CARDIAC ARRHYTHMIA DETECTION WITH MACHINE LEARNING

MASTER'S PROJECT REPORT

Alexander Lay

Performed in collaboration with Cassian Lewis and under the supervision of Henggui Zhang

Student ID: 10301226

School of Physics and Astronomy

The University of Manchester

December 2021

Contents

ABSTRACT

Two different deep neural networks were created, the first was a Convolutional Neural Network (CNN) which was used to classify different forms of cardiac beats. The CNN was a one-dimensional (1D) CNN with an input of a 1D array of data from the PTB-XL database and an output label for each of the 20 beat types in the data set. The second network was a Generative Adversarial Network (GAN), this uses the same input as the CNN along with a random array of noise. The GAN produces a set of computationally generated heart beat data in the same form as the input PTB-XL data. The 1D CNN model achieved an accuracy of 96.0% with data from PTB-XL on multi-class labeling and an accuracy of 89.2% for binary class. The CNN model then attained an accuracy of 93.7% when a combination of PTB-XL data and synthetic data from the GAN were used in binary classification.

1 Introduction

Cardiac complications affect around 7.6 million people in the UK alone (1), these complications can range from minor heart palpitations to serious life threatening conditions, such as Atrial Fibrillation (2). For many of these complications there are underlying features that can be detected with the use of an electrocardiogram (ECG) which allows for early diagnosis and prevention. Clearly, the early detection of these underlying features is of great importance to the medical world due to its life saving potential. Machine learning has been demonstrated to assist with these detections (3), it provides a quantitative assessment of ECG scans including an assessment of the probability that each beat has been detected. This has several advantages including the reduction in medical observation time. Some ECG scans may require many hours of data collection so the ability to classify the scans automatically with machine learning is of high interest to the medical world. Currently machine learning is also being used to carry out large scale data analysis in healthcare (4) as well as more general uses in areas such as social media (5).
To sufficiently train a model, as will be required in this project, a large sample size is needed and in many cases this is not possible, so a GAN (6) can be used in order to increase the training data set size. GANs are currently being used by companies such as Facebook (7) and Google (8) to assist with large scale data synthesis. In the PTB-XL (9) data set it was discovered that for specific beat types, such as Complete Right Bundle Branch Block (CRBBB) there was a relatively small sample size compared to that of normal beat types. Hence it was decided that a GAN, if correctly trained, would allow for equal amounts of each data type to be entered into the CNN for its training. Several different architectures were built throughout the semester and the effect each had on the CNN is shown in Figure 11. Please note that this represents the first semester report of a year long project and as such a decision on how to quantitatively rank the quality of data from the GAN has not yet been made so will not feature in this report but plans for how to achieve this will be detailed. Preceding construction on the GAN the architecture and hyper-parameters of the CNN(10) model were fine tuned in order to achieve maximum accuracy on the PTB-XL data set. The best model was then used with the additional synthetic data produced by the GAN. Limited studies exist on the generation of synthetic ECG signals and even fewer that incorporate the data used by this project. The GAN architectures in these studies were different so comparisons have been hard to make. However there have been many studies on CNN architecture for PTB-XL data and as such a comparison of how accurate this model is compared to other studies such as those by Sandra Śmigiel ,Krzysztof Pałczyński and Damian Ledziński (11).

2 Background

2.1 Background: Cardiac Arrhythmia and ECGs

Cardiac arrhythmia are defined as "a problem with the rate or rhythm of the heartbeat. During an arrhythmia, the heart can beat too fast, too slowly, or with an irregular rhythm."(12). During a cardiac arrhythmia it may feel like the heart is fluttering or racing heart and the consequences can be bothersome or even life threatening. These problems occur when the electrical signals controlling the heart's rhythms don't function correctly. The heart is made up four main sections, two upper left and right atria and two lower left and right ventricles. Within the heart there are two types of cell that interact with the electrical signals; conducting cells which carry the signal through the heart and muscle cells which contract upon interaction with the electrical signal (13). During a single heart beat the electrical signal starts at a group of cells called the Sinoatrial (SA) node, this pulse causes the two atrium to contract forcing blood down into the ventricles. The signal then arrives at the atrioventricular (AV) node where it is slowed down before the impulse causes ventricles to contract, the SA node then sends another signal and the cycle starts again. The SA node is sometimes known as the "Pacemaker"(14) of the heart as the rate of electrical impulse release is what dictates the persons heart rate. The full cycle can be observed in 1 below.

In order to study the heart a patient will wear an ECG, this involves attaching, depending on the type of ECG, several electrical nodes. These nodes measure the strength of the electrical impulses as they travel through the heart allowing the pulse to be seen on the ECG output. The main features of the ECG readings for a normal beat can be seen in Figure 5, with each of the P, Q, R, S and T waves matching the pulse as it moves through the heart (16). In cardiac arrhythmia anomalys will exist in the ECG signal such as a missing T wave.

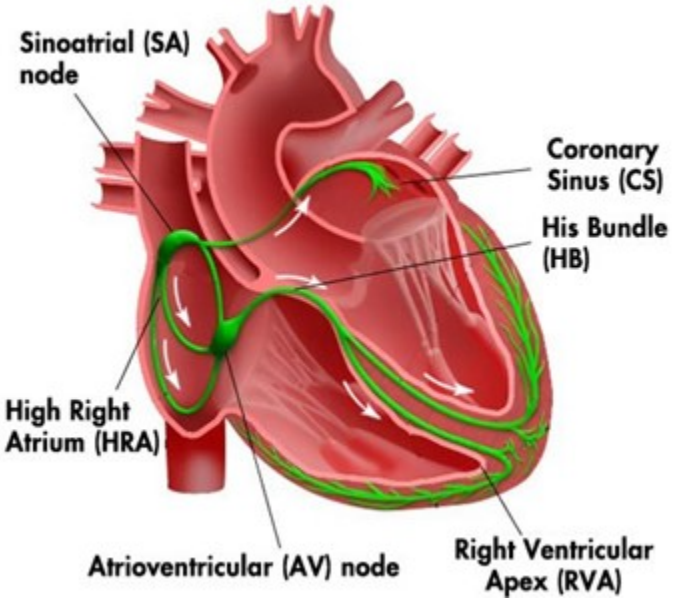

Figure 1: The figure shows the key electrical nodes within the heart along with the pathway that the electrical signal takes each cycle. The delay created at the AV node is what controls a person's heart rate and as such its electrical signal is measured during ECGs. When more than one lead is used this allows for different electrical signals at each point in the cycle to be measured. (15)

Other features that can be used to classify beats, include the time between the different waves as well as their intensity compared to previous beats. The CNNs created in this project are designed to identify these features automatically in a similar method to that of a health care professional.

