Applications of Machine Learning

by

Brosnan Yuen
B.Eng., University of Victoria, 2018

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Applications of Machine Learning

by

Brosnan Yuen
B.Eng., University of Victoria, 2018

Supervisory Committee

Dr. Tao Lu, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Mihai SIMA, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Tao Lu, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Mihai SIMA, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

In this thesis, many machine learning algorithms were applied to electrocardiogram (ECG), spectral analysis, and Field Programmable Gate Arrays (FPGAs). In ECG, QRS complexes are useful for measuring the heart rate and for the segmentation of ECG signals. QRS complexes were detected using WaveletCNN Autoencoder filters and ConvLSTM detectors. The WaveletCNN Autoencoders filters the ECG signals using the wavelet filters, while the ConvLSTM detects the spatial temporal patterns of the QRS complexes. For the spectral analysis topic, the detection of chemical compounds using spectral analysis is useful for identifying unknown substances. However, spectral analysis algorithms require vast amounts of data. To solve this problem, B-spline neural networks were developed for the generation of infrared and ultraviolet/visible spectras. This allowed for the generation of large training datasets from a few experimental measurements. Graphical Processing Units (GPUs) are good for training and testing neural networks. However, using multiple GPUs together is hard because PCIe bus is not suited for scattering operations and reduce operations. FPGAs are more flexible as they can be arranged in a mesh or toroid or hypercube configuration on the PCB. These configurations provide higher data throughput and results in faster computations. A general neural network framework was written in VHDL for Xilinx FPGAs. It allows for any neural network to be trained or tested on FPGAs.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank Minh Tu, Ahmed Magdy, Sanaul Haque, Yizhou Zhu, Luyun Gan, Saeed Farajollahi, Shahin Honari, and Elham Hosseini for their support in creating this thesis. Also, I would like to thank Prof. Tao Lu and Prof. Xiaodai Dong for supervising and mentoring me. This thesis is funded by the Natural Sciences and Engineering Research Council of Canada Graduate Scholarships-Master's Program, University of Victoria President's Scholarship, and Fortinet.

Chapter 1

Introduction

1.1 Types of Learning

In supervised learning [3], the AI is trained using a set of inputs and it must produce an exact set of outputs. The output set is explicitly crafted and is explicitly given to the AI. However, creating the output set requires significant time and effort from researchers. Moreover, the output set might be biased towards the researchers and introduces the over-fitting error. The over fitting error is one of the major problems that stops the AIs from generalizing across different datasets. On the other hand, researchers have created unsupervised AIs that do not require an output set. The unsupervised AIs are feed a set of inputs and they automatically extract useful data. Unsupervised AIs [4] are widely used in data mining as they do not require significant design time and effort from researchers. One of the drawbacks of unsupervised AIs is that they might produce garbage results because they are not guided by any output set.

Semi-supervised learning [5] is a hybrid of supervised learning and unsupervised learning. Semi-supervised AIs take a set of inputs and produce a set of outputs. However, the semi-supervised AIs do not know all of the information available in the output set. Semi-supervised AIs must try to reconstruct the full output set using only the input set and the partial information from the output set. Reinforcement learning [6] is special case of semi-supervised learning, where the AI explores a Markov chain. The goal of the AI is to reach the optimal state by traversing the Markov chain. At the start, the AI only knows very little about the Markov chain. As the AI explores more and more, knowledge about the Markov chain increases and the AI can make

better decisions. Eventually, the AI will find the optimal state.

1.2 Loss Functions

In 1801, the mean squared error (MSE) function was first used by Carl Friedrich Gauss for the celestial mechanics of Ceres [7]. MSE or L2 loss is mainly used for regression problems that have continuous outputs. When the squared function is replaced with the absolute value function, the loss function becomes the L1 loss. Cross entropy loss function was first used in "A mathematical theory of communication" [8] by Claude Elwood Shannon in 1948. Cross entropy is used for classification problems, where the outputs are probabilities. The Huber loss function was created by Peter Jost Huber [9] as an improvement to the MSE function. When optimization algorithms are applied to the MSE loss, the optimizer concentrates on lowering the loss on the outliers and ignores lowering the loss on the non-outliers because of the squared term. Huber loss fixes this problem by applying L1 loss to the outliers and L2 loss to the non-outliers.

For semi-supervised learning, the loss functions are slightly different. In generative adversarial networks (GAN) [10], the loss for the generator and the loss for the discriminator add up to zero. This means the generator and the discriminator compete against each other in a zero sum game. Furthermore, some researchers have added Wasserstein loss [11] to the GAN. On the other hand, loss functions for Q-learning [12] depends on rewards. For each action and state, there is a reward. Good actions will incur the highest reward in the future. Bad actions will incur the lowest reward in the future. As a result, the loss function for Q-learning depends on the current reward for current action as well as the future rewards for future actions.

1.3 Optimization Algorithms

Random search (RS) [13] was one of the first optimization algorithm, where the parameters of AI models are randomly chosen. If the new parameters yield a lower loss, then the optimizer moves to the new parameters. RS frequently gets stuck because it does not a high enough probability to escape the local minimums. On the other hand, simulated annealing (SA) [14] reduces this problem by allowing movements to higher losses with a probability. The probability changes as a function of temperature. High temperatures implies more movements to higher losses and low temperatures implies

less movements to higher losses. SA starts at a high temperature and gradually lowers in order to reach a global minimum. Bayesian optimization [15] is similar to simulated annealing, where the choice of the next parameters is random at the start and less random at the end. The actual model is first approximated by a surrogate model that is faster to sample. The uncertainty and the minimums of the surrogate model are determined by the acquisition function. Subsequently, the actual model is sampled at the most uncertain places, which yields the most information. Moreover, the actual model is also sampled at the minimums predicted by the surrogate model.

The optimization algorithms above are hard to parallelize since they use one particle by default. However, evolutionary and genetic algorithms [16] are built to use many particles. Multiple particles increase the diversity of the sample space. Evolutionary and genetic algorithms start with a population of random individuals, where each individual represents a set of parameters. The top 10% are selected as the elite population. Afterwards, the elite population is breed by mixing the parameters of individuals. Random mutations are added, and this produces a new population of individuals. Particle swarm optimization (PSO) [17] is similar to evolutionary and genetic algorithms as they all use multiple particles. In PSO, a set of particles is uniformly initialized with positions and velocities. At every time-step, each particle moves according to the previous position and the current velocity. The particles with the lowest losses are determined. Then the particles' velocities are perturbed towards the lowest losses. This makes the particles move towards the lowest known losses.

Ant colony optimization (ACO) [18] is an optimization algorithm used for effective traversals of graphs. Starting at vertex A, the goal is to reach vertex B with minimum cost. Firstly, ants randomly traverse the graph. Each ant selects an edge based on a probability. The probability depends on the cost and the pheromone level of the edge. Secondly, the pheromone level is updated based on the number of ants that selected this edge. Thirdly, all pheromone levels decrease due to the pheromones evaporating. The cycle repeats until the ACO converges to a single path.

Stochastic gradient descent (SGD) [19] is similar to Bayesian optimization because SGD also simplifies the model by constructing a surrogate model. SGD takes the Taylor series of the function and reduces the problem to a plane or a parabola. Then the SGD moves to the minimum predicted by the plane or the parabola. The Taylor series approximation could be first, second, or third order. The original SGD uses the first order Taylor series approximation, while ADAM [20] and L-BFGS [21] uses the second order Taylor series approximation. Moreover, some variants of SGD use

momentum similar to the PSO. SGD momentum can build up, which allows the SGD to escape local minimums.

1.4 Neural Networks

In 1943, Warren Sturgis McCulloch and Walter Pitts created the theoretical foundations of artificial neural networks "A Logical Calculus of the Ideas Immanent in Nervous Activity" [22]. It suggests information transmitted between neurons can be modeled as time delayed signals similar to modern spiking neural networks [23]. Moreover, it proposes a long chain of simple neurons could perform complex operations like the human brain. This paper also inspired the creation of fuzzy logic.

The innovation of the backpropagation algorithm [24] allowed neural networks to be trained using SGD. Soon, there were many different types of neural networks. The multi layer perception (MLP) was the first practical neural network. MLP used many different neural network layers to transform the input to the output. The convolutional neural networks (CNNs) used CNN layers to detect images. The CNN layers convolves the CNN filters with the images and the peaks of the resulting signals indicate the detections of spatial patterns. On the other hand, recurrent neural networks (RNNs) focuses on storing memories using neurons. RNNs are great for detecting temporal patterns like text and speeches. The deep belief network (DBN) is able to determine probabilities of any event in any system, which enables the DBN extract frequently occurring features from the system. Sparse neural networks are similar to MLPs. However, sparse neural networks have fewer connections between each neuron, which reduces overfitting and computational complexity. Many neural networks suffer from vanishing gradients due the backpropagation not reaching the first few layers. Residual neural networks were created to reduce the vanishing gradients. They solve this problem by adding new connections that bypasses hidden layers, of which decreases the distances from the final layer to the first few layers.

Many generative neural networks such as GAN, autoencoders, and self-organizing maps (SOMs) are able to create new samples not found in the training dataset. In autoencoders, the encoder component extracts information from the input and compresses it. Afterwards, the decoder component recreates the input at the output only using compressed information. Autoencoders are built towards data compression and data filtering. On the other hand, GANs are built towards semi supervised data generation. GANs have a generator component and a discriminator component. The

generator creates new samples, while the discriminator sorts the samples into fake and real categories. This forces the generator to create more realistic samples. SOMs are mainly used to generate visualizations of high dimensional data. Given a high dimensional input, the SOMs can project the data to a 2D plane.

1.5 Activation Functions

The history of the activation functions is presented below. The first neural network [25, 26] used the Sigmoid activation function, where the outputs of the activation functions are limited to range $[0, 1]$. The Sigmoid function is good for limiting the outputs of neural networks. The Sigmoid function belongs to the Sigmoid activation function family, of which the general Sigmoid equation [27] was developed by F. J. Richards in 1959. For the most part, the Sigmoid family is used for classifying objects, where $\hat{y} = 1$ is the object existing and $\hat{y} = 0$ is the object not existing. Other activation functions in the Sigmoid family include the step function, the Tanh function [28] and clipped function. Unlike the Sigmoid function, the step function has a discontinuity at $x = 0$ and only outputs 0 or 1. On the other hand, the Tanh function is similar to the Sigmoid function as the Tanh function is constrained to range of $[-1, 1]$.

The ReLU activation function [29] is another popular activation function. The ReLU activation function outputs $y = 0$ if $x < 0$, otherwise it outputs $y = x$. Moreover, the ReLU function is part of the ReLU activation function family, where the behaviour of all functions in the family are linear $y = x$ when $x > 0$. The ReLU activation function family is mainly used for classification and reinforcement learning (RL) problems. The identity, LeakyReLU [30], Elu [31], and Softplus [32] functions are included in this family. The identity function $y = x$ is typically used for the output layer of regression. The LeakyReLU is a version of ReLU that has a slight slope $y = \alpha x$ when $x < 0$. The slight slope is used to prevent the gradient from reaching zero. One of the problems the ReLU and LeakyReLU functions encounter is the discontinuity at $x = 0$ that produces undetermined gradients. To remove undetermined gradients, the Elu, and Softplus function are developed to have smoothness around $x = 0$.

The Gaussian activation function [33] has a bell shaped curve and is useful for modeling Gaussian distributed random variables. For example, a neural network predicting the speed of a car might use the Gaussian function for regression because the speed of a car is Gaussian distributed. Sometimes, the Gaussian function is

used for classifying the existences of objects. The Gaussian function is special case of the Radial Basis Functions (RBFs) [34], whose functions are always shaped like a bell shape curve. Other members of the RBFs include the polyharmonic spline and the bump function. Newer activation functions such as Mish [35] and Swish [36] have built-in regularization to prevent over-fitting of models. They look similar to the Softplus and Elu activation functions, however they converge to $y = 0$ when $x \to -\infty$ in order to eliminate large negative values.

Adaptive activation functions have been created by adding trainable parameters to the basic activation functions above. In this way, the optimizer decides the parameters of the activation functions instead of the researchers. PReLU [37] is an example of adaptive activation function, where the slope α of a LeakyReLU function is a trainable parameter. Bodyanskiy et al. [38] developed an adaptable RBF that can be trained in real time. Qian et al. [39] proposes adaptive ReLU functions for convolutional neural networks (CNNs). Campolucci et al. [40] proposes an adaptive spline to approximate the curves of a sigmoid function. The uniformly sampled spline uses fixed knot vector, fixed basis matrix, and the control points as the trainable parameters. The main problem with splines is the over-fitting. As the splines can fit all possible functions using the many trainable parameters, the spline may over-fit to the training set and may perform significantly worse in the testing test. Furthermore, the splines requires many additional constraints to allow continuity and differentiability.

1.6 QRS Complex Detection Problem Statement

Many heart diseases are diagnosed using electrocardiogram (ECG). The field of ECG analyses the electrical signals emitted by the heart, of which the most important ECG signals are the P wave, Q wave, R wave, S wave, and T wave. These waves could be used to detect arrhythmia, atrial fibrillation, and ventricular blocks. The direct detection of individual waves is very hard because there are many noises such as electrode contact noise, motion artifact noise, and amplifier noise. Furthermore, the search spaces of waves are very large as the ECG recordings are hours long. ECG segmentation is required in order to break the ECG signals into smaller ECG segments. The detection of the waves is easier in small ECG segments. Most of the time, the ECG signals are segmented at the QRS complex because the QRS complex is the easiest ECG signal to detect. QRS complexes are also useful for detecting the individual Q wave, R wave, and S wave because the waves are grouped up together.

The RR intervals can be calculated using the locations of the R peaks. The heart rate of a patient is determined using the mean of the RR intervals. As a result, the detection of QRS complexes is useful for diagnosing heart diseases.

1.7 Spectral Analysis Problem Statement

Spectral analysis is useful for identifying unknown substances. It has many applications in the fields of chemistry and biology. For gaseous mixtures, the individual concentrations could be determined using infrared (IR) spectra. The detection of hazardous gasses such as CO_2, CO, NO_2, and H_2S are extremely useful for industrial and medical applications. On the other hand, ultraviolet/visible (UV/Vis) spectra are used for detecting the concentrations of aqueous mixtures. For example, spectral analysis can determine the mass of sugar in soft drinks, it can predict the concentrations of proteins in blood, and it can detect new chemical compounds. Many algorithms like partial least squares (PLS), MLP, and CNN were developed to detect the concentrations of chemical compounds. The algorithms above require vast amounts of experimental data to converge. However, it is physically hard to concoct thousands of different mixtures because each mixture has different combination of compound concentrations. Moreover, experimental measurements require a lot of time and wastes many chemicals. Therefore, an algorithm is needed in order to synthesize many realistic samples from experimentally measured samples. This way, the machine learning algorithms will converge at a smaller number of experimental samples.

1.8 Neural Networks on Field Programmable Gate Array Problem Statement

Many neural networks have been implemented on graphical processing units (GPUs). All GPUs are dependent on the CPU and they communicate with each other using PCIe bus. This creates large latencies because the data in CPU's RAM has to be transferred to the GPU's RAM and vice versa. Moreover, the PCIe bus's throughput is slower than GPU RAM's read/write throughput. As the demand for processing power increases, more GPUs are clustered together using optic fiber. However, optic fiber has a few problems. Despite the optic fiber having a large bandwidth, the optic

fiber has large latencies due to the electronic to photonic conversion process. Furthermore, the GPU chips are physically far apart, of which creates wave propagation latencies. FPGAs are much more flexible because they don't depend on CPUs. FPGAs can pull data directly from hard drives and store them in RAM or in internal block RAM. This eliminates the PCIe bottleneck. Traces on the PCBs can provide direct communications between FPGAs. Moreover, Xilinx AXI interconnect protocol allows for the transfer of data between FPGAs. Furthermore, FPGAs can be arranged in grids or toroids to facilitate lower latencies.

1.9 Thesis Layout

Each chapter in this thesis contains an independent paper that solves a specific problem. Chapter 2 and Chapter 3 focuses on the detection of QRS complexes using different neural network algorithms. Chapter 2 implements the CNN-LSTM algorithm, while chapter 3 implements a machine learning pipeline. Chapter 4 and Chapter **??** discuss the applications of neural networks to spectral analysis. Chapter 4 uses the IR spectra to detect the concentrations of gasses mixtures. Subsequently, Chapter 5 details the implementation of a neural network on a FPGA. Finally, Chapter 6 gives the conclusion to this thesis.

Chapter 2

Inter-Patient CNN-LSTM ECG QRS Complex Detection

Electrocardiogram (ECG) is the most important and prevalent tool in diagnosing cardiovascular diseases. With the advancement of wearable technology, Internet of things (IoT) and mobile health, mobile wearable ECG for real-time long-term monitoring becomes increasingly possible anywhere and anytime in patients' hands. The direct result is that vast amounts of ECG data will be generated. The sheer volume of ECG recordings is prohibitive for cardiologists to handle. Therefore, accurate and automated ECG analysis is in urgent need to process the explosively growing number ECG recordings collected by wearable devices.

Computer aided ECG analysis is a field that has been developed for more than four decades. Numerous algorithms were devised and proposed for QRS complex detection and heartbeat classification in the literature [41, 42] and references therein. QRS complex detection is the critical first step, as QRS complex is the most prominent portion of a heartbeat signal and its detection facilitates the subsequent ECG analysis. In addition to heartbeat classification, basic parameters, such as RR, QT, PR intervals, etc., derived after QRS detection, are required for every ECG recording and reveals important information about heart functions. Therefore, literature is abundant with QRS complex detection.

Techniques used in QRS complex detection range from signal derivative and digital filters [43, 44, 45, 46, 47], wavelet transforms [48, 49, 50, 51, 52], Hilbert transforms [53, 54, 55], matched filters [56, 57], compressed sensing [58, 59], to machine learning and neural networks (NN) approaches [60, 61, 62, 63, 64, 65, 66, 67, 68]. Among

the many classical derivative and digital filter algorithms after the first Pan and Tompkins method [43], GQRS [47] is a simple one with superior performance by using adaptive search intervals and amplitude thresholds. Reference [50] uses wavelet transform and dynamic amplitude thresholding for QRS complex detection. The wavelet transform eliminates noise and other peaks from the ECG recordings, after which the generated pulse trains are scanned for the QRS complex peaks using the dynamic amplitude thresholding. This method has the advantage of being easy to implement and not needing a training phase. However, the wavelet transform uses a fixed filter pattern, which has the disadvantage of not adapting to different types of QRS complexes. Similarly, papers [69, 70, 71] employ noise filtering techniques to extract QRS complexes. A quadratic filter with dynamic amplitude thresholding is constructed in [71] for QRS complex detection, which has the same advantages and disadvantages of the wavelet transform filter.

