UiO : Department of Mathematics
University of Oslo

Detecting valvular event times from echocardiograms using deep neural networks

Marie Roald

Master's Thesis, Spring 2018

This master's thesis is submitted under the master's programme *Computational Science and Engineering*, with programme option *Computational Science*, at the Department of Mathematics, University of Oslo. The scope of the thesis is 60 credits.

The front page depicts a section of the root system of the exceptional Lie group E_8, projected into the plane. Lie groups were invented by the Norwegian mathematician Sophus Lie (1842–1899) to express symmetries in differential equations and today they play a central role in various parts of mathematics.

Front matter

Abstract

The timing of cardiac events is essential for the analysis of certain components of myocardial function [36]. Finding an algorithm that detects these timings has therefore been the subject of several studies[36]. In this project, deep neural networks were used to detect the valvular event times from echocardiography sequences. Three classes of neural network algorithms were tested: fully convolutional architectures [48], VGG [47] inspired architectures and recurrent neural networks. Temporal information was also incorporated by feeding the network the relative time passed since the last QRS peak. It was found that incorporating temporal information was necessary for detecting the valvular event times with an acceptable accuracy. The model providing the highest performance metrics was a VGG inspired architecture with both an RNN head and the relative time since the QRS peak. A version of this model that did not include the relative times was visualised using both guided backpropagation [48] and image occlusion [54], which demonstrated that the position and movement of the valves were important for correctly predicting valvular events.

The best model achieved a 93% test accuracy and correctly detected all the valvular events in 7 out of 11 test series with a mean error of 1.03 frames. This is not satisfactory for clinical use. However, it does indicate that deep neural networks applied to echocardiography are a promising approach for automatic valvular event time detection.

0.1 Acknowledgements

First, I would like to thank my supervisor Eigil Samset for his important feedback and support. I also want to thank Anne H Schistad Solberg for taking the time to read this thesis and contributing constructive feedback and advice. Moreover, I want to thank Cecilia Marie Futhsæther for reading the abstract and providing helpful suggestions. The help I received from Aleksandar Babic and Kristin McLeod when retrieving the augmented dataset was also much appreciated. I want to thank Vigdis Holta for reading parts of this thesis and sharing valuable insights and wisdom. Finally, I thank Yngve Mardal Moe for continuous support and invaluable advice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Information about timing of the cardiac cycle is often needed to evaluate specific parts of myocardial function. The aortic valve closure time and mitral valve closure time mark end-systole and end-diastole and can be used to estimate several parameters for assessing heart function. Examples are post-systolic shortening [36] and systolic displacement [5]. The valvular timings vary from cycle to cycle and thus, the timing cannot be extracted from a different cycle, but must be calculated for each cycle. Moreover, all the valvular event times, that is, when the cardiac valves close and open, are used to individualise left ventricular pressure for a specific subject and estimate the myocardial work [45, 44]. A method to find these valve event times automatically will, therefore, be useful and will allow parameters that require the valvular event times to be calculated without manual effort.

Several methods to calculate valvular event times exists. Aortic valve closure can be estimated by analysing velocity curves from tissue Doppler imaging, phono-cardiography of the second heart sound and empirical regression relations [1]. Speckle tracking analysis, strain curve analysis or mitral spectral Doppler assessment can be used to detect both mitral valve closure and aortic valve closure [36]. Image analysis of 2D echocardiography images of the heart have also been proposed to detect aortic valve closure and mitral valve closure. Gifani et al. use manifold learning on the grey scale echocardiography frames to detect aortic valve closure and mitral valve closure.

For this project, we want to detect the valvular event times automatically from sequences of echocardiography with convolutional neural networks. Convolutional neural networks (CNNs) are a type of neural networks that are proven very successful for image classification problems [20, 23, 47, 31, 48]. Our dataset consists of sequences of grey-scale ultrasound images depicting the cardiac cycle. The image frames are apical long-axis view, i.e. they show the left ventricle, the mitral valve and the aortic valve. Our objective is to use CNNs on the ultrasound frames to classify the aortic valve and the mitral valve as either open or closed and use this information to detect valvular event times.

1.1 Cardiac cycle

The heart has four chambers. The two upper chambers are known as the *atria*, and the two lower chambers are called the *ventricles*. The atria receive blood into the heart, and the ventricles discharge the blood out from the heart to the lungs or the veins. Figure (**??**) shows the four chambers. The atria chambers open into the ventricles via the atrioventricular valves.

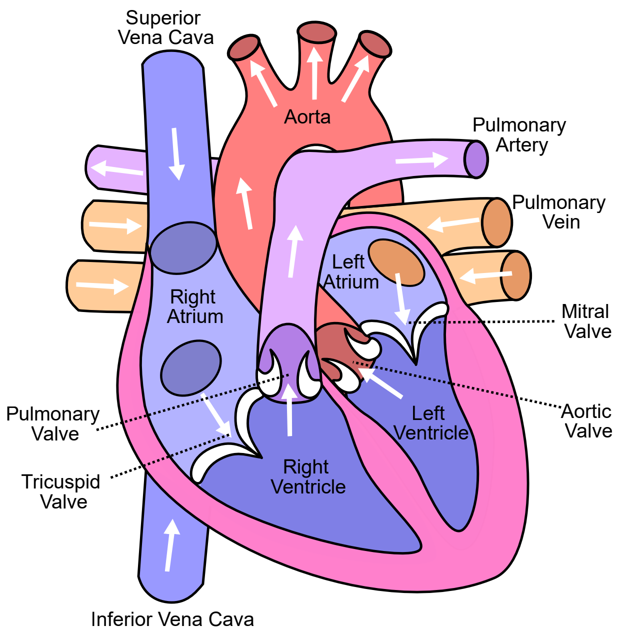


Figure 1.1: Diagram of the human heart

Illustrates the placement of the valves and chambers. The arteries and veins are also indicated and the normal direction of the blod flow is shown with white arrows. (Figure by Eric Pierce, CC-BY-SA-3.0, via Wikimedia Commons, url: ***https://en.wikipedia.org/wiki/File:Diagram_of_the_human_heart.svg*** *)*

Four valves are situated between the heart's chambers. Section 1.1 shows an illustration of the valves and their placements. The valves that connect the atria to the ventricles are known as the *atrioventricular valves*. There are two atrioventricular valves, the *tricuspid valve* and the *mitral valve*. The tricuspid valve sits between the right atrium and the right ventricle, while the mitral valve is found between the left atrium and left ventricle. At the exit of each of the ventricles lies two *semilunar valves*, the *pulmonary valve* and the *aortic valve*. The pulmonary valve rests at the base of the pulmonary artery, and the aortic valve lies at the base of the aorta.

The sequence of events that occurs every time the heart beats is known as the *cardiac cycle*. At the start of the cycle, the right atrium relaxes and fills with deoxygenated blood that flows in from the body. Then the right atrium contracts and blood is pumped from the right atrium into the right ventricle through the tricuspid valve. When the right ventricle is filled, the tricuspid valve closes and stops the blood from flowing back. The right ventricle contracts, which opens the pulmonary valve and blood is pumped out from the ventricle to the pulmonary artery through the pulmonary valve before the pulmonary valve closes. From the pulmonary valve, the blood moves to the lungs to pick up oxygen. On the left side, the left atrium relaxes and fills with oxygenated blood. Then, the left atrium contracts and the blood moves to the left ventricles through the mitral valve. When the left ventricle is filled, the mitral valve closes. The left ventricle contracts and forces open the aortic valve. The blood flows through the aortic valve into the aorta. Then the aortic valve closes, the left ventricle relaxes, and the cycle begins again. In reality, the two sides of the heart work together, and several of the steps in the cycle happens at the same time.

We can divide the cycle into *systole* and *diastole*. Until now, we have described both the left and right side of the heart. For our project, we look at images of the left heart, and thus we will just focus on the left heart from now. Systole is the contraction phase and begins with *iso-volumetric contraction* (IVCT). Iso-volumetric contraction is the short period that occurs when the ventricle is contracting, both valves are closed, so the ejection of blood from the ventricles has not begun yet, and there is therefore no change in volume. Then the aortic valve open and the blood is pumped out of the ventricle. Diastole is the relaxation phase. Early in diastole, we have *iso-volumic relaxation* (IVRT). Iso-volumetric relaxation is the period between when the aortic valve has just closed, and the mitral valve begin to open. When the mitral valve opens, the blood moves from the atrium into the ventricles.

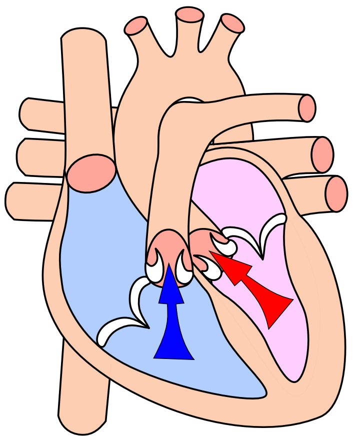
For this project, we are interested in the timing of the opening and closing of the valves. Specifically the valves in the left part of the heart, the mitral valve and the aortic valve. From the perspective of the valves the cardiac cycle has four stages:

1. Iso-volumetric contraction (both valves closed)
2. Ventricular ejection (aortic valve open, mitral valve closed)
3. Iso-volumetric relaxation (both valves closed)
4. Ventricular filling (aortic valve closed, mitral valve open)

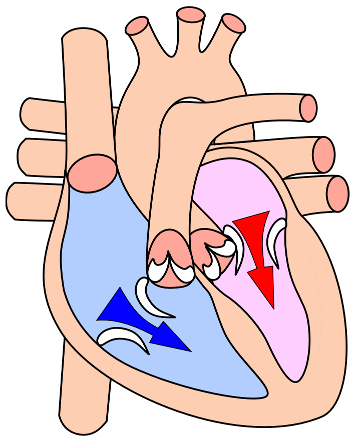
We know that these stages always happen in the same order. However, the exact timing can vary from case to case.

1.2 Motivation

Valvular event times can be useful when analysing specific parts of the cardiac cycle. The valvular opening and closing times mark the transition between different stages of the cardiac cycle. Aortic valve closure marks the transition from



(a) The heart during ventricular ejection.



(b) The heart during ventricular filling.

Figure 1.2: Illustration of the two cardiac phases

Figure 1.2a shows contraction (systole) and Figure 1.2b shows relaxation (diastole). The red arrow indicates newly oxygenated blood and the blue arrow indicates oxygen-depleted blood that is soon to be reoxygenated by the lungs. (Figures by Eric Pierce, CC-BY-SA-3.0, via Wikimedia Commons, url: `https://commons.wikimedia.org/wiki/File:Heart_systole.svg`,`https://commons.wikimedia.org/wiki/File:Heart_diasystole.svg`) ventricular ejection to the beginning of relaxation (diastole). Mitral valve opening happens at the transition point from iso-volumetric relaxation to ventricular filling, and mitral valve closure marks the transition from relaxation and ventricular filling to contraction (systole). Aortic valve opening separates the iso-volumetric contraction period from the beginning of ventricular ejection. These timings can be used for further analysis that requires information about the timing of the different cycle phases. It is beneficial to find these time points automatically to save manual effort.

An example of a task that requires valvular event times is estimating wasted myocardial work for the left ventricle. In ‘Assessment of wasted myocardial work: a novel method to quantify energy loss due to uncoordinated left ventricular contractions’, Russell et al. describes a method to measure how much myocardial energy is wasted for a dyssynchronous contraction pattern, the Wasted Work Ratio (WRR) [45]. This estimate is calculated as a ratio of the wasted work compared to the total work. Russell et al. used the area of an LV pressure/strain loop to estimate the myocardial work for a segment. To calculate this, they started by finding the left ventricular pressure (LVP) non-invasively by using a normal curve calculated from known examples and warping it in time to match the valvular event times of the subject. This method to construct a non-invasive LVP curve is described in detail in [45]. The steps are shown in Section 1.2. The LV pressure curve is multiplied with segmental shortening rate to get a measure of power. The power curve is then integrated over the iso-volumetric contraction, ventricular ejection and iso-volumetric relaxation phases to get a measure of work. This interval is the time between mitral valve closure and mitral valve opening. These steps are shown in Section 1.2 and explained in detail in [44]. So to find wasted myocardial work, we need all the valvular event times as input. Mitral valve closening and opening are needed for the final integration of the power. Furthermore, both mitral and aortic valve closing and opening are required to individualise the estimate for LV pressure.

1.3 Related Work

In ‘Automatic timing of aortic valve closure in apical tissue Doppler images’ [1], Aase et al. evaluate automatic and automated algorithms for detecting aortic valve closure time with Doppler tissue imaging (TDI). For this task, they used the motion of the mitral ring points. During the cardiac cycle, mitral ring points usually produce strong echoes and have higher velocity values. Because of this, TDI velocity/time curves from the mitral ring points are the curves most robust for noise. From the velocity/time curves, time points for mitral valve opening and early relaxation were extracted by analysing the curve, and used to define a region of interest for searching after aortic valve closure. Finally, aortic valve closure was found as a notch with high acceleration within this region of interest. Aase et al. found that in 98% of the cardiac cycles they used for validation, the automatic algorithm estimated the time point of AVC within 25 ms of the reference.

In ‘How to define end-diastole and end-systole?: Impact of timing on strain

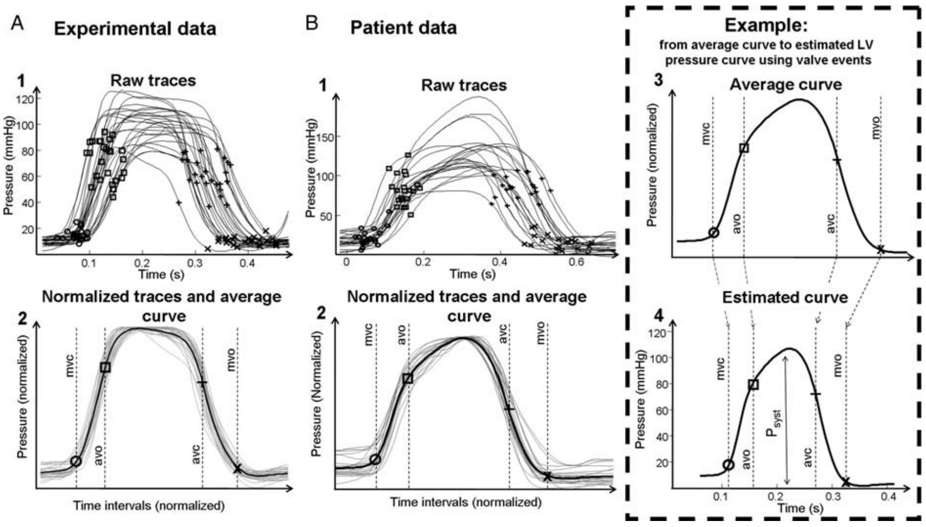


Figure 1.3: The process for non-invasive estimation of the left ventricular pressure curve [45].

A1 *and* ***B1*** *show raw left ventricular pressure data from dogs and patient respectively. The valvular event times are indicated (circle marks MVC, square marks AVO, plus marks AVC and x marks MVO).* ***A2*** *and* ***B2*** *show the result of warping the raw pressure waveforms (grey curves) along the time axis to normalise the duration of the intervals between the valvular events for all recordings. The waveforms have also been scaled in amplitude to have the same peak value. The black curve indicates the averaged waveform.* ***B3*** *shows the average waveform result from B2.* ***B4:*** *the left ventricular pressure waveform is created by stretching and compressing along the time axis so that the valvular events match the actual valvular timings for the specific subject. The waveforms have also been scaled vertically to match systolic arterial cuff pressure. Figure from [45].*

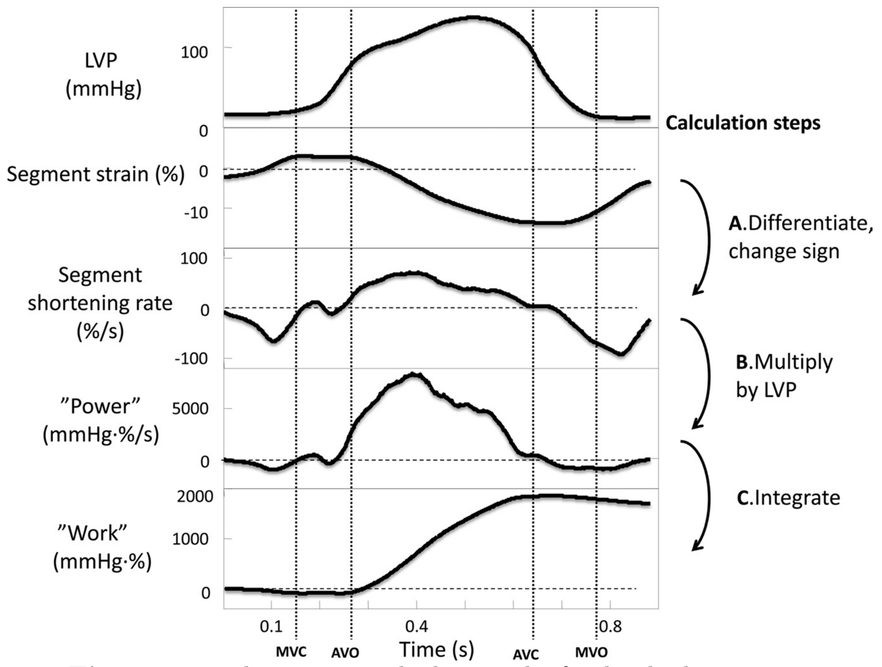


Figure 1.4: The steps to calculate work of individual segments from strain recordings and LVP [44].

The rate of segmental shortening (strain rate) was found by differentiating the strain curve. Multiplying this with LVP results in a measure of power which was then integrated over time to give work as a function of time. Losses can occur by stretching of segments in the isovolumic phases, so the work was calculated over the IVC, ejection, and IVR phases by integrating from MVC to MVO. Figure from [44] measurements'[36], Mada et al. looked at different ways to automatically define end-systole (ES) and end-diastole (ED) and how changes in the definition of these events impacted the accuracy of strain measurements. The details of the different methods are explained in ref. Sections 1.3 and 1.3 shows the mean and standard deviation of the difference between the surrogates and the reference ED and ES frames. The authors found that peak R can serve as an approximation for ED as long as the ECG morphology is normal. In all other cases, mitral valve closure should be used for correct and well-defined measurement. Similarly, a global strain or volume nadir may be used as an approximation for end-systole in hearts without regional dysfunction. If this is not the case, aortic valve closure should be used. [36] concludes that manual observation of the valve closures in the grey-scale images is sufficient for defining ES and ED, but if necessary, measuring the valve closure artefacts in aortic and mitral valve Doppler traces can replace the direct observation. For normal cases, Mada et al. found that Mitral spectral Doppler assessment of ED had a mean difference of 4 ms (and standard deviation of 10 ms) from the reference, while AVC derived from the spectral Doppler data had a mean error of 3 ms (and standard deviation of 10 ms).

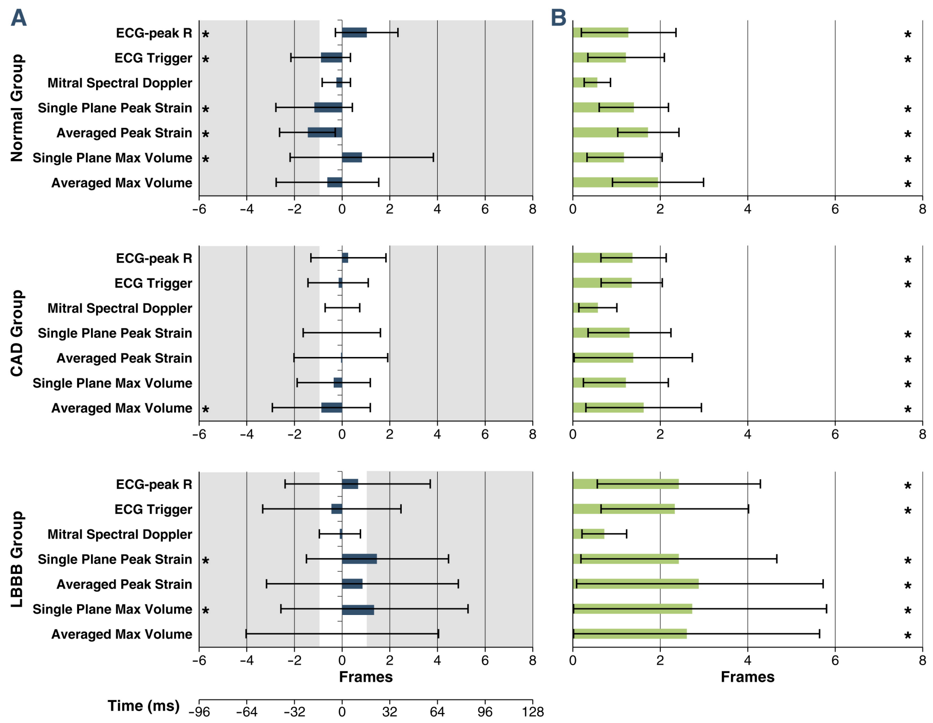


Figure 1.5: Comparison of different surrogate parameters used to define timing of end-diastole [36].

A: *Mean error compared to reference ± SD* ***B:*** *mean absolute error compared to reference ± SD. CAD = coronary artery disease, ECG = electrocardiography, LBBB = left bundle-branch block. (Figure from [36])*

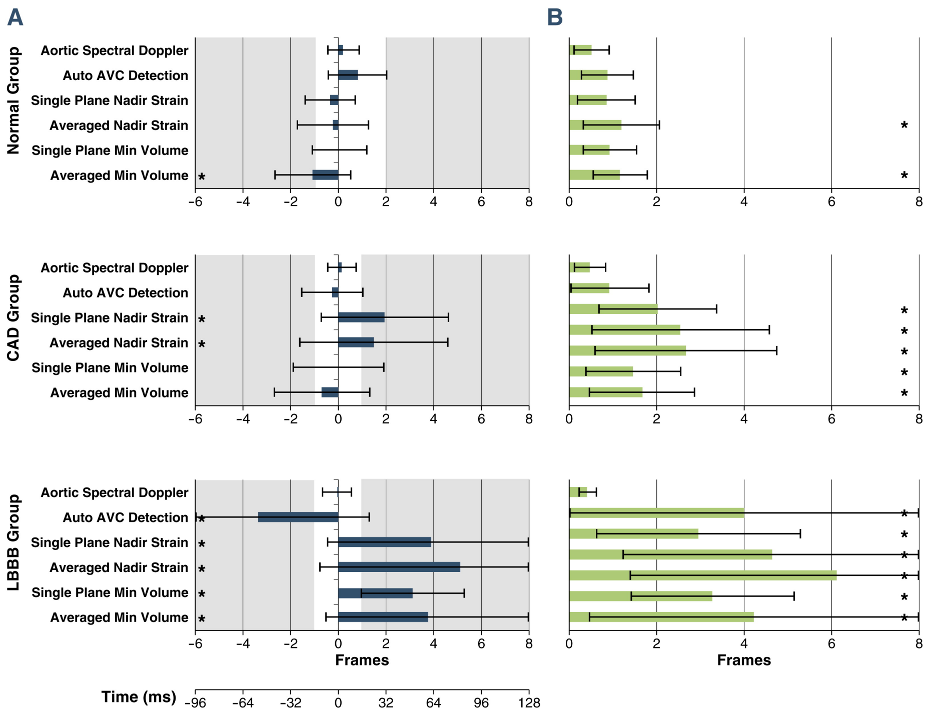


Figure 1.6: Comparison of different surrogate parameters used to define timing of end-diastole [36].

A: *Mean error compared to reference ± SD* ***B:*** *mean absolute error compared to reference ± SD. CAD = coronary artery disease, LBBB = left bundle-branch block. (Figure from [36])*

Gifani et al. applied manifold learning to a series of 2D echocardiography images depicting one cycle of heart motion to analyse the relationship between the frames and detect the end-systole (aortic valve closure) and end-diastole (mitral valve closure) frames [15]. This method only needs the ultrasound images as the input. The authors used locally linear embeddings (LLE) to represent each image as a point on a two dimensional manifold. The manifold found for some of the cycle are shown in Section 1.3. Through analysis of this manifold, they discovered three dense regions corresponding to iso-volumetric contraction, iso-volumetric relaxation and reduced filling. These are all periods in the cardiac cycle where the change in volume is low or non-existent. Gifani et al. examined the distance between consecutive frames in the manifold and extracted the minima, which corresponded to the iso-volumic frames. Example distance diagrams with the minima are shown in Section 1.3. Two of these three minima corresponds to the end-systolic and end-diastolic frames which are part the IVC and IVR phases respectively. Finally, the end-systolic and end-diastolic frames are the frames with the maximum difference, and thus minimum correlation was used to select them from the three candidates. For end-diastole (mitral valve closure) the mean difference between the manually identified frames and the frames automatically identified by this method was 1.3 frames and for end-systole (aortic valve closure), the mean difference was 0.7 frames.

There are several schemes to detect aortic valve closure and mitral valve closure, which marks the end-systole and end-diastole events respectively. For our problem, we also need the timing of aortic valve opening (AVO) and mitral valve opening (MVO). It appears that visual inspection of valvular events from sequences of two-dimensional grey-scale echocardiography of the cardiac cycle is satisfactory to detect AVC and MVC provided the resolution is adequate, and the valves are visible in the frames. It is reasonable to expect also to be able to detect AVO and MVO from such echocardiography sequences. [36] and [15] used manual inspection of echocardiography as the reference for AVC and MVC and [15] showed that the relationship between the frames in a sequence of frames depicting a cardiac cycle contain information that can be used to detect AVC and MVC. All this suggests that we can define the valvular event times by applying modern image processing algorithms like convolutional neural nets to sequences of 2D grey-scale ultrasound images of the heart valves.

Neural networks have revolutionised computer vision the past few years [31, 47, 22, 23] and it has already had its impact on medical research [13, 21]. This approach has also been shown to give state of the art accuracy on valvular event detection from MRI images [30]. Using a similar methodology to detect valvular events on echocardiogram series is therefore a logical next step. This is especially true considering that echocardiograms have been used to detect valvular events, although not (to the author's knowledge) using neural networks [36, 15]. The goal of this project is, therefore, to introduce the concepts of neural networks that are necessary to analyse video data and conclude if this is a promising approach for valvular event detection.