2.2 Background: Neural Networks

The precursor concept for neural networks were developed 1943 by McCulloch and Pitts (17) when they modelled the neurones in the brain as logic gates which take several input signals and produce a binary output if the combined signals pass a threshold, as seen in Figure 3. In essence a neural network recreates this process with the inputs being from the data and the output being labels for the data-set. To illustrate this mathematically consider a function $\phi(z)$ where z is an input with a combination of input values x and there weights w such that,

$$z = x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + ... \tag{1}$$

The function $\phi(z)$ will then output a label depending on whether or not threshold values from the input are exceeded. Therefore in order for the network to learn and identify features the weighting on each of the values must be adjusted upon each iteration. The weights are updated as follows,

$$w = w + \Delta w \text{ where, } w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ ... \end{bmatrix}. \tag{2}$$

The change in weights can be calculated via the perception rule as follows,

$$\Delta w_j = \eta(y^i - \widehat{y}^i)x_j^i.(18) \tag{3}$$

Here η is the learning rate of the network and is considered a hyper-parameter, Δw_j is the change in weight per data point x_j^i. y^i is the true class label of the ith example with $\widehat{y}^i$ being the predicted class label. By iterating this process over the data the computer is able to update the weights each epoch and hence improve

the final result, this process is considered a single layer network (18).However most neural networks take the Adaline (19) approach as seen in equation 4,

$$\phi(xw^T) = xw^T \tag{4}$$

The difference being that the weights are updated based on a linear activation function as opposed to the step function seen in figure 2. The Adaline approach introduces a cost function which is aimed to be minimised in order to learn the weights, seen as $J(w)$ in equation (5),

$$J(w) = \frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2. \tag{5}$$

The advantage here is that the cost function is differentialable allowing the gradient descent to be used to help find the weights that minimise the cost function. The idea behind gradient descent is to take a step in the opposite direction to the gradient each epoch. Hence taking a step in the opposite direction to the gradient,$\Delta J(W)$, along with equation (2) the following is found,

$$\Delta w = -\eta \nabla J(w). \tag{6}$$

All symbols carry the same meaning as defined in equation (3) and this means the update of the weight per point can be written as,

$$\Delta w_j = \eta \sum_i (y^{(i)} - \phi(z^{(i)}))x_j{}^{(i)}. \tag{7}$$

This method is sometimes referred to as batch gradient descent (20) and is the approach used throughout this study. By combining many of these neurones a multilayered neural network can be created, this is not dissimilar to how the brain functions as it links many neurones together to achieve a desired outcome. As seen in Figure 3 there is an activation at each layer which is the step or linear function described earlier.

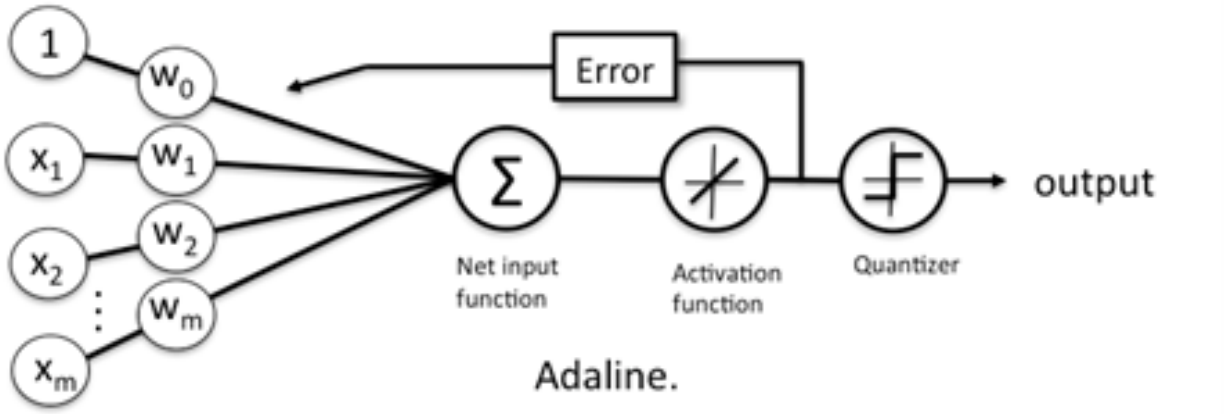

Figure 2: Shows the basic concept of neurone function. The neurone takes input values,x , along with the corresponding weights, w, a net input is then taken before passing this to the activation function allowing a new weight and error to be calculated. (18)

As there are now many more weights depending upon a final error, a back propagation (21) is preformed in order to calculate its derivative with respect to each of the weights in the network, this is one of the causes of a problematic phenomenon known as vanishing gradients. Back propagation allows the network to find the change in total cost due to the weight at a given node per each training example, as such the network is able to calculate the nodes and weights having the greatest effect on the final cost value and update them accordingly. For example the network with labels x, y and z may suggest a training example has a higher probability of label being x greater than y at the same time probability of y is equal to z and the true label is y. By backpropagation the network will drastically change the weights of the nodes leading to the incorrect

values of x and y but will not change the weights resulting in z as much as it is deemed closer to the true values. An unfortunate result of this arises if the initial value of cost is too small, as seen in sigmoid functions, the proportion of the weight per layer due to the error by become infinitesimally small resulting in gradients tending to zero hence vanishing gradients. If the reverse happens and the weights are too large, as can occur in linear functions, the gradients tend to infinity and exploding gradients occur. Back propagation is much more computationally cheap and hence is highly useful in machine learning (22).

2.2.1 Background: Convolutional Neural Networks

CNNs are a form of neural network for classifying images, discovered in 1989 (23). They function in a similar way to the human eye, the original idea came from a paper in 1960 (24) which had discovered the visual cortex has many different layers. CNNs are so good at image classification that Yann LeCun received the Turing award in 2018 for the original work (25).
All machine learning algorithms are judged by their ability to extract all the salient features from a data set, CNN algorithms are considered suitable for images as they can successfully extract features from raw data. A CNN will extract what are considered to be lower level features from the data set in the early layers, using more layers these features are combined into higher level features in a process known as feature hierarchy (26). These higher level features form what is known as a feature map where each element is formed from several adjoining points in the input data-set. Fully connected layers like those seen in Table 1 are used in order to help achieve this.

CNNs make use of convolutional layers in order to decide how to extract features from the data. A discrete convolution for two vectors x and y is mathematically defined as:

$$z = x * y \rightarrow z[i] = \sum_{k=-\infty}^{\infty} x[i-k]y[k](27) \tag{8}$$

Here the $[i]$ denotes the indexing of the vector, also note that the sum between ∞ and $-\infty$ is due to padding being used. Padding is the processes of adding an infinite number of zeros to either end of the data being used in the convolution, due to the computational cost this is not done to infinity in machine learning and is instead a finite integer length. The vector y is a random set of values and the vector size is known as the kernel size, hence a convolution can be seen as a sliding window across the data set computing values for the feature map. In this project 1D convolution was used for feature extraction due to the increased number of publications to compare results with. To assist with feature extraction Maxpooling layers were also used. These simply split the feature map up into sections and take the max value creating the higher level features mentioned earlier. A visual representation can be seen in Figure 4. At the end of this process fully connected layers and an activation function is used to produce a label for the data.

One final comment about CNN networks, is the use of regularisation in the process of reducing over fitting in complex neural networks such as this. The more complex a network becomes the more complex function it can learn to approximate. A side effect of this is that the network preforms very well on the data it is trained on but poorly on unseen examples. To help with this regularisation such as L1 and L2 (29) can be used to add penalties to the loss function of the network hence reducing the size of the weight parameters. An issue with this is the vanishing gradient problem mentioned earlier and hence a process called dropout was used in this report. Dropout is the system of severing connections between layers and in effect making the network slightly less accurate as information is lost about the previous layer. Commonly a percentage of connections are severed from one layer to the next, this process not only reduces over-fitting but also makes the network computationally cheaper as it reduces the amount of back propagation calculations being done.