There is a long history of using neural networks for ECG analysis. ECG signals are non-linear and non-stationary in nature, and hence methods that can adapt to changes are needed. Neural networks have such potential. Advancements in neural networks lead to new opportunities and design. Recently, Zihlmann et al. propose a convolutional neural network (CNN) followed by a long short-term memory (LSTM) network for ECG disease classification [72]. Jun et al. claim that a CNN with a fully connected layer classifies arrhythmia in ECG recordings [73]. Rajpurkar et al. [74] developed a 34-layer CNN for detecting arrhythmias in arbitrary length ECG time-series. For applying neural networks to QRS detection, [63] implements the first multi-layer perceptron (MLP) for QRS complex matched filtering. In [65], Xiang et al. utilize a CNN followed by a dense layer for QRS complex detection. The CNN filters the ECG signal, while the dense layer predicts the QRS complexes. The CNN has the advantage of adapting to different types of QRS complexes, but it does not directly predict the timing information of R peaks. Paper [66] segments the QRS complexes by removing the regions outside of the QRS complexes using the first CNN. Then the second CNN finds the starts and ends of the QRS complexes. Paper [67] implements an MLP with radial basis functions for QRS complex detection. Radial basis functions are better at filtering noise when compared to the regular sigmoid functions.

Despite of significant efforts, there are still unsolved challenges of QRS complex detection. First, when heavy noise, motion artifact and baseline wanders are present, robust algorithms are yet to be developed. In wearable device-based ECG measurements, signals can often be very noisy. Second, QRS complex varies from person to

person and even within one person's recording. For training-based methods such as NN, detection of new records not previously in the training dataset leads to unsatisfied performance. As mobile wearable ECG adoption increases, many patients' data are not labeled and not included in the training database. To address these challenges, this paper proposes, for the first time according to the authors' knowledge, a CNN-LSTM for QRS complex detection with the objectives of not only high classification accuracy but also small timing error. Moreover, the CNN-LSTM model developed has the ability to generalize to new patients' records. The CNN captures visual patterns and filters noises, while the LSTM detects timings of the QRS complexes. After that, an MLP formats the timing predictions and outputs the final QRS complex detection result. Finally, this paper performs inter-patient testing on the CNN-LSTM by training and testing on different ECG patient recordings. Inter-patient testing verifies the CNN-LSTM's generalization ability.

The rest of the paper is organized as follows. Section 2.1 discusses several related QRS complex detection algorithms in detail. Section 4.1 on data preparation shows the inter-patient test environment and the test parameters. The proposed CNN-LSTM neural network is presented in Section 3.3 and the simulations section, Section 2.4, compares the performance metrics of the CNN-LSTM to other QRS complex detection algorithms. Error analysis of CNN-LSTM is conducted in Section 2.5. Conclusions are given in Section 4.4.

2.1 Related QRS Complex Detection Algorithms

In this section, the following related QRS complex detection algorithms are presented: Pan and Tompkins [43], GQRS [47], Wavedet [48], Xiang et al.'s CNN [65], and Chandra et al.'s CNN [68]. The advantages and disadvantages of each algorithm are also described.

2.1.1 Pan and Tompkins

The Pan and Tompkins algorithm [43] is the first real time QRS complex detection algorithm, in which a bandpass filter is applied to reduce the noises in the ECG signals, and adaptive filters are used to detect the QRS complexes. The adaptive filters consist of an amplitude filter, a slope filter, and a width filter. In order to be marked as a QRS complex, an ECG peak must simultaneously meet all of the following criteria:

the peak's amplitude must be greater than an amplitude threshold, the peak's slope must be greater than a slope threshold, and the peak's width must fall within the range of a QRS complex width. The amplitude filter rejects the low amplitude signals, while the slope filter and the width filter eliminate the P waves and T waves. The advantages of the Pan and Tompkins algorithm are the fast processing times and low complexity. However, the filters used in the algorithm need to be engineered by hand, which requires a lot of time and expertise. Furthermore, the handcrafted filters can not adapt to different patients and environments.

2.1.2 GQRS

GQRS [47] is a classical QRS complex detection algorithm. Firstly, it calculates the means and the standard deviations of the RR intervals and the QRS complex amplitudes of the previously detected QRS. Secondly, the algorithm forms an adaptive search interval using the statistics of the RR intervals. Thirdly, the model creates an adaptive amplitude filter using the statistics of the QRS complex amplitudes. Finally, the adaptive amplitude filter is applied to the current adaptive search interval in order to detect the QRS complex. GQRS has the advantage of adapting slightly better than the Pan and Tompkins algorithm, which resulted in a better performance. However, GQRS still fails at detecting some of the QRS complexes because of its inability to adapt properly in noisy signals.

2.1.3 Wavedet

Wavedet [48] is a wavelet based QRS complex detection algorithm. It performs wavelet decomposition on the ECG signals, which produces a time series of frequencies. After the decomposition, a matched filter detects the QRS complexes by looking at the patterns of the wavelet coefficients. The matched filter allows for the analysis of many different signals at varying frequencies and time intervals, thus enabling the separation of the QRS complex signals from the non QRS complex signals. For the final QRS complex detection, it uses an adaptive amplitude filter. Wavedet performs better than GQRS under low noise conditions due to its multi-resolution analysis, but performs poorly under high noise conditions due to its ineffective matched filter and adaptive amplitude filter. The matched filter is unable to filter out the noises as it can not distinguish the false QRS complexes from the actual QRS complexes. Furthermore, the amplitude filter can not tell the difference between the noises and

the actual QRS complexes just by looking at the amplitudes.

2.1.4 Automatic QRS complex detection using two-level convolutional neural network

Xiang et al.'s paper [65] detects QRS complexes using a 2-layer CNN. The first ECG channel is obtained by applying a difference filter to the original input ECG signal. The second ECG channel is produced by applying a moving average filter and a difference filter to the original input ECG signal. After filtering, two 1x5 pixel CNN kernels are applied to the ECG channels. For the second CNN layer, it uses a 1x5 pixel CNN kernel. Finally, the MLP layers make the final QRS complex predictions. Xiang et al.'s CNN is fast and produces great results under low noise conditions. However, Xiang et al.'s CNN is ineffective under high noise conditions due to its difference filter. The difference filter is a highpass filter that allows high frequency noise through, which introduces classification errors and decreases the performance of the algorithm.

2.1.5 Robust Heartbeat Detection From MultimodalData via CNN-Based Generalizable Information Fusion

Chandra et al.'s paper [68] uniquely features an inter-patient testing scheme. In the testing scheme, the patients in the training set differ from the patients in the testing set. This testing scheme proves the generalization ability of their algorithm. Their neural network has a 1-layer CNN and an MLP. The CNN has 2 filters with a kernel size of 29 pixels. The MLP has one 200-neuron hidden layer and employs a sigmoid activation function. The model performs slightly better than Xiang et al.'s CNN due to the former's large CNN kernel size and the former's greater number of neurons. However, it was not designed for high noise conditions, and hence its performance degrades in very noisy data that often happen in wearable ECG devices.

2.2 Data Preparation

As stated in the introduction, data preparation provides the testing and training environment to compare the various QRS complex detection algorithms. The MIT-BIH arrhythmia database [75, 76] and the European ST-T database [77] were selected

for the training and testing of the QRS complex detection algorithms. The MIT-BIH database was sampled at 360 Hz, or equivalently 1 sample per 2.78 ms. In order to maintain a consistent sample rate, the European ST-T database was upsampled from 250 Hz to 360 Hz. The databases have relatively clean ECG recordings. To simulate the noisy wearable ECG devices, noise was added to the ECG recordings using the PhysioToolkit Noise Stress Test [78] software. In this paper, only the first 640,000 samples of each ECG recording were used due to the constraints of the PhysioToolkit Noise Stress Test [78]. The worst case signal to noise ratio (SNR) for most wearable ECG devices ranges from 12 dB SNR to 0 dB SNR. As a result, only the 12 dB SNR and the 0 dB SNR ECG recordings were used.

The following labels were selected for QRS complex detection: N, •, L, R, A, a, J, S, V, F, e, j, E, /, f, and Q. After the selection, the labels were converted into floats. For every individual sample that has a QRS complex label, $y = 1.0$ was assigned to that individual sample, which usually corresponds to the R peak position or very close to the R peak. The floats $y = 0.0$ were assigned to all other samples in the recording. There is only one $y = 1.0$ label for each QRS complex. All detection algorithms were restricted to using only the primary ECG lead for QRS complex detection, while Other ECG leads were not used. The usage of only the primary ECG lead was also done to mimic wearable single channel ECG devices.

Some of the ECG recordings in the databases have inconsistent label positioning. A portion of the QRS complexes were labeled at the R peak, while other QRS complexes were labeled at the start of the Q wave. For this paper, the QRS complexes labeled at the R peak were used. Moreover, a few ECG recordings have QS complexes instead of QRS complexes. The detection of QS complexes is out of the scope of this paper. The following correct ECG recordings from the MIT-BIH database were used for training and testing: 100, 101, 102, 103, 104, 105, 106, 109, 112, 113, 115, 116, 118, 119, 121, 122, 123, 201, 202, 208, 209, 212, 213, 214, 215, 217, 219, 220, 221, 222, 228, 230, 231, 232, and 234. Furthermore, the following correct ECG recordings from the European ST-T database were used for training and testing: e0103, e0104, e0111, e0112, e0113, e0115, e0116, e0118, e0123, e0127, e0136, e0147, e0151, e0154, e0159, e0161, e0166, e0170, e0203, e0204, e0206, e0207, e0208, e0210, e0212, e0303, e0306, e0404, e0406, e0408, e0409, e0410, e0411, e0417, e0418, e0509, e0601, e0606, e0607, e0609, e0610, e0611, e0612, e0613, e0615, e0704, e0818, and e1304. Patients with multiple ECG recordings in the database had only one ECG recording included in this study. The datasets were concatenated into one dataset and randomly shuffled

during the 1x10 fold testing phase. After shuffling, 14 ECG recordings were used as the training dataset and the remaining ECG recordings were grouped as the testing dataset. This way the patients from the training dataset differ from the patients in the testing dataset, realizing interpatient testing to minimize bias towards the training dataset. The following recordings were used for the MIT-BIH NST cross validation set: 107, 117, 124, and 205. These recordings were selected because they already have significant noise artifacts or ECG deformations present.

2.3 Proposed Convolutional Neural Networks With Long Short-Term Memory

In this paper, we propose a CNN-LSTM for the detection of QRS complexes in noisy ECG signals. The algorithm takes in a 2 channel ECG signal. Note that channel 1 is the filtered version of the primary ECG lead, and channel 2 is the gradient of channel 1. To mimic wearable ECG devices, the model does not use any other ECG lead besides the primary ECG lead. The model predicts QRS complexes by producing a delta function at the location of the R peak.

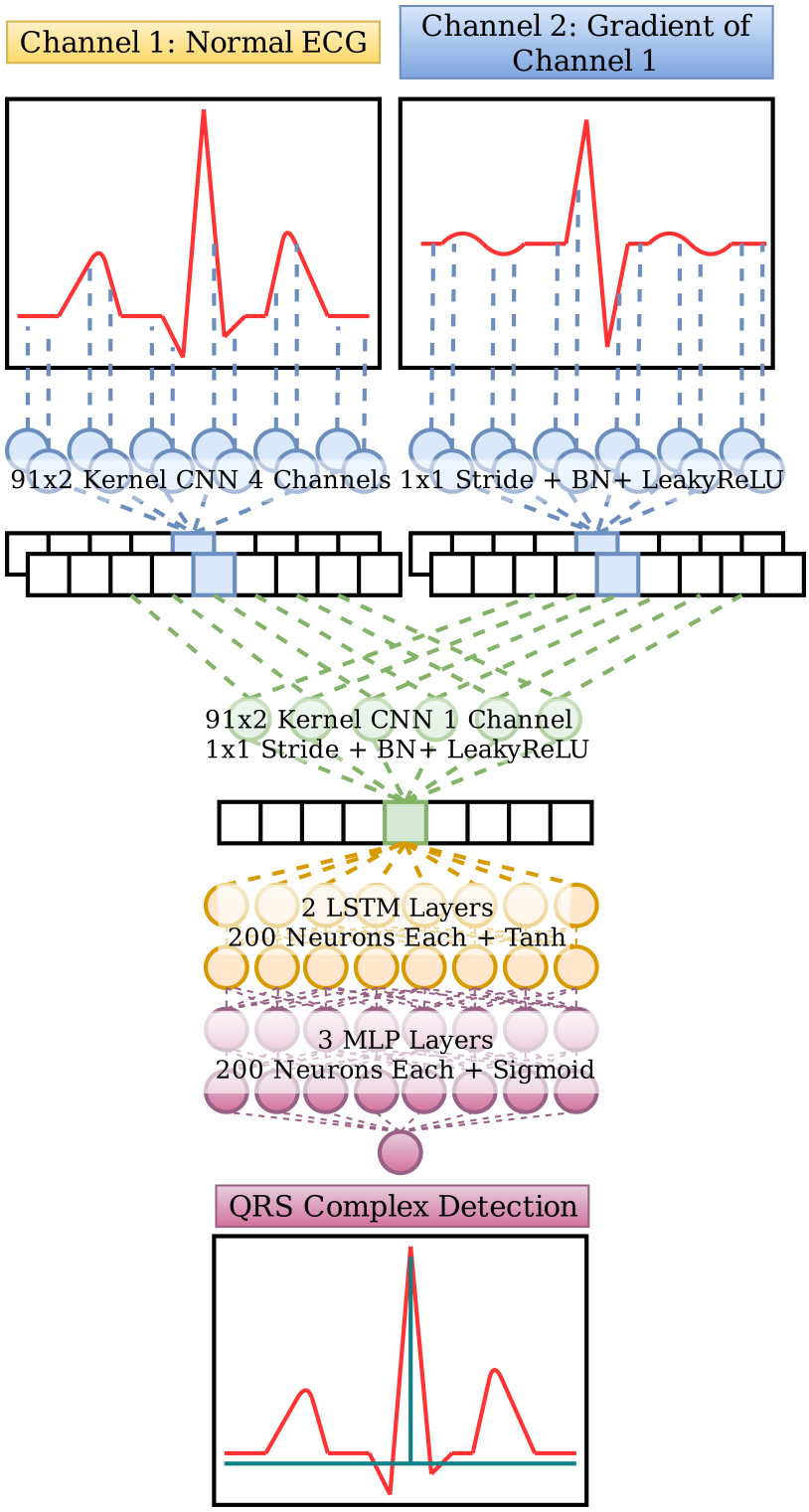

Figure 2.1: The proposed CNN-LSTM architecture.

Table 2.1: CNN-LSTM Hyperparameter Tuning.

F_1 score	CNN kernel size	CNN channels	LSTM neurons per layer	LSTM layers	MLP neurons per layer	MLP layers
0.955521	21x2	4	200	2	200	3
0.963075	41x2	4	200	2	200	3
0.974967	61x2	4	200	2	200	3
0.977523	91x2	4	200	2	200	3
0.959403	91x2	1	200	2	200	3
0.977511	91x2	2	200	2	200	3
0.977523	91x2	4	200	2	200	3
0.960813	91x2	6	200	2	200	3
0.973849	91x2	4	50	2	50	3
0.968482	91x2	4	100	2	100	3
0.977523	91x2	4	200	2	200	3
0.970380	91x2	4	300	2	300	3
0.957475	91x2	4	200	1	200	3
0.977523	91x2	4	200	2	200	3
0.968161	91x2	4	200	3	200	3
0.967221	91x2	4	200	2	200	2
0.977523	91x2	4	200	2	200	3
0.964121	91x2	4	200	2	200	4

Note: MIT-BIH NST 12 dB SNR database. Cross validation set.

In the pre-processing phase, a Butterworth highpass filter $n = 3, f_c = 5$ Hz is applied to the primary ECG lead in order to obtain channel 1. The Butterworth filter reduces the baseline wandering of the ECG signals by attenuating the signals below $f_c = 5$ Hz. After obtaining channel 1, a difference filter is applied to the channel 1 in order to obtain channel 2, as given by

$$y[t] = x[t] - x[t-1] \tag{2.1}$$

where $x[t]$ is the input ECG signal with respect to time t and $y[t]$ is the filtered output signal with respect to time t. The difference filter enhances signals that have large gradients. As the QRS complexes have large gradients, the difference filter enhances the QRS complexes. After the filtering, channel 1 and channel 2 are independently

normalized in order to compensate for the differing patients and ECG devices. First, each ECG recording is divided into ECG segments of 1,280 samples each. Second, each segment is normalized using the mean of the local maximums.

The architecture of the CNN-LSTM is shown in Fig. 2.1. It is made from a 2-layer 2D CNN, a 2-layer LSTM, and a 3-layer MLP. The purpose of the CNN layers is to extract the visual features from the ECG signals. Moreover, the CNN layers are able to filter noise from the ECG signals. The visual features extracted by the CNN layers are sent to the LSTM layers, which predict the future QRS complexes using the previous QRS complexes. Furthermore, the LSTM layers smooth out high frequency noise present in the ECG signals. The timing predictions from the LSTM layers are sent to the MLP layers, which apply thresholding to the timing predictions in order to produce the final QRS complex predictions.

The CNN-LSTM architecture is superior to the CNN counterpart because the former takes into account of the temporal correlations between the ECG samples through the LSTM. QRS complexes are quasi-periodic signals. If the period of the QRS complexes is known and position of the latest QRS complex is known, the position of the next QRS complex could be predicted. The LSTM enables the prediction of the next QRS complex position by using the previous QRS complex position and the visual features from the CNN.

2.3.1 Hyperparameter Tuning

Table 2.1 shows the hyperparameter tuning of the CNN-LSTM. Firstly, the CNN kernel size is varied until the optimal 91x2 kernel size is found. Secondly, the number of CNN channels, i.e., filters, in the first layer is varied. The optimal number of CNN channels is found to be 4 CNN channels. Thirdly, the number of LSTM and MLP neurons per layer is altered and the optimal number is found to be 200. In order to preserve the information between the LSTM layers and the MLP layers, the number of LSTM neurons per layer must equal the number of MLP neurons per layer. Finally, the optimal number of LSTM layers is found to be 2 LSTM layers, and the optimal number of MLP layers is 3.