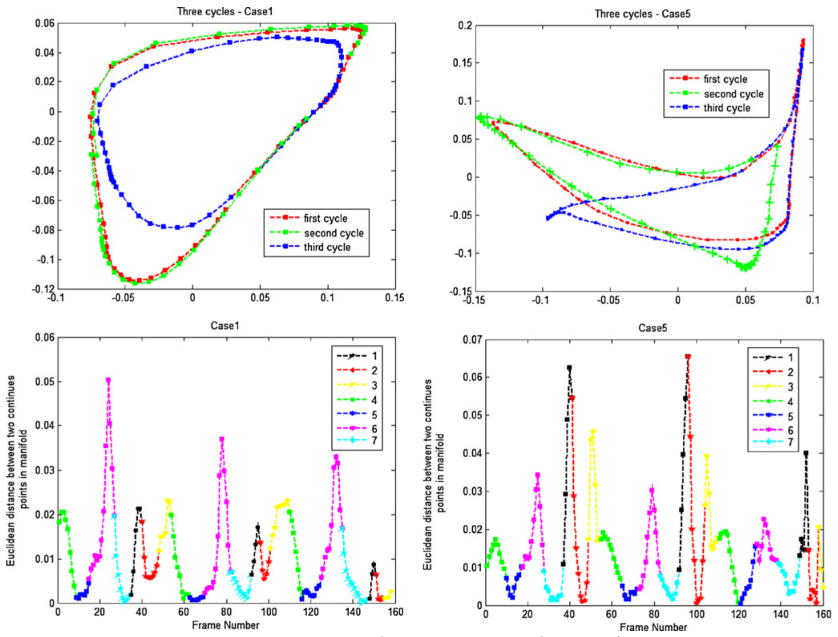


Figure 1.7: Illustration of the results from 'Automatic detection of end-diastole and end-systole from echocardiography images using manifold learning'.

TOP: *The 2D manifold found by LLE for three consecutive cycles.* ***BOTTOM:*** *The corresponding distance diagrams. The minima with red color corresponded to end-diastole frames, and the minima with blue color corresponded to end-systole frames. Seven phases of the cardiac cycle are depicted with different colors. (Figure from [15])*

Chapter 2

Background theory

2.1 Feedforward neural networks

A feedforward neural network works by composing several linear transformations, separated by several non-linear activation functions. We call each of these pairs of linear transformation and non-linear activation function for a layer. Thus, the output of a two-layer neural network can be written this way

$$\hat{\boldsymbol{y}} = \boldsymbol{W_2}\phi(\boldsymbol{W}_1 x + \boldsymbol{b}_1) + \boldsymbol{b}_2, \tag{2.1}$$

where $\hat{y}$ is the output of the network, the $\boldsymbol{W}$ matrices and $\boldsymbol{b}$ vectors are supposed to be learned, ϕ is the non-linearity function used and $\boldsymbol{x}$ is the input vector. Similarly, a three layer network can be written this way

$$\hat{\boldsymbol{y}} = \boldsymbol{W_3}\phi(\boldsymbol{W_2}\phi(\boldsymbol{W_1 x} + \boldsymbol{b}_1) + \boldsymbol{b}_1) + \boldsymbol{b}_1. \tag{2.2}$$

We call the i-th value in the vector $\phi(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1)$ for the activation of the i-th neuron in the first layer. Likewise, we call the i-th value in the vector $\phi(\boldsymbol{W}_2\phi(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)$ for the activation of the i-th neuron in the second layer. The reason for this nomenclature is that much of the theory of neural networks draw inspiration from computational neuroscience [20, 18].

The non-linear activation functions, ϕ, are used to ensure that the network can learn non-linear relationships. There are several non-linearities used in deep learning, and we will present some of the most well known functions.

A quick note about notation

It will be necessary to talk about all the parameters that go into the model. Not only the weight matrices, $\boldsymbol{W}_i$, and biases, $\boldsymbol{b}_i$, but also other components that will be introduced later. To do this, we define $\mathcal{W}$ as the collection of all parameters of the model and w_i as a single arbitrary parameter of the model.

Sigmoid

$$\phi(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

The sigmoid non-linearity was frequently used for a long time. It takes in a real number and outputs a number between 0 and 1. It is not widely used any more because the gradient is effectively zero for large positive and negative numbers. This is detrimental because the gradient of the output $\hat{y}$ with respect to the input $\boldsymbol{x}$ is, according to the chain rule of calculus, proportional to the gradient of the non-linearity. This problem is known as the vanishing gradient problem [20]. The gradients are needed to train the model because the training happens by minimising a loss function with gradient descent. Optimisation and gradient descent will be explained in detail in Section 2.4 on page 29. Additionally, the outputs are not centred around zero which can lead to undesired dynamics in the gradient updates [33].

Tanh

$$\phi(x) = \tanh(x) = 2\sigma(2x) - 1 \tag{2.4}$$

The tanh activation function was also popular for a while. Unlike the sigmoid function, the tanh has zero centred output. However, it is equivalent to a scaled sigmoid function, and as a result, it has the same problems with vanishing gradients as the sigmoid activation.

ReLU

$$\phi(x) = \max(0, x) \tag{2.5}$$

The rectified linear unit (ReLU) [18] has become popular over the last few years [31]. It works by thresholding the activation at zero, which gives favourable results [20] compared to the tanh and sigmoidal activation functions. The ReLU function is, in addition to this, less computationally expensive than the tanh or sigmoid units which involve calculating exponentials. Furthermore, it does not saturate the neurons and kill the gradients, which yields a faster convergence with stochastic gradient descent in comparison to tanh or sigmoid activations. [31].

Other activation functions

One problem with ReLU activation units is that they can be fragile. If a large gradient causes the weights to update such that the neuron never activates again for any datapoint, then the gradient from there on will be zero and the unit will never update [35]. This issue is not as problematic if the learning rate for gradient descent is properly set. Several newer activation functions try to fix this problem. A couple of examples are Leaky ReLU [35], which is similar to ReLU, but with a small negative slope instead of 0 when $x < 0$ and Maxout [19] which generalises ReLU and Leaky ReLU and returns the max of two linear functions, $\max(W_1^T x + b_1, W_2^T x + b_2)$.

2.2 Softmax classifier

We want the last layer of the network output to represent how likely it is that the input belongs to the different classes. One conventional way to do this is to use the softmax classifier [20] given by

$$\hat{y_k} = \frac{e^{f_k^{\mathcal{W}}(x)}}{\sum_j e^{f_j^{\mathcal{W}}(x)}}, \tag{2.6}$$

where $\mathcal{W}$ is the collection of all weights in the network, $\hat{y_k}$ is the softmax response for class k and $f^{\mathcal{W}}$ is the output from the last layer of a network parametrised by $\mathcal{W}$. $\hat{y}$ can be interpreted as a normalised probability assigned to each class given the network f and parameters $\mathcal{W}$, that is, $\hat{y_k} = p(y = k|x, \mathcal{W})$. The softmax function interprets the output from the last layer in the network as normalised log probabilities for each class. If we take the exponential of this, we get the unnormalised probabilities. The division gives us normalised probabilities that sums up to 1. The classifier assigns the input to the class with the highest probability.

The softmax outputs are intuitive and easy to interpret. For example, if the softmax classifier outputs 0.8 for class 0, we interpret this as 80% probability of the input belonging to class 0. So in addition to getting a prediction for which class the input belongs to, we get a measurement for how confident the classifier is of this prediction. This can be useful if we just want to use the predictions that the network is sure of, or if we want to interpret which of the inputs the network struggles to classify.

One potential drawback is that the normalisation can lead to the network seaming very sure of the classification despite low activations for all classes. For example, if the model receives input data that does not belong to either of the classes. If all classes have a low activation for some data, but one of the classes still have a higher activation relative to the others, the softmax normalisation will give that class a high probability.

In order to train a neural network we need some measurement for the error, i.e. how 'wrong' the model is. For softmax classifiers it is common to use cross-entropy [4] from information theory. The cross-entropy between the true distribution p and the estimated distribution q is

$$H(p, q) = -\sum_x p(x) \log q(x). \tag{2.7}$$

Combined with softmax we get the following cross entropy loss function for a neural network, f, with parameters $\mathcal{W}$

$$J(\mathcal{W};x) = -\sum_k \log\left(\frac{e^{f_k^{\mathcal{W}}(x)}}{\sum_j e^{f_j^{\mathcal{W}}(x)}}\right). \tag{2.8}$$

The true distribution is 1 for the correct class and 0 everywhere else. If we minimise this, we get a model that wants to give the correct class of an input a probability of 1 and the other classes a probability of 0, which is what we want. We can also interpret this as Maximum Likelihood Estimation because minimising this loss can be understood as minimising the negative log likelihood for the correct class[20].

2.3 Convolutional neural networks

When the data we want to classify consists of images, the pixels that are closer to each other are often more related. A fully connected neural network does not know this without learning it from scratch and considers two images that are identical except for a spatial shift as completely different. This is problematic because many image classification applications are invariant to position, e.g. we might want the network to detect a valve independent of its position in the image. Additionally, if the images are large, then the number of weights for a single neuron quickly become very high. An image of size 256×256, for example, would lead to neurons with $256 \times 256 = 65536$ weights each. This is not very efficient, and such a large number of parameters can easily lead to overfitting.

Convolutional neural networks aim to solve this problem by taking advantage of the spatial correlation of the pixel values. Each layer is a set of learnable convolutional filters that we slide around to every position in the image [20]. An example of a convolution operation is shown in Figure 2.1. We see from the illustration that each element of the output is computed by element-wise multiplying the highlighted area with the filter kernel and summing it up (usually followed by offsetting with bias). Each of the filters is spatially small, i.e. 3×3 and we reuse the same filter kernels for every position. Hence, the number of parameters in such a network is significantly smaller than for a fully connected network. Moreover, the parameters are shared, which means that the same features are calculated over the whole image. This allows for a representation that is invariant to the absolute spatial positions. Section 2.3 shows some examples of filter kernels learned by a convolutional neural net.

The filter kernels have the same depth as the input to the layer (i.e. the number of channels in the input image, or the number of convolutions in the previous layer for intermediate layers). This means that we, for a single convolution, have as many filter kernels as we have input channels. For example, if we have an RGB image, a single convolution will consist of three convolution kernels, one for the red, one for the green and one for the blue channel. The response of

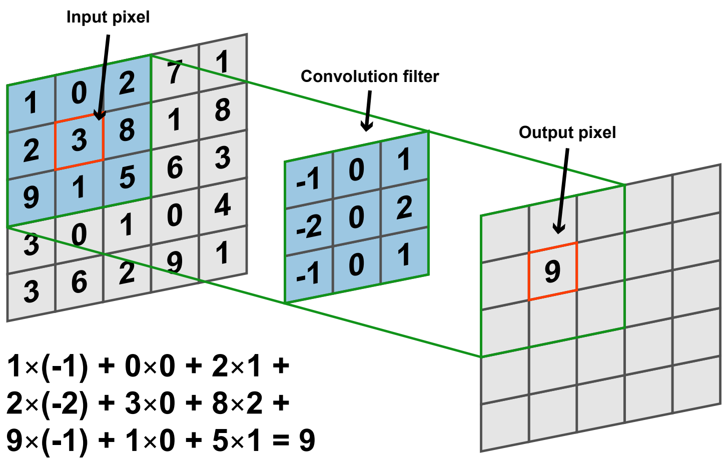


Figure 2.1: Illustration of a convolution operation with a 3×3 filter kernel.

these three kernels is then summed to give the output of a single convolution. Thus, the convolutional layers are fully connected for the channel dimension and, for a given spatial position, each output channel is dependent on all the input channels. This way of handling the channels is intuitive because we do not generally assume that the channels next to each other are related. Furthermore, handling the channels this way makes it possible to use 1×1 convolutions to “pool” features across channels [34] or to increase or decrease the feature dimension [51].

An issue with convolutions is how to deal with the boundaries. There are two conventional two to deal with this that are common in deep learning. The first is to perform the convolution only for the elements in the input where the filter kernel “fits” [7]. For a $k \times k$ filter kernel, this effectively reduces the size of the output image from $n \times m$ to $(n-(k-1)) \times (m-(k-1))$ (We assume k is an odd number which is common for convolution kernels). Another common method is to pad the image with zeros [7] so that the convolution can be done for all elements in the image. This method preserves spatial size and thus it allows convolutional operators and down-sampling operators to be separate. Therefore, it is often used in deep networks. A one-dimensional example of zero padding can be seen in Section 2.3.

We can stack several convolutional layers after each other to get more complex features. The early layers may learn a set of useful primitive features, e.g edge detectors, and then the next layer takes the previous layer as input which increases the receptive field. A deeper network with small filter kernels has been shown to give better results than a shallower network with larger filter kernels [47]. One possible explanation for this is that the learned features in such a network are limited to filters that are a combination of smaller filters separated by non-linearities. This restriction might act as a form of regularisation (see

Section 2.7) since a deep network can have the same receptive field as a shallow one while requiring fewer parameters, thus constraining the model in some sense.

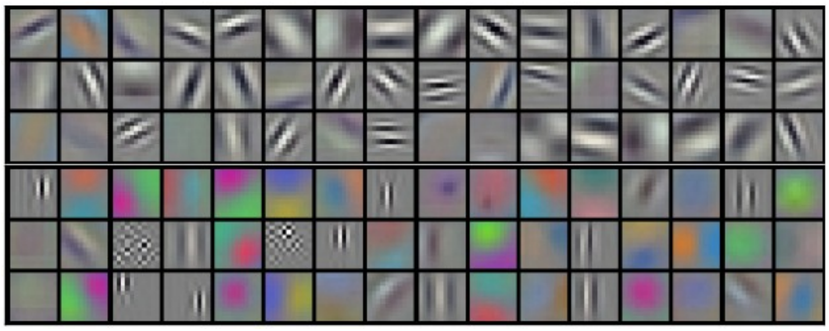


Figure 2.2: 96 Example filters from [31].

The filters are of size $11 \times 11 \times 3$ *and are learned by the first convolutional layer on* $224 \times 224 \times 3$ *input images. Each one of the filters is shared on every position of the image. This parameter sharing is intuitive. We see that, for example, some of the filters are edge detectors and it is reasonable to assume that if detecting edges is useful at some positions in the images, then it is also useful elsewhere. Thus it is unnecessary to learn from scratch to detect edges for every position in the image.*

To further reduce the number of parameters and increase the receptive field we can use strided convolutions. For a convolution with stride one, the kernel moves one pixel at a time over the input image, and the output has the same spatial size as the input (assuming zero padding at the edges). If the stride is two, the kernel jumps two pixels at a time when it slides over the input, and the output is downsampled to half the spatial size of the input. Section 2.3 shows an example of this. The stride can also be more than two, but this is uncommon in practice. An increased receptive field allows the features at the higher layers to cover a greater area of the image frame and the reduction in the number of parameters reduces memory overhead and helps to prevent overfitting.

Pooling layers also reduce the output spatially by looking at a small window of the image, commonly 2×2, at a time and replacing it with the results of a pooling operation. Pooling layers work independently on each depth slice, and the most common operator is the max operator [48]. Section 2.3 shows an example of the max pool operator. The reduced spatial size and number of parameters can help control overfitting.

2.4 Optimisation

Once we have decided on a machine learning model to use, we need to train it. By this, we mean to find the optimal parameters, $\mathcal{W}$, for the model. To do this, we use a loss function, $J(\mathcal{W})$, which tells us how "wrong" our model is for the

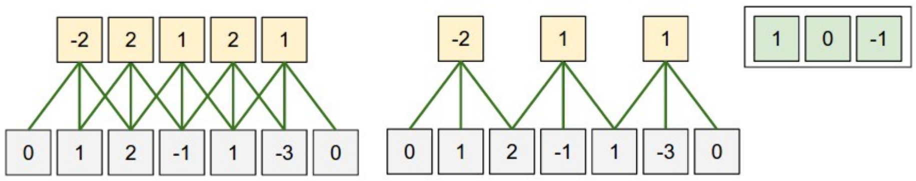


Figure 2.3: Example that shows the effect of different stride values.

For this illustration there is only one spatial dimension, the size of the filter is $k = 3$ and the input size is $n = 5$. The input is padded with zeros at the edges so that the convolution can be calculated for the entire image. The filter used is an edge filter shown to the top right. ***Left:*** *The stride is 1, and we get the same size for the output as for the input, 5.* ***Right:****The stride is 2, so the filter 'skips' one position and the output size is reduced to 3. (image from* `http://cs231n.github.io/convolutional-networks/`*)*

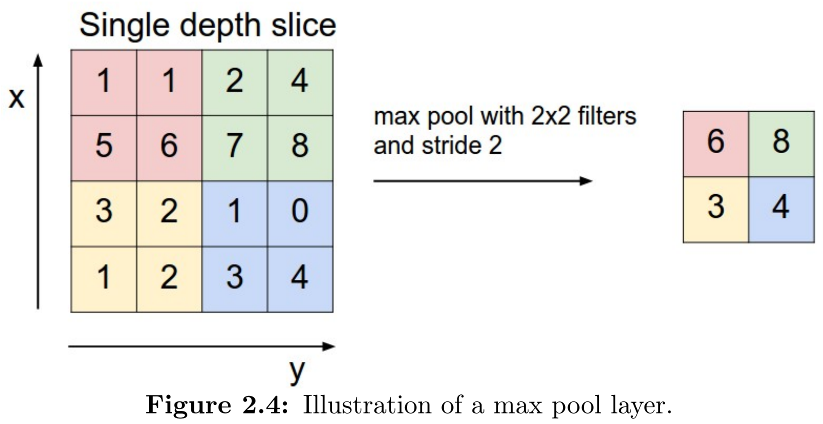


Figure 2.4: Illustration of a max pool layer.

The max pool operator replaces the numbers inside a small window with the maximum number. In this example, the window is 2×2 with stride 2 (image from `http://cs231n.github.io/convolutional-networks/`*)*

given parameters, $\mathcal{W}$. Then we try to minimise this loss function, i.e. we try to find the optimal parameters

$$\mathcal{W}^\star = \arg\min_{\mathcal{W}} J(\mathcal{W}) \tag{2.9}$$

One way to find this minimum is to use the fact that the gradient is zero for critical points and then to solve it analytically.

For example, for a simple regression model ($\hat{y_i} = w_1 x_i + w_0$), the loss function is the mean square error,

$$J(\mathcal{W}) = \frac{1}{2m}\sum_{i=0}^{m}(\hat{y_i} - y_i)^2, \tag{2.10}$$

and if insert the expression for $\hat{y}$, we get

$$J(w_0, w_1) = \frac{1}{2m}\sum_{i=0}^{m}(w_1 x_i + w_0 - y_i)^2, \tag{2.11}$$

where m is the number of samples, x_i is an input sample, y_i is the correct output for this sample and $\hat{y_i}$ is the corresponding prediction. The derivatives of this are

$$\begin{aligned}\frac{\partial J(w_0, w_1)}{\partial w_0} &= \frac{\partial}{\partial w_0}\frac{1}{2m}\sum_{i=0}^{m}(w_1 x_i + w_0 - y_i)^2 && (2.12)\\ &= \frac{1}{m}\sum_{i=0}^{m}(w_1 x_i + w_0 - y_i) && (2.13)\end{aligned}$$

and

$$\begin{aligned}\frac{\partial J(w_0, w_1)}{\partial w_1} &= \frac{\partial}{\partial w_1}\frac{1}{2m}\sum_{i=0}^{m}(w_1 x_i + w_0 - y_i)^2 && (2.14)\\ &= \frac{1}{m}\sum_{i=0}^{m}(w_1 x_i + w_0 - y_i)x_i && (2.15)\end{aligned}$$

When we know this, we can analytically find the minimum points by finding the parameters,w_0, w_1, where the derivatives are zero. We see that for this simple linear regression case, we only need to solve a quadratic equation. However, for more complex models and loss functions, it is not that easy. Because of this, we use iterative methods, like *gradient descent* to find a minimum for J instead of using analytic methods.

2.4.1 Gradient descent

In most practical cases we can not find the minimum analytically. Neural networks are one of the examples where this is the case [4]. However, even though we cannot solve for the minimum point directly, we can iteratively walk towards a minimiser of the loss function. The gradient of a function points towards the direction that the function increases the most. If we take a sufficiently small step in the opposite direction of the gradient, we move to a point where that function has a lower value. Then we can find the negative gradient of this new point and take a small step in this direction. If we continue like this, taking small steps in the opposite direction of the gradient, we will eventually converge at a local minimum or saddle point [12]. This method is known as gradient descent[20][6]. Section 2.4.1 shows an example of the gradient descent algorithm for one dimension.

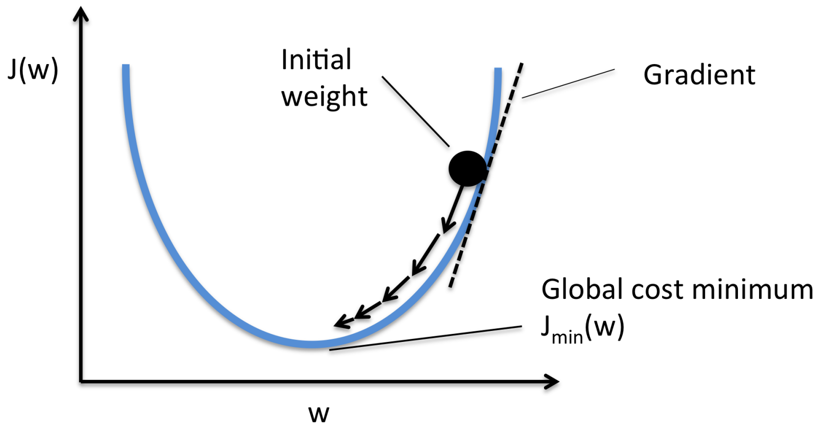


Figure 2.5: Illustration of gradient descent.

Here, the function we want to minimise is a function, $J(w)$, of just one scalar variable, w. (image from `https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html`)

The effectiveness of gradient descent is very dependent on the step length. A small step length will in most cases lead to consistent, but slow progress. We can get faster progress by taking longer steps, but we risk going too far and overshooting the target, fluctuate around the minimum and never converge. A substantial step length can increase the cost function. An optimal choice of step length, or learning rate, is as large as possible while still being small enough to converge nicely.

The update rule for gradient descent with learning rate α looks like this [20]

$$\mathcal{W}_t = \mathcal{W}_{t-1} - \alpha \nabla_{\mathcal{W}} J(\mathcal{W}_{t-1}). \tag{2.16}$$

Note that the magnitude of the gradient is higher when the function is steeper, which means that the step length decreases as $\mathcal{W}$ approach a critical point.

If the loss function is convex, gradient descent with a reasonable learning rate will always find a global minimum. Unfortunately, this is not the case for most loss functions. Even if it converges, there is no guarantee that we have found the global minimum. Gradient descent can easily get stuck at a non-optimal local minimum, or even a saddle point. Saddle points are often surrounded by a plateau of the same error and can be a more significant problem than local minimum points [12]. To avoid this, we have to be careful with how we initialise the gradient descent.

Choosing a proper learning rate is difficult, but essential for good convergence. There are a number of schemes that attempt to combat this by adaptively updating the learning rate, some of which will be described later in this text.

2.4.2 Stochastic gradient descent

Gradient descent needs to calculate the gradients at every update. For regular gradient descent we calculate the gradients for the entire dataset every single update. We call this *batch gradient descent.* If the dataset is large, batch gradient descent will be very slow and the full dataset might not fit in memory. An alternative algorithm is *stochastic gradient descent*[20].

Stochastic gradient descent uses only one training sample, $\boldsymbol{x_i}$, for each parameter update.

$$\mathcal{W} = \mathcal{W}_{t-1} - \alpha \nabla_{\mathcal{W}} J(\mathcal{W}_{t-1}; \boldsymbol{x_i}; \boldsymbol{y_i}) \tag{2.17}$$

If the dataset is massive, it often contains a lot of similar examples. This means that batch gradient descent performs unnecessary calculations when it recomputes similar gradients for every update. Stochastic gradient descent avoids this redundancy by making an update for every example. SGD is usually much faster than BGD, but it introduces considerable variance to the updates and makes the loss function fluctuate. This variance enables SGD to potentially "escape" sub-optimal local minima and jump to new and potentially better ones.[20] Though, it also complicates the convergence in general.

2.4.3 Mini batch gradient descent

Mini batch gradient descent [20] combines the best of SGD and BGD and uses a small subset of training examples for the updates. The update scheme for mini batch gradient descent is

$$\mathcal{W}_t = \mathcal{W}_{t-1} - \alpha \nabla_{\mathcal{W}} J(\mathcal{W}_{t-1}; \boldsymbol{x_{i:i+b}}; \boldsymbol{y_{i:i+b}}), \tag{2.18}$$

where b is the batch size, $\boldsymbol{x_{i:i+b}}$ is a batch of b samples and $\boldsymbol{y_{i:i+b}}$ is the corresponding batch of labels. This reduces the variance of the updates, but it is more computationally efficient than full batch gradient descent. The size of the minibatch can vary for different applications, and a typical batch size is often between 20 and 256.

2.4.4 Advanced first order optimisation algorithms

Momentum

Stochastic gradient descent struggles in areas where the surface of the loss function has a steeper slope in some dimensions than others. The gradient will be more substantial for the steeper dimensions, and so the update step will be large in this direction. The update may overshoot the minimum point and oscillate back and forth in this direction while moving slowly in the directions with a gentler slope. This is not an efficient way to update the parameters and can slow down convergence.

To combat this, momentum updates[43] adds a 'push' in the direction of the previous update. The new update can be written like this

$$\mathcal{V}_t = \gamma \mathcal{V}_{t-1} + \alpha \nabla_{\mathcal{W}} J(\mathcal{W}_t) \tag{2.19}$$
$$\mathcal{W}_t = \mathcal{W}_{t-1} - \mathcal{V}_t, \tag{2.20}$$

where $\mathcal{V}_t$ is the current update, and the momentum term γ is how much of the previous update we want to add, i.e. how big the momentum push is.

We can think of this like rolling a ball down a slope. The ball gains momentum downwards while it rolls, and moves faster and faster towards the bottom. In the same way, the parameter updates increase in the direction where the gradient is parallel with the previous update and decreases in the other directions. The momentum term,γ,can be thought of as a friction coefficient. This dampens the oscillations and results in faster convergence [43].