2.2.2 Background: Generative Adversarial Networks

The second type of deep neural network created in this project was a GAN, with the aim of creating synthetic data. In machine learning when training any classification model it is highly important to train the model on an equal number of examples of each class, failure to do so may result in incorrect labelling or a network that

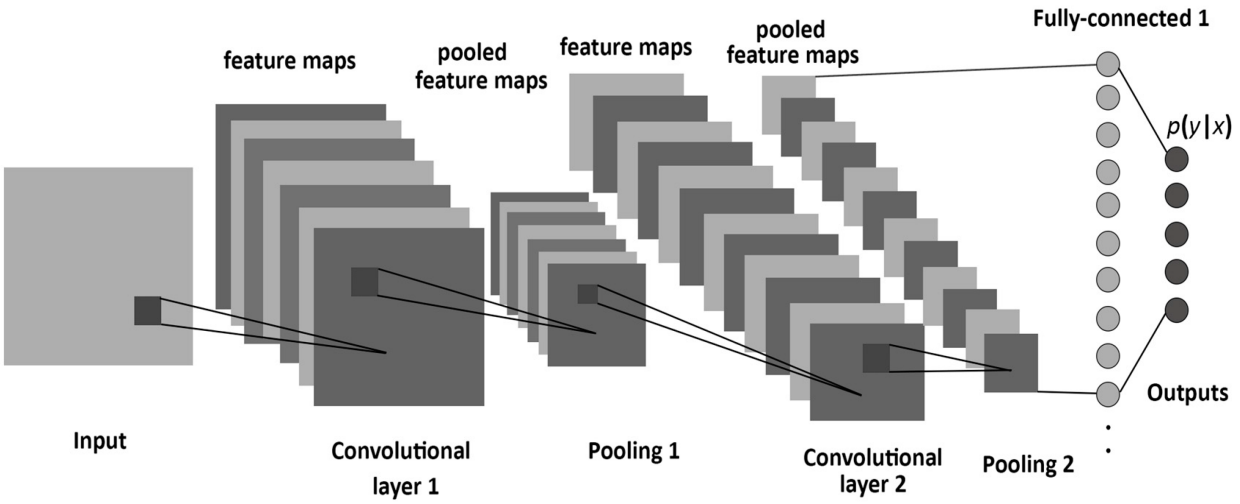

Figure 3: The figure shows the concept behind a 2D convolutional neural network for feature extraction and classification, A 1D variation was used for this project but the same concept being applied. The theory originates from the way in which the several layers in the eye feature extract from images. (28)

does poorly on some classes and better on others. For cardiac arrhythmia there are several rare beat types in comparison to others, hence less data exists and the CNN model cannot be trained as well.
To resolve this issue a GAN was built. Basic GANs, such as those built in this project, consist of two neural networks competing against each other, the first network is known as a generator and the second a discriminator. An illustration of this process can be seen in figure 5.

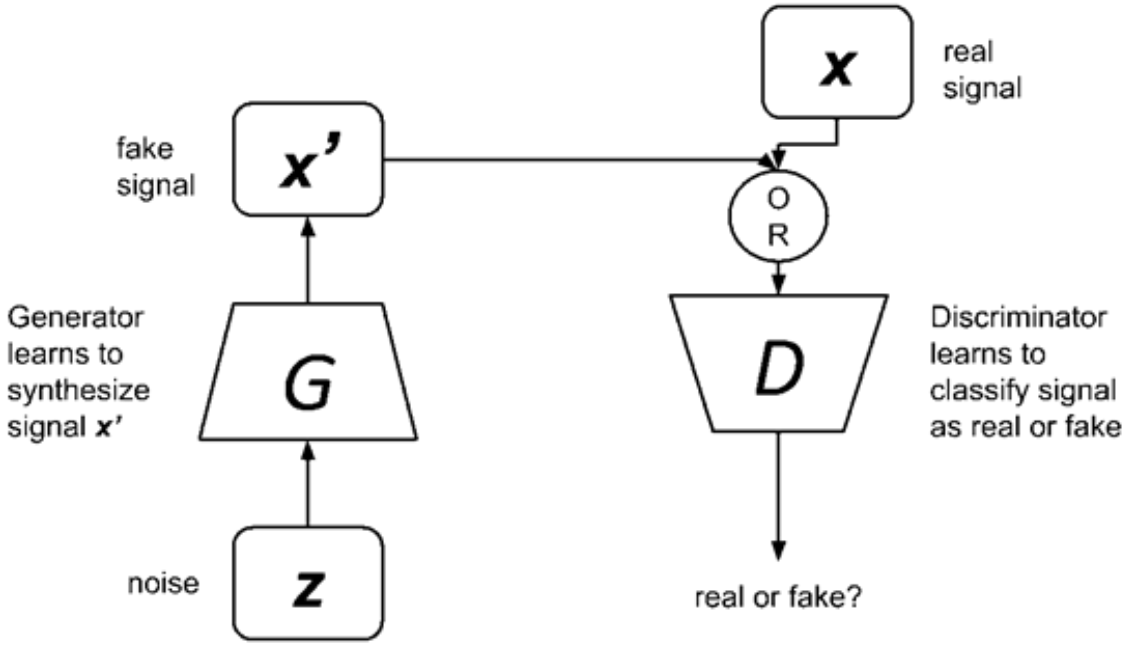

Figure 4: The figure shows the two competing networks in a GAN set up, the or gate represents the combination of fake an real data being added to the network. (18)

The generator's job is to take an input noise vector and from that create a synthetic data-set. Please note for this project the generator and discriminator were only trained on one beat type. The discriminator is then trained on the real data to allow it to learn the features of real data. The generator then feeds the discriminator the synthetic data with the goal of fooling the discriminator into classifying the synthetic data as real data. When the generator fails the weights of the generator are updated from the loss of the discriminator while at this stage the weights of the discriminator are frozen. Note the discriminator set up is similar to the CNN network seen in section 2.2.1, the discriminator has to classify between real and synthetic where real is labelled 1 and synthetic 0. Throughout this project several initial GANs were built using the binary cross entropy loss, $\mathcal{L}$, for the discriminator,

$$\mathcal{L}^{(D)}(\theta^{(D)}, \theta^{(G)}) = -E_{x\sim p_{data}} log\mathcal{D}(x) - E_z log(1 - \mathcal{D}(\mathcal{G}(z))). \tag{9}$$

The E represents the expected value of correctly identifying real data ,$log\mathcal{D}(x)$, given x distributed at p_{data} and similarly for the second term. Here $\mathcal{G}(z)$ is the fake input from the discriminator with z being the initial noise. Hence the loss function for the generator as it has no direct dependence from the real data,

$$\mathcal{L}^{(G}(\theta^{(D)}, \theta^{(G)}) = -E_z log(1 - \mathcal{D}(\mathcal{G}(z))). \tag{10}$$

Initially in this experiment the losses above were used but due to the significant dependence of one network upon another for its weight updates it is very easy if one network is out performing the other then huge instabilities can occur. This can lead to situations such as vanishing or exploding gradients as the number of epochs becomes large, the problem means that you cannot train a network for long enough to produce good data without the instability becoming too great. The solution chosen for this was the Wasserstien or Earth-Mover (EM) loss (30). Developed by Facebook for image production, the concept is to compare the difference in distribution between the real and synthetic data with the goal of minimising the distance between them for the generator and maximising for the critic. The discriminator is now called a critic as its value can be any number and is no longer constrained between 0 and 1. The full equation for EM distance can be found in the WGAN paper (30) but its simplest implemented into the loss function is as follows,

$$\mathcal{L}^{(D)}(\theta^{(D)}, \theta^{(G)}) = -E_{x\sim p_{data}} \mathcal{D}_w(x) + E_z \mathcal{D}_w(\mathcal{G}(z))) \tag{11}$$

$$\mathcal{L}^{(G)}(\theta^{(D)}, \theta^{(G)}) = -E_z \mathcal{D}_w(\mathcal{G}(z))). \tag{12}$$

Here w is the weight clipping parameter used to assist with keeping the function a 1 - Lipschitz function,

$$w \leftarrow clip(w, -1, 1) \tag{13}$$

In order make sure the generator and critic learn at the same rate it was suggested a set of 1-Lipschitz functions would be used. Lipschitz functions allow the computer to have a quantitative measure of how similar the output and input of a function are and update accordingly. Here this measure is being used to control the rate at which the critic and generator learn in order to improve stability. In the initial WGAN paper a process of weight clipping was used to constrain the network to the Lipschitz conditions, but this has been shown to have a negative effect on the ability to optimise a network (31). The suggested improvement to be used is a gradient penalty that takes advantage of the statement, "A differentiable function f is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere."(32). This allowed the new form of the Wasserstien loss equation to be used,

$$\mathcal{L}^{(D)}(\theta^{(D)}, \theta^{(G)}) = -E_{x\sim p_{data}} \mathcal{D}_w(x) + E_z \mathcal{D}_w(\mathcal{G}(z))) + \lambda E_{x\sim p_{data}}[(\|\nabla_x \mathcal{D}(x)\|_2 - 1)^2] \tag{14}$$

which has been proven to be better at maintaining stability for many different architectures of GANs(33).

