the method of using DQN works in general, 4 experiments are conducted. First, an experiment utilizing "all" the RL machinery: ϵ -greedy policy with $\epsilon = 0.2$, discounting with $\gamma = 0.95$, and 4-step bootstrapping. Then another with the same setup, except that every action is random, i.e. $\epsilon = 1$. Then another with the same setup, except that every action is random and there is no discounting, i.e. $\gamma = 0$. And lastly, the same setup, except every action is random, no discounting, and no N-step bootstrapping.

The results for each of these sets of hyper-parameters are plotted in figure 5.1. The agent is evaluated on the validation part of the Echonet dataset.

TODO: Make this a proper ablation study instead... It is hard to tell how each hyper-parameter affect the result.

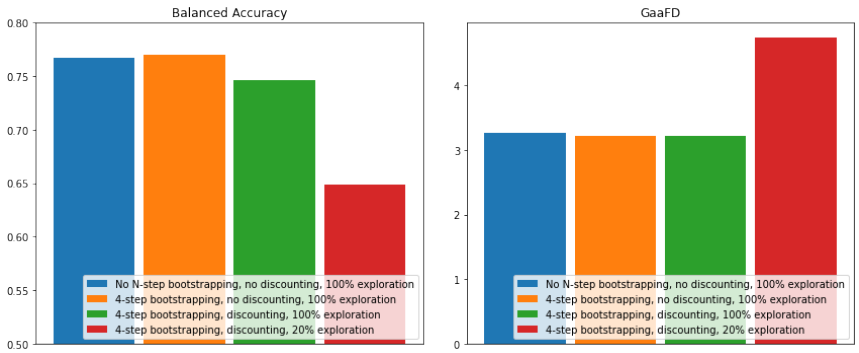

Figure 5.1: To the left: the balanced accuracy score of the agent's predictions on the Echonet validation split dataset. To the right: the GaaFD, likewise.

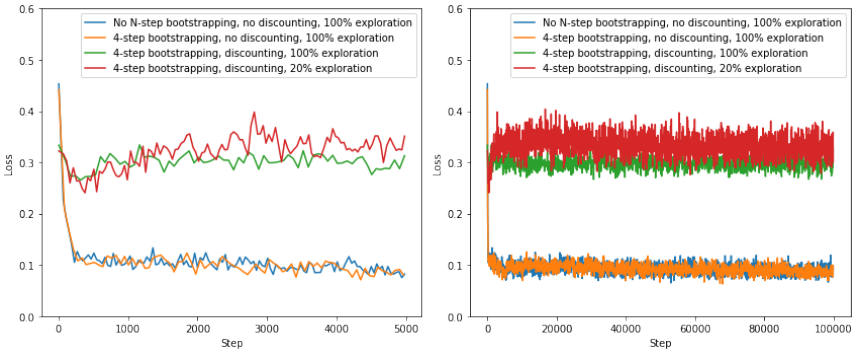

Figure 5.2: To the left: the loss over the first 5000 learner steps. To the right: the loss over all 100K learner steps.

TODO: Also with and without Importance Sampling
LOSS CURVE VARIANCE OF DIFFERENT EXPERIMENTS
THE BEST OF THESE BUT WITH PROXIMITY REWARD
THE BEST OF THESE BUT WITH MOBILENET
THE BEST OF THESE BUT WITH MORE FRAMES/CHANNELS
SHOW BEST PERFORMING IMAGES WITH Q-VALUES
SHOW WORST PERFORMING IMAGES WITH Q-VALUES

5.3 M-Mode Binary Classification Environment

Part III

Conclusion

5.4 Discussion

5.5 Conclusion and Further Work