2.3.2 CNN Description

The first CNN layer has a kernel size of 91x2. As the kernel needs to detect QRS complex gradients, the kernel size is set to the size of a QRS complex gradient.

The CNN layers' horizontal strides control how much the kernels shift at every time interval. In order to preserve the timing of the ECG signal, the horizontal strides of the CNN layers are set to 1 sample. This makes the kernels shift right by 1 sample at every time interval. When the kernels go out of the bounds of the input matrix, the ends of the input matrix are padded with zeros. The first CNN layer uses 4 channels in order to detect the 4 main QRS complex like waveforms: QRS complex, qRs complex [79], QR complex, and RS complex. The first CNN layer uses the LeakyReLU activation function with $\alpha = 0.02$ given by

$$LeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{2.2}$$

where x is the input matrix to the LeakyReLU function. The LeakyReLU function is fast due to its low computational complexity. Moreover, it prevents the loss from reaching zero. The first CNN layer also uses the batch normalization function given by

$$BN(x) = \frac{x - \mu_x}{\sigma_x} \tag{2.3}$$

where x is the input matrix, μ_x is the mean of x, and σ_x is the standard deviation of x. Batch normalization helps the neural network to converge faster. The second CNN layer is similar to the first CNN layer, with the only difference being the number of channels. The second CNN layer takes in 4 CNN channels from the first CNN layer and reduces it to 1 channel, which effectively functions as a 4 to 1 pooling layer.

2.3.3 LSTM Description

The second CNN layer connects to the first LSTM layer, which predicts the QRS complex timings using the 1D sequence of visual features from the CNN layers. The QRS complex timings allow the LSTM layers to narrow the search spaces for QRS complexes. There are 2 LSTM layers. Each has 200 neurons and uses the tanh function as the activation function. The tanh function has a range of $r \in [-1, 1]$, which allows for the negative and positive feedback in the LSTM layers without exponential feedback, which in turn allows the LSTM layers to remember different past information. The LSTM with the tanh activation function can be viewed as a smoothing filter and hence is able to smooth out high frequency noise present in the ECG signals.

2.3.4 MLP Description

The final LSTM layer fully connects to the first MLP layer. The purpose of the MLP layers is to execute the final QRS complex detection. The MLP layers apply thresholding to the QRS complex timing predictions in order to filter out the incorrect QRS complex predictions. There are 3 MLP layers, each having 200 neurons. The MLP layers use the batch normalization function and the sigmoid activation function given by

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

where x is the input matrix. The sigmoid activation function constrains the MLP layers' output to the continuous interval of $Q \in [0, 1]$. In order to produce a binary output, a final threshold $f_{thres} = 0.9$ is applied to MLP layers' output Q. If $Q > f_{thres}$, then the CNN-LSTM predicts $\hat{y} = 1.0$ to signal the presence of QRS complex, otherwise the CNN-LSTM predicts $\hat{y} = 0.0$ to signal the absence of a QRS complex.

2.3.5 Loss Function

Neural networks are trained by minimizing a defined loss function. As a result, the choice of the loss function is critical to the performance of the neural network. This work uses the weighted cross-entropy loss function expressed as

$$J(\hat{y}, y) = -\log(S(\hat{y}))(y)(W_{pos}) - \log(1 - S(\hat{y}))(1 - y) \tag{2.5}$$

where y is the QRS complex label and W_{pos} is the cross-entropy weight. The weighted cross-entropy loss function is chosen because the function allows the designer to change the ratio of false positives (FP) to false negatives (FN) by varying the cross-entropy weight W_{pos}. Each ECG recording has approximately 340 samples in between each pair of QRS complexes. Therefore, the number of true negatives (TN) is far larger than the number of true positives (TP). The imbalance is corrected by setting the cross-entropy weight to $W_{pos} = 340$. Furthermore, the predicted QRS complex detection $\hat{y}$ is matched against the actual QRS complex detection label y. If they both have the same value $\hat{y} \approx y$, then the loss function is small. If they have different values $\hat{y} \neq y$, then the loss function is large. This fulfills the design objective.

2.4 Simulations

In this paper, all algorithms described in Section 2.1 are implemented as the comparison basis for the proposed CNN-LSTM. The neural networks are implemented in Python 3 using TensorFlow 1.5 [80], while the other algorithms are implemented in MATLAB using the PhysioNet ECG-Kit [76]. The QRS complex detection algorithms are benchmarked using the noisy dataset described in Section 4.1.

2.4.1 Evaluation Metrics

The true positives (TP), false positives (FP), false negatives (FN), sensitivity (SEN), positive predictive value (PPV), F1 score (F_1), and root mean-squared error (RMSE) of the timings of the QRS complex detection algorithms are recorded. Here, SEN, PPV and F_1 are computed according to the equations below

$$SEN = \frac{TP}{TP + FN} \tag{2.6}$$

$$PPV = \frac{TP}{TP + FP} \tag{2.7}$$

$$F_1 = 2\frac{SEN \cdot PPV}{SEN + PPV}. \tag{2.8}$$

Sensitivity measures the number of false negatives in relation to the actual QRS complexes. Positive predictive value measures the number of false positives among the detected QRS complexes. If a QRS complex detection algorithm performs well, then it must have a high sensitivity $SENS \approx 1$ and a high positive predictive value $PPV \approx 1$. This in turn causes the $F_1 \approx 1$ to be high.

If a QRS complex detection algorithm predicts the R peak of a QRS complex within 50 ms of the R peak of a true QRS complex, then the predicted QRS complex counts as a true positive. If a QRS complex detection algorithm predicts a QRS complex and the R peak of a true QRS complex does not exist within 50 ms of the R peak of the predicted QRS complex, then it is counted as a false positive. If a QRS complex detection algorithm does not predict the R peak of a QRS complex within 50 ms of the R peak of a true QRS complex, then it is counted as a false negative. The true negatives are not relevant as none of the ECG metrics use them.

Another important performance measure is related to the timing accuracy of the R wave, in addition to QRS detection benchmarks. R peak timing error directly impacts

the accuracy of RR intervals, PR intervals, and heart rate variability calculations. Here, the RMSE timing metric, given by

$$RMSE = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(T_i - \hat{T}_i)^2} \tag{2.9}$$

is used for the evaluation of the QRS complex detection algorithms, where M is number of QRS complexes, T_i is the QRS complex label time, and $\hat{T}_i$ is the QRS complex prediction time.

2.4.2 CNN-LSTM Learning Curve

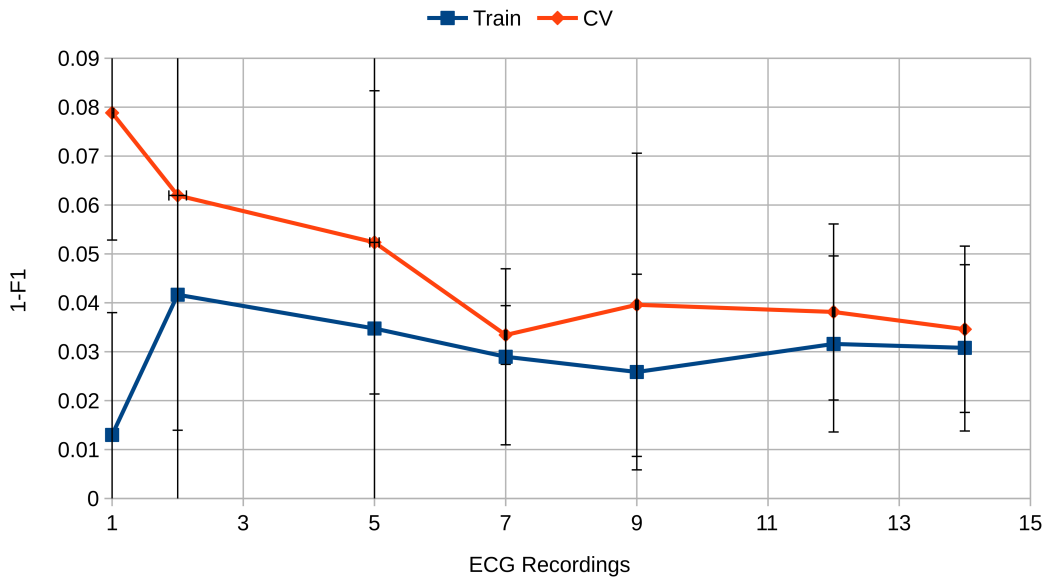

Figure 2.2: CNN-LSTM's learning curve. MIT-BIH NST 12 dB SNR. 2σ error bar.

Fig. 2.2 shows the learning curve of the CNN-LSTM. The learning curve was generated using the MIT-BIH NST 12 dB SNR database. Some of the ECG segments in the MIT-BIH NST database have low noise, while others have high noise. This discrepancy causes fluctuations in the F_1 score. Furthermore, the CNN-LSTM may perform better in certain ECG recordings, which also leads to more fluctuations in the F_1 score. The fluctuations in the F_1 score account for the large error bars in the learning curve. These errors also plague the other algorithms presented in Table 2.2, which results in large F_1 score standard deviations. The learning curve narrowing around the 12 to 14 ECG recording mark. This proves the convergence of the model.

Moreover, the learning curve also proves the CNN-LSTM is neither underfitting nor overfitting.

2.4.3 Results

Table 2.2: MIT-BIH NST Algorithm Performance, with 12 dB SNR.

Algorithm	GQRS [47]	Pantom [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	46702 ± 1904	40140 ± 4564	43809 ± 3556	45411 ± 6626	46993 ± 2877	46591 ± 3120
FP	8652 ± 1404	2265 ± 2582	12590 ± 2226	2739 ± 2326	4738 ± 3186	2218 ± 1742
FN	1650 ± 626	8164 ± 3184	3945 ± 1830	2766 ± 5020	625 ± 751	1155 ± 1693
SENS	0.9658 ± 0.013	0.8305 ± 0.069	0.9172 ± 0.039	0.9419 ± 0.107	0.9868 ± 0.016	0.9757 ± 0.035
PPV	0.8436 ± 0.025	0.9462 ± 0.062	0.7765 ± 0.044	0.9444 ± 0.042	0.9089 ± 0.058	0.9550 ± 0.033
F_1 score	0.9005 ± 0.019	0.8844 ± 0.062	0.8409 ± 0.041	0.9418 ± 0.043	0.9460 ± 0.032	0.9650 ± 0.017
Timing RMSE	12.40 ± 0.08	6.98 ± 0.80	4.01 ± 0.66	1.98 ± 0.80	1.58 ± 0.44	1.76 ± 0.50

Note: Confidence interval of 2σ. Timing RMSE units in samples.

Table 2.3: MIT-BIH NST Algorithm Performance, with 0 dB SNR.

Algorithm	GQRS [47]	Pantom [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	39941 ± 2008	9758 ± 1573	39106 ± 1911	37006 ± 24831	42216 ± 1268	43384 ± 3019
FP	23746 ± 1206	6533 ± 217	20716 ± 699	14552 ± 10141	16952 ± 2431	14355 ± 2280
FN	8307 ± 684	37906 ± 2928	8576 ± 593	10947 ± 25316	5641 ± 1158	4189 ± 1402
SENS	0.8277 ± 0.015	0.2048 ± 0.035	0.8201 ± 0.008	0.7732 ± 0.518	0.8822 ± 0.020	0.9117 ± 0.031
PPV	0.6270 ± 0.023	0.5981 ± 0.032	0.6536 ± 0.018	0.7212 ± 0.045	0.7137 ± 0.031	0.7513 ± 0.038
F_1 score	0.7135 ± 0.019	0.3049 ± 0.042	0.7274 ± 0.012	0.7036 ± 0.469	0.7890 ± 0.023	0.8237 ± 0.033
Timing RMSE	12.16 ± 0.09	5.57 ± 0.20	4.69 ± 0.42	2.64 ± 0.81	2.57 ± 0.51	2.57 ± 0.56

Note: Confidence interval of 2σ. Timing RMSE units in samples.

Table 2.4: European ST-T NST Algorithm Performance, with 12 dB SNR.

Algorithm	GQRS [47]	Pantom [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	65835 ± 2726	55556 ± 4387	66374 ± 2135	58909 ± 8037	66101 ± 2813	65802 ± 2847
FP	9511 ± 897	2241 ± 713	18451 ± 1752	2283 ± 1666	3293 ± 1338	1789 ± 1087
FN	1206 ± 599	11517 ± 2522	1616 ± 352	6752 ± 8403	297 ± 145	849 ± 822
SENS	0.9819 ± 0.008	0.8281 ± 0.037	0.9761 ± 0.005	0.8974 ± 0.126	0.9955 ± 0.002	0.9873 ± 0.012
PPV	0.8737 ± 0.011	0.9612 ± 0.011	0.7824 ± 0.020	0.9633 ± 0.022	0.9525 ± 0.019	0.9735 ± 0.015
F_1 score	0.9247 ± 0.009	0.8896 ± 0.024	0.8686 ± 0.014	0.9277 ± 0.063	0.9735 ± 0.010	0.9803 ± 0.006
Timing RMSE	12.66 ± 0.30	10.11 ± 0.55	11.81 ± 0.34	0.75 ± 0.37	0.83 ± 0.22	1.07 ± 0.27

Note: Confidence interval of 2σ. Timing RMSE units in samples.

Tables 2.2-2.5 show the results of the 1x10 fold testing on the MIT-BIH NST and the European ST-T NST databases. For both databases, the proposed CNN-LSTM outperforms GQRS [47], Pan and Tompkins [43], Wavedet [48], Xiang et al.'s CNN [65], and Chandra et al.'s CNN [68] in terms of F_1 score. For example, for the 12 dB SNR MIT-BIH NST database, the proposed CNN-LSTM's F_1 score of 0.9650 is greater than GQRS's F_1 score of 0.9005, Pan and Tompkins's F_1 score of 0.8844, Wavedet's F_1 score of 0.8409, Xiang et al.'s CNN's F_1 score of 0.9418, and

Table 2.5: European ST-T NST Algorithm Performance, with 0 dB SNR.

Algorithm	GQRS [47]	Pantom [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	58611 ± 1680	15671 ± 2187	57813 ± 1419	57121 ± 6175	62234 ± 2475	60620 ± 3002
FP	32359 ± 980	8101 ± 762	31697 ± 711	17375 ± 7378	19569 ± 3184	13415 ± 5102
FN	8354 ± 854	53038 ± 2386	9913 ± 624	8880 ± 4375	3882 ± 829	4909 ± 2706
SENS	0.8752 ± 0.012	0.2280 ± 0.030	0.8536 ± 0.007	0.8650 ± 0.069	0.9412 ± 0.012	0.9251 ± 0.040
PPV	0.6442 ± 0.012	0.6586 ± 0.035	0.6458 ± 0.010	0.7683 ± 0.076	0.7609 ± 0.032	0.8197 ± 0.052
F_1 score	0.7421 ± 0.011	0.3386 ± 0.038	0.7353 ± 0.007	0.8130 ± 0.054	0.8414 ± 0.019	0.8688 ± 0.033
Timing RMSE	12.38 ± 0.16	12.03 ± 0.39	12.35 ± 0.38	1.75 ± 0.45	1.73 ± 0.23	1.65 ± 0.21

Note: Confidence interval of 2σ. Timing RMSE units in samples.

Chandra et al.'s CNN's F_1 score of 0.9460. Also shown in these tables, the most recent machine learning based algorithms, [65], [68] and the proposed CNN-LSTM, have clear advantages over the previous filter and wavelet based algorithms, which demonstrates the effectiveness of neural networks. The proposed model performs consistently better than the other NN based QRS complex detection algorithms for noisy data because our CNN-LSTM model has larger CNN kernels than the latter. The larger CNN kernels help the CNN-LSTM to filter out the noise better, thus reducing the number of false positives. Furthermore, the LSTM layers improve the F_1 score of the CNN-LSTM model by predicting the future QRS complexes correctly. Finally, the proposed model has a greater number of neurons than the other NN. The greater number of neurons allows the CNN-LSTM to detect more complex patterns, which improves the F_1 score.

2.4.4 Wide QRS Complexes

Fig. 2.4 and Fig. 2.5 show patients with wide QRS complexes. The ECG signals have QRS complex widths of 80 samples (222 ms) and 90 samples (250 ms) respectively. The smaller CNN kernel sizes have trouble detecting large QRS complexes because they can not capture the entire QRS complex. Thus, the large 91x2 CNN kernels were used to detect the large QRS complexes. This results in an increase of F_1 as shown in Table 2.1.

2.4.5 CNN-LSTM Limitations

The proposed CNN-LSTM has a few limitations. The timing RMSE of the model is similar to that of Xiang et al.'s CNN [65] and Chandra et al.'s CNN [68] at low SNRs, but slightly worse at high SNRs. The timing errors of the proposed model are due to the large 91x2 CNN kernels. All CNN kernels have a trade off between spatial

frequency uncertainty and position uncertainty. The 91x2 CNN kernels has low spatial frequency uncertainty at the cost of high position uncertainty. Another limitation of the proposed model is the computational complexity. At every time interval n, the CNN performs one convolution with kernel width W and kernel height H at a cost of $O(W, H) = WH$ computations. If the number of channels C is considered, then the cost is $O(W, H, C) = WHC$ computations. The cost for the entire time interval n is $O(n) = WHCn$ computations. With the addition of many CNN layers L_{CNN}, the cost becomes $O(n) = L_{CNN}WHCn$ computations.

Now, consider the computational complexity of the LSTM. For a single gate $G = 1$ at a single time interval $n = 1$, the gate has a cost of $O(m, p) = mp$ computations, where m and p are the height and width of the gate's weight matrix respectively. For multiple gates G and time intervals n, the cost is $O(n) = Gmpn$ computations. With the addition of many LSTM layers L_{LSTM}, the cost becomes $O(n) = L_{LSTM}Gmpn$ computations. The MLP layers have the same weight dimensions as the LSTM layers. Thus, the computational cost of the MLP layers is $O(n) = L_{MLP}mpn$ computations, where L_{MLP} is the number of MLP layers. Finally, the total computational complexity of the CNN-LSTM is

$$O(n) = L_{CNN}WHCn + L_{LSTM}Gmpn + L_{MLP}mpn. \tag{2.10}$$

The computational complexity of the CNN-LSTM is higher than the computational complexities of other QRS complex detection algorithms. As a result, the proposed model detects QRS complexes at a slower rate than the rest of the QRS complex detection algorithms. The CNN-LSTM also requires more ECG recordings for the training phase. The proposed model requires at least 11 ECG recordings for the training phase as shown in Fig. 2.2. The rest of the QRS complex detection algorithms only require 100,000 ECG samples for the training phase. These limitations can be largely overcome by today's powerful computing machines such as GPU during training.