The final piece of background that will be provided is the Long Short Term Memory (LSTM) (34) layer that was used in the latter part of the project. After several poor results from earlier GANs, research was carried into the use of a bidirectional LSTM layers which have proven to be effective at feature extraction on 1D time-dependent data (35). LSTMs are an extension of Recurrent Neural Networks (RNN), these networks make a calculation based on not only the current input but also inputs that have come previously. The issue is RNN networks do not have a long enough memory such that the number of earlier inputs that it can take information from is very small. LSTM networks mitigate this problem by storing large amounts of previous inputs weighting them highly and carrying them through are therefore more suitable for this project.

3 Experimentation

3.1 Preprocessing

Before any models could be tested the ECG data had to be prepocessessed and divided into different class types and then into individual beats. The PTB-XL data set comes as a set of 5 super classes and 19 sub classes this allowed the project to choose between training networks to more varied or to more specific beat types. The dataset contained 118967 examples from 12 different leads, this totalled from 21837 individual ECG recordings. Training the GAN involved the use of only one sub class type at a time, however training the CNN involved up to all 5 super classes. In PTB-XL each of the 12 leads relates to each of the ECG sensors, this allows for networks to be trained and tested on a varying number of leads. For testing the data was split into two parts, 80% for training and 20% for testing the accuracy of the network on unseen data.

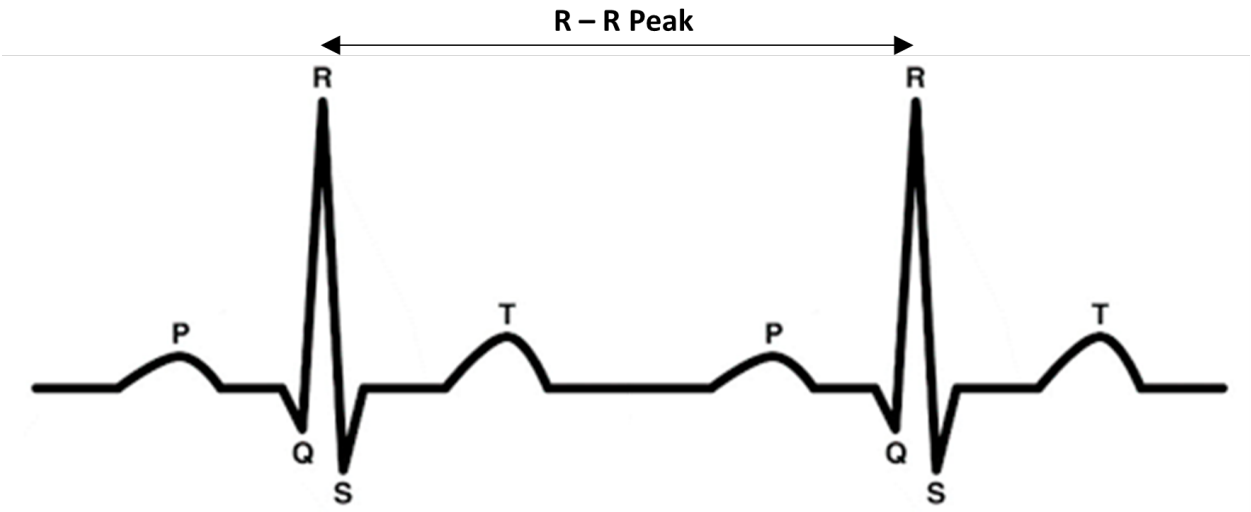

Figure 5: The Figure shows the R-R peak interval used for spliting the samples into different beats (36). Once split these beat could be used for the networks constructed.

Before splitting the data all non-numerical values were removed as Tensorflow has not been coded to accept NAN values. Following this in order to standardise the data set a normalisation function was applied to set the signal between an upper and lower bound, this was required due to the varying levels of baseline amplitude of the signals. Once this was complete the readings had to to be divided into different individual beats, to do this the R-R interval was used.

The R-R interval is the time between one R peak and the next as illustrated in Figure 8. To identify these peaks, first a rolling variance was passed over the data to remove smaller peaks from the set, then the maximum value from the recording was extracted. Finally the peaks for each lead were found and the mode location identified. Following this the distance from one beat peak to the next was measured, the R-R interval. To label the data as a beat the R-R distance was segmented into thirds. From here a section two thirds beyond the peak and one third preceding was classified as a single beat. This method was highly successful in capturing the main features of the beats. The final step was to normalise of the voltage value of the data between one and zero. This is essential in order to allow for sigmoid activation functions to be used in the CNN and GAN models which are shown to perform better.

3.2 CNN Architectures

Initially conversion of a simple Residual neural network (ResNet) (37) architecture, which was built for two dimensional images, to that of a one dimensional system was done. These networks take the input of layer l as a combination of the outputs from layers $l-1$ and $l-2$, this simplifies the network and the learning rate increases, as well as assisting with the vanishing gradient problem. This was deemed to be highly successful

when applied to single lead data sets but struggled to work with the 12 lead set up desired by the project. As a result two new models were built based on AlexNet (38) design which is known for its ability to classify on a higher number of features and class sets after it won the Large Scale Visual Recognition Challenge in 2012 (39). The success comes from the use of overlapping Maxpooling layers that were found to improve feature extraction for multi channel data sets. The final architecture tested can viewed in Table 1. A bidirectional LSTM layer was added later on due to its success in GAN architecture but was then chosen to be removed after it was found to be adding nothing to accuracy but was in fact increasing computational cost and run times. To optimise this network the hyper-parameters were adjusted until the highest accuracy was attained. Hyperparameters are parameters that are applied to the model before any training has begun, for this project the learning rate, Leakyrelu alpha value and dropout were hyperparameters. All networks in this project where build using pythons Tensorflow (40) package and as such layers will be denoted accordingly.