Nesterov momentum

Nesterov momentum [38] is similar to regular momentum updates but instead of calculating the gradient at the current position it 'looks ahead' and calculates the gradient for an approximation of the future position. From Equations (2.19) and (2.20), we know that the next parameter is given by $\mathcal{W}_t = \mathcal{W}_{t-1} - \mathcal{V}_t$ and we can approximate this with $\mathcal{W}_{t-1} - \gamma \sqsubseteq_{t-1}$. Then we can evaluate the gradient at the approximation which gives us the following update rule

$$\mathcal{V}_t = \gamma \mathcal{V}_{t-1} + \alpha \nabla_{\mathcal{W}} J(\mathcal{W}_{t-1} - \gamma \mathcal{V}_{t-1}) \tag{2.21}$$
$$\mathcal{W}_t = \mathcal{W}_{t-1} - \mathcal{V}_t \tag{2.22}$$

This anticipatory way of updating the parameters helps to avoid going too fast and overshooting near the minimum point[38].

Momentum and Nesterov momentum allows us to adapt the parameter updates to the slope of the loss function. However, it would be even more beneficial to adapt the updates to each parameter and make a larger or smaller update depending on the parameters importance.

Adagrad

Adagrad updates the learning rate for every time step for each parameter, $\mathcal{W}_i$, based on the gradients calculated at previous timesteps [8].

$$g_{i,t} = g_{i,t-1} + \frac{\partial}{\partial w_i} J(\mathcal{W}_{t-1})^2 \tag{2.23}$$

$$w_{i,t} = w_{i,t-1} - \frac{\eta}{\sqrt{g_{i,t-1} + \epsilon}} \cdot \frac{\partial}{\partial w_i} J(\mathcal{W}_{i,t-1}) \tag{2.24}$$

ϵ is added in the denominator to avoid dividing by zero. $g_{t,i}$ is a cache that keeps track of the sum of former gradients for the parameter $\mathcal{W}_i$ and is then used to adjust the learning rate for that parameter. If one direction has a large accumulated gradient, the learning rate becomes smaller. So the optimisation takes smaller steps for more frequent parameters. The benefit of Adagrad is that the learning rate adapts, so it is not necessary to manually tune it.

The problem with Adagrad is the cache. It accumulates the square gradients which are always positive, and so the cache will keep growing during training. Eventually, it will become very large, and hence the learning rate will become very small. Then the parameters will cease to update, and the model will no longer learn.

RMS prop

RMS prop [14] is similar to Adagrad but uses a cache that is an exponentially decaying average of squared gradients instead of a sum.

$$g_{i,t} = \gamma g_{i,t-1} + (1-\gamma) \frac{\partial}{\partial w_i} J(\mathcal{W}_{t-1})^2 \tag{2.25}$$

$$w_{i,t} = w_{i,t-1} - \frac{\eta}{\sqrt{g_{i,t} + \epsilon}} \cdot \frac{\partial}{\partial w_i} J(\mathcal{W}_{t-1}) \tag{2.26}$$

γ is the amount of decay for the average and is often set to 0.9 [14]. This prevents the learning rate to shrink radically and become infinitesimally small like for Adagrad.

Adam

Similarly to Adagrad and RMS prop, Adaptive moment estimation (Adam) also calculates adaptive learning rates for each parameter. Adam also keeps track of an exponentially decaying average of past squared gradient like RMS prop, but it also stores exponentially decaying average of past gradients [29].

$$m_{i,t} = \beta_1 m_{i,t-1} + (1-\beta_1)\frac{\partial}{\partial w_i}J(\mathcal{W}_{t-1}) \tag{2.27}$$

$$v_{i,t} = \beta_2 v_{i,t-1} + (1-\beta_2)\frac{\partial}{\partial w_i}J(\mathcal{W}_{t-1})^2 \tag{2.28}$$

m_t is an estimate of the first moment (the mean) of the gradient, and v_t is an estimate of the second moment (the uncentred variance) of the gradient. The square of the gradient is element-wise This is where Adam gets its name (Adaptive moment estimation). It has been observed that when m and v are initialised as zero, they are biased towards zero. Because of this, the authors behind Adam calculate biased corrected versions of the moment estimates.

$$\hat{m}_{i,t} = \frac{m_{i,t}}{1-\beta_1^t} \tag{2.29}$$

$$\hat{v}_{i,t} = \frac{v_{i,t}}{1-\beta_2^t} \tag{2.30}$$

These are the estimates that are then used for the updates:

$$w_{i,t} = w_{i,t-1} - \frac{\eta}{\sqrt{\hat{v}_{i,t}} + \epsilon}\hat{m}_{i,t} \tag{2.31}$$

β_1^t and β_2^t is normally 0.9 and 0.999. usually is 0.9 and 0.999. Adam is shown empirically to work well in practice, and it works comparatively better than other adaptive methods for parameter updates.

2.4.5 Initialisation of the parameters

Proper initialisation of the weight parameters is essential for the signal to reach deeper layers of the network. If the weights begin too small, then the output values will shrink for each layer until they are too small to be useful. Also, if the weights instead begin too large, then the output values will grow until they are too large to be useful. Good initialisation schemes help the weights to remain within a useful range throughout the network.

One method for initialising the weights is to just set all the weights to zero. The problem with this is that then all the neurons will give the same outputs. So then all the neurons in the network will have the same gradients and learn the same features. What we want is to break symmetry so that each neuron is initialised differently and learns a different feature.

To do this it is common to draw the weight values from a Gaussian distribution with zero mean and a small standard deviation. For example. This breaks symmetry and allows the features to be nicely distributed around zero. The problem with this approach is that the variance of the distribution for the output will grow when the number of inputs increases.

To prevent this, it is common to use a standard deviation of $\mathrm{STD}(W_i) = \sqrt{1/n_{\text{in}}}$, where n_{in} is the number of inputs to the layer.[17] Then the variance is

$$\mathrm{Var}(W_i) = \frac{1}{n_{\text{in}}}. \tag{2.32}$$

This helps to ensure that all neurons begin with approximately the same output distribution. In 'Understanding the difficulty of training deep feedforward neural networks' [17], Glorot and Bengio analyse the variance of both the output and the backpropagated gradients and suggests to use

$$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}} \tag{2.33}$$

In practise, this can be troublesome to implement because we need to know the number of neurons in the next layer that uses the output of the current one. So Equation (2.32) is more commonly used.

A more recent paper, 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification' [23], by He et al. builds upon the initialisation scheme proposed by Glorot and Bengio [17] and suggest to use

$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}}}. \tag{2.34}$$

This makes sense because a ReLU activation will zero out half the inputs, which means that one needs to double the size of the weight variance to cancel this and still get constant variance for the outputs.

2.5 Backpropagation

To train our network, we need to calculate the gradient of the loss function with respect to the parameters for every layer. To do this, we use backpropagation which is an efficient way of computing the gradient of a function with recursive application of the chain rule of calculus [20].

Let us look at a straightforward example. Say we want to compute the gradient of a two-layer neural network, z_2, given by

$$h_1 = w_1 x + b_1 \tag{2.35}$$
$$z_1 = \max(0, h_1) \tag{2.36}$$
$$h_2 = w_2 z_1 + b_2 \tag{2.37}$$
$$z_2 = \max(0, h_2) \tag{2.38}$$

Let us start by computing $\frac{\partial z_2}{\partial w_2}$;

$$\frac{\partial z_2}{\partial w_2} = \frac{\partial z_2}{\partial h_2}\frac{\partial h_2}{\partial w_2}. \tag{2.39}$$

When computing this, we see that the derivative of z_2 with respect to h_2 is given by

$$\frac{\partial z_2}{\partial h_2} = \begin{cases} 1, & h_2 > 0 \\ 0, & \text{otherwise} \end{cases}, \tag{2.40}$$

furthermore, the derivative of h_2 with respect to w_2 is given by

$$\frac{\partial h_2}{w_2} = z_1. \tag{2.41}$$

Combining this and Equation (2.39) we get

$$\frac{\partial z_2}{\partial w_2} = \begin{cases} z_1, & (w_2 z_1 + b_2) > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{2.42}$$

Now, observe that z_1 is in the expression for the derivative of z_2 with respect to w_2. The output of layer $n-1$ is almost always part of the derivative of a neural network with respect to a parameter in layer n. Therefore, we can save time if we want to compute many such gradients by first computing the output of the network and storing the outputs of all the intermediate layers.

Let us now differentiate z_2 with respect to a variable in the first layer. To do this we use the chain rule and get

$$\frac{\partial z_2}{\partial w_1} = \frac{\partial z_2}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{h_1}{w_1}. \tag{2.43}$$

Note here that we have already computed $\frac{\partial z_2}{h_2}$ when computing $\frac{\partial z_2}{\partial w_2}$. Therefore, we can save time on computations if we compute the derivative of z_2 with respect to w_1 after the derivative with respect to w_2. Again, this behaviour is seen for neural networks in general. Whenever we compute the derivative of parameters in layer n, we need to compute some of the same terms as when we computed the derivative of parameters in layer $n+1$.

We will not bother writing out the full expression for this derivative, but rather summarise the key concept of backpropagation. Backpropagation is a method of computing the gradient of a neural network. It works by first computing the output of the network and storing the outputs of each layer. This output is then used to compute the derivative of the network with respect to all parameters in the final layer, intermediate terms are stored and used to compute the derivative with respect to the second to last layer. This process is then repeated until the full gradient of the network is computed.

2.6 Splitting the dataset for training, testing and validation

We need some method to evaluate the performance of our model, both for fine-tuning and to have an idea of how well it will perform on unseen data. If we measure the effectiveness of our model on the training data, we will likely get an overly optimistic result. This is because the model can memorise aspects of the training data rather than learning generalisable features [20]. To get a better evaluation of the model we split the dataset into two parts. One part used to train the model, the training set, and one part used to evaluate the model and select hyperparameters, the validation set.

In addition to a separate dataset for fine-tuning the model, we need a third dataset that is separate from both the training and validation set and that we only use to evaluate the finished model. When we fine-tune with the help of the validation set, we indirectly give the model information about validation data. For example by selecting a hyper-parameter because it yields a high validation score. Then, even though the model never trains on the validation data it will still adjust to fit these images, and if we evaluate the model on the validation data, we might overestimate its performance. Therefore, to get a better estimate on how well the model performs on unseen data, we need a separate test set that we keep hidden until we have finished tuning and training.

2.7 Regularisation

A central challenge for machine learning in general and neural networks, in particular, is to create a model that will perform well on not just the data provided during training, but also on new inputs. A complicated model may adapt too well to the training set and possible memorise aspects of the training data that is not relevant to the task [20]. This overfitting will result in a low error on the training data, but a high error on other data. There are many strategies that are designed to reduce the test error to improve generalisation. In machine learning, we call these strategies for regularisation. There is a large number of different regularisation methods, and we will focus on two popular methods that are relevant to our problem, early stopping and dropout.

2.7.1 Early stopping

When the model we are training is complex enough to have the capacity to overfit to the training data, we may observe the validation loss start to increase again after decreasing for a while, even though the training loss is still decreasing. This increase in validation loss happens when the model begins to overfit, and we can combat this by stopping training before the validation loss increases. This early stopping works as a hyperparameter selection, where the number of training steps is the hyperparameter we want to fine tune. A simple way to implement this is to store a copy of the model weights at some points during training. Then we can compare the validation loss for different iterations and chose the model weights from the iteration with the lowest validation error. Early stopping is a computationally inexpensive strategy, and the only cost is the memory it takes to store the model weights. However, this simple strategy helps reduce the chances of overfitting and improves generalisation[20].

2.7.2 Dropout

Dropout is a simple and computationally inexpensive way of reducing the chances of overfitting for neural network models [20, 49]. Combining the predictions of an ensemble of different models is a very efficient way to reduce test errors [31, 51].However, training a deep neural network is very time consuming, and so combining several different trained models is often impractical. Dropout approximates an ensemble of different models by setting the output of each hidden neuron in a layer to zero with a set probability (often around 50% [49]). The "dropped out" neurons do not contribute to the forward pass or backpropagation. This means that for every iteration, the network samples a slightly different sub-architecture, but all these architectures share weights. When evaluating the model, we use all the neurons, but we multiply their outputs by the dropout probability (i.e. 0.5 if the dropout probability is 50%). This gives us a reasonable approximation to taking the mean of the predictions produced by the many subnetworks trained by "dropping out" random nodes.[49]

Dropout also regularises each hidden neuron to be a useful feature that is helpful in many contexts. A neuron cannot rely on the presence of specific other neurons, since they may be "dropped out", and therefore it is forced to learn features that are beneficial in combination with several different subsets of the other neurons. This leads to more robust features that helps the model generalise better to unseen data and helps to prevent overfitting [31, 25, 49].

2.8 Batch normalisation

During training, the distribution of the input to every layer of a neural network changes every time the parameters of the previous layers updates. This change in distribution is known as *internal covariance shift*[28]. Internal covariance shift complicates training because all the layers have to keep adapting to this new distribution. Moreover, because the input to a layer is affected by the parameters for all the previous layers, small changes will be amplified for layers deeper into the network. Thus, this problem is particularly severe for deep neural network models.

In 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'[28], Ioffe and Szegedy et al. introduce a method to reduce internal covariate shift by normalising the input to each layer to keep the distribution constant. During training, this can be done by calculating the mean, $\mu_{\mathcal{B}}$, and variance, $\sigma_{\mathcal{B}}$, and scale the input by these values like so

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.44}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \tag{2.45}$$

$$\hat{x_i} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{2.46}$$

$$\mathrm{BN}_{\gamma,\beta}(x_i) = \gamma\hat{x_i} + \beta \tag{2.47}$$

Where $\mathcal{B}$ is the mini-batch we use during training, m is the size of this mini-batch and ϵ is added to the denominator in Equation (2.46) to avoid division by zero. We scale by two additional parameters α and γ to allow the network to learn different distributions. When applying batch normalisation to convolutional layers, we want different locations of the same feature map to be normalised the same way, so we define $\mathcal{B}$ to consist of the all the elements across both the mini batch and the spatial locations. The transformation defined by Equations (2.44) to (2.47) is differentiable and can be backpropagated through during training [28]

During evaluation of the network, we can normalise based on the mean and variance for the entire dataset instead of just a mini-batch. To be able to evaluate the model while it trains we can use moving averages to update these values.

Ioffe and Szegedy showed that applying batch normalisation accelerated training and improved classification accuracy compared to similar models without batch normalisation. Additionally, normalising layer inputs throughout the network prevents small adjustments to amplify into greater and suboptimal changes in the gradients that are used to update the parameters, and this allows for larger learning rate which can further speed up convergence. So batch normalisation can lead to accelerated training just by itself but also by enabling a higher learning rate.

Batch normalisation can also work as a form of regularisation. During training for a model with batch normalisation, a sample will be normalised together with the other samples in the mini-batch. This means that while training, the model will not produce the same deterministic value for a given training sample. Ioffe and Szegedy found that this gave a beneficial regularising effect and reduced the need for other regularisation strategies like dropout.

2.9 Architectures for convolutional neural networks

A popular architecture for image classification is the VGG architecture introduced by Simonyan and Zisserman in 'Very Deep Convolutional Networks for Large-Scale Image Recognition' [47]. The VGG architecture focuses on layers with small filter kernels stacked after one another in a deep neural network and uses exclusively convolutional filter kernels of size 3×3. Simonyan and Zisserman reasoned that a combination of two 3×3 kernels has the same receptive field as one 5×5 kernel. This means that we can get similar results as for a larger filter kernel, but we keep the advantages of smaller filter kernels. Among the advantages are a smaller number of parameters, which makes it possible to add a higher number of layers without the number of parameters growing to a size that is unfeasible to train. This type of architecture also has more non-linear activation functions which adds more opportunities for non-linearity. So, the VGG network architecture is a simple, but effective architecture and it has been shown to achieve excellent results on both localisation and classification tasks [47].

The main downside of the VGG architecture is the final fully connected layers. There are two main reasons for this, firstly, the fully connected layers requires the output images of the final convolutional layer to be reshaped into a vector. This reshaping requires the output of the final convolutional layer to always have the same size, which again, requires that the input image is always of the same size. The other, and probably the main, downside of the VGG architecture is the sheer amount of parameters in the fully connected layers. There are, after removing the final fully connected layer, over 16 million parameters in the fully connected layers of the VGG architecture [47]. Which opens for a high degree of overfitting [34, 48].

Another popular method for designing neural networks to solve image recognition problems is by using so-called fully convolutional architectures. In a fully convolutional architecture, all the layers are convolutional layers as opposed to mixing convolution layers with some intermediate max-pooling layers and some concluding fully connected layers (preceded by a flattening of the image into a vector) [34, 48]. The main benefit of these algorithms is that they have relatively low complexity and are amongst the convolutional neural network algorithms that are the easiest to implement.

There are two main parts of a conventional convolutional network that need to be replaced in a fully convolutional network, downsampling via max-pooling and the fully connected layers at the end. The way downsampling is performed in a fully convolutional network is generally through *strided convolutions*, described in Section 2.3. Fully connected layers can be replaced in fully convolutional networks in two ways. They can either be replaced by a convolution of the same size as the image, or, more commonly used, through a global average pool followed by a 1×1 convolution [48, 26]. A global average pool simply consists of taking the average over all spatial positions in the image. The reason why a global average pool followed by a 1×1 convolution is used is that it allows the network to be used to classify images of varying dimensions.

2.10 Recurrent neural nets

Recurrent neural nets are a type of neural nets that are particularly suited to model sequences of data and thus highly relevant for ultrasound images [20]. Like CNNs, RNNs utilise parameter sharing. Convolutional networks reuse the convolution kernels at different spatial positions in the image and RNNs use the same weights at different temporal positions in the sequence. The state at each step is a function of both the current input and the previous state.

$$h_t = f_W(h_{t-1}, x_t), \tag{2.48}$$

where h_t and x_t is state and input at step t. This allows the model to remember previous states and use that to affect the next prediction which is particularly useful for sequences. For example, when classifying a word in a sentence, the previous word might provide useful context. Moreover, perhaps unsurprisingly, RNNs have provided excellent results for natural language processing problems[20].

For the most straightforward recurrent net, f_w is a linear combination of the previous state, h_{t_1} and the current input x_t followed by a tanh activation function.

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \tag{2.49}$$

If we want to output a prediction for every time step for example when classifying each frame in a series of frames, we can get y_t like this

$$y_t = W_{hy}h_t \tag{2.50}$$

Section 2.10 shows the connections in a simple recurrent net.

tanh is used for RNNs because it squashes the output values between -1 and 1. ReLU does not limit the size of the values. For every step, we multiply with the same weight matrix. If the largest eigenvalue of the weight matrix is larger than one, the values might increase with each multiplication and eventually explode. The values are guaranteed to explode if the smallest eigenvalue of the weight matrix is less than one. The use of tanh stops the values from exploding this way.

The gradient may still explode when propagating backwards. Just like the values are multiplied by the weight matrix at each step when propagating forwards, the

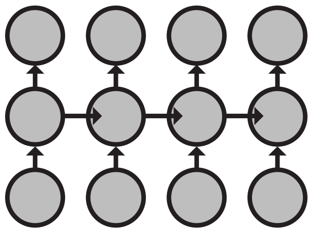


Figure 2.6: Example of the structure of a simple RNN over four time steps

The bottom row is the inputs, the middle row is a layer of hidden states with recurrent connections and the top row show the outputs for each time step

gradient is also multiplied by the weight matrix when propagating backwards. to simply crop the gradients if they are above some threshold. This gives the effect of setting up a wall that prevents huge jumps when updating the parameters[42].

2.10.1 Gated recurrent units

RNNs have trouble learning long-range dependencies[10]. This is because the information is disappearing exponentially between steps. After only a few time steps the information can disappear altogether, and inputs from far away will not contribute to what the network learns. Gated Recurrent Units (GRU) [10] tries to solve this problem by adding a gate that controls how much of the previous state should be remembered and how much of the current candidate state should be added. The new state is a linear combination of the previous state and the current candidate state. Equations (2.51) - (2.55) show the formulas for an RNN with GRU units.

$$z_t = \sigma(x_t U^z + s_{t-1} W^z) \tag{2.51}$$
$$r_t = \sigma(x_t U^r + s_{t-1} W^r) \tag{2.52}$$
$$h_t = \tanh(x_t U^h + (s_{t-1} \circ r_t) W^h) \tag{2.53}$$
$$s_t = (1 - z_t) \circ h_t + z_t \circ s_{t-1} \tag{2.54}$$
$$y_t = W_{sy} s_t \tag{2.55}$$

Where $\circ$ is element-wise multiplication, σ is a logistic sigmoid function, s_t is the new state, and h_t is the current candidate calculated similarly to a regular recurrent unit. z is the gate that tells us how much of the previous state should be remembered and how much of the current candidate should be included. The new state is a linear interpolation between the previous state and the candidate. r tells us how much of the previous state should be included in the current candidate. Finally, W_{sy} is the weights transforming the current state to the current output.

This architecture gives the network the possibility to chose what to remember, which makes it easier for the network to remember previous states further back. The GRU cells just add to or remove from the previous state instead of transforming the state completely

2.10.2 Bidirectional recurrent neural net

The simple recurrent network architecture in Equation (2.49) can only represent causal relationships in time. A prediction for one time step, y_t is only dependent on the model's previous inputs, $x_1, x_2, \ldots, x_{t-1}$ and the current input x_t. In some circumstances, it is helpful to also include information about future input. Bidirectional neural nets do precisely this; they combine two recurrent nets on top of each other [20]. One network moves forwards in time from the beginning of the sequence to the end, and one moves backwards from the end of the sequence to the beginning. Then, both the results are combined to get the output for the current time step.

$$h_t^f = f_W^f(h_{t-1}^f, x_t) \tag{2.56}$$

$$h_t^b = f_W^b(h_{t-1}^b, x_t) \tag{2.57}$$

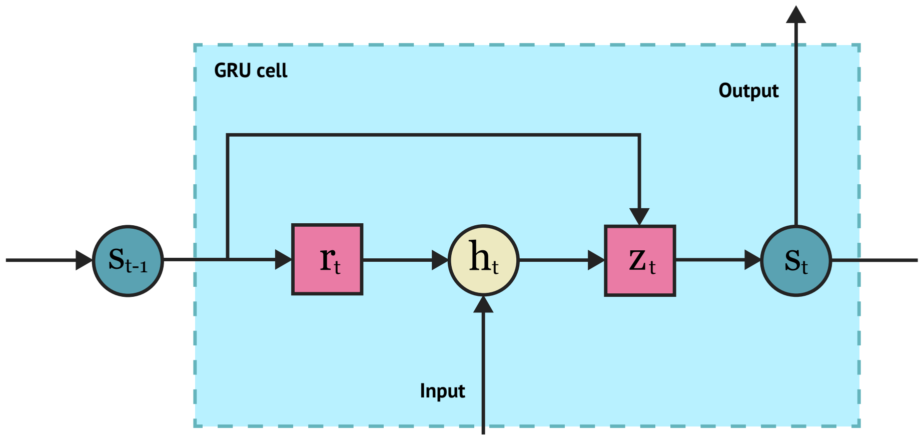
$$y_t = g(h_t^f, h_t^b) \tag{2.58}$$

h_t^f is the forward state, and h_t^b is the backward state. This architecture can be useful if the state we want to classify is dependent on both previous and future states. E.g. when classifying the state of the valves in an ultrasound image. For example, if a valve is open both in the near future and the near past, then it is most likely also currently open, and we know that the valvular events follows a particular pattern.

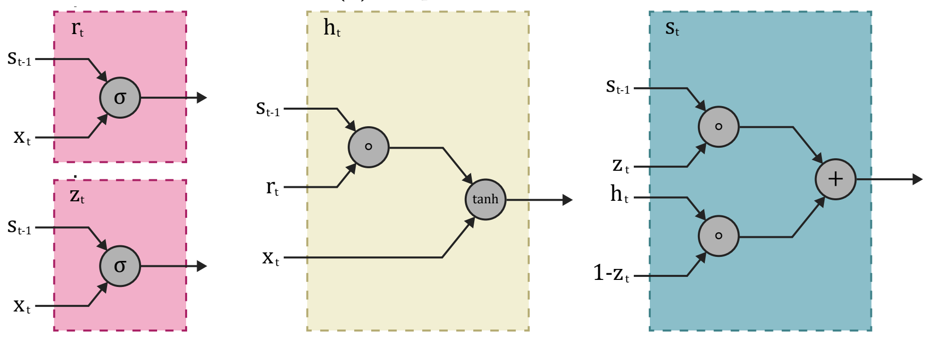
2.11 Visualisation of neural networks

Neural networks are known as *black box models* [9] and it is beneficial to have an understanding of how machine-learning algorithms make decisions before adapting them. In our case, for example, we are interested in knowing whether the network finds and focus on the valves in the frames. We mainly consider two such methods; *image occlusion* [54] and *guided backpropagation* [48].

Image occlusion was introduced by Zeiler and Fergus in 'Visualizing and Understanding Convolutional Networks' [54] with the goal of discovering what parts of an image are important when the network tries to classify it. More precisely, we place a grey box at every possible position in the image and use the network to classify the occluded images. Both the predicted labels and the label probabilities for the original class are stored. Using this, we can plot a heatmap that shows how certain the predictions are as a function of where the grey box is centred. Likewise, we can plot the class labels as a function of where the



(a) Simplified GRU cell



(b) Components of a GRU cell

Figure 2.7: Illustration of a GRU cell

Figure 2.7a shows a simplified GRU cell that illustrates how the information "flows" through the cell. The inputs that "alter" the r and z gates is omitted here. Figure 2.7b shows the inputs, outputs and inner workings of each of the GRU components. s_t *is the new state calculated as a linear interpolation between the previous state and the candidate.* h_t *is the current candidate state that is determined by the input and the previous state.* z_t *is a gate that tells us how much of the previous state should be remembered by the new state and how much of the current candidate should be included and* r_t *is a gate that tells us how much of the previous state should be included in the current candidate.*

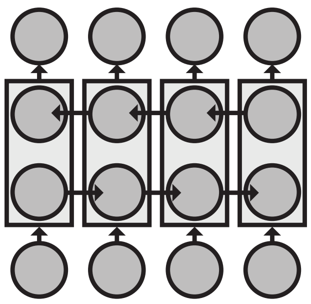


Figure 2.8: Example of the structure of a simple bidirectional RNN.