2.5 Error Analysis

A detailed error analysis of the CNN-LSTM indicates the following 5 main errors: QRS complex like artifact created by noise, P wave and T wave misclassified as QRS complex, QRS complex amplitude too small, atrial flutter/atrial fibrillation,

and actual QRS complex distorted by noise. Fig. 2.3 shows the CNN-LSTM's error distribution.

2.5.1 QRS complex like artifact created by noise

This error type occurs when QRS complex like artifacts are introduced by the noises, generated using the PhysioToolkit Noise Stress Test [78], and is the main source of error accounting for 35.12% of total number of errors. The criteria

$$(FP) \wedge (RMSE(ECGclean[t], ECGnoisy[t]) > 0) \tag{2.11}$$

is used to classify the error, where the error is a false positive $FP = True$ and large amounts of noises are introduced $RMSE(ECGclean[t], ECGnoisy[t]) > 0$. $ECGclean[t]$ and $ECGnoisy[t]$ represent the ECG signals before and after the additions of the noises respectively. The generated artifacts are almost indistinguishable from the actual QRS complexes. The artifacts could be minimized by employing more filters or advanced neural networks. For example, the filters could minimize the number of false positives by rejecting false QRS complexes before they reach the CNN-LSTM.

2.5.2 P wave and T wave misclassified as QRS complex

P waves and T waves in the ECG signals sometimes look similar to the QRS complexes, especially when they become larger than QRS complex in amplitude. This error type happens when a P wave or a T wave is misclassified as a QRS complex. The criteria

$$(FP) \wedge ((label == P) \vee (label == T)) \tag{2.12}$$

is used to classify the error, where the error is a false positive $FP = True$ and a P wave or a T wave is within 50 ms of the error. The P waves and T waves could be removed using a P wave and T wave detector. However, the detector may introduce more errors.

2.5.3 QRS complex amplitude too small

The CNN-LSTM uses thresholding to detect QRS complexes. If a QRS complex amplitude is above the threshold, then it gets detected; Otherwise it does not get

detected. This error type happens when a QRS complex amplitude is too small, which results in a false negative error. The criteria

$$(FN) \wedge (E[A_{QRS}] > A_{QRS}) \tag{2.13}$$

is used to classify the error, where the error is a false negative $FN = True$ and the expected value of the QRS complex amplitudes $E[A_{QRS}]$ is greater than the current QRS complex amplitude A_{QRS}. This could be reduced by using better normalization algorithms. However, the normalization algorithms introduce a chicken and egg problem. The QRS complex detection algorithm requires a normalization algorithm in order to increase the QRS detection accuracy. On the other hand, the normalization algorithm needs the actual QRS complex amplitude because the noise peaks could be higher than the actual QRS complexes.

2.5.4 Atrial flutter/Atrial fibrillation

When atrial flutters or atrial fibrillations occur, the ECG signals look like triangular waves or saw-tooth waves. This significantly distorts the QRS complexes and introduces detection errors. The criteria

$$(FN) \wedge ((label == AFIB) \vee (label == AFL)) \tag{2.14}$$

is used to classify the error, where the error is a false negative $FN = True$ and the ECG segment is labeled as an atrial flutter or an atrial fibrillation. The misclassification errors may be resolved by increasing the cross entropy weights of the segments that have atrial flutters or atrial fibrillations. Moreover, the false negative errors could be reduced by using a specialized CNN-LSTM just for the detection of the atrial flutters and the atrial fibrillations.

2.5.5 Actual QRS complex distorted by noise

This error type occurs when the actual QRS complex is distorted by the noises generated by the PhysioToolkit Noise Stress Test [78]. The distorted QRS complex does not resemble any normal QRS complex, thus resulting in a classification error. The criteria

$$(FN) \wedge (RMSE(ECGclean[t], ECGnoisy[t]) > 0) \tag{2.15}$$

is used to classify the error, where the error is a false negative $FN = True$ and the actual QRS complex is distorted by the noises $RMSE(ECGclean[t], ECGnoisy[t]) > 0$. The error could be minimized by adding more filters to the model. More filters could mean better detection of distorted QRS complexes.

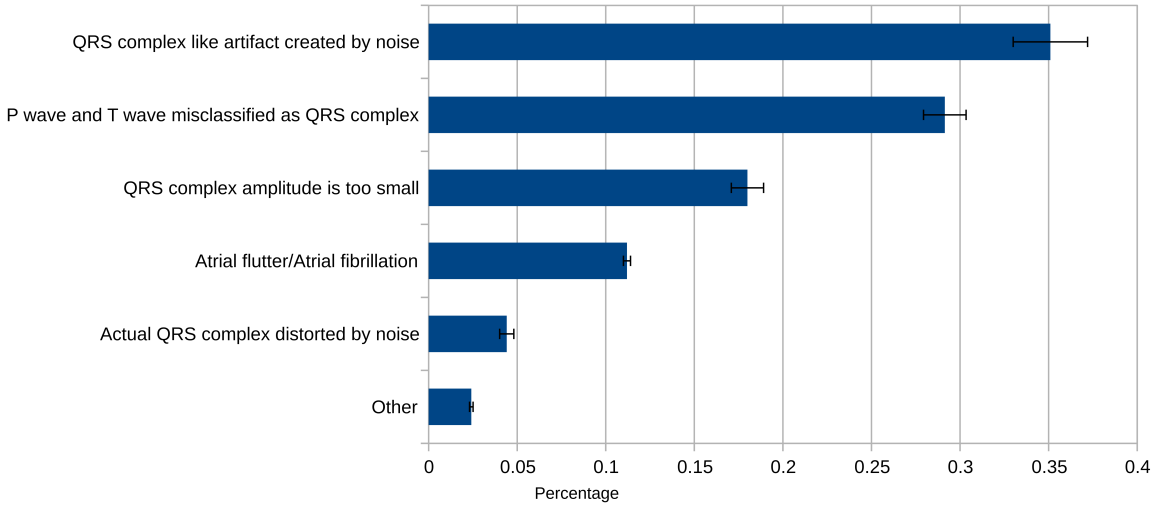

Figure 2.3: CNN-LSTM's error distribution. MIT-BIH NST 12 dB SNR. 2σ error bar.

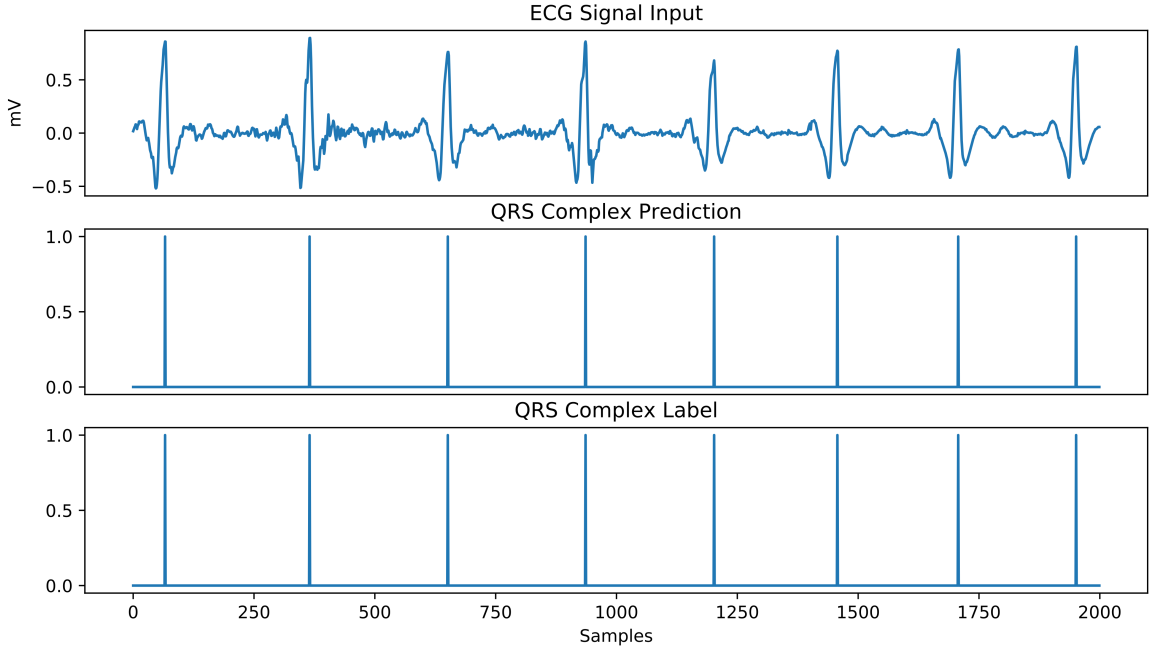

Figure 2.4: MIT-BIH NST CNN-LSTM QRS complex detection. QRS complex width 80 samples (222 ms).

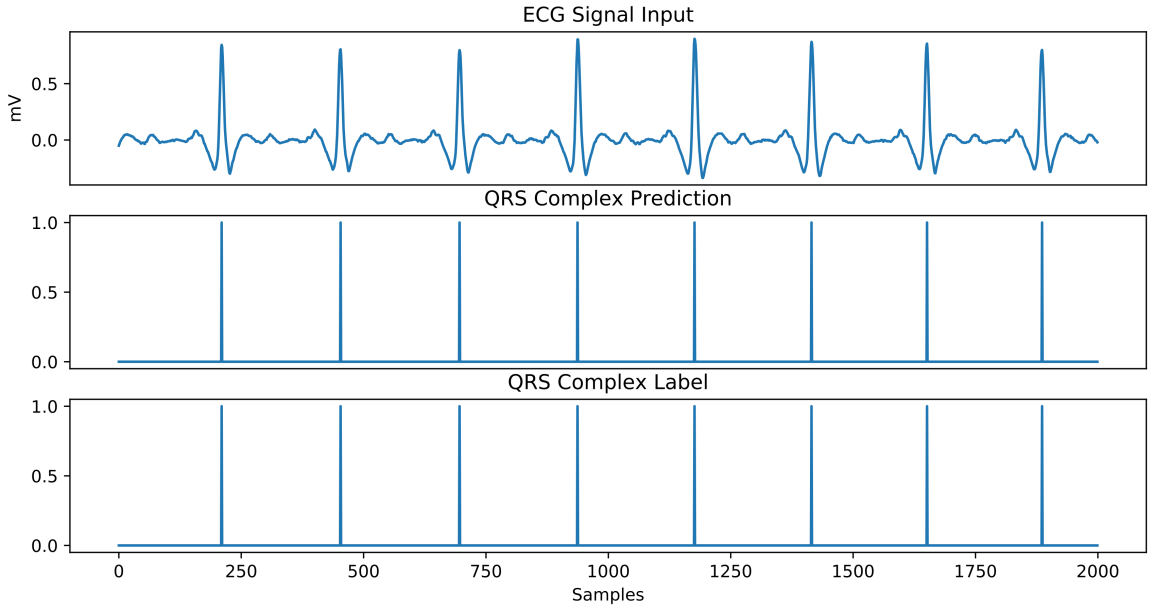

Figure 2.5: MIT-BIH NST CNN-LSTM QRS complex detection. QRS complex width 90 samples (250 ms).

2.6 Conclusion

This paper has presented a novel CNN-LSTM structure for the detection of QRS complexes in noisy ECG signals. Moreover, an inter-patient training/testing procedure has been devised to prove the generalization ability of the CNN-LSTM. The generalization ability of the CNN-LSTM is particularly useful for automatic analysis of ECG data collected by mobile wearable devices, where manual labeling of individual patients' records is unrealistic. Inside the stacked network, the CNN layers extract visual features and filter out noise from the noisy ECG signals. The LSTM layers predict the QRS complex timings. The subsequent MLP layers execute the final QRS complex detections and format the outputs of the network. Simulations using MIT-BIH NST and European ST-T NST databases have demonstrated that the proposed CNN-LSTM outperforms the existing algorithms in the literature in terms of F_1 score. As a result, the proposed CNN-LSTM is a promising solution for use in noisy wearable ECG devices.

Chapter 3

Detecting Noisy ECG QRS Complexes using WaveletCNN Autoencoder and ConvLSTM

Many cardiovascular diseases are diagnosed using electrocardiogram (ECG) recordings. For example, cardiovascular diseases such as coronary artery disease, arrhythmia, and heart valve disease are detected using ECG recordings. However, accurate diagnoses of cardiovascular diseases require large amounts of ECG recordings. Wearable ECG devices were invented in order to gather numerous amounts of ECG recordings for the cardiologists. The downside to the vast stores of ECG recordings is the disease classification time. As the number of ECG recordings increases, the amount of time the cardiologists spend on disease classification increases.

Automated ECG disease classification was created to expedite the cardiologists' diagnoses. More recently, deep neural networks were applied to ECG disease classification. Zihlmann et al. [72] proposed a convolutional neural network (CNN) followed by a long short-term memory (LSTM) network for ECG disease classification. Andersen et al. [81] developed a CNN-LSTM network that can detect atrial fibrillation (AF) in real time as it can process 24 h recordings in less than 1 second. Furthermore, a CNN-LSTM is created by Verma et al. [82] for classifying between normal, AF, noisy signals, and other signals. Pourbabaee et al. [83] tested many CNNs with support vector machines (SVMs) and multilayer perceptron (MLPs) for paroxysmal atrial fibrillation detection. The CNN-LSTM generally performs better than the MLP and the SVM because the former is able to detect spatial-temporal patterns in the

ECG signal.

In order to diagnose cardiovascular diseases using the ECG recordings, the ECG recordings must first be segmented. QRS complex detection is required for ECG segmentation because the QRS complexes mark the starts and the ends of the cardiac rhythm measurements and ECG segments. As a result, QRS complex detection is paramount for beat classification.

Many QRS complex detection algorithms have been created over the past few decades. The literature [41, 42] and references therein show a vast array of QRS complex detection algorithms. Most of the QRS complex detection algorithms use digital filters to detect the QRS complexes. The main digital filters [41, 42] for QRS complex detection are the amplitude filter, the time series filter, the matched filter, and the frequency filter. Amplitude filters use thresholds to find the QRS complexes. If an ECG peak is above a threshold, then the amplitude filter classifies the peak as a QRS complex. Certain portions of Pan and Tompkins [43], GQRS [47], and ecgpuwave [84] employ the amplitude filter. The algorithms presented above perform within 0.1% of each other under low noise conditions. As a result, the algorithms have reached the Bayes error rate for QRS complex detection.

The matched filter convolves a predefined template with the ECG signal in order to produce a QRS complex detection signal. These papers [56, 57] show classical applications of the match filters for QRS complex detection. On the other hand, more advanced neural network based match filters have been developed. These references [60, 61, 62, 63, 64, 65, 66, 67, 68] contain neural network based match filters. The frequency filters allow for decomposition of signals based on frequency and time. As a result, certain frequencies could be attenuated in order to reduce the noises. Moreover, the frequency filters allow for analysis of QRS complexes as QRS complexes have unique frequency signatures. This improves the QRS complex detection accuracy. Wavedet [48] and WQRS [85] are examples of ECG frequency filters. Other examples of frequency filters include [49, 50, 51, 52, 53, 54, 55, 86, 87, 88].

The QRS complex detection algorithms presented above are well suited for detecting QRS complexes in clean ECG signals. However, the algorithms above perform poorly in very noisy wearable ECG devices. Noises [89] such as baseline wandering, power line noise, electrode contact noise, and instrumentation noise create many problems. Baseline wandering introduces many low frequency noises to the ECG signal, which distorts the amplitudes of the ECG peaks. The other noises introduce high frequency noises, which creates false QRS complexes. Moreover, the high frequency

noises make the actual QRS complexes undetectable by disfiguring them.

This paper proposes a novel machine learning pipeline to solve the problems above. The machine learning pipeline consists of a Butterworth filter, two wavelet convolutional neural network (WaveletCNN) autoencoders, an optional QRS complex inverter, a Monte Carlo k-nearest neighbours (k-NN), and a convolutional long short-term memory (ConvLSTM). The Butterworth filter removes the baseline wandering of the ECG signals by attenuating the low frequency noise components. The WaveletCNN Autoencoders are advanced bandpass filters, which essentially eliminate the false QRS complexes and enhance the actual QRS complexes. A QRS complex inverter is used to flip the inverted QRS complexes for better detection. The Monte Carlo k-NN applies automatic gain control to the ECG signals in order to normalize the peaks of the ECG signals to 1 mV. The ConvLSTM executes the final QRS complex detection by using the advantages of a CNN and a LSTM. As a result of the machine learning pipeline, the detection of QRS complexes in noisy wearable ECG devices is feasible.

The rest of the paper is organized as follows. Section 3.1 discusses several related QRS complex detection algorithms in detail. Section 3.2 shows the data preparation and the test environment. The proposed machine learning pipeline is presented in Section 3.3. Section 3.4 compares the performance metrics of the proposed machine learning pipeline to the other QRS complex detection algorithms. Finally, conclusions are given in Section 4.4.

3.1 Related QRS Complex Detection Algorithms

In this section, the following related QRS complex detection algorithms are presented: Pan and Tompkins [43], GQRS [47], Wavedet [48], Xiang et al.'s CNN [65], and Chandra et al.'s CNN [68]. The advantages and disadvantages of each algorithm are also described.

3.1.1 Pan and Tompkins

The Pan and Tompkins algorithm [43] is the first real time QRS complex detection algorithm, in which a bandpass filter is applied to reduce the noises in the ECG signals, and adaptive filters are used to detect the QRS complexes. The adaptive filters consist of an amplitude filter, a slope filter, and a width filter. In order to be marked as a

QRS complex, an ECG peak must simultaneously meet all of the following criteria: the peak's amplitude must be greater than an amplitude threshold, the peak's slope must be greater than a slope threshold, and the peak's width must fall within the range of a QRS complex width. The amplitude filter rejects the low amplitude signals, while the slope filter and the width filter eliminate the P waves and T waves. The advantages of the Pan and Tompkins algorithm are the fast processing times and low complexity. However, the filters used in the algorithm need to be engineered by hand, which requires a lot of time and expertise. Furthermore, the handcrafted filters can not adapt to different patients and environments.