Layer	Dimensions	Parameters	Filter	Strides	Kernel	Activation
Convolutional1D	(None,48,96)	12768	96	11	4	-
Batch normalisation	-	384	-	-	-	LeakyRelu
Maxpooling1D	(None, 24, 96)	-	-	2	-	-
Convolutional1D	(None, 24, 256)	123136	256	-	5	-
Batch normalisation	-	1024	-	-	-	LeakyRelu
Maxpooling1D	(None, 12, 256)	-	-	2	-	-
Convolutional1D	(None, 12, 384)	295296	384	-	3	-
Batch normalisation	-	1536	-	-	-	LeakyRelu
Convolutional1D	(None, 10, 384)	442752	384	-	3	-
Batch normalisation	-	384	-	-	-	LeakyRelu
Convolutional1D	(None, 8, 256)	295168	256	-	3	-
Batch normalisation	-	1024	-	-	-	LeakyRelu
Convolutional1D	(None, 8, 128)	65664	128	-	2	-
Batch normalisation	-	512	-	-	-	LeakyRelu
Maxpooling1D	(None, 4, 128)	-	-	2	-	-
Globalaveragepooling1D	(None, 128)	-	-	-	-	-
Dense	(None, 128)	-	-	-	-	LeakyRelu
Dropout	-	-	-	-	-	-
Dense	(None, 128)	-	-	-	-	LeakyRelu
Dropout	-	-	-	-	-	-
Dense	(None, 12)	-	-	-	-	Sigmoid

Table 1: Above is the final CNN architecture used for beat classification, the parameters column shows the number of parameters in each layer. The activation column shows the the activation function used within each layer. The total number of parameters was 1,397,474 with only 3,008 being non-trainable. This network was used when assessing the effectiveness on accuracy of adding GAN data.

3.3 GAN Architectures

Throughout the project a large number of different variations of GANs were produced however only the most stable four were chosen to be compared in this report. The intial GANs built were based on Deep Convolutional GANs (DCGANs) (41), these take the idea of a CNN based discriminator and generator due to their known success with feature extraction. These networks struggled with stability due to inconsistent learning rates between competing networks so the Wasserstien loss function was added. One of these networks is shown in Table 2. The GAN built shows implementation of a LSTM layer in the generator due to the success shown in using these layers to produce sine waves (42).

The network shown uses the idea of a Variational Auto Encoder (VAE), this is the most parameter even network built and is the basis of future project work. The principle of a VAE is to first encode the data down to a small number of nodes and then decode the data out into the final result. For this network this was achieved by using convolutional layers to encode the data into a higher level feature map. From there the data was decoded using transposed convolutional layers which extract from a feature map and reverse the process

carried by the convolutional layer. When the data is decoded the aim is now for the higher level features to decode out from the desired features in a more precise manner. The main inspiration for this network was taken from work completed at The University of Nevada (43) with additions from earlier experimentation done in this project.

Layer	Dimensions	Parameters	Filter	Strides	Kernel	Activation
Convolutional1D	(None,200,32)	544	32	1	16	LeakyRelu
Droput	-	-	-	-	-	-
Convolutional1D	(None,200,64)	32832	64	1	16	LeakyRelu
Droput	-	-	-	-	-	-
Maxpooling1D	(None, 100, 64)	-	-	2	-	-
Convolutional1D	(None,100,128)	131200	128	1	16	LeakyRelu
Droput	-	-	-	-	-	-
Convolutional1D	(None,100,256)	524544	256	1	16	LeakyRelu
Droput	-	-	-	-	-	-
Maxpooling1D	(None, 50, 256)	-	-	2	-	-
Flatten	(None, 12800)	-	-	-	-	-
Dense	(None, 1)	12801	-	-	-	-
Bidirectional LSTM	(None,50,128)	33792	-	-	-	-
Convolutional1D	(None,50,64)	131136	64	1	16	LeakyRelu
Convolutional1D	(None,50,32)	32800	128	1	16	LeakyRelu
Convolutional1D	(None,50,16)	8208	256	1	16	LeakyRelu
Convolutional1DTranspose	(None,50,16)	4112	16	1	16	LeakyRelu
Convolutional1DTranspose	(None,50,32)	8224	32	1	16	LeakyRelu
Convolutional1DTranspose	(None,50,64)	32832	64	1	16	LeakyRelu
Convolutional1DTranspose	(None,50,64)	131200	128	1	16	LeakyRelu
Upsampling	(None,200,64)	-	-	-	-	-
Convolutional1D	(None,200,12)	24588	12	1	16	LeakyRelu

Table 2: Encoder - Decoder LSTM GAN network architecture, the Encoder - Decoder is the same as above with the bidirectional LSTM layer removed. The critic had 707,553 parameters with the generator having 674,796. There were no non-trainable parameters, the bidirectional layer had 64 units and padding was same throughout to reduce data loss. The first section of the table is critic and the second is the generator architectures respectively.

All four networks tested use an input noise vector array drawn from a random distribution. They all utilise dense fully connected layers to bring all the nodes back together to produce both the critic values and the synthetic data. In order to reduce the effect of overfitting and to improve the computational cost of the network, a series of dropout layers were added for all networks. This was seen to be highly effective even when the Wasserstein loss was not in effect. Batch sizes were kept to 64 as seen for the original WGAN paper (30) and a RMSProp optimisation with a learning rate of 0.00001 was in use. Note that in accordance with the WGAN paper throughout this section of the project the critic was trained seven times per epoch before the gradient penalty was applied and any Wasserstien loss was calculated. This was in an attempt to allow the network to function alone and update its weights manually, as it is a known stable feature extraction network and does not require Wasserstien loss when training on its own.

4 Results

4.1 Results: CNN

As noted in section 3.2, to optimise the CNN architecture each of the main hyperparameters were adjusted in order to find the highest possible accuracy for the network. Then a comparison with other studies on this data set could be made in order to asses the success of the network.

4.1.1 Hyperparameters

4.1.2 Hyperparameters: Learning rate

The learning rate of a network is defined as the size of the update steps per epoch. The size of the gradient update is set with 1 = 100%. In machine learning a network is aiming to minimise a function, hence the size of these steps is highly important. Too large and an algorithm may miss the true minimum of the function however too small and it may get stuck in small local minimum from which it cannot escape. The learning rate will also affect the speed of training. A smaller rate will require much longer training times but the accuracy may be better with the adverse for larger learning rates. As illustrated in Figure 6, larger learning rates such as 0.5 and 0.1 are unsuccessful, achieving less than 70% accuracy. Figure 6 also shows us that 0.00001 is too low and the network has become stuck within a local minimum. Clearly there exists an optimum value between 0.001 and 0.0001, looking to Figure 6 this appears to be 0.0003 achieving around 84% accuracy after only 70 epochs. After completing a K-fold validation it was found 0.001 was a more optimal value and as such was used for further training. Note the use of 70 epochs for comparison was due to the fact the network was found to consistently converge after this point and further training had little effect.

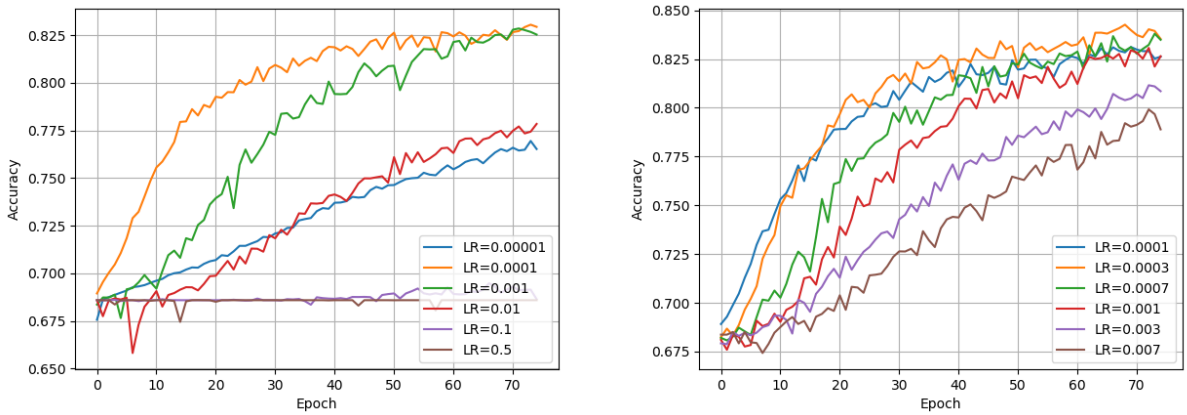

Figure 6: Both plots show the learning rate of the network and its effect on accuracy per epoch. The left plot is of the initial testing showing poor results for high learning rates, as such the right plot is further test around the values found to perform well.