The bottom row is the input. One set of hidden states has forward recurrent connections (second row from bottom), and one set of hidden states has recurrent connections that go backwards (third row from bottom). The output at each step depends on both (top row).

grey box is to get an understanding of how occluding different parts of an image changes the predictions.

Before we introduce guided backpropagation [48], we describe the visualisation methods it builds on, saliency maps [46] and (the somewhat misleadingly named) deconvnet [54]. The saliency maps algorithm is performed by simply differentiating a class probability with respect to the input image. This gives a heatmap that shows how we can change the image to most efficiently change the probability of the chosen class. The output of this heatmap, thus, in some way demonstrate the parts of the image that are important and the parts that are not.

Deconvnet, on the other hand, works by modifying the way the gradient "flows" through the ReLU nodes. When we differentiate the output probability with respect to the input image, we have to differentiate the ReLU nodes. The gradient of a ReLU function is, in reality, 0 for all pixels that were rectified during the forward pass and 1 for all pixels that were not rectified. However, Springenberg et al. show that setting the "derivative" of the ReLU function to be a ReLU function itself and using this modified "derivative" during backpropagation yields a less noisy output image than saliency maps [48]. One way to interpret this is that the deconvnet algorithm only displays how pixels positively influenced neurons in the network.

One problem with deconvnet is that the output of neurons that were rectified during the forward pass still influence the way we perform visualisation. This can lead to noisy visualisations [48]. The way guided backpropagation solves this is by combining saliency maps and deconvnet. One essentially sets the "derivative" of the ReLU function ϕ to be $\phi'(x) = \phi'(\phi(x))$. Thus the "derivative" of a ReLU nonlinearity is set to 0 if the value of a pixel were rectified on the forward pass or if it were negative during the backward pass [48]. See Figure 2.9 for a more detailed demonstration of how "differentiating" a ReLU nonlinearity differs between guided backpropagation and deconvnet and Figure 2.10 for a

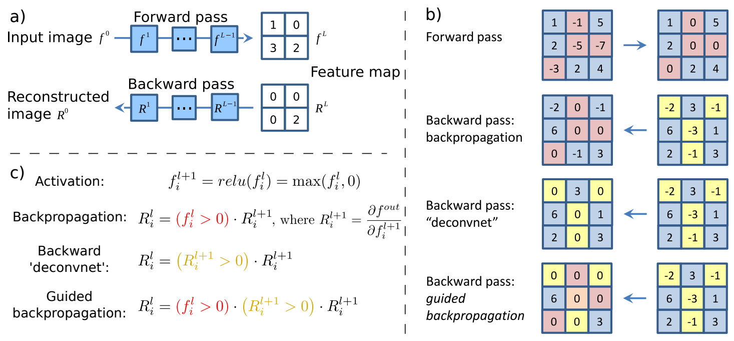
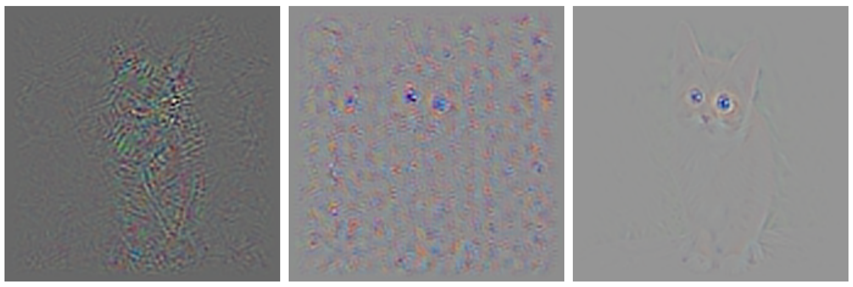

a)
Input image f^0
Forward pass
f^1
f^{L-1}
f^L
Feature map
Reconstructed image R^0
Backward pass
R^1
R^{L-1}
R^L
b)
Forward pass
Backward pass: backpropagation
Backward pass: "deconvnet"
Backward pass: guided backpropagation
c)
Activation: $f_i^{l+1} = relu(f_i^l) = \max(f_i^l, 0)$
Backpropagation: $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$, where $R_i^{l+1} = \frac{\partial f^{out}}{\partial f_i^{l+1}}$
Backward 'deconvnet': $R_i^l = (R_i^{l+1} > 0) \cdot R_i^{l+1}$
Guided backpropagation: $R_i^l = (f_i^l > 0) \cdot (R_i^{l+1} > 0) \cdot R_i^{l+1}$

Figure 2.9: Illustration of how the gradient flows through a ReLU unit in saliency maps, deconvnets and guided backpropagation 'Striving for Simplicity: The All Convolutional Net'.



(a) Saliency map **(b)** Deconvnet **(c)** Guided backprop.

Figure 2.10: Comparison of different gradient-based visualisation methods [48].

comparison of the different gradient-based visualisation methods.

2.12 TensorFlow

We use Tensorflow for the implementation of the neural net. TensorFlow is an open source software library for numerical computation and machine learning[52]. TensorFlow was developed by researchers and engineers at Google for research in machine learning and deep neural networks. The library uses dataflow graphs for the calculations, the graph nodes represent mathematical operations, and the edges represent the tensors passed on between them. The framework performs computation efficiently and can deploy on one or more CPUs or GPUs to further speed it up. TensorFlow is also well documented, easy to use and can be controlled by a simple Python API which makes it a great choice for this project.

Chapter 3

Methods

3.1 The dataset

The dataset used in this project consisted of 241 series of ultrasound images of the heart. Each series contained between 39 and 307 frames and depicted the motion of the valves during the cardiac cycle as described in Section 1.1 on page 14. The mean number of frames per series was 179, and each series contained between one and four cycles. Each frame is an image of size 256x256 pixels, and there were approximately 30 000 frames in total. The frames were all apical long-axis images that showed the opening and closing of the mitral valve and the aortic valve. Section 3.1 shows an example image from the dataset.

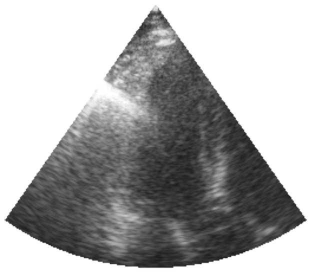


Figure 3.1: Example APLAX ulstrasound image from dataset

Our goal was to detect the valvular event frames from these sequences of frames with a convolutional neural net. A simple first step was to classify each frame as one of three possible states. Because the frames contain two valves, the aortic valve and the mitral valve, and only one can be open at a time, the only valid states are *both valves closed*, *aortic valve open* and *mitral valve open.* These states correspond to three possible classes which were used for image classification of the frames. Class 0 was defined to be *both valves closed* (BVC), class 1 to be *aortic valve open* (AVO) and class 2 to be *mitral valve open*

(*MVO*).

The dataset was manually labelled to provide ground truth labels for supervised machine learning. A labelling tool was developed in Jupyter notebook, and we manually assigned a class to each frame in 66 of the series yielding a total of 11626 frames (this labelling tool will be explained in detail in Section 3.1.3 on page 52). In total, 94 series were examined, and 28 series were excluded.

3.1.1 Training, validation and test-set

We split the labelled data into training, validation and test sets as described in Section 2.6 on page 39. The training set consisted of 44 series, while the validation and test sets consisted of 11 series each. To ensure uncorrelated training, validation and test sets, we separated the data by series and not by frames.

3.1.2 The classes

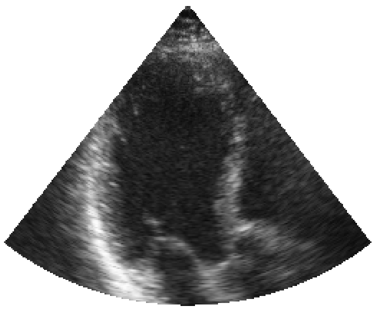
All the images in the dataset were apical long axis view and showed the left ventricle as well as the mitral valve and aortic valve. Most of the images had the valves in roughly the same position, at the bottom of the images towards the left. However, some of the images were more zoomed out and had a different placement of the valves. The images were never mirrored, and the mitral valve was always located to the left of the aortic valve. Section 3.1.2 shows some examples of typical valve position and size.

Some of the series had to be excluded. The exclusion criteria were: One or more of the valves not visible within the image, wrong cardiac view, or poor image quality. The series that met one or more of the criteria for exclusion were manually removed from the dataset during the labelling process. In total 28 of the 94 examined series were excluded.

After studying the images, three classes were found to be consistent throughout the dataset. These were defined by whether each of the two valves were open or closed in the frame. Each valve can either be open or closed and if one of the valves is open, the other has to be closed. Both valves can never be open at the same time so there was only three possible classes. Each frame can only belong to one of the classes. The definition of the three classes and an overview of their characteristics are provided here.

Both Valves Closed (Class 0)

Both valves are closed. Two examples of frames belonging to this class is shown in Section 3.1.2. This class occurred between the closing of one valve and the opening of the other, i.e. during iso-volumetric contraction and iso-volumetric


(a) Typical echocardiogram with the valves towards the bottom part of the frame

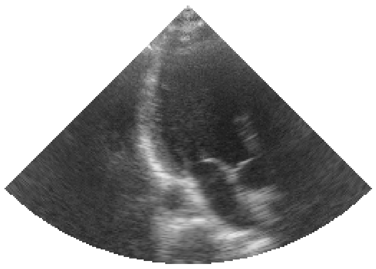
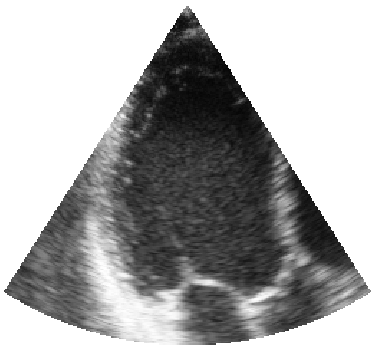

(b) More zoomed out echocardiogram. The valves are smaller and higher up in the image

Figure 3.2: Echocardiograms with valves of different sizes and at different locations

relaxation. This state generally lasted for between 3 and 8 frames. As a consequence of this short duration, most of the frames belonging to this class showed the heart when the valves were close to a transition between being open or closed.


(a) Echocardiogram with both valves clearly closed

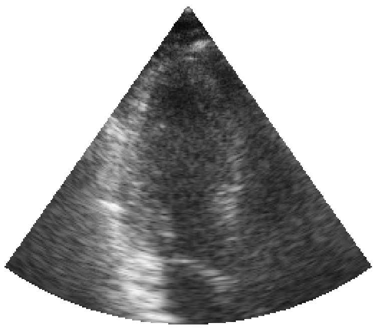
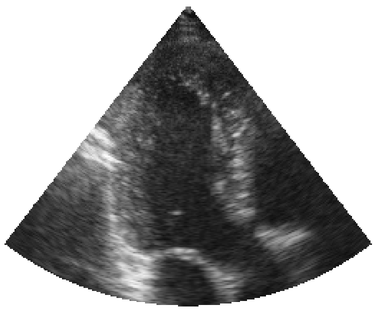

(b) Echocardiogram where shadows makes it difficult to tell that the aortic valve has closed

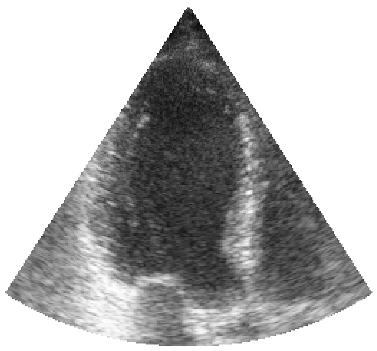
Figure 3.3: Two examples of frames where both valves are closed

Aortic Valve Open (Class 1)

During ventricular ejection, the aortic valve opens. An example of two frames belonging to this class is displayed in Section 3.1.2. The aortic valve is smaller than the mitral valve. And the opening and closing of the valve happens in a quick and small movement. The aortic valve also spends less time open than



(a) Echocardiogram where the aortic valve clearly is open



(b) Echocardiogram where noise gives appearance of a closed aortic valve

Figure 3.4: Two examples of frames where the aortic valve is open

the mitral valve, so the duration of class 1 is shorter than class 2.

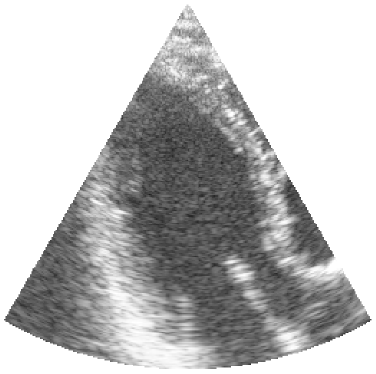
Mitral Valve Open (Class 2)

This class occurs during ventricular filling when the mitral valve is open. An example of two frames belonging to this class is showed in Section 3.1.2. The mitral valve opens with a large movement and stays open for a longer time than the aortic valve. The valve is also reasonably large and easily detected in the image. There is often a "dip" during ventricular filling where it visually appears as if the valve is about to close before it quickly opens again. After this "dip", the valve stays open for some more time before it truly closes. An example of this phenomenon is presented in Figure 3.4b

3.1.3 Labelling tool

To label the data we developed a tool to browse all the frames and determine the state of the valves. The tool was developed using a Jupyter Notebook with interactive widgets. Section 3.1.3 shows a screenshot of our final labelling tool. In the screenshot, we see one frame as well as our Jupyter widget toolbar. To navigate the data efficiently, we created several buttons. We made two buttons, "next" and "previous" to move forwards and backwards in the dataset. There is also a button "Go to idx" to go directly to the frame for a given index, and a button, "Go to series", to go directly to the first frame of a given series.

For the actual labelling, two toggle buttons were created. One for each valve with the states "Closed" and "Open". The person labelling the images can use these buttons to indicate the state of the valves in the frame. The states from these buttons are converted to a classification label corresponding to one


(a) Echocardiogram where the mitral valve clearly is open

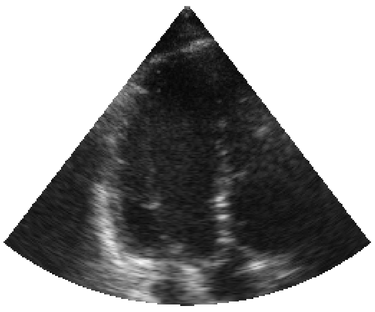

(b) Echocardiogram where the mitral valve appears to close before it actually does

Figure 3.5: Two examples of frames where the mitral valve is open

of our three classes, "both valves closed (class 0)", "aortic valve open (class 1)" and "mitral valve open (class 2)". If both valves are marked as open, an error message appears because this is not a valid state. We also included a field to display and enter the label directly as the number value.

To make the labelling process more efficient (taking advantage of the fact that two adjacent frames were likely to have the same label), we created a "sticky next" button that, when toggled, automatically labels a frame with the same label as the previous frame. This automatic labelling allows the person labelling to indicate the label only when it changes, and just click "next" when the label stays the same. We found that this considerably reduces the number of clicks, and thus also the time it takes to go through and label the data.

It can, be difficult to determine the class of a frame if we do not have information about the previous or following frames. To help with this, we chose to also display a visualisation of the labels for the entire current series below the frame. An example of this visualisation is shown towards the bottom of Section 3.1.3. The person labelling can look at this visualisation instead of clicking back and forth when determining the label for a problematic frame. This helps to reduce the number of clicks it takes to label a frame, thus further speeding up the labelling process.

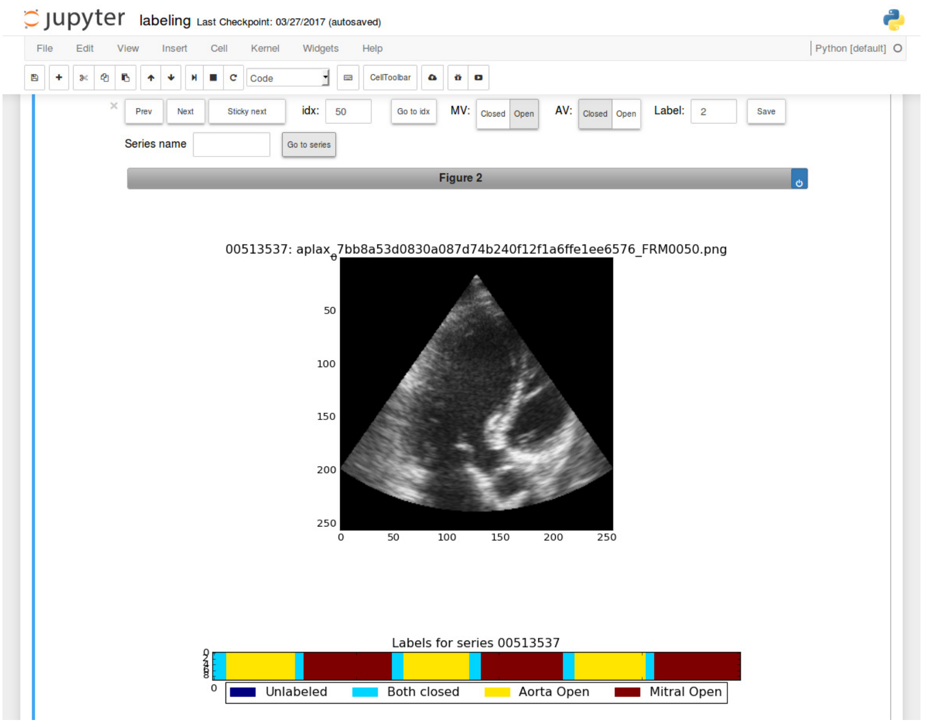

jupyter labeling Last Checkpoint: 03/27/2017 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python [default]
Code
CellToolbar
Prev
Next
Sticky next
idx: 50
Go to idx
MV: Closed Open
AV: Closed Open
Label: 2
Save
Series name
Go to series
Figure 2
00513537: aplax_7bb8a53d0830a087d74b240f12f1a6ffe1ee6576_FRM0050.png
Labels for series 00513537
Unlabeled
Both closed
Aorta Open
Mitral Open

Figure 3.6: Screenshot of the labelling tool used to label the data.

Towards the top, we see the Jupyter widget toolbar we created to click through and assign labels to the frames, in the middle we display the current frame being labelled, and towards the bottom, we display a visualisation of the labels for the entire cycle

Type	Output Filters	Size/Stride	Output Size
Conv1	32	3x3/1	252x252
Conv2	64	3x3/2	126x126
Conv3	64	3x3/2	63x63
Conv4	128	3x3/2	32x32
averagepool	128		1x1
dropout			
conv5	3	1x1/1	3
softmax			3

Table 3.1: Network architecture for a fully convolutional network with five layers

3.2 Classification with convolutional neural nets

3.2.1 Fully convolutional architectures

We first chose a straightforward strategy to detect the valvular event frames, and fed in a single frame at a time to a convolutional neural net, which classified it as either both valves closed (class 0), aortic valve open (class 1) or mitral valve open (class 2). From the classification results, one can then detect the frames where the valves open or close as the frames were the class label changes. Several convolutional neural network architectures were trained and validated on the training set and validation set. All architectures were designed with a low memory overhead to work on the hardware we had available which had limited RAM.

For single frame classification, three fully convolutional architectures (described in Section 2.9 on page 42) were explored with 5, 9 and 11 layers. All architectures used batch-normalisation (described in Section 2.8 on page 40) to normalise the distribution of the input for each layer. Batch-normalisation was applied after the ReLU activation to account for the fact that the average activation will be positive the ReLU nonlinearity. Thus, by performing batch-normalisation after this, the network has a chance to set the mean activation back to zero. A dropout layer (described in Section 2.7.2 on page 40) was implemented before the final layer for regularisation. Tables 3.1 to 3.3 show the architectures in detail and Table 3.4 show the hyperparameters used by all the models.

3.2.2 VGG-like architectures

The VGG architecture is, as described in Section 2.9 on page 42, a popular architecture for image classification. However, the two fully connected layers lead to large memory requirements which made it unfeasible to use with our equipment. We, therefore, decided to test one architecture inspired by the VGG11 architecture, but with significantly smaller filters and fully connected layers. Additionally, a high degree of downsampling was used to reduce the

Type	Output Filters	Size/Stride	Output Size
Conv1	32	3x3/1	252x252
Conv2	64	3x3/2	126x126
Conv3	64	3x3/2	63x63
Conv4	128	3x3/2	32x32
Conv5	128	3x3/2	16x16
Conv6	256	3x3/2	8x8
Conv7	512	3x3/2	4x4
Conv8	512	3x3/2	2x2
averagepool	512		1x1
dropout			
conv9	3	1x1/1	3
softmax			3

Table 3.2: Network architecture for a fully convolutional network with nine layers

Type	Output Filters	Size/Stride	Output Size
Conv1	32	3x3/1	252x252
Conv2	64	3x3/2	126x126
Conv3	64	3x3/2	63x63
Conv4	128	3x3/2	32x32
Conv5	128	3x3/2	16x16
Conv6	128	3x3/2	8x8
Conv7	256	3x3/2	4x4
Conv8	512	3x3/2	2x2
Conv9	512	3x3/2	1x1
Conv10	512	3x3/1	1x1
averagepool			
dropout			
conv11	3	1x1/1	3
softmax			3

Table 3.3: Network architecture for a fully convolutional network with eleven layers

Hyperparameter	Value
Batch size	32
ADAM-η	0.001
ADAM-β_1	0.9
ADAM-β_2	0.999
Dropout-p	0.5
Random crop size	252×252
Initialisation	Glorot

Table 3.4: Hyperparameters used when training all the networks.

Type	Output Filters	Size/Stride	Input Size
Conv1	32	3x3/1	252x252
Conv2	64	3x3/2	126x126
Conv3	64	3x3/2	64x64
Conv4	128	3x3/2	32x32
Maxpool1	128	3x3/1	32x32
Conv5	128	3x3/1	32x32
Conv6	128	3x3/2	16x16
Conv7	256	3x3/2	8x8
Maxpool2	256	2x2/1	8x8
Conv8	512	3x3/2	4x4
Conv9	512	3x3/2	2x2
Conv10	3	3x3/1	2x2
FC1	-	-	12
dropout			
FC2	-	-	3
softmax			3

Table 3.5: VGGlike13-f5 architecture

model's memory footprint. Our VGG-like architecture used 16 neurons in the hidden layer, instead of 4096. The architecture is described in detail in Table 3.5.

3.2.3 Incorporating temporal information

Three methods to incorporate temporal information were designed and tested. The first method was to input several consecutive frames at a time and trying to predict the state of the middle image. We used the same architecture that had the best performance when classifying frame by frame and trained it on five frames at a time. The only modification needed from the single frame classification was to the input of the network. Instead of one frame as the input image, we fed the network five frames as one image with five channels. The centre channel corresponded to the "current" frame, i.e. the one that we wanted to classify, the two first channels were the two preceding frames, and the two last channels were the two following frames.

The second method we used to incorporate temporal information was to include timestamps relative to the QRS complex as input to the model. In the metadata for the images, we had access to each frames timestamp and also the timestamps of the QRS complex from the electrocardiogram (ECG). The QRS complex is a combination of three waves seen on a typical electrocardiogram. A schematic of a QRS complex is shown in Section 3.2.3. To get a normalised feature for our model, we calculated the relative timestamp to the QRS trigger for every frame to get a number between 0 and 1. We concatenated this number together with the features from the last convolutional layer to get an additional feature. Then, we fed the combined features to the first fully-connected layers in the best performing architecture. We had two fully connected layers after the time-stamp

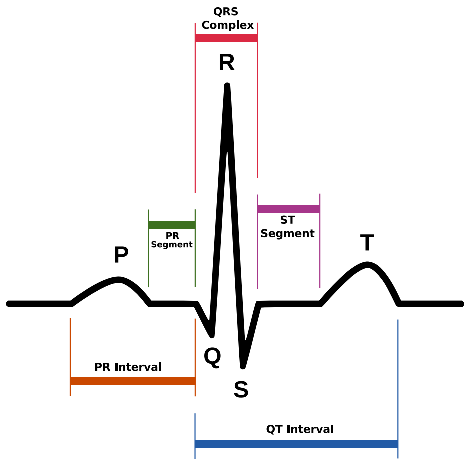


Figure 3.7: Schematic of ECG with QRS complex shown.

(Figure by Anthony Atkielski, Public domain, via Wikimedia Commons, url: `https://commons.wikimedia.org/wiki/File:SinusRhythmLabels.svg`*)*

feature is concatenated to let the network learn nonlinear relationships between the time-stamps and valvular event-times. The full architecture is shown in Table 3.6.

Lastly, we incorporated temporal information via bidirectional recurrent neural networks (as described in Section 2.10.2 on page 45). Training an RNN from scratch on sequences of image frames requires both a lot of memory and time and was not feasible with the hardware we had available. Therefore, we extracted the output from the second to last layer of the VGG-like convolutional neural networks and used this as features to train a recurrent neural network. This means that we first trained a convolutional neural network for single-frame classification, by removing the final layer, we got a function that maps an image to a 16-dimensional vector. We then fed all the frames into this function and got a sequence of 16-dimensional vectors per series, and it was this series of vectors that was fed into the RNN.

The RNN we used consisted of two bidirectional GRU layers [10] (Sections 2.10.1 and 2.10.2 on page 44 and on page 45) of 64 nodes each. After the GRU layers, we had a fully connected layer that outputted a score for each class for each frame. Because of memory considerations, we only fed the network 32 frames. 32 frames did not contain a complete cardiac cycle but were generally enough to contain one or more valvular events. We trained one bidirectional RNN for each of the VGG-like architectures described above.

3.2.4 Preventing local minima

We have previously discussed how local minima and saddle points are detrimental when training neural networks Section 2.4.1 on page 32. As a measure

Type	Output Filters	Size/Stride	Input Size
Conv1	32	3x3/1	252x252
Conv2	64	3x3/2	126x126
Conv3	64	3x3/2	64x64
Conv4	128	3x3/2	32x32
Maxpool1	128	3x3/1	32x32
Conv5	128	3x3/1	32x32
Conv6	128	3x3/2	16x16
Conv7	256	3x3/2	8x8
Maxpool2	256	2x2/1	8x8
Conv8	512	3x3/2	4x4
Conv9	512	3x3/2	2x2
Conv10	3	3x3/1	2x2
FC1	-	-	15
concat time			16
FC2			16
dropout			
FC3	-	-	3
softmax			3

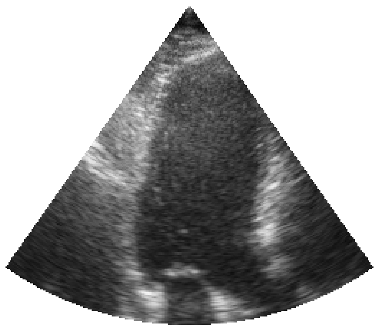
Table 3.6: VGGlike13-f5-t architecture

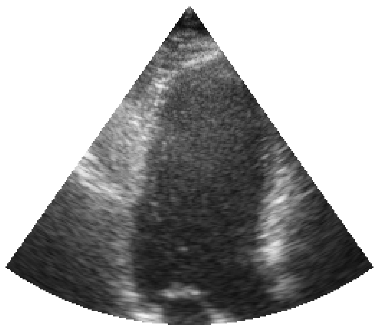
to improve the reliability of our results and avoid local minima and saddle point, all architectures were tested at least two times, and some were tested more.