3.1.2 GQRS

GQRS [47] is a classical QRS complex detection algorithm. Firstly, it calculates the means and the standard deviations of the RR intervals and the QRS complex amplitudes of the previously detected QRS. Secondly, the algorithm forms an adaptive search interval using the statistics of the RR intervals. Thirdly, the model creates an adaptive amplitude filter using the statistics of the QRS complex amplitudes. Finally, the adaptive amplitude filter is applied to the current adaptive search interval in order to detect the QRS complex. GQRS has the advantage of adapting slightly better than the Pan and Tompkins algorithm, which resulted in a better performance. However, GQRS still fails at detecting some of the QRS complexes because of its inability to adapt properly in noisy signals.

3.1.3 Wavedet

Wavedet [48] is a wavelet based QRS complex detection algorithm. It performs wavelet decomposition on the ECG signals, which produces a time series of frequencies. After the decomposition, a matched filter detects the QRS complexes by looking at the patterns of the wavelet coefficients. The matched filter allows for the analysis of many different signals at varying frequencies and time intervals, thus enabling the separation of the QRS complex signals from the non QRS complex signals. For the final QRS complex detection, it uses an adaptive amplitude filter. Wavedet performs better than GQRS under low noise conditions due to its multi-resolution analysis but performs poorly under high noise conditions due to its ineffective matched filter and adaptive amplitude filter. The matched filter is unable to filter out the noises as it can not distinguish the false QRS complexes from the actual QRS complexes.

Furthermore, the amplitude filter can not tell the difference between the noises and the actual QRS complexes just by looking at the amplitudes.

3.1.4 Automatic QRS complex detection using two-level convolutional neural network

Xiang et al.'s paper [65] detects QRS complexes using a 2-layer CNN. The first ECG channel is obtained by applying a difference filter to the original input ECG signal. The second ECG channel is produced by applying a moving average filter and a difference filter to the original input ECG signal. After filtering, two 1x5 pixel CNN kernels are applied to the ECG channels. For the second CNN layer, it uses a 1x5 pixel CNN kernel. Finally, the MLP layers make the final QRS complex predictions. Xiang et al.'s CNN is fast and produces great results under low noise conditions. However, Xiang et al.'s CNN is ineffective under high noise conditions due to its difference filter. The difference filter is a highpass filter that allows high frequency noise through, which introduces classification errors and decreases the performance of the algorithm.

3.1.5 Robust Heartbeat Detection From MULTIMODAL DATA via CNN-Based Generalizable Information Fusion

Chandra et al.'s paper [68] uniquely features an inter-patient testing scheme. In the testing scheme, the patients in the training set differ from the patients in the testing set. This testing scheme proves the generalization ability of their algorithm. Their neural network has a 1-layer CNN and an MLP. The CNN has 2 filters with a kernel size of 29 pixels. The MLP has one 200-neuron hidden layer and employs a sigmoid activation function. The model performs slightly better than Xiang et al.'s CNN due to the former's large CNN kernel size and the former's greater number of neurons. However, it was not designed for high noise conditions, and hence its performance degrades in very noisy data that often happen in wearable ECG devices.

3.2 Data Preparation

As stated in the introduction, data preparation provides the testing and training environment to compare the various QRS complex detection algorithms. The MIT-

BIH arrhythmia database [75, 76] (https://physionet.org/content/mitdb/1.0.0/), the European ST-T database [77] (https://physionet.org/content/edb/1.0.0/), and the Long Term ST database [90] (https://physionet.org/content/ltstdb/1.0.0/) are selected for the training and testing of the QRS complex detection algorithms. Noise is added to the ECG recordings using the PhysioToolkit Noise Stress Test [78] (https://physionet.org/content/nstdb/1.0.0/).

3.2.1 Pre-processing Procedure

The following labels are selected for QRS complex detection: N, •, L, R, A, a, J, S, V, F, e, j, E, /, f, and Q. Some of the ECG recordings in the databases have inconsistent label positioning. A portion of the QRS complexes are labeled at the R peak, while other QRS complexes are labeled at the start of the Q wave. For this paper, the QRS complexes labeled at the R peak are used. For every individual sample that has a QRS complex label, $y = 1.0$ is assigned to that individual sample, which usually corresponds to the R peak position or very close to the R peak. The floats $y = 0.0$ are assigned to all other samples in the recording. There is only one $y = 1.0$ label for each QRS complex.

All detection algorithms are restricted to using only the primary ECG lead for QRS complex detection, while other ECG leads are not used. The usage of only the primary ECG lead is done to mimic wearable single channel ECG devices. The datasets are trained and tested on a patient level. Patients with multiple ECG recordings in the database had only one ECG recording included in this study. The total dataset contains N number of patients. For training, the total dataset is randomly shuffled and X number of patients are randomly selected for the training dataset. The remaining $N - X$ number of patients are used for the testing dataset. This way the patients from the training dataset differ from the patients in the testing dataset, minimizing bias towards the training dataset. The process above repeats for 10 times for 1x10 fold testing.

3.2.2 PhysioToolkit Noise Stress Test

The datasets have relatively clean ECG recordings. To simulate the noisy wearable ECG devices, noise is added to the ECG recordings using the PhysioToolkit Noise Stress Test [78] (NST). The NST produces baseline wandering noises, white Gaussian noises, muscle noise, and electrode contact noises based on the given signal to noise

ratio (SNR). Baseline wandering noises consist of low frequency high amplitude noises, while electrode contact noises are high frequency signals that look similar to QRS complexes. In this paper, only the first 640,000 samples of each ECG recording are used due to the constraints of the NST. The worst case SNR for most wearable ECG devices ranges from 12 dB SNR to -6 dB SNR. As a result, only the 12 dB SNR, the 0 dB SNR, and the -6 dB SNR ECG recordings are used.

3.2.3 MIT-BIH NST

For the MIT-BIH NST, the following recordings are used for training and testing: 100, 104, 108, 113, 117, 122, 201, 207, 212, 217, 222, 231, 101, 105, 109, 114, 118, 123, 208, 213, 219, 223, 232, 102, 106, 111, 115, 119, 124, 203, 209, 214, 220, 228, 233, 103, 107, 112, 116, 121, 200, 205, 210, 215, 221, 230, and 234. The QRS complex inverter is applied to the MIT-BIH as it has many inverted QRS complexes. For the training and the testing process, 17 random patient recordings are used as the training dataset and the remaining 30 patient recordings are grouped as the testing dataset.

3.2.4 European ST-T NST

The MIT-BIH database is sampled at 360 Hz, or equivalently 1 sample per 2.78 ms. In order to maintain a consistent sample rate, the European ST-T database is upsampled from 250 Hz to 360 Hz. Furthermore, the following ECG recordings from the European ST-T database are used for training and testing: e0103, e0104, e0111, e0112, e0113, e0115, e0116, e0118, e0123, e0127, e0136, e0147, e0151, e0154, e0159, e0161, e0166, e0170, e0203, e0204, e0206, e0207, e0208, e0210, e0212, e0303, e0306, e0404, e0406, e0408, e0409, e0410, e0411, e0417, e0418, e0509, e0601, e0606, e0607, e0609, e0610, e0611, e0612, e0613, e0615, e0704, e0818, and e1304. The QRS complex inverter is not used on the European ST-T database as most of European ST-T's QRS complexes are not inverted. For the training and the testing process, 14 random patient recordings are used as the training dataset and the remaining 34 patient recordings are grouped as the testing dataset.

3.2.5 Long Term ST NST

Similar to the European ST-T database, the Long Term ST database is upsampled from 250 Hz to 360 Hz. Furthermore, the following ECG recordings from the Long Term ST database are used for training and testing: s20011, s20031, s20091, s20101, s20111, s20131, s20151, s20201, s20211, s20231, s20251, s20261, s20271, s20281, s20291, s20321, s20331, s20341, s20351, s20361, s20371, s20401, s20411, s20431, s20451, s20461, s20471, s20521, s20551, s20631, s30671, s30701, s30741, and s30801. The QRS complex inverter is not used on the Long Term ST database. For the training and the testing process, 14 random patient recordings are used as the training dataset and the remaining 20 patient recordings are grouped as the testing dataset.

3.3 Proposed Machine Learning Pipeline

Noisy ECG signals introduce large amounts of errors into the QRS complex detection process. Noises such as baseline wandering, electrode contact noise, and instrumentation noise are present in ECG signals. This paper proposes a novel machine learning pipeline to denoise the ECG signals and to detect the QRS complexes.

Fig. 3.1 shows the machine learning pipeline for detecting QRS complexes. The input ECG signal contains low frequency baseline wandering signals. Firstly, the Butterworth high pass filter eliminates the baseline wandering signals by attenuating the low frequency signals. This effectively stabilizes the ECG signal. Secondly, the WaveletCNN Autoencoder 1 filters out the non QRS complex signals. WaveletCNN Autoencoder 1 applies wavelet decomposition to the ECG signal in order to obtain the wavelet coefficients. After that, the wavelet coefficients feeds into a CNN autoencoder. The CNN autoencoder filters the noisy wavelet coefficients and produces the clean wavelet coefficients. Subsequently, wavelet reconstruction recovers the clean single channel ECG signal from the clean wavelet coefficients. Thirdly, the difference filter sharpens and enhances the QRS complexes. This is done by attenuating the low frequency components, while maintaining the high frequency components resembling the QRS complexes. Fourthly, the ECG signal passes through the WaveletCNN Autoencoder 2. The WaveletCNN Autoencoder 2 performs the same filtering operations as the WaveletCNN Autoencoder 1. For some datasets, a QRS complex inverter is applied to flip the inverted QRS complexes. Fifthly, the Monte Carlo k-NN normalizes the ECG signals by scaling the ECG peaks to 1 mV. Normalization is required in

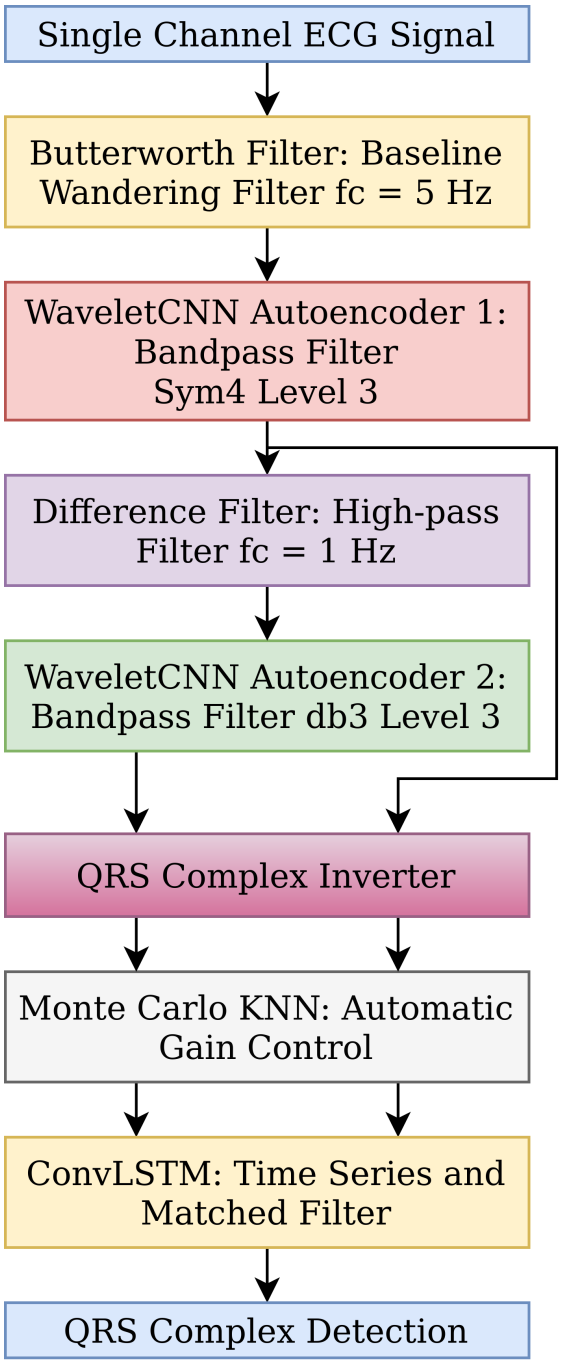

Figure 3.1: Machine learning pipeline for detecting QRS complexes.

order to compensate for the differing patients and ECG devices. Lastly, the ConvLSTM uses the filtered ECG signals to predict the QRS complexes. The ConvLSTM uses the timing accuracy of a LSTM combined with the pattern matching ability of a CNN to detect spatial temporal signals such as the QRS complexes.

3.3.1 Butterworth Filter: Baseline Wandering Filter

Baseline wandering signals create noises and distortions in the ECG signals. Furthermore, baseline wandering signals randomizes the amplitudes of the P waves, the R waves, and the T waves. As a consequence, the P waves and the T waves are often misclassified as the QRS complexes. In order to reduce the baseline wandering signals, a Butterworth highpass filter [91] of order $n = 3$ with $f_c = 5$ Hz is designed. The filter attenuates the low frequency baseline wandering signals, while preserving the high frequency components. The Butterworth filter essentially takes in a single channel baseline wandering ECG signal and outputs a single channel baseline removed ECG signal to the WaveletCNN Autoencoder 1.

3.3.2 WaveletCNN Autoencoder 1: Bandpass Filter

The QRS complex has a unique signature in the frequency and time domain. The wavelet filter [92] is able to isolate the QRS complexes from the other ECG signals by filtering out certain frequencies at certain time intervals. This lowers the false positive rate of the QRS complex detection algorithm. As a result, the algorithm achieves a higher detection accuracy.

Wavelet filtering starts with the wavelet decomposition process. Firstly, the approximation coefficients $A[t]$ are calculated by convolving the input function $x[t]$ with the lowpass filter $g[t]$, i.e.,

$$A[t] = \sum_{k=-\infty}^{\infty} x[k]g[t-k] = x[t] * g[t]. \tag{3.1}$$

Secondly, the detail coefficients $D[t]$ are computed by convolving the input function $x[t]$ with the highpass filter $h[t]$, i.e.,

$$D[t] = \sum_{k=-\infty}^{\infty} x[k]h[t-k] = x[t] * h[t]. \tag{3.2}$$

Thirdly, both sets of coefficients $A[t], D[t]$ are downsampled by 2. That is,

$$A_{down}[t] = A[2t+1] \tag{3.3}$$

$$D_{down}[t] = D[2t+1] \tag{3.4}$$

Finally, the downsampled detail coefficients $D_{down}[t]$ are kept, while the downsampled approximation coefficients $A_{down}[t]$ are fed as input $x[t]$ into the next wavelet level. The process above repeats until the wavelet coefficients of the desired level are obtained.

For every wavelet level, the detail coefficients $D_{down}[t]$ represent a fixed frequency component of the ECG signal. By attenuating certain detail coefficients $D_{down}[t]$, the non QRS complex signals such as instrumentation noises are reduced. This improves the performance of the proposed machine learning pipeline. Furthermore, certain wavelet coefficients are amplified in order to sharpen the QRS complexes. The process effectively decreases false negative rate and increases the true positive rate.

One way to filter the wavelet coefficients is by using the CNN [93]. The wavelet coefficients could be viewed as a 2D image, of which the CNN filters. The CNN removes the noises by detecting invalid spatial patterns in the wavelet coefficients and eliminating them. The CNN equations shown in

$$V^i_{y,z} = \sum_{j=1}^{N} \sum_{k=1}^{M} W^i_{j,k} X^i_{j+y-1,k+z-1} \tag{3.5}$$

$$O^i = A(V^i + B^i) \tag{3.6}$$

depict 2D convolution between the input matrix X^i and the kernel weights W^i. The kernel weights W^i slide across the input data X^i to produce V^i. After computing V^i, V^i is added to bias B^i. Then the resulting matrix passes through the activation function A and produces the output matrix O^i.

The WaveletCNN Autoencoder 1 receives the single channel baseline removed ECG signal from the Butterworth filter and outputs a filtered single channel ECG signal. The WaveletCNN Autoencoder 1 effectively functions as a bandpass filter by removing the false P waves, R waves, and T waves.

Fig. 3.2 shows the architecture of the WaveletCNN Autoencoder 1. Firstly, the WaveletCNN Autoencoder 1 performs wavelet decomposition on the noisy single channel ECG signal in order to produce the wavelet coefficients. WaveletCNN Autoencoder 1 uses the symlets 4 level 3 wavelet for wavelet decomposition. The symlets 4 wavelet approximates the shape of the QRS complex, which allows for the analyses of the QRS complexes. Secondly, the CNN encoder filters the wavelet coefficients using the 3x51 CNN kernels. After filtering, the CNN encoder downsamples the wavelet coefficients. The downsampling forces the WaveletCNN Autoencoder 1 to extract the

Table 3.1: Hyper-parameter Tuning of WaveletCNN Autoencoder 1.

RMSE	Kernel Size	Channels	Bottleneck Stride	α
0.141181	3x21	4	4x4	0.35
0.138602	3x31	4	4x4	0.35
0.134275	3x41	4	4x4	0.35
0.120745	3x51	4	4x4	0.35
0.121464	3x61	4	4x4	0.35
0.129826	1x51	4	4x4	0.35
0.125492	2x51	4	4x4	0.35
0.120745	3x51	4	4x4	0.35
0.122911	4x51	4	4x4	0.35
0.140361	3x51	1	4x4	0.35
0.123508	3x51	2	4x4	0.35
0.124838	3x51	3	4x4	0.35
0.120745	3x51	4	4x4	0.35
0.141749	3x51	4	1x1	0.35
0.131968	3x51	4	2x2	0.35
0.120745	3x51	4	4x4	0.35
0.125231	3x51	4	4x4	0.01
0.122039	3x51	4	4x4	0.1
0.121347	3x51	4	4x4	0.2
0.120745	3x51	4	4x4	0.35
0.123383	3x51	4	4x4	0.5

Note: RMSE units in mV. MIT-BIH NST 0 dB SNR.

most important features and to eliminate the unnecessary wavelet coefficients.

Thirdly, the CNN decoder reconstructs the original wavelet coefficients from the downsampled wavelet coefficients. The decoder uses the 3x51 kernel transpose CNN to upsample the wavelet coefficients. After upsampling, the wavelet coefficients are filtered by a CNN again. Fourthly, the neural network soft thresholding layer applies smoothing to the wavelet coefficients. Smoothing removes the high frequency noise. Finally, WaveletCNN Autoencoder 1 performs wavelet reconstruction on the filtered wavelet coefficients. In the end, a clean single channel ECG signal is produced.

Table 3.1 shows the hyper-parameter tuning of the WaveletCNN Autoencoder 1. The kernel size of the CNN layers is varied until the optimal kernel size of 3x51 is determined. Afterwards, the optimal number of channels is found to be 4. The bottleneck stride controls the amount of filtering that happens on the wavelet coefficients.