4.1.3 Hyperparameters: Dropout

As discussed earlier the dropout of a network is used to remove connections between layers and to assist with the overfitting issue that many networks suffer from. Over-fitting occurs when the network becomes accurate at identifying the data it has been trained on but fails to classify unseen data as successfully. The dropout rate is the percentage of values that get set to zero (severed) with 1 = 100%. As seen in Figure 7, the accuracy of the network improves as the dropout decrease up to 0.15 where it appears to have a negative impact on the accuracy, suggesting that key features are being lost. After a K-fold validation had been complete it was decided 0.2 would be used as the final value for training.

4.1.4 Hyperparameters: Alpha

In each layer the activation function used was a LeakyRelu (44), again to assist with the vanishing and sparse gradients, mathematically this can be written as,

$$f(x) = \alpha x \text{ if } x < 0 \quad \text{and} \quad f(x) = x \text{ if } x \geq 0. \tag{15}$$

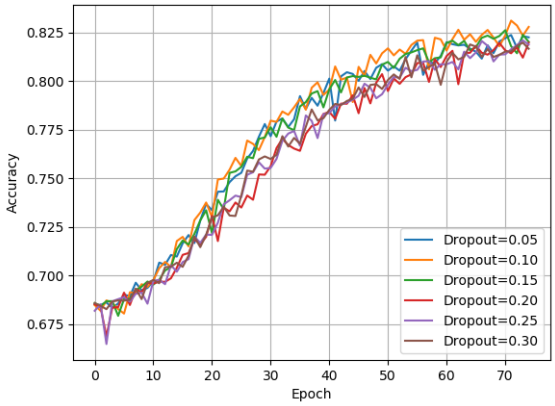

Figure 7: Plot shows the effect of dropout on the model's accuracy per epoch, the most successful value was found to be 0.1 but only by a relatively small amount.

This highlights to another hyperparameter α that can be be optimised for the network. For $\alpha = 1$ there is a linear activation function so it is logical for the tuning to have $0 < \alpha < 1$ as for $\alpha = 0$ we have the original Relu (44) function. As shown in Figure 8, the accuracy increases as α decreases. But as the value gets too small the network reverts to the exploding gradient issue and the accuracy begins to drop off again. The best value chosen for further experimentation extracted from the data in Figure 8 was 0.01.

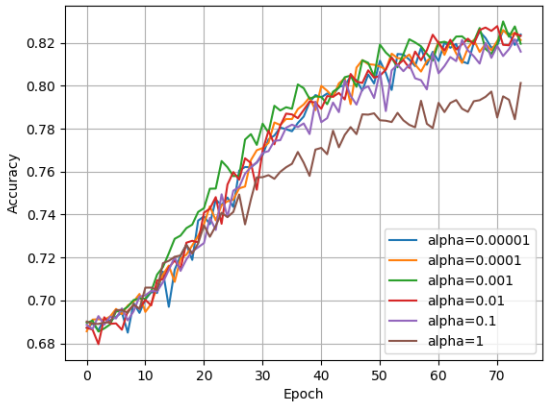

Figure 8: Plot shows the effect of alpha value on the model's accuracy per epoch, the most successful value was found to be 0.01 but only by a relatively small amount. With linear activation, alpha = 1, performing significantly poorly.

4.1.5 Further Tuning

As well as tuning hyperparameters, the effect of the type of optimiser used and the use of batch normalisation layers was tested. Using the data illustrated in Figure 9 it was concluded that Adamax (45) was the best optimiser. This is an updated version of the Adam optimiser that is better able to deal with noise in the previous epochs weights as it updates the network. The use of batch normalisation (46)was tested due to its known success in stabilising and accelerating networks. It achieves this by normalising layer inputs through re-scaling and re-centering, the reason it works so well is still under discussion but its effectiveness is echoed in Figure 9 where it can be seen to improve the accuracy of the network in this study by 3%.

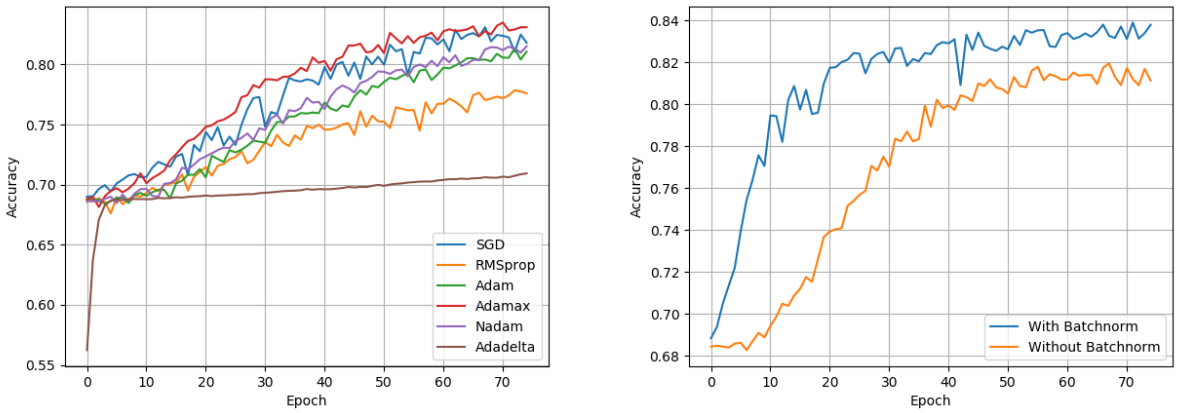

Figure 9: Plot on the left shows the effect of optimiser on the network's accuracy per epoch, with Adamax found to be the most accurate during training. The plot on the right shows the effect of the use of batch normalisation with a clear improvement being demonstrated.

4.2 Comparison with other studies

Once the model was correctly tuned it was run on the PTB-XL dataset in both binary class and multiclass set ups. To compare with other studies the F1 score was used, this metric will only be used in binary classification. There are four key parameters; probability true positive (TP), probability false positive (FP), probability true negative (TN) and probability false negative (FP). With these the following metrics can be formed,

With this the F1 score can be calculated,

$$PRE = \frac{TP}{TP + FP} \tag{16}$$

$$REC = \frac{TP}{FN + TP} \tag{17}$$

$$F1 = 2\frac{PRE \times REC}{PRE + REC} \tag{18}$$

The F1 score is a measure of the quality of the network on a given data set and hence comparisons between other networks can be drawn. Additionally Area Under Curve (AUC) was calculated as it a widely used measure of classification success. Due to the PTB-XL data being a multi-class dataset it was felt a binary distance measure between the binary encoded labels would be a more accurate comparison, as such a Jaccard score (47) was used.

Study	Task	Accuracy (%)	Jaccard Score	F1	AUC
Smigiel et el	Binary Classification	**89.2**	-	**0.891**	0.960
This Project	Binary Classification	82.5	-	0.825	**0.825**
Smigiel et el	5 multi-label super-classes	76.5	-	0.680	0.910
This Project	5 multi-label super-classes	**84.2**	0.665	**0.727**	**0.872**
Smigiel et el	20 multi-label super-classes	69.8	-	0.339	**0.815**
This Project	20 multi-label super-classes	**96.0**	**0.659**	**0.730**	0.918

Table 3: Table shows the final results of the CNN network in comparison to that of Smigiel et al, a clear improvement can be seen in multi label classification but a poorer performance is found for binary classification.