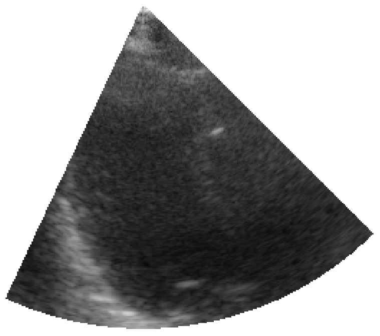
3.3 Data augmentation

One of the main challenges to training the convolutional networks was a small number of labelled data and small variation in the dataset. Frames from the same series were very similar, and so with just 44 labelled series in the training set, we got little variation, which is bad for generalisation. Generalisation is a common challenge of neural nets, which have many parameters and often requires massive datasets to prevent overfitting. The simplest and most common method to alleviate this problem is to use data augmentation to simulate more data. One popular method for image augmentation is to take a random crop of the image at every epoch of training[31]. Inspired by this we extracted a random 252×252 patch of each image in every batch for each iteration. This method created several shifted images for every image in the dataset that can help the model generalise better to small changes.

Because we used ultrasound images, there was room for more problem-specific augmentation. The images in our dataset were collected as images in polar space, where the rows represent radius, and the columns represent the angle. These polar space images were then scanconverted to cartesian coordinates. We had access to the images before they were scanconverted, so we could do the augmentations on the polar space images. Then we scanconverted the aug-


(a) Original image (depth 0.13)


(b) Depth cut to 0.1205


(c) Original image (width 1.222)

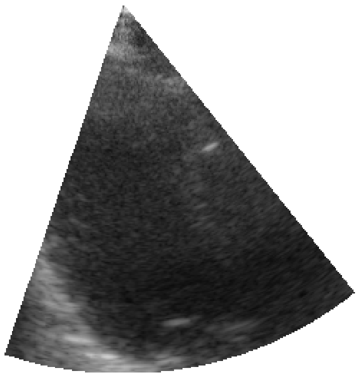

(d) Width shrinked to 1.0

Figure 3.8: Examples of augmentations before scan conversion

mented images and got augmented images that were very close to a natural ultrasound image. We implemented four different augmentation methods to the non-scanconverted images. Sections 3.3 and 3.3 gives an overview of these augmentations.

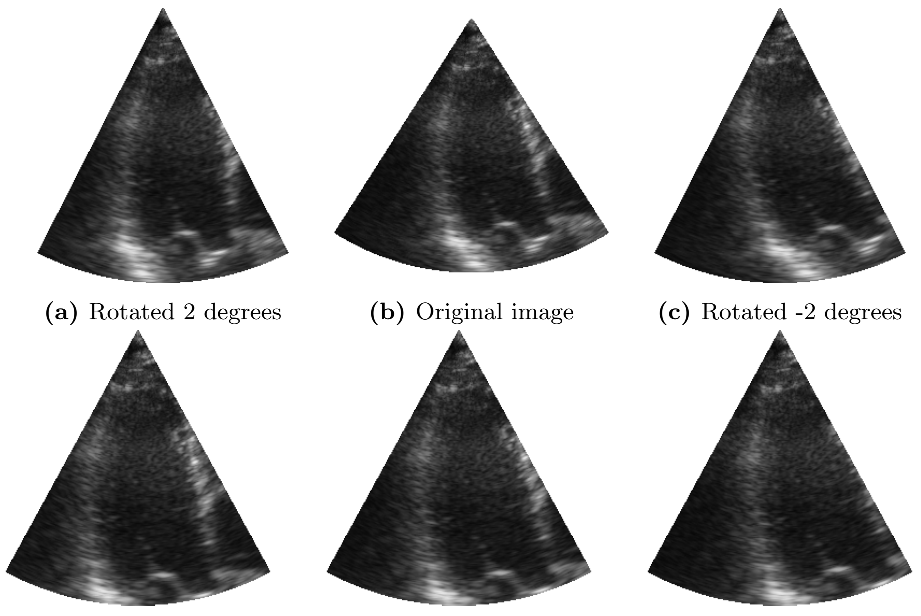
Cut Depth We cut the depth of the non scanconverted image (cropped part of the bottom image and changed the depth). To keep the image similar to a natural ultrasound image we only crop the bottom part of the image and keep the top part.

Shrinked Width We shrinked the width of the non scanconverted image (cropped the sides of the image and changed the value of the width.

Rotation Rotated the non scanconverted image and cropped to remove black spaces

Lateral translation After shrinking the width as described above, we took left, right and centre crops to get three translated versions of the image.

Some of the images in the dataset had more possibilities for augmentations than others. If the valves were located close to the edge of the image, they would not



(a) Rotated 2 degrees **(b)** Original image **(c)** Rotated -2 degrees

(d) Right crop **(e)** Centre crop **(f)** Right crop

Figure 3.9: More examples of augmentations

be visible if we cut the depth or shrinked the width too aggressively. To avoid this, we visually inspected the images in the dataset with the smallest values for depth and width and determined the minimum width and depth necessary for the valves to remain visible within the frame. The width and depth of the image were available to us in the metadata. We defined augmentation step sizes to avoid the images having to little visible changes between the different augmentations. Choosing this step size was done by visually inspecting the changes to the image with different augmentation schemes.

We also wanted to avoid augmentations with too little visible change in the image, so we defined augmentation step sizes by visually inspecting the results from different levels of augmentation. Then, for each image, we created as many augmentations as possible in the range between the original images depth and width and the minimum depth and width with the defined step sizes. We found that rotating the image very quickly led to a significant amount of cropping and, thus we chose only to do two small rotations for every image. The parameters we chose for augmentations are shown in table Section 3.3.

Some of the series in the dataset had ample room for augmentations. To avoid the number of augmentations to become excessively high for these series, we defined a max number of shrunk widths and cut depths and adjusted the augmentation step size when the max was exceeded. We wanted the augmented dataset to be feasible to train within a reasonable time, and many similar images will not provide additional information to the network. Besides, a too high number of augmentations for just one or a few of the series can lead to the network focusing too much on the details of those particular series. Because of

Augmentation parameter	value
Min rotation	-2
Max rotatation	2
Rotation step size	2
Min width	1.0
Width step size	0.02
Max width augmentations	5
Min depth	0.16
Depth step size	0.002
Max depth augmentations	5

Table 3.7: Parameters used for augmentation

this, the number of augmentations were capped to a maximum of 3 (rotations) x 5 (depth cuts) x 5 (shrunk widths) x 3 (translations) = 225. We also created a dataset containing ten augmentations at random per series.

The final model we trained on augmented data was a VGG-like architecture with timestamps but with the weights initialised by first training the model on non-augmented data. The model was trained without augmented data for the first 50 000 iterations and with augmented data for the following 50 0000 iterations.This allowed us to investigate if the augmented data can help the model further improve on what it have learned from non-augmented data.

For all model trained on augmented data, we only performed augmentation on the data used for training. All evaluation were done on the original images, using a single centred crop of 252x252 per frame.

3.4 Methodology for evaluating the model

3.4.1 Accuracy

Accuracy is the proportion of samples for which the model produces the correct classification [20]. We calculated this proportion by merely counting the number of samples in the validation set that were correctly classified, and finding which percent of the validation set this represented. Accuracy is also what our convolutional neural net trained to maximise. We used cross-correlation loss for optimisation, which pushed the model towards classifying as many frames as possible correctly, and every frame was classified independently.

Accuracy was not very useful for the problem at hand. Our final goal was to find the valvular event times. This meant that we mostly cared about the frames where there was a transition from one label to a different label, i.e. from *both valves closed* (class 0) to *aortic valve open* (class 1). The frames where there was no change of class label was not that interesting to us. The problem was that there were comparatively few of these transition frames in a series. A model could make mistakes only near these points and still get a high accuracy score since the percent miss-classification would be small. This was not good, and we, therefore, needed alternative ways to evaluate how well our model predicts the correct transition points between the labels.

3.4.2 Measuring performance on valvular event time detection

For our problem, we wanted to extract the frames corresponding to the valvular event times described in Section 1.1 on page 14, so it was natural to focus on these frames when examining the model. The valvular event frames are the frames where the class label transition between two different classification labels and thus we wanted to examine these transitions. To begin with, let us establish some notation

The dataset consisted of a set of series. Each series, $\boldsymbol{X}$, contains n frames. Every frame, $\boldsymbol{x_j}$ has a class label y_j and these class labels can be collected in a vector,

$$\boldsymbol{y} = [y_0, y_1, \dots, y_n]. \tag{3.1}$$

For every frame, we also have a predicted class label, $\hat{y_j}$,

$$\boldsymbol{\hat{y}} = [\hat{y_0}, \hat{y_1}, \dots, \hat{y_n}]. \tag{3.2}$$

The transition points, $\mathcal{O}$, were found by calculating the difference between neighbouring labels, y_j and y_{j+1}. A number o is in $\mathcal{O}$ if $y_o \neq y_{o+1}$. Similarly, the predicted transition points, $\hat{\mathcal{O}}$, as the numbers $\hat{o}$ such that $y_{\hat{o}} \neq y_{\hat{o}+1}$.

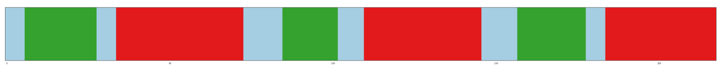


Figure 3.10: Visualisation of the class labels for a series of frames.

The horizontal axis is time in frames. Class 0 (Both valves closed) is shown as blue, Class 1 (Aortic valve open) is shown in green and class 2 (Mitral valve open) is red. This series has three cycles and 11 transition points between different classes corresponding to 11 valvular event times.

A problem when evaluating valvular event detection models that are based on frame classification is *false transitions* and an example of this problem is demonstrated in Section 3.4.2. To find how many false transitions there is in the predictions, we iterated trough all predicted transitions, $\hat{o}_l$ and checked if was is a valid matching true transition, o_k. For o_k to be a valid match, the following must be true:

$$y_{o_k} = \hat{y}_{\hat{o}_l}, \tag{3.3}$$
$$y_{o_k+1} = \hat{y}_{\hat{o}_l+1}, \tag{3.4}$$
$$|o_k - \hat{o}_l| < d, \tag{3.5}$$

Equations (3.3) and (3.4) ensures that the transition is the same transition for both the predicted and the true class labels, and Equation (3.5) ensures that the transition is within an acceptable distance, d. We chose $d = 7$ frames. We counted the number of predicted transitions that did not have a valid matching true transition. From this, we found the percent predicted transitions that were false and, which gives us a measurement of the specificity of the model (that is good the model is at avoiding false predictions).

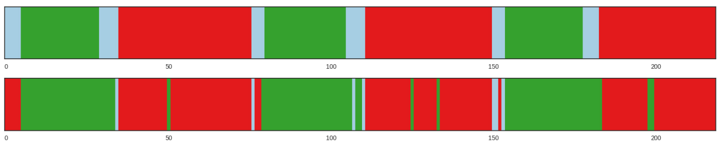


Figure 3.11: Illustration of how the classification results are visualised.

Top: *Ground truth classification labels for a series of frames.* ***Bottom:*** *Predicted classification labels for the same frames by a convolutional network. This prediction has dropped the transitions to and from class 0 at the beginning and towards the end of the series (the first iso-volumetric contraction and the final iso-volumetric relaxation). There are also several spikes of miss-classified frames throughout the series. These problems made it very difficult to use the predictions to extract the points of transition between valve states despite relatively high classification accuracy (86.76%)*

It was also necessary to measure how many of the true transitions that the model correctly predicted. We examined all true transitions, o_k and checked if there was a valid matching prediction, $\hat{o}_l$. $\hat{o}_l$ is a valid match for o_k if Equations (3.3) to (3.5) holds. We then found the percent true transitions that the model predicted, which gave us a measurement of the sensitivity of the model (that is, how many of the true transitions the model finds).

Furthermore, we wanted to measure how accurately the model placed the transitions it predicted correctly. To do this, we looked at the true transitions that were correctly predicted and found how far apart in time the predicted transition were from the actual transition. I.e. we calculate $o_k - \hat{o}_l$. We got a measurement of how accurately in time the model placed the transition points by taking the average of this distance for all the correctly predicted transitions in the dataset.

For our problem, some measurements are more critical. To be able to use the program to extract the time points for transitions between the valve states, the percent false predicted transitions have to be 0%, and the number of true transitions that were correctly predicted has to be at 100%. To get a final evaluation measurement for the model, we counted how many of the series that were valid according to these criteria, which gave us an indication of the overall efficiency of the algorithm.

3.4.3 Visualising the model

Another way we evaluated the effectiveness of the model was to visualise what it had learned. There are a plethora of methods that aim to give an understanding of what neural networks consider when they classify images [39]. Visualisations can give us knowledge about what the model focuses on which also gives us information about how well the model generalises and this tells us about how it will perform on unseen data. To evaluate our models, we chose to implement visualisation by *image occlusion*[54] and *guided backpropagation*[48]

Image Occlusion

Image occlusion was performed on the best model that did not include any temporal information from Section 3.2.2. The high number of model evaluations were too time-consuming for the limited hardware available. We, therefore, moved the box over the image with stride four yielding a 16 times speedup. The increased stride gave us a downsampled output map of probabilities and labels. However, the resolution was good enough to analyse the outputs. We ran image occlusion on the two frames that were classified correctly with the highest certainty (i.e. the highest softmax score) from the validation set.

Guided backpropagation

Guided backpropagation was performed on the same model. To demonstrate what the model considered important, we chose two frames from from the validation set that the model classified with high certainty.

3.5 Post processing with a median filter

Several of the problems with false transitions as described above were small "spikes" of miss-classified frames. These spikes were very short, lasting only one or a few frames each and they often occured in the middle of a more extended phase of the cardiac cycle when the class label was expected to stay constant. For example, when the mitral valve was open, the model would sometimes classify a few frames in the middle of this phase as *both valves closed.* As if the valve suddenly closed and then reopened again. We used a simple way of combating this without retraining the model; we added a post processing step after the classification which used a *median filter* to attempt to remove these spikes.

Median filtering consists of taking a window of fixed size, k, and sliding it over a signal and replacing the signal at each position with the median of the points within the window [2]. We find the median of a set of points by placing them in order by value and find the middle value of that order. In our case, we applied the median filter to the labels of a series, $\boldsymbol{y} = y_0, y_1, \ldots, y_n$ so we replaced each label, y_j, with $\mathrm{MED}(y_{j-k/2}, \ldots, y_j, y_{j+k/2})$

The "spikes" of one or a few miss-classified frames in a series were similar to a signal corrupted by salt-and-pepper noise. Which means sharp and sudden disturbances that are sparsely distributed in the signal[2]. Median filters are an effective method for removing this kind of noise [2]. Additionally, median filters are, unlike for exam ple Gaussian blur, known to be edge preserving in the cases where there is little noise, and the noise is well-separated from the boundary [3]. This is important because we did not want to smooth the transition points of the labels. Because of these qualities, median filtering can be a useful processing step after classification to remove miss-classified spikes before extracting the transition points and an example of this is shown in Section 3.5.

We used median filtering with window size 5 as a post processing step after classification. The window size was chosen by evaluating the performance for window sizes 3, 5 and 7 on the validation data and we found that a window size of 5 eliminates false transitions without removing an entire band of class 0 (both valves closed).

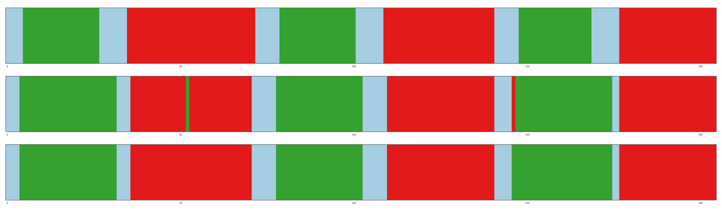


Figure 3.12: Illustration of how median filtering can improve event detection.

Top: *Ground truth classification labels for a series of frames.* ***Middle:*** *Predicted classification labels for the same frames by a convolutional network.* ***Bottom:*** *The predicted classification labels after a median filtering with window size, $k = 3$. For this example, the median filter takes care of the two spikes of misslabelled frames without affecting the other labels.*

Chapter 4

Results

4.1 Dataset statistics

We labelled in total 66 series and split them into a training set, a validation set and a test set. The training set comprised of 44 series or 7901 frames. Of these 16.54% were labelled as class 0, 31.49% as class 1 and 51.97% as class 2. In the smaller validation set of 11 series (2014 frames), 17.38% of the frames were labelled as class 0, 30.14% as class 1 and 52.48% as class 2. Finally, the test set had 11 series (1754 frames), 18.98% labelled as class 0, 31.19% as class 1 and 49.77% as class 2. Section 4.1 shows the final distribution of the classes in the different datasets. These labelled sets of data are what was used to train and evaluate the neural nets.

4.2 Single frame classification

We tested single frame classification with a 5, 9, and 11 layer architecture as described in Section 3.2 on page 55. The resulting accuracy scores for both the training set and the validation set are presented in Section 4.2. Section 4.2 shows the development of the accuracy score and loss for the training set and

	Train	Validation	Test
no. Series	44	11	11
no. frames	7901	2014	1754
Class 0	16.54%	17.38%	18.98%
Class 1	31.49%	30.14%	31.19%
Class 2	51.97%	52.48%	49.77%

Table 4.1: The distribution of the different classes

Architecture	%Validation accuracy	%Train accuracy
conv5	66.71	92.74
conv9	80.66	99.65
conv11	77.84	99.75

Table 4.2: Accuracy scores for single frame classification on validation data and training data

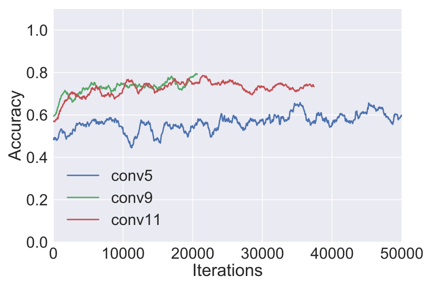
config.	Mdiff	%PT	% FP	Mdist	# accepted
conv9	17.3636	78.65	48.82	1.9393	1/11

Table 4.3: Custom metrics evaluated on the validation set for the single frame classification with the highest accuracy score

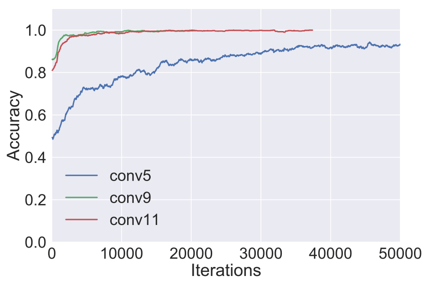
Mdiff: *Mean difference in number of true transitions and number of predicted transitions.*
%PT: *Percentage of the true transitions that were correctly predicted.*
%FP: *Percentage of the predicted transitions that were false*
Mdist: *Mean distance between the correctly predicted true transitions and the corresponding predictions*
#accepted: *total number of complete series that had 0%FP and 100%PT.*

the validation set during training.

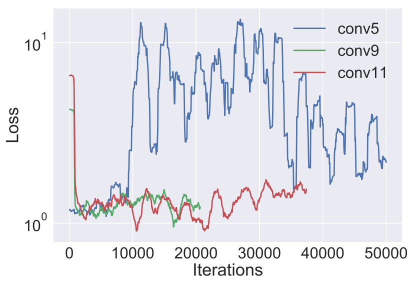
The best architecture for single frame classification was a nine-layer architecture. This model obtained a 80% accuracy on the validation set.Section 4.2 presents the results of the metrics described in Section 3.4.1 on page 62. Only 1/11 of the series in the validation set where acceptably classified by the requirements laid out in Section 3.4.2 on page 64. Sections 4.2 and 4.2 show visualisations of the classification for two of the series in the validation set.



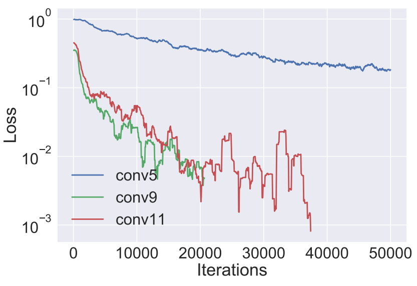
(a) Validation accuracy



(b) Train accuracy



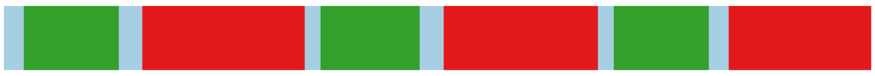
(c) Test loss



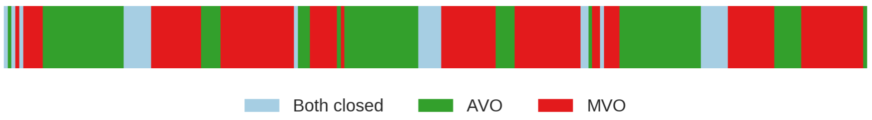
(d) Train loss

Figure 4.1: Validation and training accuracy and loss for single frame classification models

Plot of the accuracy scores and loss for the different single frame classification models evaluated on the training and validation set during training. The conv5 model (blue) was trained for 50000 iterations, the conv9 model (green) trained for 20650 iterations and the conv11 model (red) trained for 37400 iterations. To make the plot clearer, we have smoothed the accuracy scores and the loss values with a running average filter with a window size covering 30 iterations.



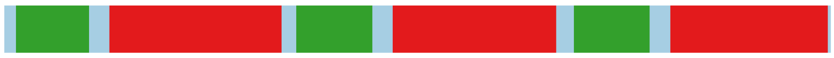
(a) True labels



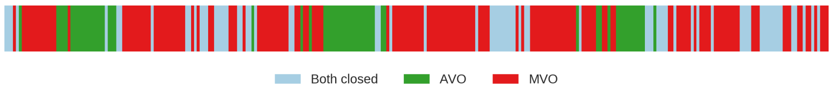
(b) conv9 predictions

Figure 4.2: Results from series 92460530

4.2a: *Visualisation of the true labels for series 92460530.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***4.2b:*** *Visualisation of the corresponding labels predicted by the conv9 model*



(a) True labels



(b) conv9 predictions

Figure 4.3: Results from series 17864142

4.3a: *Visualisation of the true labels for series 17864142.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***4.3b:*** *Visualisation of the corresponding labels predicted by the conv9 model*

Architecture	%Validation accuracy	%Train accuracy
conv9-5f	90.04	100.00
conv11-5f	90.07	99.99
VGGlike13-5f	94.49	99.94
VGGlike13-5f-t	94.74	99.94

Table 4.4: Accuracy scores for architectures incorporating temporal information evaluated on validation data and training data

4.3 Architectures incorporating temporal information

The results from Section 3.2.3 on page 57 can be seen in Sections 4.3 and 4.3 and Sections 4.3 and 4.3. The accuracy improved by 10%. The model had a high number of false transitions (37.66%) and the number of accepted series was 2/11.

The results from adding relative timestamps as a feature (Section 3.2.3 on page 57) are displayed in Section 4.3 and Figures 4.6c and 4.7c. This architecture had 5% better validation accuracy than conv9-5f. The number of accepted series in the validation set was the same as for conv9-5f, 2/11. The percentage of falsely predicted transitions was reduced from 37.66% to 25.52% and the percentage of true transitions that were correctly predicted increased from 89.39% to 92.08%.

The added timestamp feature did not result in a measurable improvement in accuracy as seen in Section 4.3. It did, however, result in fewer misclassifications of the mitral valve in the midst of ventricular filling, when the mitral valve sometimes appears to close briefly and open again before it properly closes as reported in Section 3.1.2 on page 52. An example of this is shown in Section 4.3. The added temporal information contributed to lowering the rate of false predicted transitions from 38.21% to 14.69% and raising the number of accepted predicted series from 2/11 to 4/11 (see Section 4.3).

The results from both the median filtering post-processing (Section 3.5 on page 65) and the bidirectional RNN based post processing (Section 3.2.3 on page 58) can be seen in Sections 4.3 and 4.3 and Sections 4.3 and 4.3. Median filtering reduced the number of false predicted transitions ("spikes" of misclassified frames) and increased both the classification accuracy and the number of accepted series. Bidirectional recurrent network yielded even better results, and both reduced the number of false predicted transitions while increasing the number of true transitions that where correctly predicted. The bidirectional recurrent network architecture trained on the features from the VGG inspired architecture with timestamps (VGGlike13-5f-t) got the highest number of accepted series for the validation set (8/11) among all the tested architectures.

config.	Mdiff	%PT	%FP	Mdist	# accepted
conv9-5f	9.8182	89.39	37.66	1.3418	2/11
conv11-5f	10.4545	97.52	38.21	1.4260	1/11
VGGlike13-5f	5.3636	92.08	25.52	0.7490	2/11
VGGlike13-5f-t	3.2727	92.08	14.69	0.9639	4/11

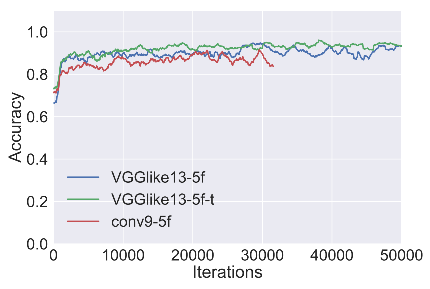
Table 4.5: Custom metrics evaluated on the validation set for the architectures incorporating temporal information

Mdiff: *Mean difference in number of true transitions and number of predicted transitions.*
%PT: *Percentage of the true transitions that were correctly predicted.*
%FP: *Percentage of the predicted transitions that were false*
Mdist: *Mean distance between the correctly predicted true transitions and the corresponding predictions*
#accepted: *total number of complete series that had 0%FP and 100%PT.*

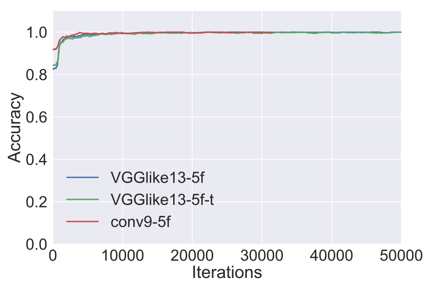
Architecture	%Validation accuracy	%Train accuracy
conv9-5f(m)	91.21	100.00
VGGlike13-5f(m)	95.63	99.94
VGGlike13-5f-t(m)	94.95	99.94
BRNN	95.03	99.41
BRNN-t	95.28	99.37

Table 4.6: Validation and training accuracy for architectures that incorporate temporal information.