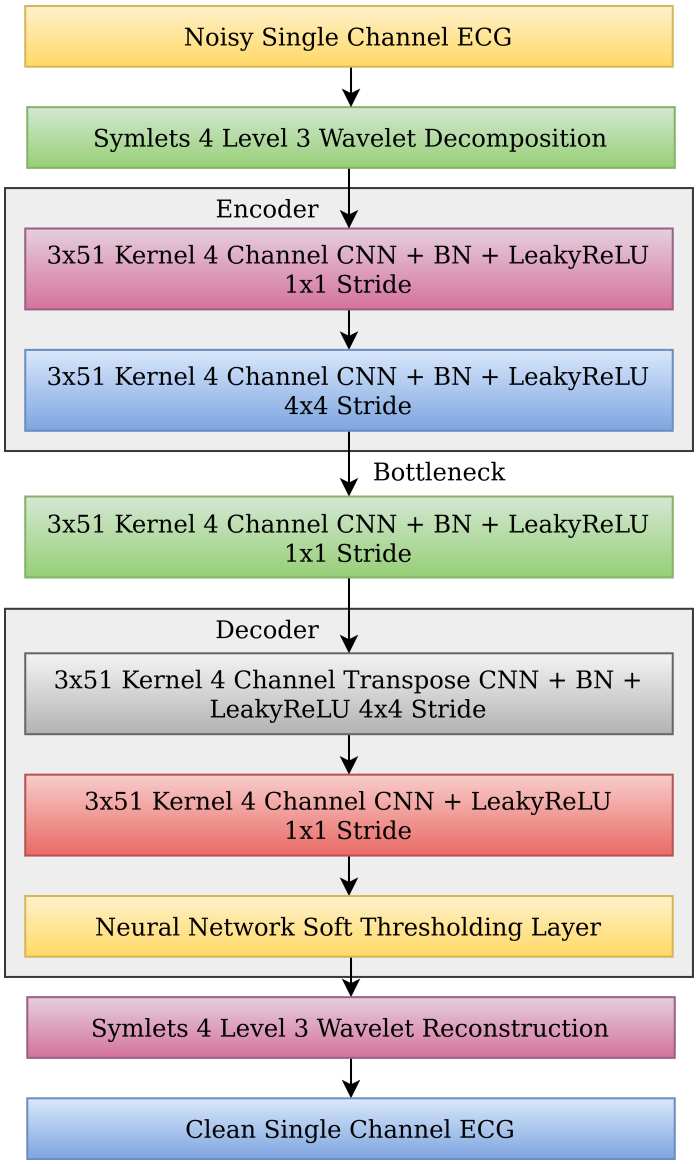

Figure 3.2: WaveletCNN Autoencoder 1 architecture.

The optimal bottleneck stride is 4x4. Each CNN layer uses the LeakyReLU activation function given by

$$LeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{3.7}$$

where x is the input matrix and α is the slope of the negative region. The LeakyReLU function prevents the loss function from reaching zero, which in turn prevents the optimizer getting stuck. The optimal slope determined by hyper-parameter tuning is

$\alpha = 0.35$. Each CNN layer also uses the batch normalization function given by

$$BN(x) = \frac{x - \mu_x}{\sigma_x} \tag{3.8}$$

where μ_x, σ_x are the mean and the standard deviation of x respectively. Batch normalization helps the neural network to converge faster, which results in a faster training process. The neural network soft thresholding layer $\epsilon = 1 \times 10^{-6}$ shown in

$$ReLU(x) = max(0, x) \tag{3.9}$$

$$\hat{y_i} = \frac{x}{|x| + \epsilon} ReLU(|x| - V_{thres}) \tag{3.10}$$

applies smoothing to the ECG signals. All input $|x|$ values are shifted down by the threshold V_{thres}. Subsequently, all input $|x|$ values below the threshold V_{thres} are set to zero. The WaveletCNN Autoencoder's output ECG signal $\hat{y_i}$ is produced by this layer.

The Adam optimizer [20] trains the WaveletCNN Autoencoder 1 using the root mean square error (RMSE) loss function stated below.

$$J(\hat{y}, y) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y_i})^2} \tag{3.11}$$

where $\hat{y_i}$, y_i are the predicted and noiseless ECG samples respectively, and N is the number of ECG samples. The loss function forces the network to reconstruct a clean ECG signal from a noisy ECG signal. If the predicted de-noised ECG signal $\hat{y}$ differs from the ground truth ECG signal y, then the loss is large, otherwise the loss is small. Fig. 3.4 shows the WaveletCNN Autoencoder 1 removing noises from the single channel ECG signal.

3.3.3 Difference Filter: High-pass Filter

In the ECG recordings, the R waves have sharper peaks than the P waves and the T waves. Therefore, the R waves have higher frequency components than the P waves and the T waves. The difference filter eliminates the P waves and the T waves by attenuating the low frequency components. As a result, the residual signal only consists of the high frequency components such as the R waves. The equation for the

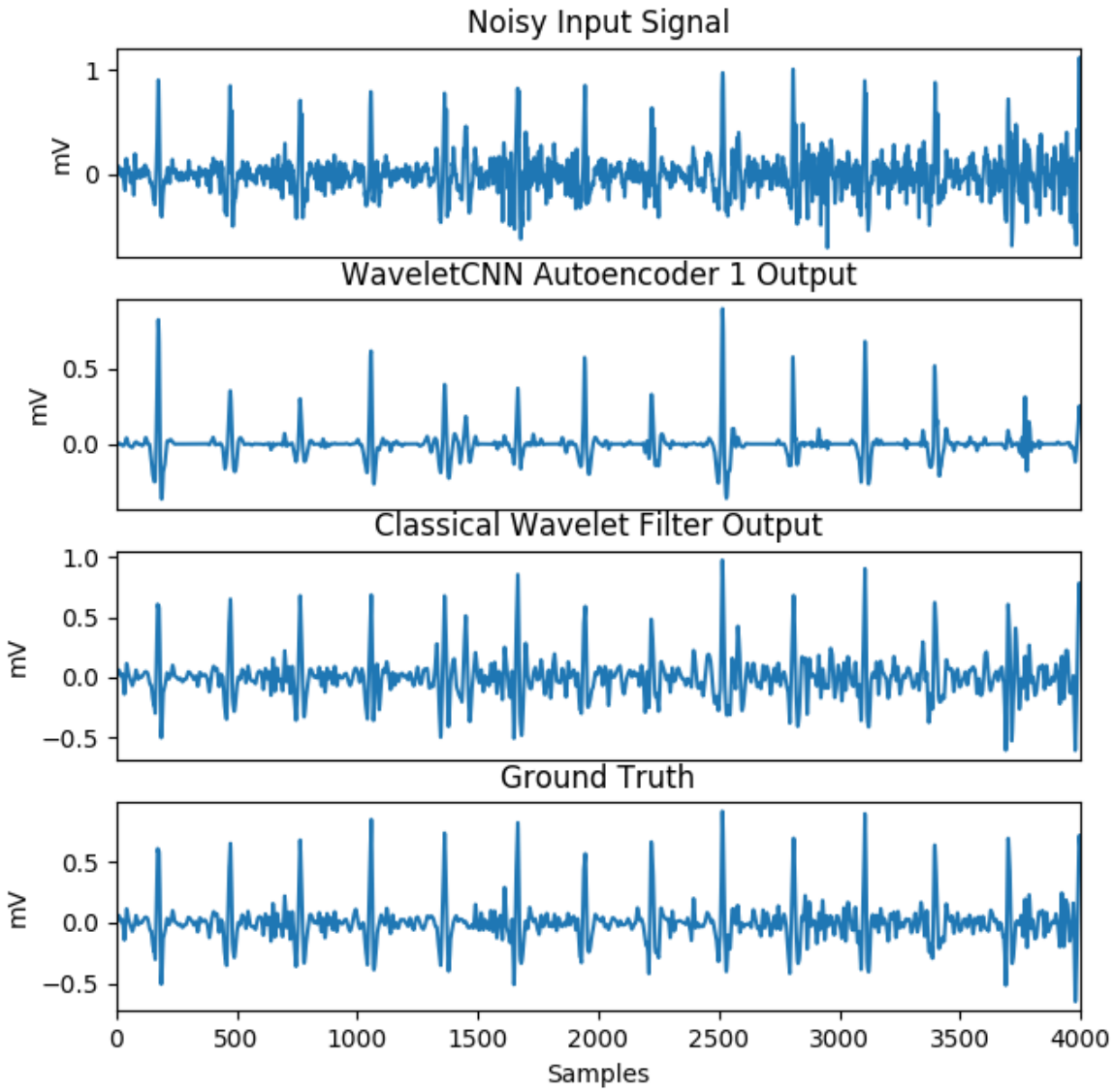

Figure 3.3: Comparison of the WaveletCNN Autoencoder 1 and the classical wavelet filter. 6 dB test SNR.

filter is presented below

$$y[t] = x[t] - x[t-1] \tag{3.12}$$

where $x[t]$ and $y[t]$ are the input and the output functions respectively. Alternatively, the filter could be thought as taking the gradient of the input function $x[t]$. In the end, the difference filter takes in the single channel ECG signal from the WaveletCNN Autoencoder 1 and sends the gradient of the ECG signal to the WaveletCNN Autoencoder 2.

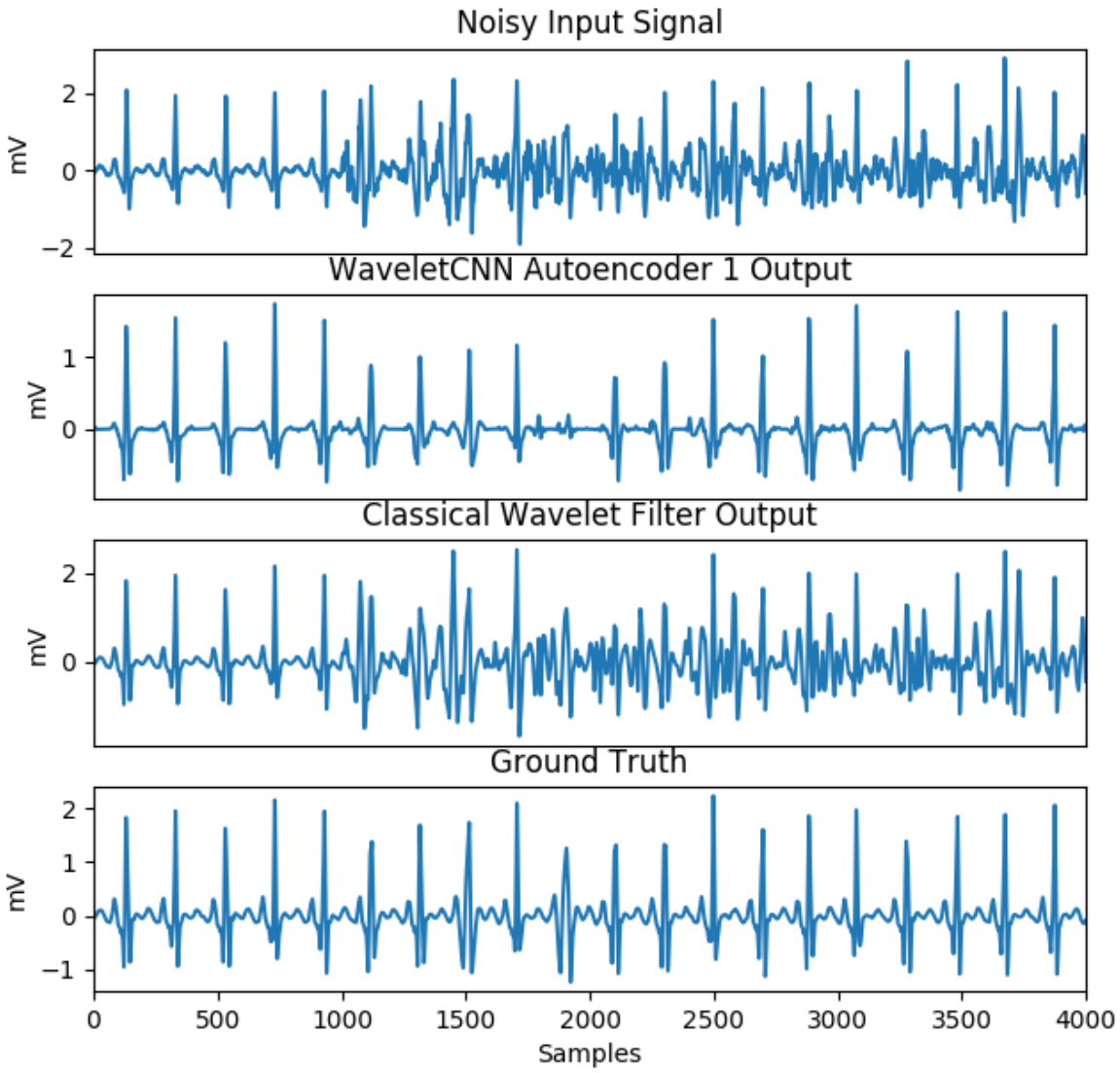

Figure 3.4: Comparison of the WaveletCNN Autoencoder 1 and the classical wavelet filter. 0 dB test SNR.

3.3.4 WaveletCNN Autoencoder 2: Bandpass Filter

The WaveletCNN Autoencoder 2 takes in the single channel ECG signal from the difference filter and applies another layer of filtering. During filtering, it removes the noises from the ECG signals while preserving the gradient of the QRS complex. WaveletCNN Autoencoder 2 and WaveletCNN Autoencoder 1 have the exact same architecture. However, the wavelet scaling functions are different. WaveletCNN Autoencoder 2 uses the Daubechies 3 level 3 wavelet, which excels at capturing the gradient of the QRS complex. Fig. 3.5 shows the WaveletCNN Autoencoder 2 fil-

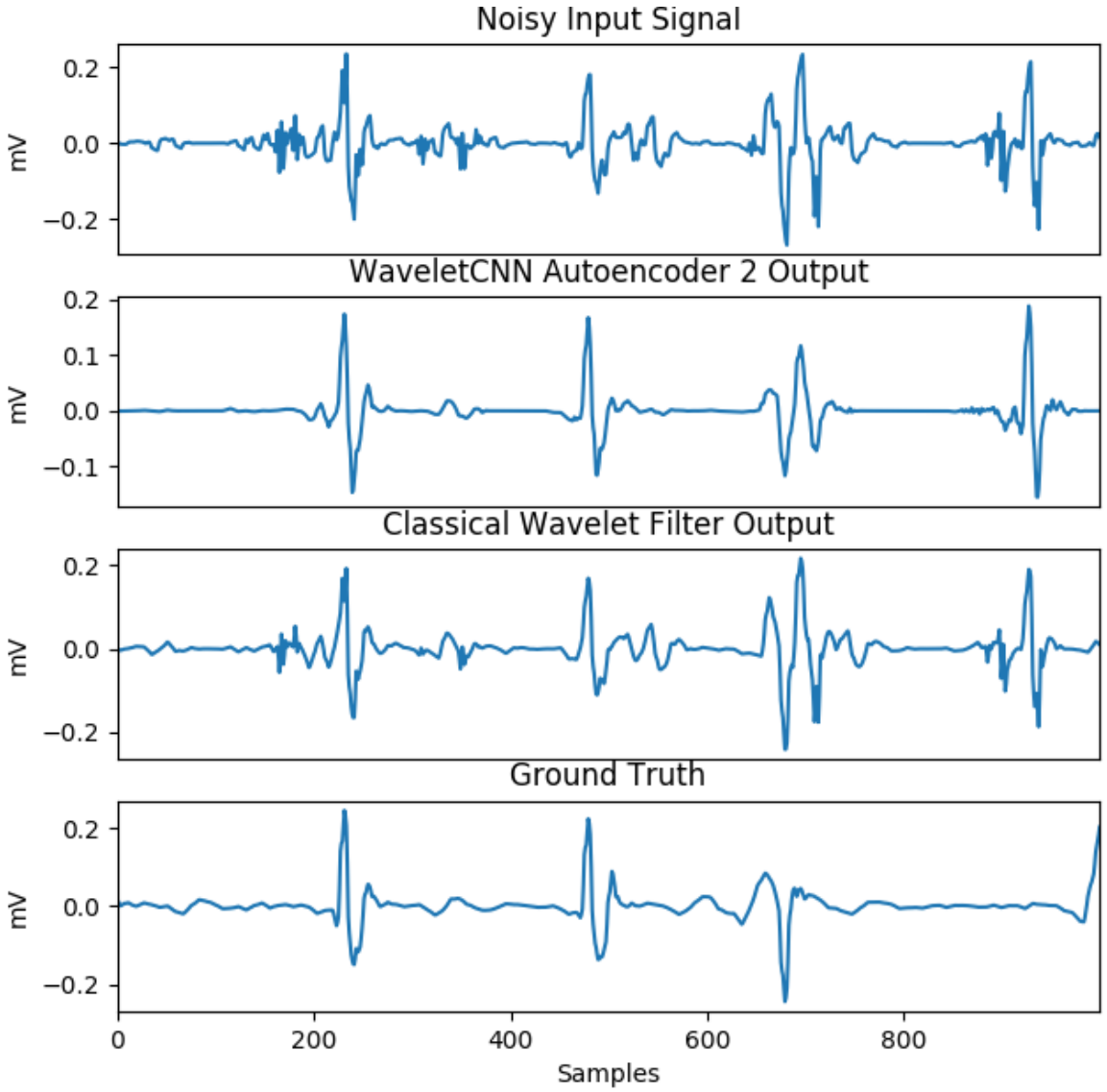

Figure 3.5: Comparison of the WaveletCNN Autoencoder 2 and the classical wavelet filter. 0 dB test SNR.

tering the single channel ECG signal. The ground truth is obtained by taking the derivative of the original noiseless ECG signal.