4.3 Results: GAN

To assess the quality of each GAN's data, the CNN network designed in section 4.1 was trained on each synthetic data set. The Accuracy was first compared for 2000 fake samples and the results for 75 epochs of training are summarised in Table 4. The effect of number of fake samples versus accuracy was then measured, this was completed by adding 250 fake samples and retraining the CNN. This was completed for a total of 6000 synthetic samples.

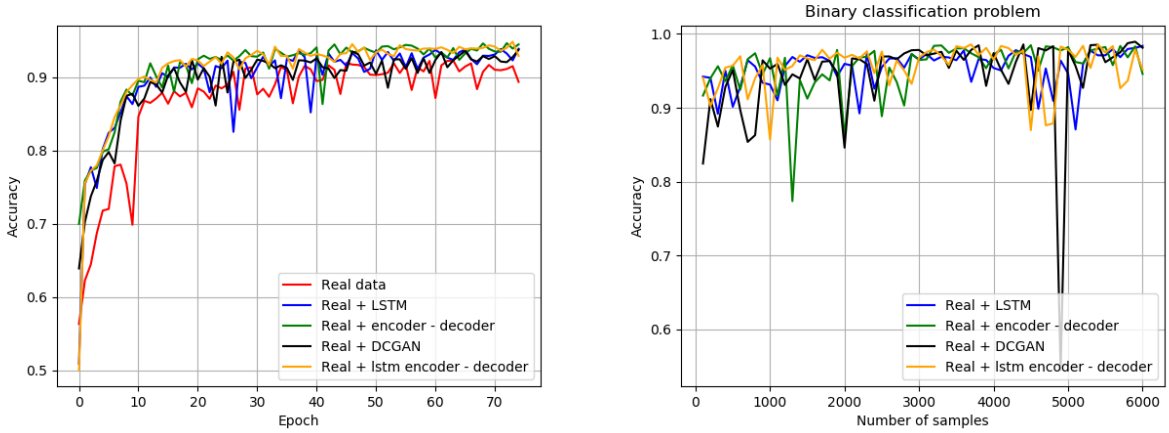

Figure 10: Plot on the left shows the effect of 2000 samples of synthetic data on the network accuracy per epoch, with encoder preforming the best. The plot on the right shows the effect of the number of synthetic samples versus the accuracy of the network.

Similar to section 4.2 in order to compare the effect of the data on the CNN's training and accuracy the AUC and the F1 scores were calculated. The final results can be seen in Table 4 with an improvement for F1 score as synthetic data is added but an undesirable increase occurring.

GAN Model	Accuracy (%)	F1 Score	AUC Score
Non	89.2	0.891	**0.960**
Basic LSTM	92.8	0.934	0.981
Basic DCGAN	90.2	0.967	0.995
Encoder - Decoder	94.1	0.971	0.993
Encoder - Decoder with LSTM	**93.7**	**0.976**	0.995

Table 4: Table shows the final results from adding fake data from each GAN data, for comparison 2000 samples of fake data were used.

5 Discussion

5.1 Discussion: CNN

As discussed earlier, in this report the issue of vanishing or exploding gradients is the source of failure in many neural networks. This phenomenon occurs when the back propagating weights of the network become infinitesimally small due to the original weight being too small, the opposite occurs for exploding gradients. These govern the stability of a network allowing it to achieve higher accuracy without collapsing. For the network discussed here the final result can be considered to be highly stable and accurate for multiclass classification but performing at a lower quality for binary classification. This is most likely due to the complexity of the network. The network has been designed to be able to classify a high number of multiclass labels and as such struggles when preforming binary classification. The network is clearly struggling to select common features in a data set with so much variation in the features compared to that of a single beat type.

Another issue arises in the seeming lack of improvement on the F1 score, this could be due to the network still struggling to classify those variations in the beats for which there are few examples and hence the GANs planned may be able to assit going forward. This network, regardless of the issues discussed, can still be considered to be an improvement on previous work.

5.2 Discussion: GAN

It can be concluded for the GANs built in this project the additional synthetic data data helped to improve the accuracy compared to a network without said data. The encoder - decoder network appear to perform best with the LSTM layer adding to the accuracy. The simple LSTM network did however perform better than the simple DCGAN suggesting the LSTM network should be further explored. It is also worth noting the poor performance of the CNN network in binary classification found in the early stages of the project so this can be considered to be very successful as all added data improved accuracy above 90.2% compared to 89.2% with the majority above 92.8%. The effect on the accuracy due to n increase number of samples appears to show a peak accuracy and stability around 3000 samples. This is interesting as the number of real samples was 3565 suggesting that using a number of synthetic samples above or below this creates a bias to one data set or the other. If this is the case then mode collapse, where a network produces data that is consistently similar and with too little variation, may have occurred. Examples of data produced by the network can be seen in Figure 11 clearly showing a successful recreation of real data. Finally it is known an optimal value for F1 score is 1, as such table 4 shows the LSTM Encoder - Decoder scoring highly here. However, as a desirable AUC score is 0.8 - 0.9 the CNN without any data appears to preform best suggesting that a bias has appeared in the dataset.

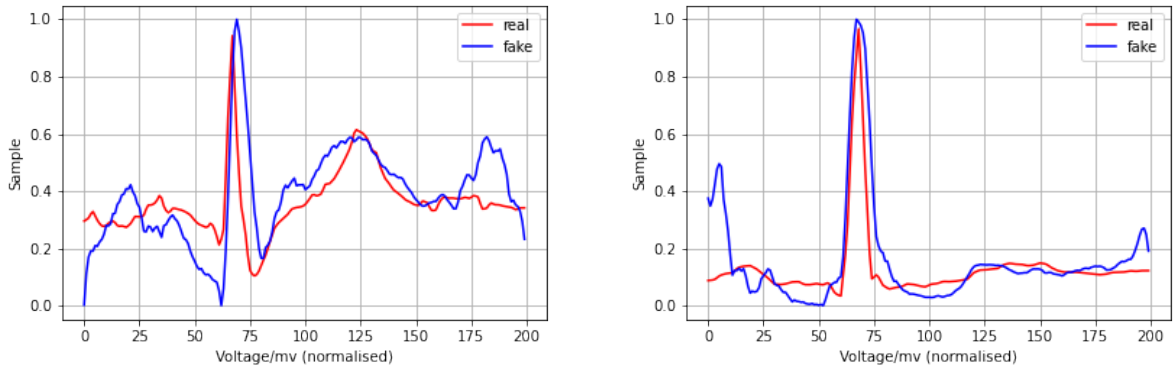

Figure 11: Plots shows examples of fake data versus the synthetic produced by the GANs. The left plot is data from the Encoder - Decoder without a bidirectional LSTM layer and the right is with the layer added.

6 Future work

With the CNN section of this project being such a success, it is felt that only minor adjustments would be required in future before commercial use can be recommended. The main concern for the CNN is the lack of data for certain data types such as CRBBB hence the main focus will be on the GAN. Currently the GAN suggested can only process one type of beat for each training cycle, this results in the requirement for numerous training runs. In future an Auxiliary classifier GAN (48) could be used, this would allow for all the data to be inputted at once with its corresponding label. Then a trained model would be able to produce a given number of a specified beat type on demand. A draw back of these GANs is their ability to extract the key characteristics of a beat, these being the peaks and troughs. Previous research groups have used DWT transformations into a 2D heat map to help isolate these peaks better, building on that and using an encoder with weightings added to the initial data may allow a clear feature map to be formed. The idea is based in CycleGANs (49) which are built to measure how successful a network is at encoding and creating a feature

map and then another network is used to measure how effective it is a recreating the initial data set. These auto-encoders are finely tuned for very specific feature extraction. Another possible development is the use of a Stacked GANs (50), again these feature extract from the data set but once features are identified it allows the input noise array to specify the size of values of these features in the synthetic example. For the data in this project, this would be the size of the peaks or if three beats were used the distance between them. Along with this incorporating aspects of a multi-label GAN (51) would allow multi-label classification to be completed with generated data. All these networks have been seen to be highly effective in their respective fields and show promise to aid with the research in this project.