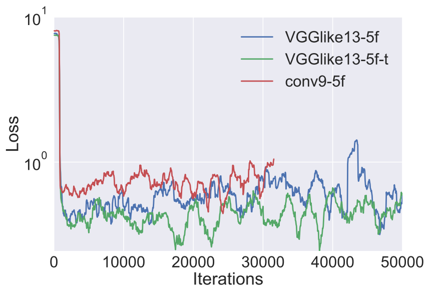
Accuracy scores for architectures post processed by median filtering and bidirectional RNN.The BRNN architecture was trained on features extracted from the second to last layer of the VGGlike13-5f model and BRNN-t is the same architecture but trained on features extracted from the VGGlike13-5f-t model instead.



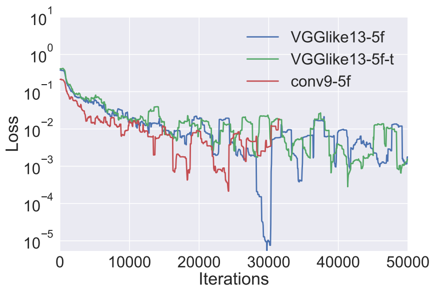
(a) Validation accuracy



(b) Train accuracy



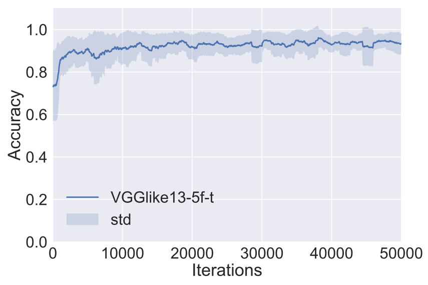
(c) Test loss



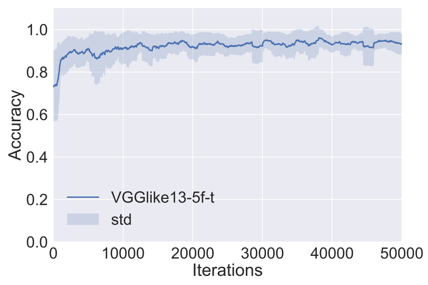
(d) Train loss

Figure 4.4: Validation and training accuracy and loss for classification models incorporating temporal information

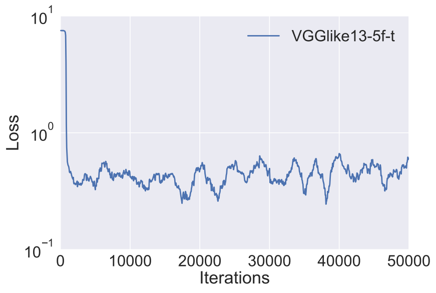
Plot of the accuracy scores and loss for the different models incorporation temporal information evaluated on the training and validation set during training. The conv9-5f model (red) was trained for 31 550 iterations, the VGGlike13-5f model (blue) and the VGGlike13-5f-t model (green) trained for 50 000 iterations. To make the plot clearer, we have smoothed the accuracy scores and the loss values with a running average filter with a window size covering 30 iterations.



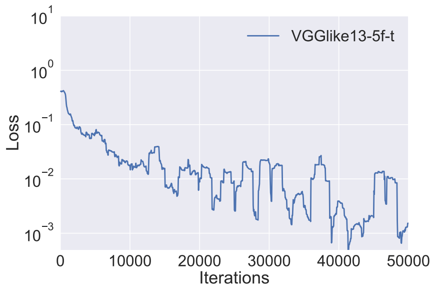
(a) Validation accuracy



(b) Train accuracy



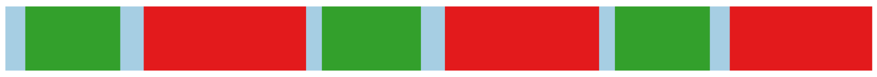
(c) Test loss



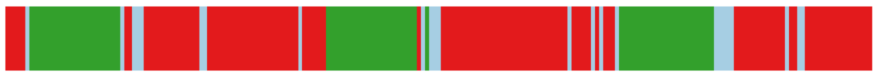
(d) Train loss

Figure 4.5: Validation and training accuracy and loss for the best performing architecture

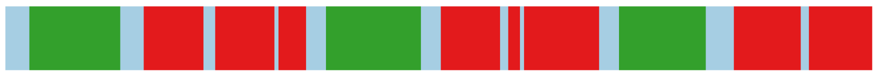
Plot of the accuracy scores and loss for the architecture that yielded the highest accuracy on the validation set.To make the plot clearer, we have smoothed the accuracy scores and the loss values with a running average filter with a window size covering 30 iterations. We display the corresponding running standard deviation in light blue



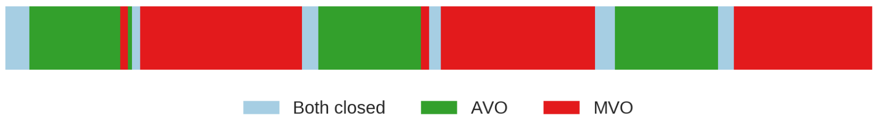
(a) True labels



(b) conv9-5f



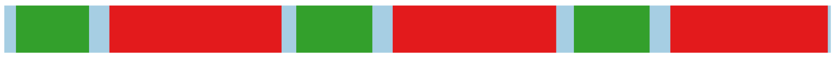
(c) VGGlike13-5f



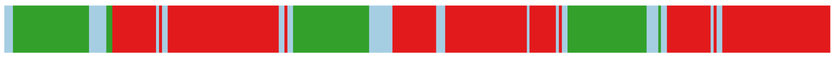
(d) VGGlike13-5f-t

Figure 4.6: Results from series 92460530

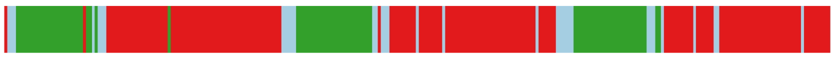
4.6a: *Visualisation the true labels for series 92460530.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***b-d:*** *Visualisation of the corresponding labels predicted by the different architectures incorporating temporal information*



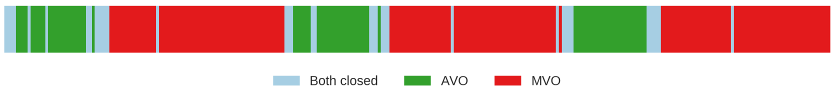
(a) True labels



(b) conv9-5f



(c) VGGlike13-5f



(d) VGGlike13-5f-t

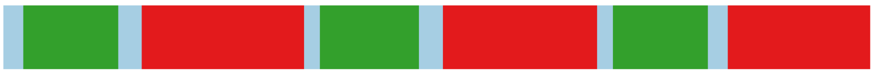
Figure 4.7: Results from series 17864142

4.7a: *Visualisation of the true labels for series 17864142. The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***b-d:*** *Visualisation of the corresponding labels predicted by the different architectures incorporating temporal information*

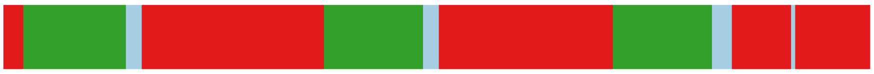
config.	Mdiff	%PT	%FP	Mdist	# accepted
conv9-5f(m)	2.5455	84.50	21.15	0.9407	5/11
VGGlike13-5f-t(m)	0.5455	89.60	6.68	0.7513	5/11
VGGlike13-5f(m)	0.4545	92.15	6.24	0.6372	7/11
BRNN	1.8182	96.21	15.42	0.6198	6/11
BRNN-t	0.4545	95.39	2.44	0.8314	8/11
BRNN-t(m)	0.3636	93.73	2.43	0.8110	8/11

Table 4.7: Custom metrics evaluated on the validation set for the architectures post processed with median filtering and bidirectional RNN.

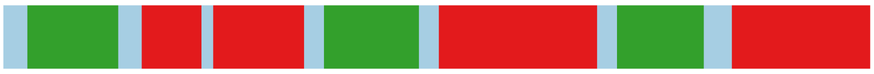
The BRNN architecture was trained on features extracted from the second to last layer of the VGGlike13-5f model and BRNN-t is the same architecture but trained on features extracted from the VGGlike13-5f-t model instead.
Mdiff: *Mean difference in number of true transitions and number of predicted transitions.*
%PT: *Percentage of the true transitions that were correctly predicted.*
%FP: *Percentage of the predicted transitions that were false*
Mdist: *Mean distance between the correctly predicted true transitions and the corresponding predictions*
#accepted: *total number of complete series that had 0%FP and 100%PT.*



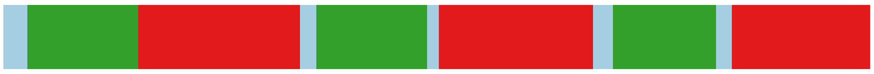
(a) True labels



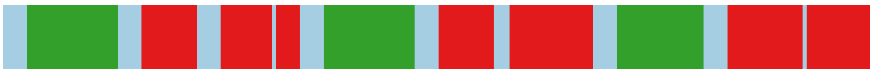
(b) conv9-5f(m)



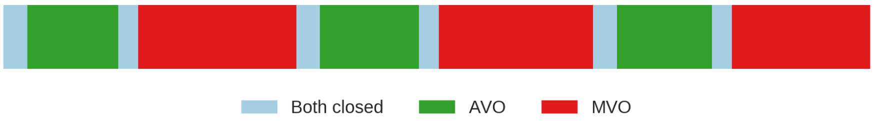
(c) VGGlike13-5f(m)



(d) VGGlike13-5f-t(m)



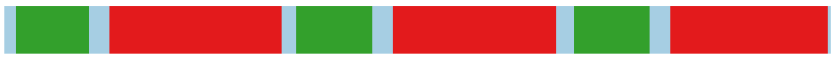
(e) BRNN



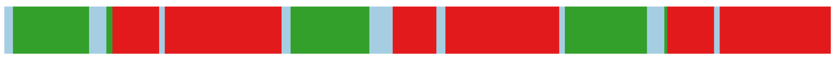
(f) BRNN-t

Figure 4.8: Results from series 17864142

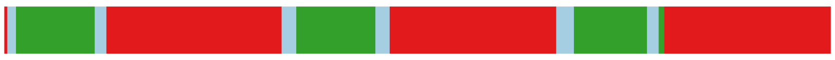
4.8a: *Visualisation of the true labels for series 92460530. The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***b-e:*** *Visualisation of the corresponding labels resulting from post processing with median filtering* (m) *and bidirectional recurrent networks* (BRNN)*. The BRNN architecture was trained on features extracted from the second to last layer of the VGGlike13-5f model and BRNN-t is the same architecture but trained on features extracted from the VGGlike13-5f-t model instead.*



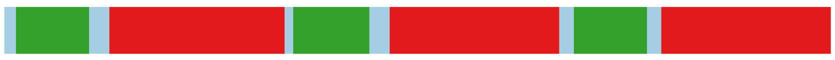
(a) True labels



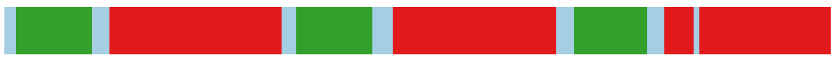
(b) conv9-5f(m)



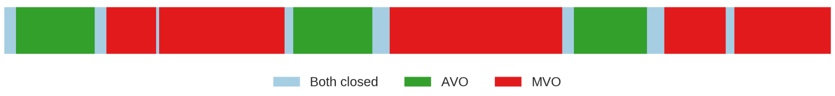
(c) VGGlike13-5f(m)



(d) VGGlike13-5f-t(m)



(e) BRNN



(f) BRNN-t

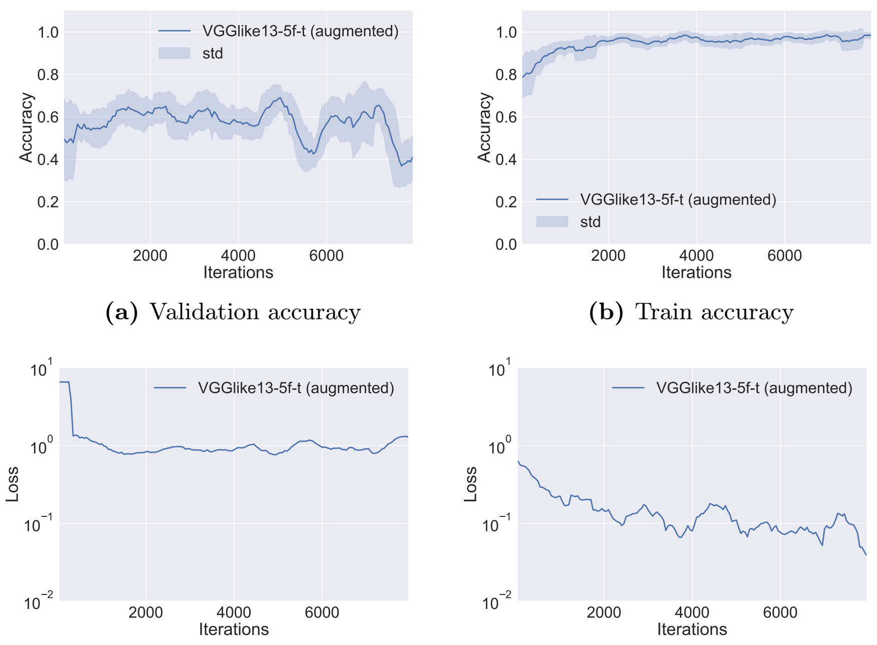
Figure 4.9: Results from series 17864142

4.9a: *Visualisation of the true labels for series 17864142. The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***b-e:*** *Visualisation of the corresponding labels resulting from post processing with median filtering* (m) *and bidirectional recurrent networks* (BRNN)*. The BRNN architecture was trained on features extracted from the second to last layer of the VGGlike13-5f model and BRNN-t is the same architecture but trained on features extracted from the VGGlike13-5f-t model instead.*

4.4 Training with augmented data

Several experiments were ran with augmented data, all used the VGG-like architecture. First, we trained the best performing model without timestamps using all the augmentations as described in Section 3.3 on page 61. This resulted in a very high training accuracy and very low validation accuracy. Exactly the same behaviour was in the model trained with only ten augmentations per series as described in Section 3.3 on page 61. We also trained a model using the VGG-like architecture with timestamps and only ten augmentations per series and, once again, we got low accuracy. Plots depicting the loss and accuracy as a function of iteration number for this model (ten augmentations and timestamps) can be seen in Section 4.4

The plots depicting the accuracy and loss for the pretrained augmentation training described in Section 3.3 on page 62 are shown in Section 4.4.

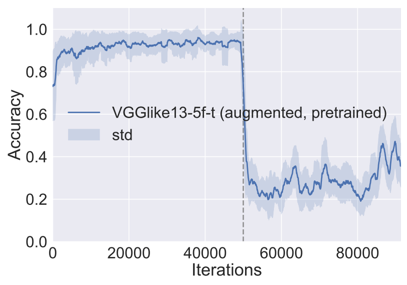


(a) Validation accuracy **(b)** Train accuracy

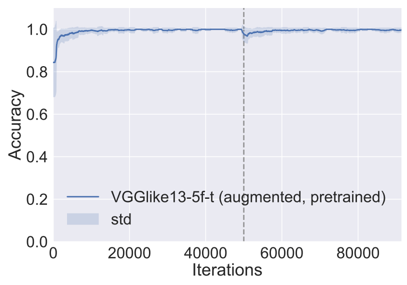
(c) Test loss **(d)** Train loss

Figure 4.10: Validation and training accuracy and loss for the architecture trained on augmented data

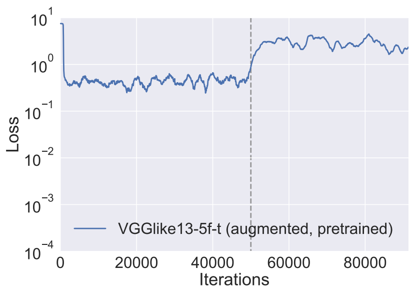
Plot of the accuracy scores and loss for the architecture that yielded the highest accuracy on the validation set trained on augmented data. To make the plot clearer, we have smoothed the accuracy scores and the loss values with a running average filter with a window size covering 10 iterations. We display the corresponding running standard deviation in light blue



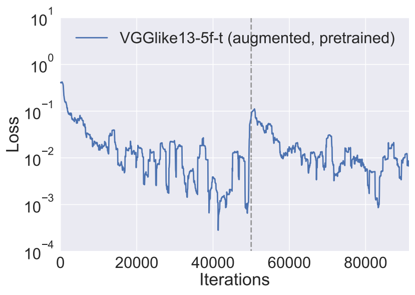
(a) Validation accuracy



(b) Train accuracy



(c) Test loss



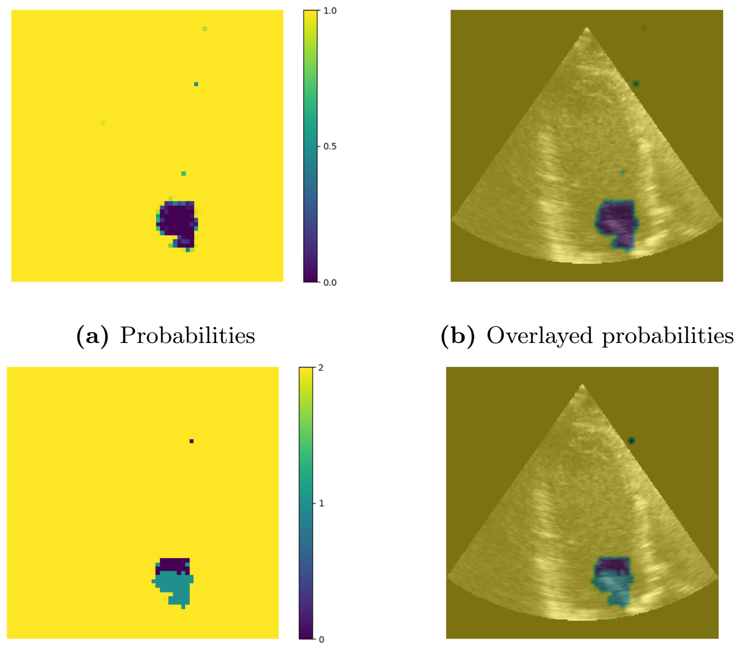
(d) Train loss

Figure 4.11: Validation and training accuracy and loss for the architecture trained on augmented data (pretrained on non-augmented data)

Plot of the accuracy scores and loss for the architecture that yielded the highest accuracy on the validation set pretrained on non augmented data for 50 000 iterations and then trained further on augmented data. The end of pretraining and beginning of the training on augmented data is shown with a grey line. To make the plot clearer, we have smoothed the accuracy scores and the loss values with a running average filter with a window size covering 30 iterations. We display the corresponding running standard deviation in light blue

4.5 Visualisation

The results of image occlusion, described in section Section 3.4.3 are shown in Sections 4.5 and 4.5. Figures 4.12a and 4.13a show how the certainty of the network changed as a function of the centre position of the occluding box. Figures 4.12c and 4.13c display the outputted classification label as a function of the centre position of the occluding box. From the figures, we see that the image occlusion had the most influence on the network classification when the box was occluding the mitral valve.



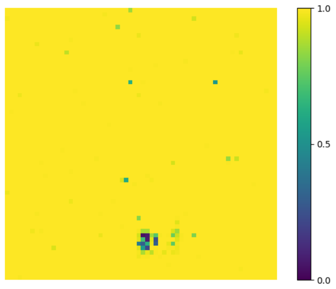
(a) Probabilities **(b)** Overlayed probabilities

(c) Labels **(d)** Overlayed labels

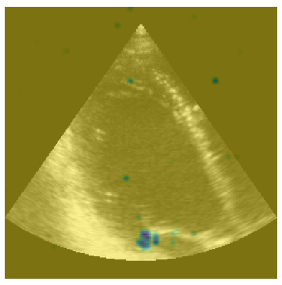
Figure 4.12: Results from image occlusion experiments

Image occlusion for VGGlike13 on one image frame from the validation set. The figure shows a selected example that the model classified correctly with high confidence, i.e high softmax score (the unoccluded image frame was correctly classified as mitral valve open*). 4.12a displays the outputted probability for the correct class for each position of the occlusion box and 4.12c displays the output label for each position of the occlusion box. 4.12b displays the probabilities from 4.12a overlayed on the original image frame and shows overlayed with the labels from 4.12d*

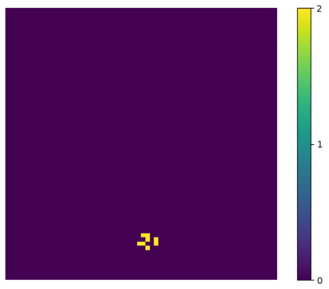
In Section 4.5 we see the results of guided backpropagation described in Section 3.4.3 on page 65. The visualisation shows guided backpropagation for the VGGlike13-5f model.



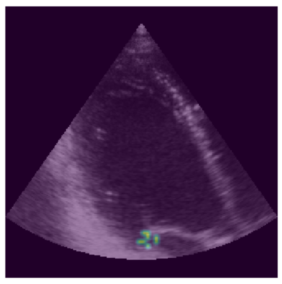
(a) Probabilities



(b) Overlayed probabilities



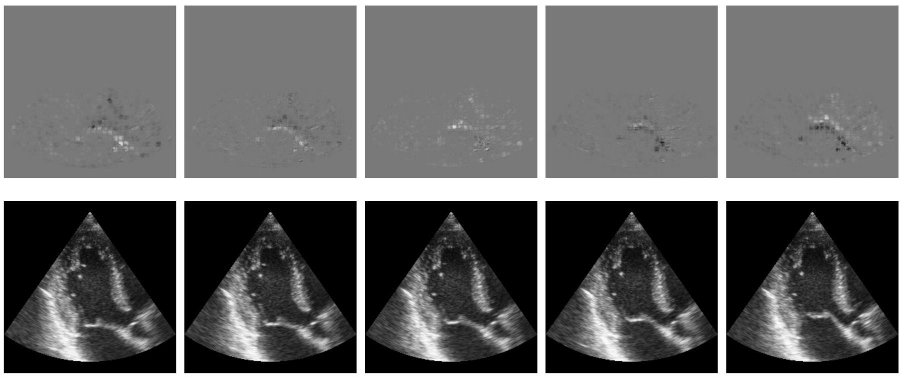
(c) Labels



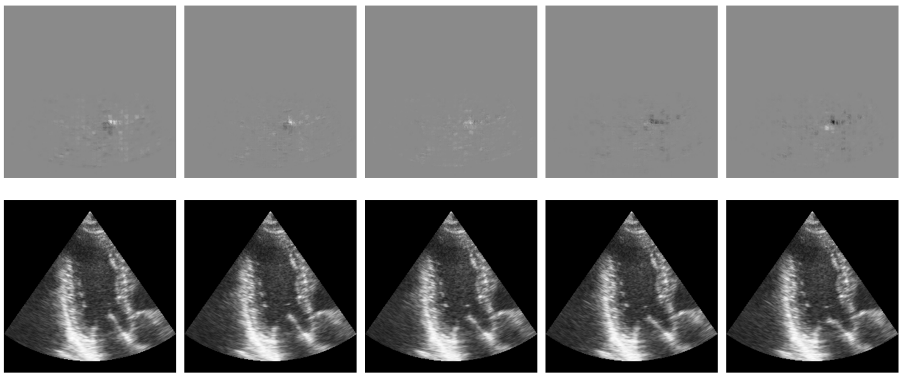
(d) Overlayed labels

Figure 4.13: Results from image occlusion experiments

Image occlusion for VGGlike13 on one image frame from the validation set. The figure shows a selected example that the model classified correctly with high confidence, i.e high softmax score (the unoccluded image frame was correctly classified as both valves closed*). 4.13a displays the outputted probability for the correct class for each position of the occlusion box and 4.13c displays the output label for each position of the occlusion box. 4.13b displays the probabilities from 4.13a overlayed on the original image frame and shows overlayed with the labels from 4.13d*



(a) Centre frame correctly classified as *both valves closed* with 99% certainty



(b) Centre frame correctly classified as *mitral valve open* with 99% certainty

Figure 4.14: Results from guided backpropagation

Figures 4.14a and 4.14b show visualisations of the gradients of the output class activations with respect to five input frames. The ***top*** *rows show the gradients and the* ***bottom*** *rows show the corresponding input frames. For this visualisation, we chose the two frames that where correctly classified with the highest softmax probability score*

Architecture	%Test accuracy
BRNN-t	93.15 %

Table 4.8: Accuracy scores for the best model evaluated on the test

config.	Mdiff	%PT	% FP	Mdist	# accepted
BRNN-t	0.7273	95.45	10.24	1.0259	7/11

Table 4.9: Custom metrics evaluated on the test set for the model with the highest validation performance

Mdiff: *Mean difference in number of true transitions and number of predicted transitions.*
%PT: *Percentage of the true transitions that were correctly predicted.*
%FP: *Percentage of the predicted transitions that were false*
Mdist: *Mean distance between the correctly predicted true transitions and the corresponding predictions*
#accepted: *total number of complete series that had 0%FP and 100%PT.*

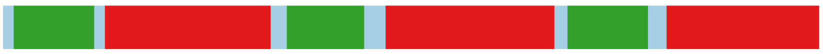
4.6 Test results

Finally, we show the results of evaluating the model with highest validation performance on the test data. The accuracies are shown in Section 4.6 and the custom metrics from Section 3.4.2 on page 63 are shown in Section 4.6. Three examples of predictions for test data are shown in Sections 4.6 to 4.6. For the test set we also extracted the timestamps of the predicted transitions and the corresponding true transitions and calculated the distance between them in seconds. The results of this are shown in Section 4.6

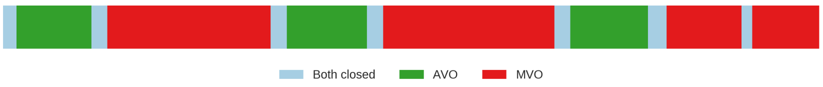
config.	Mean	Std	Max
BRNN-t	11 ms	13 ms	50 ms

Table 4.10: Timing errors for the best performing architecture.