3.3.5 QRS Complex Inverter (Optional)

Normal QRS complexes have positive R peak amplitudes. On the other hand, inverted QRS complexes have negative R peak amplitudes. Inverted QRS complexes create problems for the Monte Carlo k-NN gain control and the ConvLSTM detector

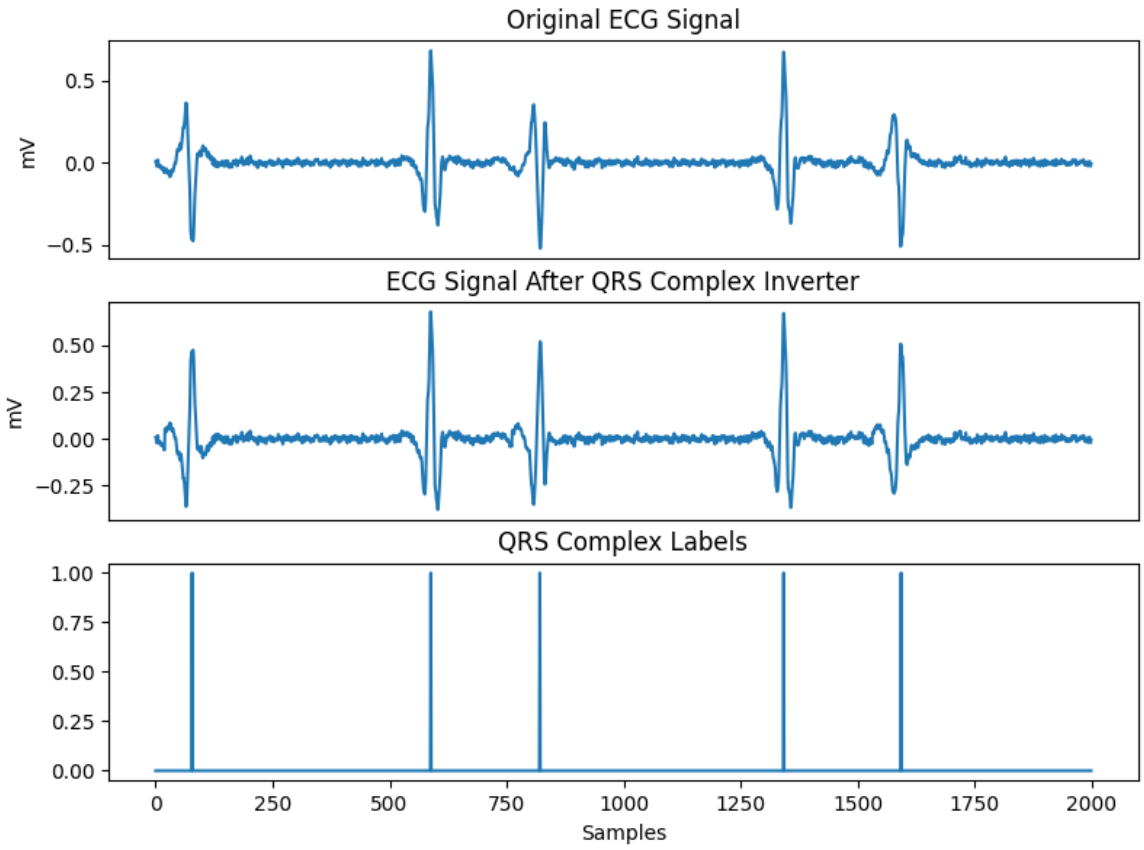

Figure 3.6: QRS complex inverter flipping inverted QRS complexes.

because both of them expect the R peaks to be positive. An optional QRS complex inverter is created to flip the inverted QRS complexes. If a patient has many inverted QRS complexes, this method improves the detection accuracy, otherwise it introduces errors and lowers the detection accuracy.

The QRS complex inverter is very simple. It has a 60 sample wide window. If the absolute value of the minimum value of the window is greater than the maximum value of the window

$$|\min(window)| > \max(window) \tag{3.13}$$

the window moves to position of the minimum value, otherwise the window increments by 20 samples. If the window reaches a negative R peak, the window is inverted.

$$window \leftarrow -window \tag{3.14}$$

The process above repeats until the window reaches the end of the ECG recording.

Fig. 3.6 shows the QRS complex inverter flipping inverted QRS complexes, while ignoring the normal QRS complexes.

3.3.6 Monte Carlo k-NN: Automatic Gain Control

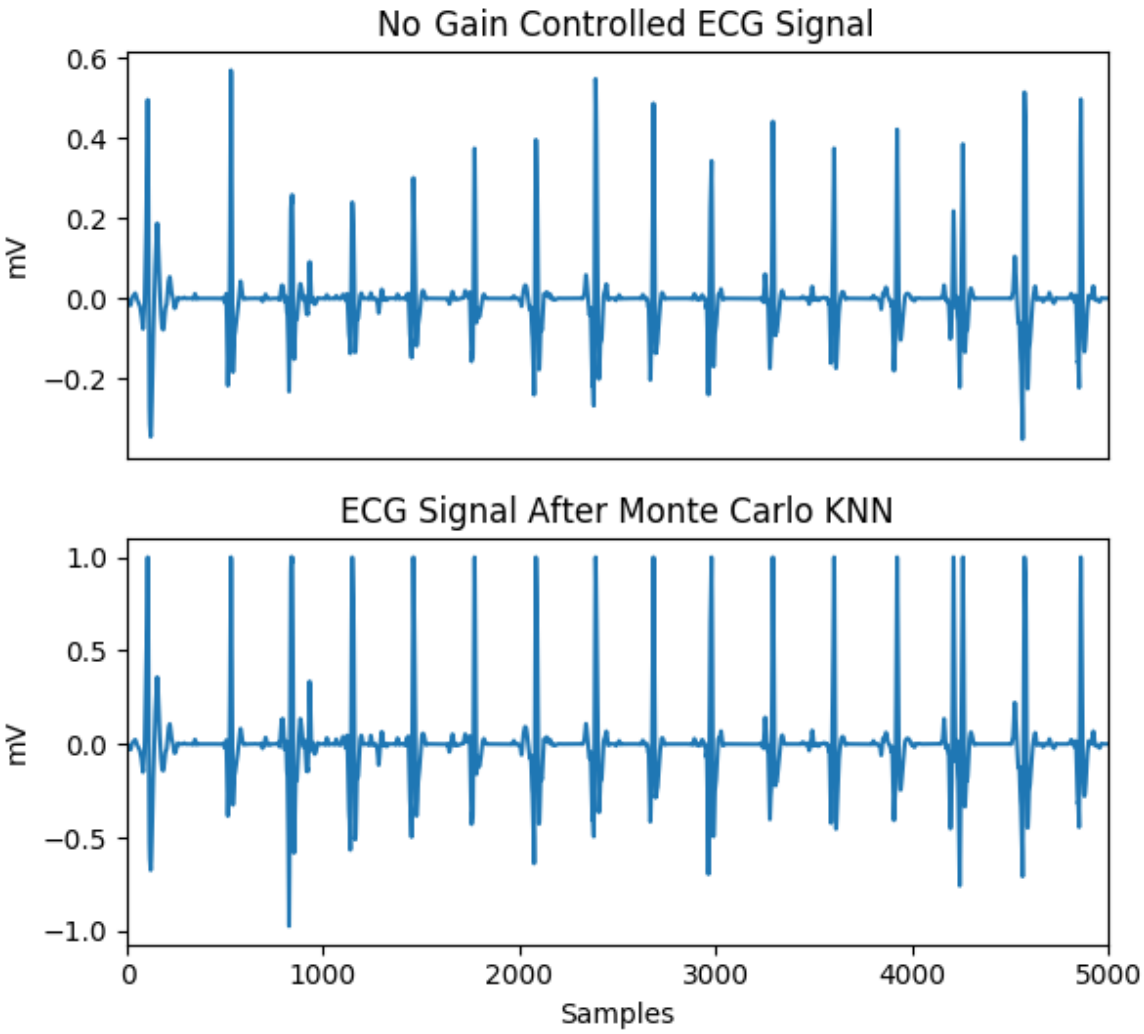

Figure 3.7: Application of the Monte Carlo k-NN.

Differing patients, instruments, and recording environments cause fluctuations in the QRS complex amplitudes. Normally, QRS complex detection algorithms use thresholds to detect the QRS complexes. If a fluctuation causes a QRS complex amplitude to drop below the threshold, then the algorithm does not detect it. This creates a false negative error.

In order to minimize the fluctuations, the Monte Carlo k-NN normalizes the QRS complex amplitudes to 1 mV. Firstly, the window start variable $winstr$ is randomly selected form a uniform distribution

$$winstr \in [0, len - winsize] \tag{3.15}$$

where len is the length of the input ECG signal $ECG1[t]$. Monte Carlo k-NN then calculates the window end

$$winend = winstr + winsize \tag{3.16}$$

using the window start and window size $winsize$. The window represents a time interval on the single channel ECG signal $ECG1[t]$. Secondly, k-NN clusters all the points in the window into two categories: R peak like signals and non R peak like signals. R peak like signals includes P waves, R waves, and T waves. The remaining signals represent the non R peak like signals. k-NN uses the Euclidean distances of the points' amplitudes for cluster assignment. Thirdly, the mean of the R peak like cluster

$$\mu_{Rpeak} = \frac{1}{N} \sum_{i=0}^{N} Rpeak_i \tag{3.17}$$

is computed, where $Rpeak_i$ is the amplitude of a R peak. Fourthly, the ECG signal between window start and window end

$$\begin{aligned} ECG1[winstr : winend] \leftarrow \\ \frac{ECG1[winstr : winend]}{\mu_{Rpeak}} \end{aligned} \tag{3.18}$$

is normalized using the R peak mean. Fifthly, the process repeats for X number of loops. The $winsize$ slowly decreases by $decsize$ for every Z loops. Finally, the single channel gain controlled $ECG2[t]$ signal is produced in a similar manner. Fig. 3.7 shows an example of the Monte Carlo k-NN with $X = 100len$ loops, $winsize = 2600$ initial window size, $decsize = 360$, and $Z = 20len$. In conclusion, the Monte Carlo k-NN takes in a single channel ECG signal from each autoencoder and outputs a 2 channel gain controlled ECG signal to the ConvLSTM.

3.3.7 ConvLSTM: Time Series and Matched Filter

The filtering process above is good at cleaning and enhancing the ECG signals. However, classification errors still persist at this stage. Therefore, a final QRS complex detector is needed in order to reduce the classification errors. Using the ConvLSTM [94] layer, the combination of a CNN layer and a LSTM layer on the same layer, the spatial-temporal patterns of the QRS complexes are detected. The ConvLSTM layer performs better than the CNN-LSTM layers because the ConvLSTM detects spatial-temporal patterns concurrently instead of detecting spatial patterns in CNN layer and temporal patterns in the LSTM layer separately. Furthermore, the architectural design of the ConvLSTM is much simpler than the CNN-LSTM due the ConvLSTM needing less layers.

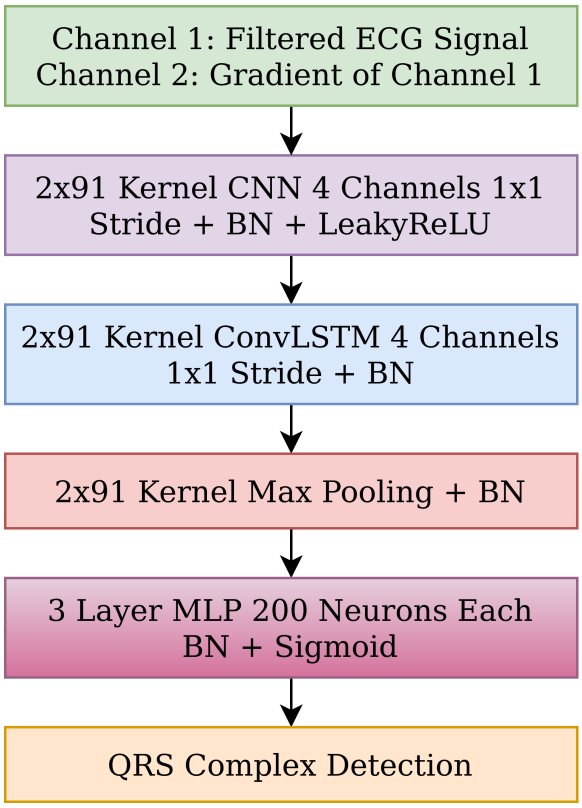

Figure 3.8: The ConvLSTM architecture.

Fig. 3.8 shows the ConvLSTM architecture, which has the same hyper-parameters as our previous paper [95]. The architecture consists of 1 CNN layer, 1 ConvLSTM layer, 1 max pooling layer, and 3 MLP layers. The CNN layer extracts features from the ECG signals, while filtering out the noises. After that, the ConvLSTM layer

predicts the QRS complex timings. Subsequently, the max pooling layer reduces the 4 channel signal to a 1 channel signal. At the end, the MLP layers format the data and produce a pulse train marking the locations of the QRS complexes.

The following paragraphs describe the ConvLSTM network in detail. Firstly, the network takes in the 2 channel ECG signal from the Monte k-NN. Channel 1 is the normal filtered ECG signal, while channel 2 is the filtered gradient version of channel 1. Secondly, the CNN layer applies a 2x91 kernel with a 1x1 stride to the data. The 2x91 kernel matches the 91 sample long gradient of the QRS complex. The 1x1 stride preserves the timing information of the ECG signal. Moreover, the layer uses 4 channels because the ECG signals contain 4 main waveforms: P wave, QRS complex, T wave, and QS complex. The activation function is the LeakyReLU function with $\alpha = 0.02$. Thirdly, the ConvLSTM layer parses the features gathered by the CNN layer. The layer is a combination of the CNN layer and the LSTM layer. The ConvLSTM layer has the timing advantages of the LSTM, where it predicts the future QRS complex locations before they appear based on the previous locations. Moreover, the spatial detection abilities of the CNN allow the network to distinguish QRS complexes from the noises. The following equations show the functionality of the ConvLSTM.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.19}$$

$$i_t = S(W_{xi} * X_t + W_{hi} * H_{t-1} + W_{ci} \circ C_{t-1} + b_i) \tag{3.20}$$

$$f_t = S(W_{xf} * X_t + W_{hf} * H_{t-1} + W_{cf} \circ C_{t-1} + b_f) \tag{3.21}$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \tanh(W_{xc} * X_t + W_{hc} * H_{t-1} + b_c) \tag{3.22}$$

$$o_t = S(W_{xo} * X_t + W_{ho} * H_{t-1} + W_{co} \circ C_{t-1} + b_o) \tag{3.23}$$

$$H_t = o_t \circ \tanh(C_t) \tag{3.24}$$

where W and b are the weight and bias matrices. $S(x)$ is the sigmoid activation function with respect to input matrix x. i_t, f_t, o_t, and H_t are the input gate, forget gate, hidden gate, and hidden state at time t respectively. X_t is the input image at time t and C_t is the output image at time t. $*$ denotes convolution and $\circ$ denotes element wise matrix multiplication. The ConvLSTM layer has 4 channels. Each channel has a 2x91 kernel with a 1x1 stride. The max pooling layer comes after the ConvLSTM layer, in which it selects the maximum value from the 2x91 image. Furthermore, the max pooling layer reduces the number of channels from 4 to 1 in

order to speed up the network convergence. After that, the MLP layers execute the final QRS complex detection using the data from the max pooling layer. There are 3 MLP layers. Each MLP layer has 200 neurons. Each neuron uses the sigmoid activation function shown in Eqn. (3.19). The sigmoid activation function constrains the output of the network to the $\hat{y} \in [0, 1]$ interval. The ConvLSTM network produces a detection signal $\hat{y}_t$ with respect to time t. If the ConvLSTM detects a QRS complex, then it predicts $\hat{y}_t = 1.0$, otherwise it predicts $\hat{y}_t = 0.0$. The ConvLSTM uses the weighted cross-entropy loss function

$$J(\hat{y}, y) = -\log(S(\hat{y}))(y)(W_{pos}) - \log(1 - S(\hat{y}))(1 - y) \tag{3.25}$$

for training, where y is the actual QRS complex and W_{pos} is the cross-entropy weight. The cross-entropy weight is $W_{pos} = 300$ because the RR interval is approximately 300 samples wide. Fig. 3.9 shows the ConvLSTM predicting a QRS complex using both ECG channels.

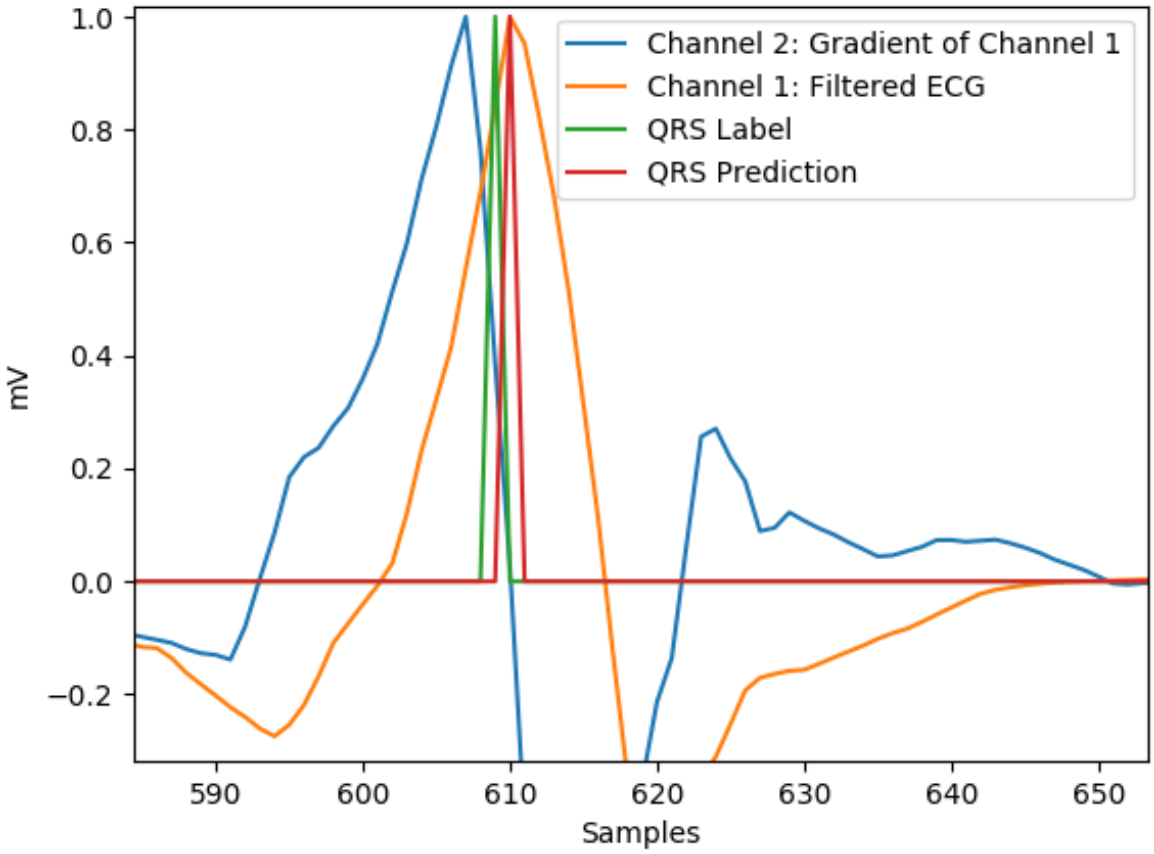

Figure 3.9: ConvLSTM QRS complex prediction using both ECG channels.