Currently once the data is created the only method for measuring its quality in comparison to true data is to use the CNN network produced in this project. As a result, the quality of this metric may not be as valuable. In order to improve this a more quantitative metric to measure the synthetic data produced is required. In other work, measures such as Inception Score and Image Retrieval Performance have been used or additionally the Generative Adversarial Metric (52). Once a metric is chosen and a quality value for the data has been assigned it would be possible to weight each synthetic data set compared to the true data values when training any CNN network. This would potentially help reduce overfitting and help with identifying unseen data as the more fake data added the more the model can only detect examples that exist within the original data set. As with any project a larger sample size will assist with training. Part of this would involve being able to generalise a network to take in more than one type of data set, such as with a large number of labels or with a varying number of lead inputs.

7 Conclusion

In this project a 1D CNN network was built that has been more successful than previous works in classification of cardiac a arrhythmia for one dimensional 12 lead single beat data sets. This section will likely undergo limited extension over the next part of the project as the focus now shifts towards improving the data inputted into the network. The multiclass classification was found to be the most accurate and as such being able to produce data of this type would be highly useful. The aim is to build on the success of the binary GAN to help prepare the CNN network for real world application. Having built upon a firm base this project has already had some significant successes from which further research can be conducted.

References

[1] British Hear Foundation. Facts and figures.

[2] Elad Anter, Mariell Jessup, and David J. Callans. Atrial fibrillation and heart failure. *Circulation*, 119(18):2516–2525, 2009.

[3] Prajwal Shimpi, Sanskruti Shah, Maitri Shroff, and Anand Godbole. A machine learning approach for the classification of cardiac arrhythmia. In *2017 international conference on computing methodologies and communication (ICCMC)*, pages 603–607. IEEE, 2017.

[4] Arwinder Dhillon and Ashima Singh. Machine learning in healthcare data analysis: a survey. *Journal of Biology and Today's World*, 8(6):1–10, 2019.

[5] Vikas S Chavan and SS Shylaja. Machine learning approach for detection of cyber-aggressive comments by peers on social media network. In *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2354–2358. IEEE, 2015.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[7] Arantxa Casanova, Marlène Careil, Jakob Verbeek, Michal Drozdzal, and Adriana Romero-Soriano. Instance-conditioned GAN. *CoRR*, abs/2109.05070, 2021.

[8] David Berthelot, Tom Schumm, and Luke Metz. BEGAN: boundary equilibrium generative adversarial networks. *CoRR*, abs/1703.10717, 2017.

[9] Nils Bousseljot Ralf-Dieter Kreiseler Dieter Lunze Fatima I. Samek Wojciech Schaeffter Tobia Wagner, Patrick Strodthoff. Ptb-xl, a large publicly available electrocardiography dataset. 2020.

[10] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. *Object Recognition with Gradient-Based Learning*, pages 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[11] Sandra Śmigiel, Krzysztof Pałczyński, and Damian Ledziński. Ecg signal classification using deep learning techniques based on the ptb-xl dataset. *Entropy*, 23(9):1121, 2021.

[12] Blood National Heart and Lung Institute. Arrhythmia.

[13] Fang Chan-Dewar. The cardiac cycle. *Anaesthesia & Intensive Care Medicine*, 13(8):391–396, 2012.

[14] Mark R Boyett, Haruo Honjo, and Itsuo Kodama. The sinoatrial node, a heterogeneous pacemaker structure. *Cardiovascular research*, 47(4):658–687, 2000.

[15] Zhihao Jiang, Miroslav Pajic, and Rahul Mangharam. Cyber–physical modeling of implantable cardiac medical devices. *Proceedings of the IEEE*, 100(1):122–137, 2011.

[16] Rexford Kennamer and Myron Prinzmetal. The cardiac arrhythmias. *New England Journal of Medicine*, 250(13):562–571, 1954.

[17] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[18] Mirjalili Vahid Raschka, Sebastian. 2. training simple machine learning algorithms for classification, 2019.

[19] Bernard Widrow. *Adaptive" adaline" Neuron Using Chemical" memistors.".* 1960.

[20] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[21] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28, 1988.

[22] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. *Advances in Neural Information Processing Systems*, 29:4125–4133, 2016.

[23] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[24] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[25] A.M. TURING Award. For conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

[26] Swalpa Kumar Roy, Gopal Krishna, Shiv Ram Dubey, and Bidyut B Chaudhuri. Hybridsn: Exploring 3-d–2-d cnn feature hierarchy for hyperspectral image classification. *IEEE Geoscience and Remote Sensing Letters*, 17(2):277–281, 2019.

[27] Philipp Walk, Peter Jung, and Götz E. Pfander. On the stability of sparse convolutions. *Applied and Computational Harmonic Analysis*, 42(1):117–134, 2017.

[28] Saleh Albelwi and Ausif Mahmood. A framework for designing the architectures of deep convolutional neural networks. *Entropy*, 19(6), 2017.

[29] Robert C Moore and John DeNero. L1 and l2 regularization for multiclass hinge loss models. 2011.

[30] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[31] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[32] Jonathan Hui. Gan — wasserstein gan wgan-gp.

[33] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans: A consistency term and its dual effect. *arXiv preprint arXiv:1803.01541*, 2018.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[35] Juzheng Liu, Yue Yin, Hanjun Jiang, Huili Kan, Zongwang Zhang, Ping Chen, Binjie Zhu, and Zhihua Wang. Bowel sound detection based on mfcc feature and lstm neural network. In *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4, 2018.

[36] Naval Kishore and Sukhmanpreet Singh. Cardiac analysis and classification of ecg signal using ga and nn. *International Journal of Computer Applications*, 127(12):23–27, 2015.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[39] Jia Deng. *Large scale visual recognition*. PhD thesis, Princeton University, 2012.

[40] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[41] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[42] Sangwook Park, David K Han, and Hanseok Ko. Sinusoidal wave generating network based on adversarial learning and its application: synthesizing frog sounds for data augmentation. *arXiv preprint arXiv:1901.02050*, 2019.

[43] Khondker Fariha Hossain, Sharif Amit Kamran, Alireza Tavakkoli, Lei Pan, Xingjun Ma, Sutharshan Rajasegarar, and Chandan Karmaker. Ecg-adv-gan: Detecting ecg adversarial examples with conditional generative adversarial networks. *arXiv preprint arXiv:2107.07677*, 2021.

[44] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[45] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[46] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015.

[47] Ran Shi, King Ngi Ngan, and Songnan Li. Jaccard index compensation for object segmentation evaluation. In *2014 IEEE international conference on image processing (ICIP)*, pages 4457–4461. IEEE, 2014.

[48] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*, pages 2642–2651. PMLR, 2017.

[49] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.

[50] Xun Huang, Yixuan Li, Omid Poursaeed, John E Hopcroft, and Serge J Belongie. Stacked generative adversarial networks. In *CVPR*, volume 2, page 3, 2017.

[51] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. In *Machine learning for healthcare conference*, pages 286–305. PMLR, 2017.

[52] Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generative adversarial metric. 2016.