Mean absolute difference, std absolute difference and max absolute difference between the true transitions that were correctly predicted and the corresponding predicted transitions evaluated on the test set for the model with the highest validation performance.



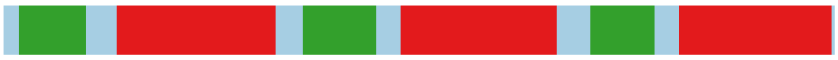
(a) True labels



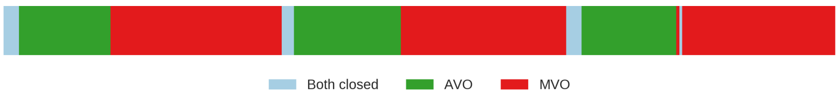
(b) RNN-t predictions

Figure 4.15: Results from series 03315171

4.15a: *Visualisation of the true labels for series 03315171.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***4.15b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*



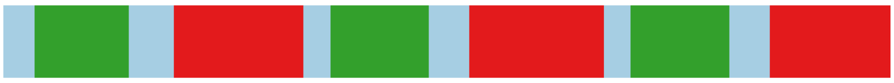
(a) True labels



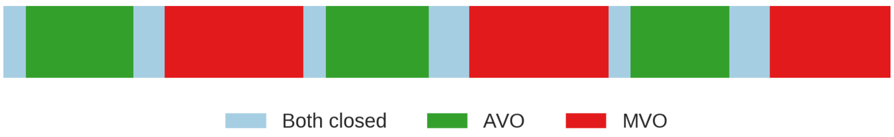
(b) RNN-t predictions

Figure 4.16: Results from series 73496926

4.16a: *Visualisation of the true labels for series 73496926.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***4.16b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*



(a) True labels



(b) RNN-t predictions

Figure 4.17: Results from series 23029318

4.17a: *Visualisation of the true labels for series 23029318.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***4.17b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*

Chapter 5

Discussion

We have demonstrated that deep neural networks are promising for solving the problem of valvular event detection. Furthermore, three components were found to improve the efficiency of these algorithms - maxpooling layers, RNN-heads and ECG information. The best network that were trained was a convolutional network that contained all of these components. It correctly classified 93% of the frames in the test series and detected the valvular events in 7 out of 11 test series. These results are not good enough for use in the clinic, however, they demonstrate that neural networks are a venue to investigate further - especially given the hardware constraints of this project.

5.1 Classification with convolutional neural nets

The single-frame architectures did not perform sufficiently. However, the fact that the nine-layer architecture performed so much better than both the five-layer architecture and the eleven layer architecture is notable. At first glance, it seems that the five-layer architecture under fitted, because it did not achieve good accuracy on the training data, and the eleven-layer architecture over fitted, because it achieved near-perfect results on the training data and poor results on the validation data. Consequently, it appears that one should choose an algorithm less complex than the eleven layer architecture, but still more complex than the five-layer architecture.

This conclusion, however, is put into question by the fact that the VGG-like architecture, which has a total of 13 layers, outperformed both the eleven-layer and the nine-layer fully convolutional architectures. Because the VGG-like architecture was trained on five frames, it begs the question of whether the eleven layer architecture would also perform well when trained on five frames. Something we see from table Section 4.3 on page 72 that is not the case. This result indicates that a VGG-like architecture which uses max-pooling layers in addition to the convolutional layers does indeed yield a more apt algorithm for classifying the frames than a fully convolutional network.

Because the single frame architectures were not satisfactory for valvular event detection, three methods to incorporate temporal information were designed and tested. Firstly, we inputted several consecutive frames at a time and classified the state of the middle image. This allowed the network to take into account the previous and next frames and learn to find the difference and motion between the frames, which can help to remove small mistakes like false transitions. When classifying the images manually, we found that looking at the difference between a few frames directly after each other helped the person labelling to decide the correct class. If, for example, the mitral valve is moving upwards it is most likely open. By feeding these extra frames to the model, it can learn short time relationships between the frames without requiring a more complicated architecture.

Motivated by this, we used the same architecture that had the best performance when classifying frame by frame and trained it on five frames at a time. The only modification needed from the single frame classification was to the input of the network. Instead of one frame as the input image, we fed the network five frames as one image with five channels. Five frames were used because it gave us some temporal span without the memory requirements exceeding the capabilities of the hardware we had available. The centre channel corresponded to the "current" frame, i.e. the one that we wanted to classify, the two first channels were the two preceding frames, and the two last channels were the two following frames.

We see from Sections 4.2 and 4.3 on page 68 and on page 72 that the five-frames architectures outperformed the single single-frame architectures by a large margin. The best single-frame architecture had only one accepted series, whereas the best five-frame architecture had two accepted series (five after post-processing by median filtering). The fact that inspecting multiple frames was of use to the network is unsurprising as we found that it was necessary to view multiple frames at once when labelling the images manually.

The second method we used to incorporate temporal information was feeding the model the timestamps relative to the QRS complex as an additional feature. Mada et al. showed that peak R, which is part of the QRS complex, could be used to approximate end-diastole when the ECG morphology is normal [36] (see Section 1.3 on page 20). Hence, the QRS trigger times may contain information useful for classification of valvular event times.

Incorporating temporal information in the form of the relative time passed since the QRS complex further improved the performance of the model. However, it did double the number of accepted validation series from two to four. Upon inspecting where this improvement happened, we see that it occurs most often during the middle of the ventricular relaxation. Here, the model only trained on image frame data sometimes misclassifies a few frames as mitral valve closed when the mitral valve is still open. The model that was given relative timestamps as an additional feature makes this mistake much more rarely. This error, intuitively makes sense, as there is a short period during the ventricular relaxation phase where it visually looks like the mitral valve closes. By adding this temporal information, we see that the networks pick up on the fact that the mitral valve does not close near the middle of two QRS peaks.

Even after incorporating temporal information in the form of timestamps relative to the QRS complex, the network still predicted several false transitions. The resulting periods of falsely classified frames often had a short duration which motivated us to use a median filter for post-processing. From Sections 4.3 and 4.3 on page 72 and on page 76 we see that the median filter removed over half of the falsely predicted transitions (14.69% was reduced to 6.68%) for the best performing classification model (without an RNN-head). Unfortunately, some true transitions where also removed. This removal occurred because IVC and IVR last for a short time and can, therefore, be removed by the median filter (especially if the network predicts the transition from MVO to BVC late and then the transition from BVC to CVO early during IVC or vice versa during IVR). In total, median filtering increased the number of accepted validation series from 4/11 to 5/11.

Lastly, we incorporated temporal information via bidirectional recurrent neural networks. Recurrent neural network models are especially suited for sequences of data as described in Section 2.10 on page 43. However, training an RNN from scratch on sequences of image frames required both a lot of memory and time and was not feasible with the hardware we had available.

To save both training time and memory, we instead took the output we got from the second to last layer of our best performing convolutional neural network and used this as features to train a recurrent neural network. The final layer of the convolutional neural network outputs the predicted class for an input frame. The layer before that outputs the features used for this classification and we can save these features and use them as input features for an RNN model. This method gave us 16 features instead of 256x256, and these features were already trained to contain compact and useful information about the state of the valves. Then the RNN could learn to use relationships in time between the frames to do the classification. For example that one valve has to close before the other valve can open. Using the features from the trained convolutional neural net is similar to transfer learning, which has shown that CNNs can learn helpful features that are transferable to other visual tasks [40].

The RNN architectures gave a significant improvement over the frame classification algorithms. There was no significant increase in accuracy, but the number of accepted validation series increased from 4/11 to 8/11. When inspecting the results, we found that the RNN architectures “cleaned up” the classifications and removed physically impossible transitions between labels. In Section 4.3 on page 77 we see an example of this, the network without the RNN head creates a transition from AVO to MVO, but the network with the RNN head predicts that both valves are closed inbetween (Section 4.3 shows the VGG architecture after median filter, results before median filtering were even worse and can be seen in Section 4.3 on page 75). This improvement is reasonable as recurrent architectures are particularly suitable to model sequences [11]. Thus, we argue that the model has learned temporal features that allowed it to recognise which patterns of classes that are allowed for a valid heart cycle, and to remove invalid transitions.

However, the RNN trained on features from the model incorporating timestamps often outperforms the RNN without timestamp information during ventricular relaxation. This result is interesting, as it indicates that our bidirectional RNN did not manage to learn adequate temporal features to discern that the mitral valve stays open during ventricular relaxation. One possible remedy to this problem might be to train the RNN with longer time-series and use gradient-clipping to avoid the exploding gradient problem [41].

Median filtering did not yield further improvements on the results from the recurrent networks (see Section 4.3 on page 76), which suggests that the bidirectional RNN learned to correct the same mistakes corrected by the median filters. This is ideal because the median filter eliminates short periods of class 0 (both valves closed). Hence, we do not recommend applying a median filter as post-processing after RNN classification

5.2 Errors made by the network

We inspected the validation series that our best performing model (RNN-t) failed on, and found two kinds of mistakes. The first mistake is at the beginning and end of the series when the model sometimes misclassifies one or a few frames. This mistake is found on all the erroneous validation series and is unsurprising since recurrent neural networks base the output classification on previous hidden states, and thus the predictions become more certain further into a sequence. Moreover, we used a bidirectional RNN, so this problem also applies to the end of a sequence. We argue that the misclassification of the boundaries is not an issue, because, in a clinical setting, a valvular-event detection program should run over enough time for it to be sensible to crop away the predictions at the beginning and end of a series. In fact, we argue that the same amount of time should be used to crop at the beginning and end of a series as the number of time steps used during training of the recurrent neural network.

The second type of mistake the network does is in the midst of MVO , where it seems like the mitral valve is closing and then opens again. This mistake occurs for both the validation data and test data. An example of this can be seen in Section 4.6 on page 85. This mistake is not anatomically possible and takes place away from the correct transitions, and thus it would be straightforward to create a post-processing tool that removes all of these transitions.

Lastly, on the test set, there are two instances of a mistake that is more detrimental. The network has, for two of the series, trouble seeing when the aortic valve closes. An example is shown in Figure 4.16b on page 85. We manually inspected the series and noticed that it is easy to discern when it opens and closes when we view many frames after each other. However, it is not so obvious when we only view one or a few frames at a time. Perhaps the network has not learned good enough temporal features to classify the aortic valve in these frames. The AVO class and the BVC class are also the classes with the least training data, so it is natural that the network would struggle with separating these specific classes. Luckily, this mistake is also non-physical, so it is easy to detect. However, we see no simple solution as to how to resolve this problem other than giving the model more examples for BVC (class 0).

5.3 Visualising the model

The first network visualisation algorithm we used was guided backpropagation (Sections 3.4.3 and 4.5 on page 65 and on page 81). The result is shown in Section 4.5 on page 83. Here, we see that for the main frame in the middle, the model appears to focus on detecting the valve. Moreover, the future and past frames have gradients similar to edge filters, and we can interpret this as the model detecting the motion of the valve in these frames. This motion detection is reasonable because, during the manual labelling of the dataset, the valvular motion was found to be a useful feature to determine the correct class. Thus, the model appears to inspect the frames similarly to how a human would do it manually. The main frame is used for basic detection of the valves, the past and future frames are used to determine the motion, and then this information affects the final classification of the frame.

The second algorithm used to visualise what the network focused on was image occlusion (Sections 3.4.3 and 4.5 on page 65 and on page 81), and the results from the first image can be seen in Section 4.5 on page 81. For this frame, we see that the network only misclassifies the occluded frame when the occlusion box covers the right-hand side of the mitral valve. This misclassification makes intuitive sense, as this is the most prominent part of the mitral valve, and it is natural that the network will focus on it when classifying a frame as "mitral valve open" (class 2).

On the other hand, the fact that the network classifies this frame as BVC when the mitral valve is covered, makes less sense. When the mitral valve is covered, it is solely black - there is no mitral valve in the frame. Thus, if the network is detecting the mitral valve, it would make sense for the network to classify a frame without mitral valve as if the mitral valve was open. However, the findings from guided backpropagation might shine some light on this. Section 4.5 on page 83 shows that the image looks for movement between the frames, and the image-occlusion box covers all five frames and eliminates all movement inside the box. Thus, we argue that because there is no movement of the mitral valve, the network considers it closed.

From Section 4.5 on page 81 we also see that the aortic valve is sometimes classified as open when the mitral valve is covered, which is less intuitive. The aortic valve is decidedly closed in the image, so classifying it as being open seems strange. We can only find one probable explanation for this. We hypothesise that the occluding box makes the image sufficiently different from any of the training examples for the classification to be almost as uncertain as random guessing. This, however, does not explain why the other image occlusion results seem to make sense and why we did not see this behaviour when the box covered other parts of the image.

It appears that the model considers the mitral valve to be an essential part of the image frame. The second image occlusion example and the guided backpropagation results support the importance of the mitral valve. In the second image occlusion example (Section 4.5 on page 82), both valves are closed, but the network only misclassified when the box occluded the mitral valve, not the aortic valve. Furthermore, the guided backpropagation (Section 4.5 on page 83) shows low gradient values for the aortic valve compared with the mitral valve. During the labelling process, we found the mitral valve to be the easiest valve to spot and therefore a helpful feature for deciding the class label. Thus it is reasonable to assume that the network has learned to be dependent on the mitral valve to classify an image frame.

For the second image occlusion example, (Section 4.5 on page 81) covering the mitral valve had a different effect than for the first example (Section 4.5 on page 81). For the first example, occluding the mitral valve caused it to seem closed, but for this frame, covering the mitral valve made it appear to the network to be open. Thus, in this case, the lack of mitral valve motion did not influence the network to classify the valve as closed. The only conclusion we can reach from this is that there is no one feature of the mitral valve that the network considers when trying to figure out whether it is open or closed, and both detection and motion can affect the result.

5.4 Data augmentation

We have not found any reasonable explanation for the substantial drop in validation accuracy that arose when training on augmented data. We see from Section 4.4 on page 79 that the validation accuracy is low, while the training accuracy is high which indicates that the network has overfitted and memorised the training data instead of learning useful features. First, we believed that the labels were somehow shuffled, either in the validation set or in the training set which would make it impossible for the model to learn anything. However, after inspecting both the training set and the validation set, we saw that they were labelled correctly.

Another possible explanation for the drop in validation accuracy might be that the added variance the augmentation provides to the training set changes the required learning rate. However, we trained the network with a learning rate ten times lower than the learning rate used without augmented data. The change in learning rate did not change the results noticeably, so it is unlikely this is the cause of the lack of validation accuracy.

We also hypothesised that there were too many augmented frames or that the augmentation obscured the valves. We limited the amount of augmentation to at most ten different augmentations per series as described in Section 3.3 on page 61 and visually inspected several hundred frames to make sure the valves were still visible within the frame. However, limiting the amount of augmentations did not affect the poor results.

The last step we took when exploring why training on the augmented data did not yield a sufficient accuracy, was to initialise the weights by first training on non-augmented data. Here we saw a sharp, but continuous, decrease in validation accuracy. A plot of the accuracy and loss is shown in Section 4.4 on page 80. This sharp decrease indicates that using the augmented data somehow destroys the information first learned by training on the non-augmented data. It was, due to the scope of this project, not feasible to inspect this phenomenon further.

5.5 Limitations

5.5.1 Challenges during labelling

An abundance of data are integral to train deep neural networks [31]. However, the dataset consists of only 66 labelled series with a total of 11626 frames. 44 of these series were used during training. This is a sizeable number of frames, however, they are highly correlated, which might increase the chance of overfitting.

Another factor that might impede our classification model is the consistency of the ground truth labels used for training. Because of the timeframe of the project, every frame was examined just one time each by someone with no medical background, which puts the quality of the labels in question. Furthermore, parts of the dataset were found to be a challenge to label accurately. Below we discuss the difficulties that arose during the labelling process for each of the different classes.

Both valves closed (class 0)

Both valves stay closed for a short period which means that most of the frames belonging to both valves closed (class 0) are near the transition to or from another class. The frames from the transition periods are a challenge to label. Sometimes the image is blurry, or there are noise and shadows in the image, which makes the exact transition point difficult to decide. Figure 3.3b on page 51 shows an example this difficulty. In addition to this uncertainty, the short duration of the class results in BVC (class 0) being the class with the least number of frames in the dataset, so the consistency of the labels is especially important. Because of these two problems, creating consistent ground truth labels for class 0 was a challenge.

Aortic valve open (class 1)

Because of noise and shadows, it is sometimes challenging to find the small aortic valve in the frames, which makes it difficult to separate between AVO (class 1) and BVC (class 0). Sometimes the valve is slightly behind a shadow or some noise that makes the valve look closed or open when it is the opposite. An example is shown in Figure 3.4b on page 52. When this happens, it is difficult to label the image consistently. The frames are also sometimes blurry around the valve which further complicates the labelling and the classification. These problems are particularly challenging when the valve is about to open or close, which are also the most relevant frames for our purpose.

Mitral valve open (class 2)

The large and obvious movement of the mitral valve and long duration of the ventricular relaxation phase makes this the class that was both the easiest to label and also the class with the most data in the dataset. Similar to class 0 and 1 there is sometimes shadows or noise that makes the labelling challenging around the start and end of the class, but since the valve is larger and clearer to spot, this is not as big of a problem as for the other classes. However, there is a "dip" when the valve appears to close before it opens again in the middle of ventricular relaxation. An example of this is shown in Figure 3.5b on page 53. This “dip” is hard to visually distinguish from the actual closing period when inspecting still frames, and it requires more frames or some temporal information to classify correctly.

We see that except for the “dip” during class 2, the frames that were demanding to label, are also the ones that are the most important for our problem. For valvular event detection, the transition frames between the classes are the most essential. Thus, inconsistent labelling of these frames can lead to an uncertain detection of the valvular events. Therefore, there might be some improvements to gain by creating more consistent ground truth labels. Mainly for the metrics measuring the precision of the transition point detection such as Mdiff (see Section 4.3 on page 76)

5.5.2 Challenges during training

It is apparent, from the semi-log plots in Section 4.2 on page 69 that the training loss for single frame classification networks had not fully converged at the point where we stopped it, and this makes it more difficult to compare this algorithm with the other algorithms. One might, for example, imagine that the networks got stuck in a saddle point, and allowing them to train for longer would improve their efficiencies. However, the test loss appears to increase for both the 5, 9 and 11 layer architectures, and the training accuracy reached near-perfect accuracy for all but five layers. This makes it unlikely that further training iterations would improve the validation results. Thus, because of hardware limitations, we ceased training for this architecture to allow us to test more algorithms.

A testament to the reliability of our results is that all the network models were trained more than once, but the results did not change noticeably between different runs. Multiple training runs are especially critical because dropout regularisation was applied. Had we only trained the networks once, dropout could make it harder to draw accurate conclusions about our results. Because adding more randomness in the training process with dropout might make a comparison of different algorithms less reliable. A single training run also has the possibility of getting stuck in a non-optimal local minimum point or a saddle point. However, we trained the networks several times and got persistent accuracy, which can be seen as an argument for our results being reliable.

Training a recurrent neural network from scratch on the image frames was found to be unfeasible with our limited hardware. The recurrent networks were, as mentioned in Section 3.2.3 on page 58, trained in a two-step fashion. First, a convolutional network was trained with the goal of classifying the images, then the features from the second-to-last layer were extracted for all images and used to train a bidirectional recurrent neural network. This was done to save both training time and memory. The two-step process is not the ideal way of training an RNN, however, it was the only way feasible with the available equipment and yielded good results.

Lastly, it is worth noting that training neural networks does not give a monotone increase in model quality. There are several reasons for this. Firstly, training is done using mini-batches which adds randomness to the training. The optimal weights for a mini-batch might be significantly different from the optimal weights of the whole dataset, and thus, a single iteration decreases overall model performance. Secondly, the networks overfit to the data. Regularisation through early stopping as described in Section 2.7 on page 39 is used to combat this issue of non-monotonicity.

Early stopping requires storing a cache of previous states of the network. Saving this state to disk is not only time consuming, but also space consuming as the model parameters might span several hundred megabytes. Saving these parameters every iteration is therefore impractical as much of the time spent training will be spent writing the parameters to disk. Additionally, hard disk space would quickly run out if the parameters are saved for each iteration. Therefore, we only saved the model every hundred iterations and only stored the last ten checkpoints.

Ten checkpoints is unfortunately not an immense amount, and ideally, more checkpoints should be stored to ensure that the best model is chosen. However, the goal of this project was not to make a state of the art model for valvular event detection, but rather a proof of concept. For this, the last ten iterations gave acceptable results.

5.5.3 Challenges evaluating the model

How we measure the effectiveness of a model should be motivated by the problem we want to solve. In our case, we want to detect the valvular event times which is the frames where the classification label changes from one frame to the next. Finding a metric that quantifies how well the model performs on this task is therefore critical to examine and compare different models

The frame-classification accuracy is not a sufficient metric to evaluate a models ability to detect valvular events. We see from Section 4.3 on page 71 that the VGG like architecture trained without timestamps got a high accuracy on the validation set (94%), but still just 2/11 accepted series (see Section 4.3 on page 72). Furthermore, from Sections 4.3 and 4.3 on page 71 and on page 72, we see that adding an RNN head to the best performing CNN model did not yield a significant improvement in accuracy (94.74% to 95.28%), yet the number of accepted series doubled from 4/11 to 8/11 (see Sections 4.3 and 4.3 on page 72 and on page 76). A high accuracy score is, therefore, not a good indicator of valvular event detection performance. This also begs the question if minimising the cross-entropy loss is sufficient. A better method might be to have a loss-function that directly tries to find the transitions.

The number of accepted series is also not a sufficient metric to evaluate the models. At least not when the validation and test sets are as small as the ones we have. To demonstrate this, we note that the VGG-like architecture has the same number of accepted series as the nine-layer fully convolutional architecture (when five frames were used for input). We see, in Section 4.3 on page 72, that the VGG-like architecture score better than the nine-layer fully convolutional network on all but metrics but the number of accepted series. From this, we can tell that the VGG-like architecture is better at not missing the true transitions and not predicting false transitions than the fully convolutional network despite the number of accepted series being the same.

For valvular event detection it is more problematic when our model completely misses a transition than when it predicts a false transition. Particularly if the false transition is clearly separated from the true transitions, e.g. when the mitral valve is falsely predicted as closed for a few frames during ventricular relaxation. False transitions like these are straightforward to fix with post processing. It is, however, more difficult to fix a dropped transition, e.g. a missing IVR or IVC phase (Figure 4.16b on page 85 shows an example of this).

The exact timing of the valvular events is not critical to estimate the left ventricular pressure curve needed to calculate a measure of myocardial work as described in 'A novel clinical method for quantification of regional left ventricular pressure–strain loop area: a non-invasive index of myocardial work'. In 'Assessment of wasted myocardial work: a novel method to quantify energy loss due to uncoordinated left ventricular contractions' Russell et al. inspects how sensitive this estimate is to incorrect timing of valvular events by calculating the pressure-strain loop area with early (+30 ms) and late (-30 ms) timings of AVC and AVO. Russell et al. found that this slight deviation from the correct timing had little effect on the pressure-strain loop area which they required for myocardial work estimation. Our best performing model had a mean absolute deviation of 11 ms ($\pm$ 13 ms) for the detected valvular events, which is adequate for creating an LV pressure curve for estimating myocardial work (it is worth noting that we calculated the mean over all detected valvular event times in the test set, not just AVC and AVO).

5.6 Further work

5.6.1 Improving the dataset

One obvious downside with this study is the limited dataset discussed in Section 5.1. Further studies should, therefore, work with a more extensive dataset with an increased number of labelled series. To speed up the acquisition of labelled data, we propose that the labelling process is executed in two steps. First, the bidirectional RNN with the relative time since the QRS peak is used to label all the images. Then, an experienced investigator can manually assess every frame and refine any errors done by the network. In this way, our model can save manual effort and facilitate the creation of an extended dataset

Furthermore, the ground truth labels used for this project was created by someone with no medical background. To ensure more consistent labels, we argue that two experienced observers inspect should every frame. Then, if the observers disagree on timings, the average can be used. This would ensure more consistent ground truth for the valvular event times, which would help the network to learn to detect the valvular event frames with more precision. Two observers would also allow for analysis of intraobserver reliability [32] which could be used to compare the reliability of the predictions with that of the ground truth labels.

5.6.2 Improving the methodology for evaluating the model

The metrics we used to measure the performance of the network when detecting valvular event times (see Section 3.4.1 on page 62)are averaged over all the events. To investigate what the network finds most challenging, it would be of interest to examine the different valvular events separately. This is especially useful for applications, such as measuring the wasted myocardial work[45], where the accurate predictions of some valvular event times are more critical than others.

Further studies should be done to investigate the VGG-like model trained on QRS relative timestamps to determine how strongly the model relies on the timestamps for prediction. We only visualised the VGG like model that was trained without timestamps as a feature. It would be interesting to analyse how much weight the networks give the image features compared to the timestamp feature and if the timestamp affects where in the image frame the network looks. This could be done straightforwardly by extracting and examining the weights of the final layers after the timestamp feature is concatenated or inspecting how an erroneous timestamp impacts performance.

5.6.3 Improving the model

Skip connections has not been used in any of the network that we tested. They have however been shown to yield state of the art performance on image classification tasks [22, 24, 50, 27]. The idea behind these networks is that it is difficult for a layer to learn the identity operator, which means that features learned in earlier layers will not be used during classification. By adding "skip connections", where the output of earlier layers are added [22, 24, 50] or concatenated [27] to the output of later layers, we allow the network to combine low-level and high-level features for classification. Using such skip connections allows for efficient training of very deep networks (more than 50 layers, up to 1000 layers [22]). We did not have the opportunity to train very deep networks because of memory concerns, however, we suggest that future studies investigate if there is anything to gain by using very deep networks with skip connections.

Our simple approach with a bidirectional RNN seems promising (Section 5.1). The bidirectional RNN provided a substantial improvement compared to the single-frame classification methods. Not only did it remove erroneous valvular events, but it also detected events that the feedforward networks missed. This performance increase suggests that recurrent architectures are a venue worth further study. Moreso considering that our model was trained in a two-step fashion, where the convolutional features were not trained to be used with an RNN. Therefore, we propose training both the convolutional features and the GRU-weights at the same time

We observed that the bidirectional RNN sometimes fails to understand long temporal relationships, especially when trained without timestamps as a feature (Section 5.1). This problem could be because we only feed the network 32 frames at a time which is not enough to contain a full cardiac cycle. We consequently suggest training a bidirectional RNN with large enough timestep to encompass a complete cycle. A greater temporal span could allow the network to recognise the pattern of valvular events that are consistent in all cycles and reduce the unnatural mistakes.

Another way of incorporating temporal information that we did not look into is to consider the video as a 3D image, and performing 3D convolutions on it. Such an approach was studied in 'Learning Spatiotemporal Features with 3D Convolutional Networks'[53] and yielded good results. It is straightforward to implement in TensorFlow, however it requires much memory, which is why we did not explore that approach.

We also discussed that single frame accuracy is not an apt measure for valvular event detection (Section 5.5.3), which is problematic because cross entropy focuses on this. Therefore, we propose that work is done to create a better loss function. This has been explored in other fields of image analysis, such as image segmentation [37], which is known to have a considerable class imbalance. The loss function in 'V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation' [37] focus on the sensitivity and specificity of the network, rather than the accuracy. This is beneficial for our problem as the sensitivity cannot be high if the network misclassifies a substantial amount of any class. As an effect of this, the loss function will penalise misclassifications of the frames where both valves are closed more than the other frames (as there are fewer BVC frames than AVO and MVO frames), which incidentally is the class we found most difficult to label correctly.

Other loss functions might also be considered. One could, for example, create networks that directly tries to find the valvular events, rather than classifying each frame. Such an approach could be based on the region proposal network introduced in [16], which propose bounding boxes for where objects are in an image. A similar approach was explored by Kong et al. to detect valvular events on MRI images and yielded state of the art results [30]. It would therefore be logical to test this approach on echocardiography series.

5.6.4 Examining the data augmentation results

We did not find an explanation for why training on the polar space augmented data yielded such poor results (Section 5.4). A possible next step could be to examine the types of augmentations one by one. This will ascertain if one of the augmentation methods are beneficial to training. Training with one augmentation method at a time can also determine if it was just one or a few of the augmentation schemes that caused the network to overtrain.

Chapter 6

Conclusion

With this project, we demonstrated that neural network models perform well on valvular event detection from sequences of echocardiography images. This was done through several experiments that compared different convolutional and recurrent neural network architectures. The best of which, achieved a 93% test accuracy and correctly detected all the valvular events in 7 out of 11 test series with a mean error of 1.03 frames (11ms($\pm$13ms)). While this model is not satisfactory for clinical use, it demonstrates that even relatively simple networks trained on limited data perform well on valvular event detection and can be used to save manual effort and open up opportunities for further automatic analysis of myocardial function.

Furthermore, two algorithms were used to visualise which parts of the images that are important for classifying frames; image occlusion [54] and guided back-propagation [48]. Both of these algorithms confirm that the network indeed detects the valves and utilises their location and motion for classification. We also observed that the model mainly focuses on the mitral valve. This focus suggests that it is more challenging for the model to detect the aortic valve.

Finally, we found two components of deep neural networks that have a definite positive effect on correctly predicting valvular events. The first component is max-pooling layers, which improved the performance of our convolutional networks. The second component is adding the relative time since the last QRS peak as a feature. Without this time, the network sometimes erroneously predicted both valves as closed during ventricular relaxation. This effect was seen both with and without an RNN head to the network. However, the RNN head was trained with a limited number of frames at a time. We, therefore, propose that further studies test the effect of adding the QRS-times when training RNNs with a greater number of continuous frames.

We have, to summarise, found that deep neural networks are a promising approach for automatic detection of valvular event times. There are, however, still a few notable hurdles to overcome. Firstly, an improved dataset is required. By this, we mean not only an increased quantity of labelled data but also more consistent ground truth labels. Our model may be used in a semi-automatic fashion when acquiring new data to facilitate the labelling process. Secondly, the experiments performed in this text were done under severe hardware restrictions, which prevented us from testing out state of the art algorithms in deep learning. Studies that explore the problem at hand without this restriction is, therefore, the logical next step to create a more efficient fully automatic algorithm.

Appendices

Appendix A

Results from the test series

A.1 Dataset statistics

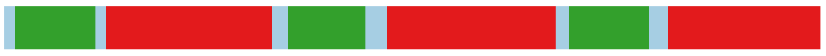
Series	diff	$\%PT$	$\%FP$	$Mdist$	accepted
03315171	2	100.0	15.4	0.182	NO
13399480	0	100.0	0.0	−0.091	YES
22053327	0	100.0	0.0	−0.091	YES
06186356	2	100.0	28.6	1.400	NO
13181261	0	100.0	0.0	0.182	YES
04796480	0	100.0	0.0	0.000	YES
02876314	2	100.0	28.6	−0.200	NO
23029318	0	100.0	0.0	−0.273	YES
52486393	0	100.0	0.0	−0.500	YES
55111328	0	100.0	0.0	1.091	YES
73496926	2	50.0	40.0	0.000	NO

Table A.1: Custom metrics evaluated on the test set for the model with the highest validation performance

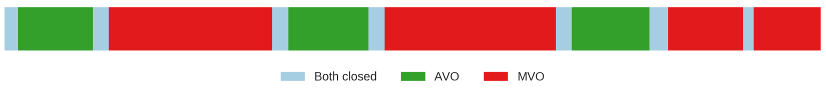
diff: *Difference in number of true transitions and number of predicted transitions.*
%PT: *Percentage of the true transitions that were correctly predicted.*
%FP: *Percentage of the predicted transitions that were false*
Mdist: *Mean distance between the correctly predicted true transitions and the corresponding predictions*
accepted: *Whether or not the series had 0%FP and 100%PT.*

A.2 Classification plots

03315171



(a) True labels

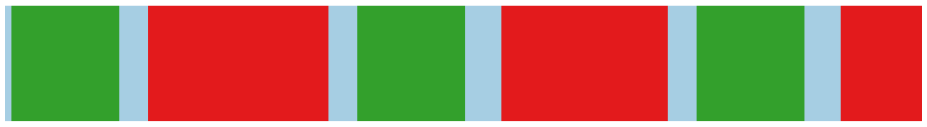


(b) RNN-t predictions

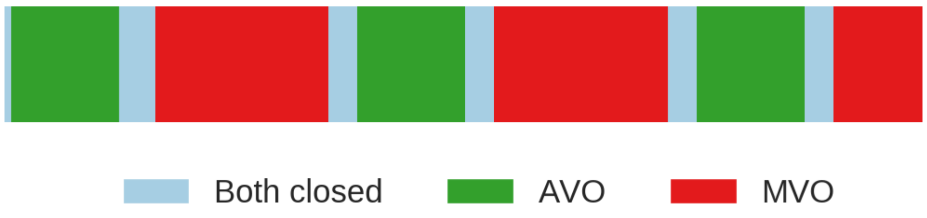
Figure A.1: Results from series 03315171

A.1a: *Visualisation of the true labels for series 03315171.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.1b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*

13399480

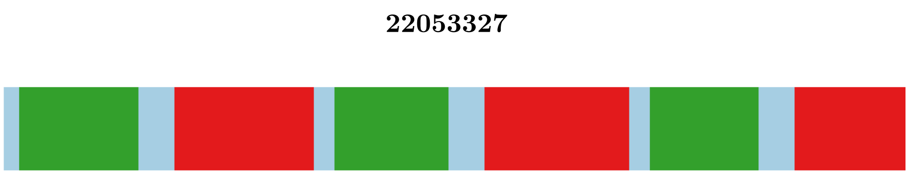


(a) True labels

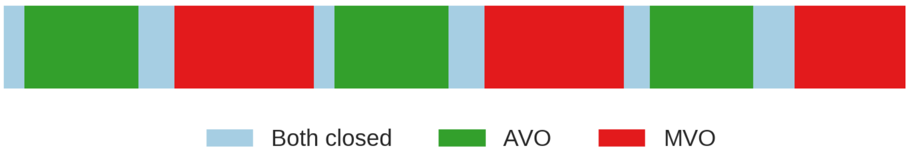


(b) RNN-t predictions

Figure A.2: **A.2a:** Visualisation of the true labels for series 13399480.The horizontal axis represents time in frames and the label at that frame is represented with a colour. **A.2b:** Visualisation of the corresponding labels predicted by the RNN-t model



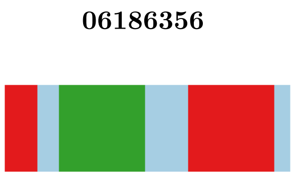
(a) True labels



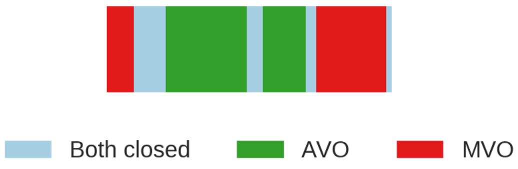
(b) RNN-t predictions

Figure A.3: Results from series 22053327

A.3a: *Visualisation of the true labels for series 22053327.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.3b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*



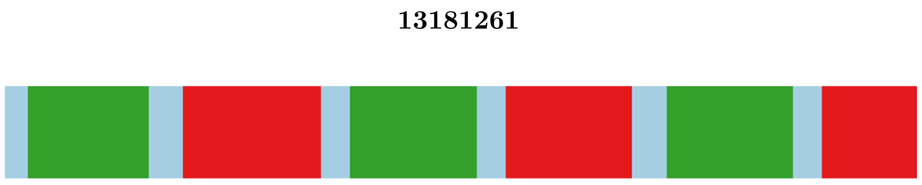
(a) True labels



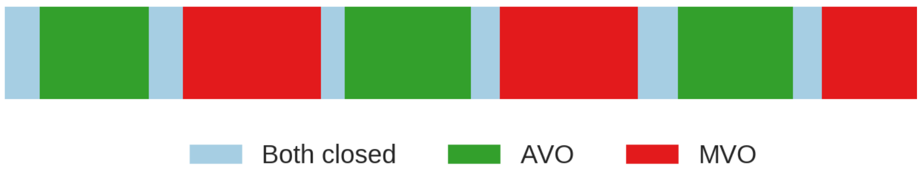
(b) RNN-t predictions

Figure A.4: Results from series 06186356

A.4a: *Visualisation of the true labels for series 06186356.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.4b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*



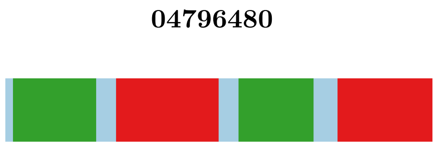
(a) true labels



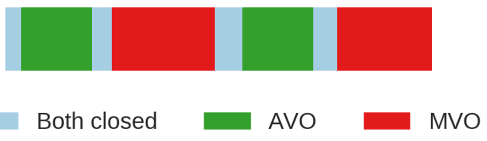
(b) RNN-t predictions

Figure A.5: Results from series 13181261

A.5a: *Visualisation of the true labels for series 13181261. The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.5b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*



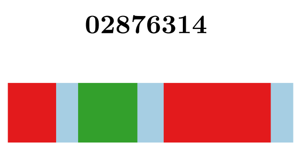
(a) True labels



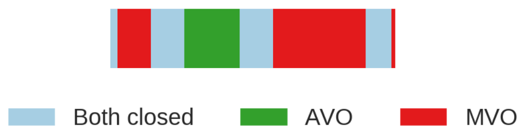
(b) RNN-t predictions

Figure A.6: Results from series 04796480

A.6a: *Visualisation of the true labels for series 04796480. The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.6b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*



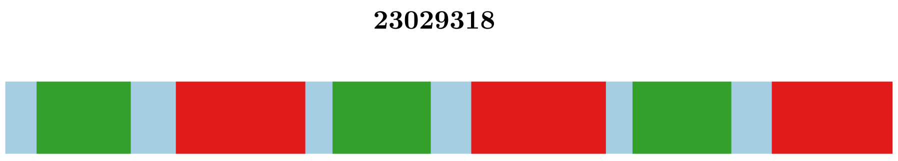
(a) True labels



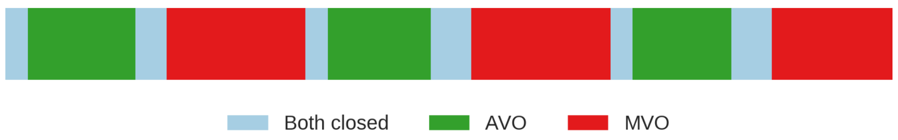
(b) RNN-t predictions

Figure A.7: Results from series 02876314

A.7a: *Visualisation of the true labels for series 02876314.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.7b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*



(a) True labels

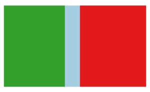


(b) RNN-t predictions

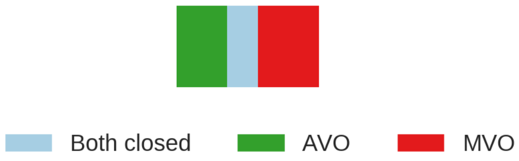
Figure A.8: Results from series 23029318

A.8a: *Visualisation of the true labels for series 23029318.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.8b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*

52486393



(a) True labels

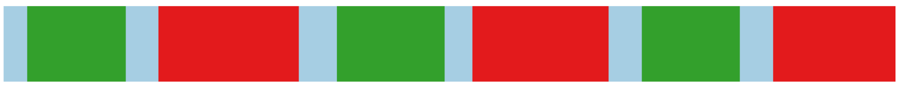


(b) RNN-t predictions

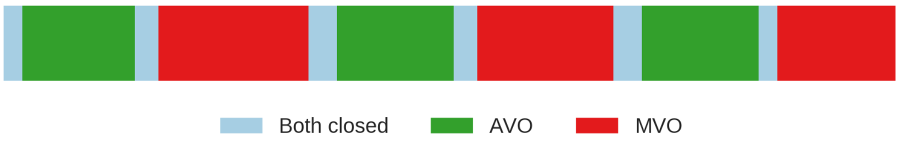
Figure A.9: Results from series 52486393

A.9a: *Visualisation of the true labels for series 52486393.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.9b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*

55111328



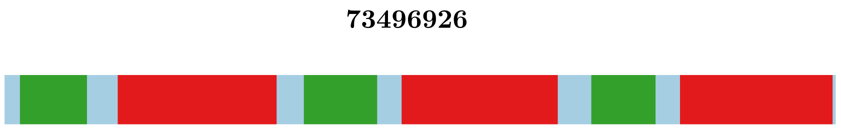
(a) True labels



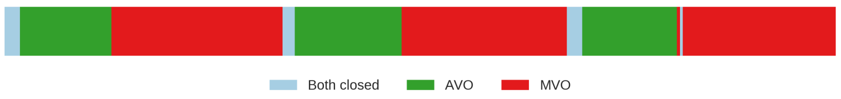
(b) RNN-t predictions

Figure A.10: Results from series 55111328

A.10a: *Visualisation of the true labels for series 55111328.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.10b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*



(a) True labels



(b) RNN-t predictions

Figure A.11: Results from series 73496926

A.11a: *Visualisation of the true labels for series 73496926.The horizontal axis represents time in frames and the label at that frame is represented with a colour.* ***A.11b:*** *Visualisation of the corresponding labels predicted by the RNN-t model*

Bibliography

[1] Svein Arne Aase et al. 'Automatic timing of aortic valve closure in apical tissue Doppler images'. In: *Ultrasound in Medicine & Biology* 32.1 (2006), pp. 19–27. ISSN: 0301-5629. DOI: `https://doi.org/10.1016/j.ultrasmedbio.2005.09.004`. URL: `http://www.sciencedirect.com/science/article/pii/S0301562905003613`.

[2] Gonzalo R. Arce. 'Median and Weighted Median Smoothers'. In: *Nonlinear Signal Processing.* John Wiley & Sons, Inc., 2005, pp. 80–138. ISBN: 9780471691853. DOI: `10.1002/0471691852.ch5`. URL: `http://dx.doi.org/10.1002/0471691852.ch5`.

[3] E. Arias-Castro and D. L. Donoho. 'Does median filtering truly preserve edges better than linear filtering?' In: *ArXiv Mathematics e-prints* (Dec. 2006). eprint: `math/0612422`. URL: `https://arxiv.org/pdf/math/0612422.pdf`.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[5] Steinar Bjaerum, Bjorn Olstad and Kjell Kristoffersen. *Ultrasound display of displacement.* US Patent 6,592,522. July 2003.

[6] L. Bottou, F. E. Curtis and J. Nocedal. 'Optimization Methods for Large-Scale Machine Learning'. In: *ArXiv e-prints* (June 2016). arXiv: `1606.04838 [stat.ML]`.

[7] W. Burger and M.J. Burge. *Digital Image Processing: An Algorithmic Introduction Using Java.* Texts in Computer Science. Springer London, 2016. ISBN: 9781447166849. URL: `https://books.google.no/books?id=YpzWCwAAQBAJ`.

[8] John C. Duchi, Elad Hazan and Yoram Singer. 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization'. In: 12 (July 2011), pp. 2121–2159.

[9] Shan Carter et al. 'Experiments in Handwriting with a Neural Network'. In: *Distill* (2016). DOI: `10.23915/distill.00004`. URL: `http://distill.pub/2016/handwriting`.

[10] KyungHyun Cho et al. 'On the Properties of Neural Machine Translation: Encoder-Decoder Approaches'. In: *CoRR* abs/1409.1259 (2014). arXiv: `1409.1259`. URL: `http://arxiv.org/abs/1409.1259`.

[11] Junyoung Chung et al. 'Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling'. In: *CoRR* abs/1412.3555 (2014). arXiv: `1412.3555`. URL: `http://arxiv.org/abs/1412.3555`.

[12] Yann Dauphin et al. 'Identifying and attacking the saddle point problem in high-dimensional non-convex optimization'. In: *CoRR* abs/1406.2572 (2014). URL: `http://arxiv.org/abs/1406.2572`.

[13] Andre Esteva et al. 'Dermatologist-level classification of skin cancer with deep neural networks'. In: *Nature* 542.7639 (2017), p. 115.

[14] Kevin Swersky Geoffrey Hinton Nitish Srivastava. *Lecture 6a Overview of mini-batch gradientdescent*. Coursera Slide. URL: `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[15] Parisa Gifani et al. 'Automatic detection of end-diastole and end-systole from echocardiography images using manifold learning'. In: *Physiological Measurement* 31.9 (2010), p. 1091. URL: `http://stacks.iop.org/0967-3334/31/i=9/a=002`.

[16] Ross B. Girshick. 'Fast R-CNN'. In: *CoRR* abs/1504.08083 (2015). arXiv: `1504.08083`. URL: `http://arxiv.org/abs/1504.08083`.

[17] Xavier Glorot and Yoshua Bengio. 'Understanding the difficulty of training deep feedforward neural networks'. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 249–256. URL: `http://proceedings.mlr.press/v9/glorot10a.html`.

[18] Xavier Glorot, Antoine Bordes and Yoshua Bengio. 'Deep Sparse Rectifier Neural Networks'. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 315–323. URL: `http://proceedings.mlr.press/v15/glorot11a.html`.

[19] Ian J. Goodfellow et al. 'Maxout Networks'. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, pp. III-1319–III-1327. URL: `http://dl.acm.org/citation.cfm?id=3042817.3043084`.

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

[21] Seung Seog Han et al. 'Deep neural networks show an equivalent and often superior performance to dermatologists in onychomycosis diagnosis: Automatic construction of onychomycosis datasets by region-based convolutional deep neural network'. In: *PloS one* 13.1 (2018), e0191493.

[22] Kaiming He et al. 'Deep Residual Learning for Image Recognition'. In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[23] Kaiming He et al. 'Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification'. In: *CoRR* abs/1502.01852 (2015). arXiv: `1502.01852`. URL: `http://arxiv.org/abs/1502.01852`.

[24] Kaiming He et al. 'Identity Mappings in Deep Residual Networks'. In: *CoRR* abs/1603.05027 (2016). arXiv: 1603.05027. URL: http://arxiv.org/abs/1603.05027.

[25] Geoffrey E. Hinton et al. 'Improving neural networks by preventing co-adaptation of feature detectors'. In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[26] Jie Hu, Li Shen and Gang Sun. 'Squeeze-and-Excitation Networks'. In: *CoRR* abs/1709.01507 (2017).

[27] Gao Huang, Zhuang Liu and Kilian Q. Weinberger. 'Densely Connected Convolutional Networks'. In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[28] Sergey Ioffe and Christian Szegedy. 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[29] Diederik P. Kingma and Jimmy Ba. 'Adam: A Method for Stochastic Optimization'. In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[30] Bin Kong et al. 'Recognizing end-diastole and end-systole frames via deep temporal regression network'. In: *International conference on medical image computing and computer-assisted intervention*. Springer. 2016, pp. 264–272.

[31] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. 'ImageNet Classification with Deep Convolutional Neural Networks'. In: 25 (Jan. 2012). URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[32] J. Richard Landis and Gary G. Koch. 'The Measurement of Observer Agreement for Categorical Data'. In: *Biometrics* 33.1 (1977), pp. 159–174. ISSN: 0006341X, 15410420. URL: http://www.jstor.org/stable/2529310.

[33] Yann LeCun et al. 'Efficient BackProp'. In: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, 1998, pp. 9–50. ISBN: 3-540-65311-2. URL: http://dl.acm.org/citation.cfm?id=645754.668382.

[34] Min Lin, Qiang Chen and Shuicheng Yan. 'Network In Network'. In: *CoRR* abs/1312.4400 (2013). arXiv: 1312.4400. URL: http://arxiv.org/abs/1312.4400.

[35] Andrew L. Maas, Awni Y. Hannun and Andrew Y. Ng. 'Rectifier Nonlinearities Improve Neural Network Acoustic Models'. In: 2013.

[36] Razvan O Mada et al. 'How to define end-diastole and end-systole?: Impact of timing on strain measurements'. In: *JACC: Cardiovascular Imaging* 8.2 (2015), pp. 148–157.

[37] Fausto Milletari, Nassir Navab and Seyed-Ahmad Ahmadi. 'V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation'. In: *CoRR* abs/1606.04797 (2016). arXiv: 1606.04797. URL: http://arxiv.org/abs/1606.04797.

[38] Yurii Nesterov. 'A method of solving a convex programming problem with convergence rate O (1/k2)'. In: *Doklady ANSSSR (translated as Soviet.Math.Docl.)* 1983, pp. 543–547.

[39] Chris Olah, Alexander Mordvintsev and Ludwig Schubert. 'Feature Visualization'. In: *Distill* (2017). https://distill.pub/2017/feature-visualization. DOI: 10.23915/distill.00007.

[40] M. Oquab et al. 'Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks'. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition.* June 2014, pp. 1717–1724. DOI: 10.1109/CVPR.2014.222.

[41] Razvan Pascanu, Tomas Mikolov and Yoshua Bengio. 'On the difficulty of training recurrent neural networks'. In: *International Conference on Machine Learning.* 2013, pp. 1310–1318.

[42] Razvan Pascanu, Tomas Mikolov and Yoshua Bengio. 'Understanding the exploding gradient problem'. In: *CoRR* abs/1211.5063 (2012). arXiv: 1211.5063. URL: http://arxiv.org/abs/1211.5063.

[43] Ning Qian. 'On the momentum term in gradient descent learning algorithms'. In: *Neural Networks* 12.1 (1999), pp. 145–151. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(98)00116-6. URL: http://www.sciencedirect.com/science/article/pii/S0893608098001166.

[44] Kristoffer Russell et al. 'A novel clinical method for quantification of regional left ventricular pressure–strain loop area: a non-invasive index of myocardial work'. In: *European Heart Journal* 33.6 (2012), pp. 724–733. DOI: 10.1093/eurheartj/ehs016. eprint: /oup/backfile/content_public/journal/eurheartj/33/6/10.1093_eurheartj_ehs016/3/ehs016.pdf. URL: +%20http://dx.doi.org/10.1093/eurheartj/ehs016.

[45] Kristoffer Russell et al. 'Assessment of wasted myocardial work: a novel method to quantify energy loss due to uncoordinated left ventricular contractions'. In: *American Journal of Physiology-Heart and Circulatory Physiology* 305.7 (2013). PMID: 23893165, H996–H1003. DOI: 10.1152/ajpheart.00191.2013. eprint: https://doi.org/10.1152/ajpheart.00191.2013. URL: https://doi.org/10.1152/ajpheart.00191.2013.

[46] Karen Simonyan, Andrea Vedaldi and Andrew Zisserman. 'Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps'. In: *CoRR* abs/1312.6034 (2013).

[47] Karen Simonyan and Andrew Zisserman. 'Very Deep Convolutional Networks for Large-Scale Image Recognition'. In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[48] Jost Tobias Springenberg et al. 'Striving for Simplicity: The All Convolutional Net'. In: *CoRR* abs/1412.6806 (2014). arXiv: 1412.6806. URL: http://arxiv.org/abs/1412.6806.

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

[50] Rupesh Kumar Srivastava, Klaus Greff and Jürgen Schmidhuber. ‘Training Very Deep Networks’. In: *CoRR* abs/1507.06228 (2015). arXiv: `1507.06228`. URL: `http://arxiv.org/abs/1507.06228`.

[51] Christian Szegedy et al. ‘Going Deeper with Convolutions’. In: *CoRR* abs/1409.4842 (2014). arXiv: `1409.4842`. URL: `http://arxiv.org/abs/1409.4842`.

[52] *TensorFlow An open-source software library for Machine Intelligence.* `https://www.tensorflow.org/`. Accessed: 2017-11-01.

[53] D. Tran et al. ‘Learning Spatiotemporal Features with 3D Convolutional Networks’. In: *ArXiv e-prints* (Dec. 2014). arXiv: `1412.0767 [cs.CV]`.

[54] Matthew D. Zeiler and Rob Fergus. ‘Visualizing and Understanding Convolutional Networks’. In: *CoRR* abs/1311.2901 (2013). arXiv: `1311.2901`. URL: `http://arxiv.org/abs/1311.2901`.