3.4 Simulations

In this paper, a few algorithms described in Section **??** are implemented for comparison to our machine learning pipeline. The QRS complex detection algorithms are benchmarked using the noisy dataset described in Section 3.2. The true positives (TP), false positives (FP), false negatives (FN), sensitivity (SEN), positive predictive value (PPV), F1 score (F_1), and RMSE timings of the QRS complex detection algorithms are recorded. Here, SEN, PPV and F_1 are computed according to the equations below,

$$SEN = \frac{TP}{TP + FN} \tag{3.26}$$

$$PPV = \frac{TP}{TP + FP} \tag{3.27}$$

$$F_1 = 2\frac{SEN \cdot PPV}{SEN + PPV}. \tag{3.28}$$

Sensitivity measures the number of false positives in the predicted QRS complexes. Subsequently, positive predictive value measures the number of false negatives. If a QRS complex detection algorithm performs well, then it must have a high sensitivity $SENS \approx 1$ and a high positive predictive value $PPV \approx 1$. This in turn causes the $F_1 \approx 1$ to be high. If a QRS complex algorithms performs poorly then all the metrics are low $SENS \approx 0, PPV \approx 0, F_1 \approx 0$.

If a QRS complex detection algorithm predicts a QRS complex within 50 ms of a true QRS complex, then the prediction counts as a true positive. If a QRS complex detection algorithm predicts a QRS complex and a true QRS complex does not exist within 50 ms of the predicted QRS complex, then the predicted QRS complex counts as a false positive. If a QRS complex detection algorithm does not predict a QRS complex within 50 ms of a true QRS complex, then the true QRS complex counts as a false negative. The true negatives are not relevant as none of the ECG metrics use them.

Table 3.2 shows the RMSE of the WaveletCNN Autoencoder 1 and the classical wavelet filter. Table 3.2 proves the WaveletCNN Autoencoder 1 filters out noises better than the classical wavelet filter because the WaveletCNN Autoencoder 1 has a lower RMSE mean. The performance improvement of the WaveletCNN Autoencoder 1 is due to the complex CNN autoencoder structure. Table 3.3 shows the WaveletCNN Autoencoder 2 performs slightly worse than the classical wavelet filter as the WaveletCNN Autoencoder 2 has a higher RMSE mean. The result is caused by

Table 3.2: WaveletCNN Autoencoder 1 and Classical Wavelet RMSE.

SNR	WaveletCNN Autoencoder 1	Classical Wavelet Filter
12 dB	0.0617 ± 0.010	0.0702 ± 0.004
6 dB	0.0917 ± 0.016	0.1476 ± 0.020
0 dB	0.1158 ± 0.020	0.2829 ± 0.026
-6 dB	0.1623 ± 0.024	0.5737 ± 0.026

Note: MIT-BIH NST. Confidence interval of 2σ. Units in mV.

Table 3.3: WaveletCNN Autoencoder 2 and Classical Wavelet RMSE.

SNR	WaveletCNN Autoencoder 2	Classical Wavelet Filter
12 dB	0.0129 ± 0.002	0.0112 ± 0.001
6 dB	0.0169 ± 0.001	0.0173 ± 0.001
0 dB	0.0206 ± 0.002	0.0214 ± 0.002
-6 dB	0.0301 ± 0.010	0.0317 ± 0.004

Note: MIT-BIH NST. Confidence interval of 2σ. Units in $\frac{mV}{Samples}$.

Table 3.4: MIT-BIH NST 12 dB SNR Algorithm Performance.

Algorithm	GQRS [47]	Pan-Tompkins [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	65189 ± 2738	54866 ± 3735	66837 ± 2553	58484 ± 31567	60972 ± 27072	65751 ± 2935
FP	12916 ± 1568	1175 ± 328	14077 ± 1511	4402 ± 3057	8745 ± 6331	3148 ± 1489
FN	3526 ± 555	14077 ± 4563	1829 ± 551	10349 ± 31103	6365 ± 25768	1980 ± 687
SENS	0.9486 ± 0.008	0.7961 ± 0.061	0.9733 ± 0.007	0.8490 ± 0.454	0.9036 ± 0.393	0.9707 ± 0.010
PPV	0.8345 ± 0.021	0.9789 ± 0.006	0.8259 ± 0.020	0.9284 ± 0.032	0.8708 ± 0.085	0.9543 ± 0.021
F_1 score	0.8879 ± 0.014	0.8778 ± 0.038	0.8935 ± 0.012	0.8640 ± 0.371	0.8743 ± 0.285	0.9624 ± 0.013
Timing RMSE	12.32 ± 0.16	6.97 ± 0.57	4.29 ± 0.42	3.75 ± 5.03	4.06 ± 4.76	2.98 ± 0.71

Note: Confidence interval of 2σ. Timing RMSE units in samples.

Table 3.5: MIT-BIH NST 0 dB SNR Algorithm Performance.

Algorithm	GQRS [47]	Pan-Tompkins [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	54774 ± 3361	11343 ± 1517	56384 ± 2011	51730 ± 35180	59868 ± 2820	62917 ± 2462
FP	35214 ± 1680	9440 ± 580	29608 ± 849	21180 ± 15049	24388 ± 6237	12806 ± 3724
FN	12977 ± 919	57163 ± 1823	12565 ± 869	16559 ± 34578	8809 ± 1349	5330 ± 1324
SENS	0.8083 ± 0.016	0.1654 ± 0.017	0.8177 ± 0.010	0.7566 ± 0.508	0.8717 ± 0.019	0.9219 ± 0.018
PPV	0.6085 ± 0.025	0.5452 ± 0.043	0.6556 ± 0.013	0.7387 ± 0.181	0.7112 ± 0.060	0.8311 ± 0.044
F_1 score	0.6943 ± 0.021	0.2538 ± 0.025	0.7277 ± 0.011	0.6923 ± 0.463	0.7831 ± 0.041	0.8739 ± 0.024
Timing RMSE	12.12 ± 0.10	6.05 ± 0.28	4.89 ± 0.36	3.12 ± 1.63	3.63 ± 0.67	3.86 ± 0.63

Note: Confidence interval of 2σ. Timing RMSE units in samples.

the ECG signal after the difference filter being relatively clean and the WaveletCNN Autoencoder 2 introducing slight noises.

Fig. 3.10 shows the learning curve of the ConvLSTM. The ConvLSTM approximately converges at 4480000 training samples. The minimum number of training sam-

Table 3.6: European ST-T NST 12 dB SNR Algorithm Performance.

Algorithm	GQRS [47]	Pan-Tompkins [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	65835 ± 2726	55556 ± 4387	66374 ± 2135	58909 ± 8037	66101 ± 2813	65696 ± 1736
FP	9511 ± 897	2241 ± 713	18451 ± 1752	2283 ± 1666	3293 ± 1338	2137 ± 676
FN	1206 ± 599	11517 ± 2522	1616 ± 352	6752 ± 8403	297 ± 145	628 ± 413
SENS	0.9819 ± 0.008	0.8281 ± 0.037	0.9761 ± 0.005	0.8974 ± 0.126	0.9955 ± 0.002	0.9905 ± 0.006
PPV	0.8737 ± 0.011	0.9612 ± 0.011	0.7824 ± 0.020	0.9633 ± 0.022	0.9525 ± 0.019	0.9685 ± 0.009
F_1 score	0.9247 ± 0.009	0.8896 ± 0.024	0.8686 ± 0.014	0.9277 ± 0.063	0.9735 ± 0.010	0.9794 ± 0.006
Timing RMSE	12.66 ± 0.30	10.11 ± 0.55	11.81 ± 0.34	0.75 ± 0.37	0.83 ± 0.22	1.46 ± 0.23

Note: Confidence interval of 2σ. Timing RMSE units in samples.

Table 3.7: European ST-T NST 0 dB SNR Algorithm Performance.

Algorithm	GQRS [47]	Pan-Tompkins [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	58611 ± 1680	15671 ± 2187	57813 ± 1419	57121 ± 6175	62234 ± 2475	63130 ± 1382
FP	32359 ± 980	8101 ± 762	31697 ± 711	17375 ± 7378	19569 ± 3184	9991 ± 3527
FN	8354 ± 854	53038 ± 2386	9913 ± 624	8880 ± 4375	3882 ± 829	2922 ± 514
SENS	0.8752 ± 0.012	0.2280 ± 0.030	0.8536 ± 0.007	0.8650 ± 0.069	0.9412 ± 0.012	0.9557 ± 0.007
PPV	0.6442 ± 0.012	0.6586 ± 0.035	0.6458 ± 0.010	0.7683 ± 0.076	0.7609 ± 0.032	0.8637 ± 0.043
F_1 score	0.7421 ± 0.011	0.3386 ± 0.038	0.7353 ± 0.007	0.8130 ± 0.054	0.8414 ± 0.019	0.9072 ± 0.023
Timing RMSE	12.38 ± 0.16	12.03 ± 0.39	12.35 ± 0.38	1.75 ± 0.45	1.73 ± 0.23	1.98 ± 0.46

Note: Confidence interval of 2σ. Timing RMSE units in samples.

Table 3.8: Long Term ST NST 0 dB SNR Algorithm Performance.

Algorithm	GQRS [47]	Pan-Tompkins [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	36393 ± 1457	8054 ± 1828	37534 ± 1617	39488 ± 3270	40114 ± 1261	41205 ± 1647
FP	22031 ± 611	4975 ± 485	17973 ± 969	10615 ± 2243	9605 ± 1409	5075 ± 1946
FN	6458 ± 265	35151 ± 3286	5958 ± 325	2937 ± 2394	2208 ± 586	1450 ± 346
SENS	0.8492 ± 0.005	0.1867 ± 0.047	0.8629 ± 0.005	0.9305 ± 0.058	0.9478 ± 0.013	0.9659 ± 0.008
PPV	0.6228 ± 0.013	0.6165 ± 0.050	0.6761 ± 0.021	0.7887 ± 0.024	0.8069 ± 0.023	0.8904 ± 0.040
F_1 score	0.7186 ± 0.010	0.2862 ± 0.060	0.7581 ± 0.014	0.8533 ± 0.017	0.8716 ± 0.014	0.9265 ± 0.024
Timing RMSE	12.08 ± 0.35	6.16 ± 0.36	4.95 ± 0.22	1.65 ± 0.11	1.60 ± 0.12	1.74 ± 0.31

Note: Confidence interval of 2σ. Timing RMSE units in samples.

Table 3.9: Long Term ST NST -6 dB SNR Algorithm Performance.

Algorithm	GQRS [47]	Pan-Tompkins [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]	Proposed
TP	32520 ± 824	2412 ± 73	33985 ± 1491	35727 ± 2218	34898 ± 1093	37774 ± 1827
FP	28147 ± 276	7685 ± 294	22126 ± 657	18910 ± 2549	19016 ± 1214	14343 ± 3373
FN	10285 ± 323	40508 ± 1397	9657 ± 476	6397 ± 2184	7292 ± 655	5011 ± 1078
SENS	0.7597 ± 0.003	0.0562 ± 0.002	0.7787 ± 0.003	0.8481 ± 0.051	0.8272 ± 0.012	0.8828 ± 0.025
PPV	0.5360 ± 0.006	0.2389 ± 0.007	0.6056 ± 0.016	0.6541 ± 0.033	0.6473 ± 0.018	0.7252 ± 0.052
F_1 score	0.6285 ± 0.004	0.0910 ± 0.002	0.6813 ± 0.010	0.7384 ± 0.033	0.7262 ± 0.011	0.7962 ± 0.042
Timing RMSE	12.07 ± 0.30	9.84 ± 0.16	5.29 ± 0.21	2.54 ± 0.19	2.71 ± 0.08	2.41 ± 0.25

Note: Confidence interval of 2σ. Timing RMSE units in samples.

ples is at least 4480000 training samples, of which 4480000 training samples translates to 7 different ECG recordings.

The Bayes factor K [1, 96] is a statistical tool that compares the relative probabilities of the alternative hypothesis H_1 and the null hypothesis H_0. The Bayes factor

Table 3.10: Bayes Factor K Applied to Tables 3.4-3.9s' F_1 Scores.

Database	GQRS [47]	Pan-Tompkins [43]	Wavedet [48]	Xiang et al. [65]	Chandra et al. [68]
Proposed on MIT-BIH NST 12 dB SNR	> 1000	> 1000	> 1000	10.59	9.200
Proposed on MIT-BIH NST 0 dB SNR	> 1000	> 1000	> 1000	6.266	> 1000
Proposed on European ST-T NST 12 dB SNR	> 1000	> 1000	> 1000	23.56	5.421
Proposed on European ST-T NST 0 dB SNR	> 1000	> 1000	> 1000	> 1000	> 1000
Proposed on Long Term ST NST 0 dB SNR	> 1000	> 1000	> 1000	> 1000	> 1000
Proposed on Long Term ST NST -6 dB SNR	> 1000	> 1000	> 1000	64.62	> 1000

Table 3.11: Interpretation of Bayes Factor K [1]

Bayes Factor K	Interpretation
1 to 3.2	Barely worth mentioning
3.2 to 10	Substantial evidence against the null hypothesis H_0
10 to 100	Strong evidence against the null hypothesis H_0
¿100	Decisive evidence against the null hypothesis H_0

K is defined by

$$K = \frac{P(D|H_1)}{P(D|H_0)} = \frac{P(H_1|D)}{P(H_0|D)} \frac{P(H_0)}{P(H_1)} \tag{3.29}$$

and is calculated using the observed data D in Tables 3.4-3.9, which show the classification performances of the QRS complex detection algorithms. Alternative hypothesis H_1 assumes the proposed method's F_1 score is higher than the other QRS

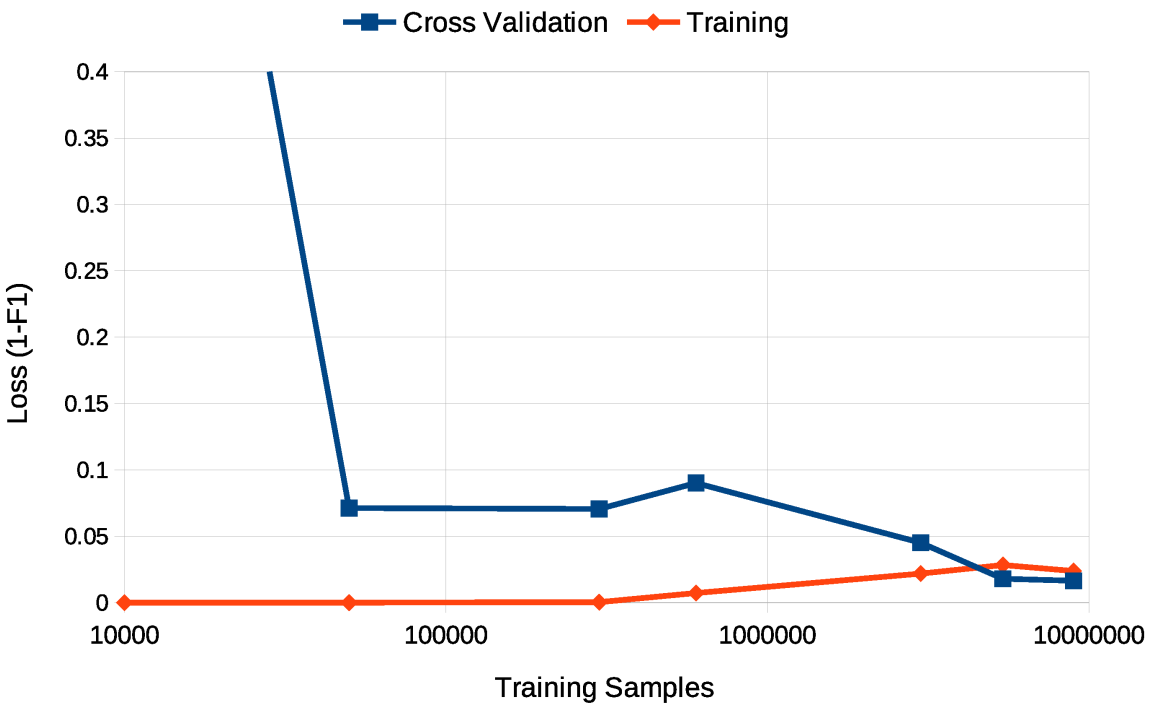

Figure 3.10: ConvLSTM's learning curve. MIT-BIH NST 12 dB SNR.

complex algorithms' F_1 scores. Null hypothesis H_0 assumes the other QRS complex algorithms' F_1 scores are higher than the proposed method's F_1 score. Assume the prior probabilities of the hypotheses are equal $P(H_0) = P(H_1)$. Assume the F_1 scores follow a truncated Gaussian distribution in the range of $[0, 1]$.

Table 3.10 shows the Bayes factors K for each QRS complex algorithm comparison. Two algorithms are compared at the same time and one algorithm is always the proposed method. Table 3.11 shows the degree of evidence against the null hypothesis H_0. For the MIT-BIH NST 12 dB SNR, all the Bayes factors are above $K > 3.2$. According to the Table 3.11, there is substantial evidence against the null hypothesis H_0. Therefore, the null hypothesis H_0 is rejected. There is a lack of strong or decisive evidence against the null hypothesis H_0 because the variances in Xiang et al.'s F_1 score [65] and Chandra et al.'s F_1 score [68] are large. Similar to MIT-BIH NST 12 dB SNR, all the Bayes factors K in MIT-BIH NST 0 dB SNR are above $K > 3.2$. As a result, the null hypothesis H_0 is rejected. For the comparison between the proposed method and Xiang et al., there is only substantial evidence against the null hypothesis H_0 because Xiang et al.'s F_1 score has high variance.

In the European ST-T NST 12 dB SNR, all the Bayes factors are $K > 10$ except for Chandra et al.'s F_1 score. This is due to the database having a high SNR and

Chandra et al.'s method having a similar performance to the proposed method. The null hypothesis H_0 is rejected due to all Bayes factors being $K > 3.2$ and having substantial evidence against the null hypothesis H_0. When the SNR decreases in the European ST-T NST 0 dB SNR, all the Bayes factors are $K > 1000$. As a result, there is decisive evidence against the null hypothesis H_0 and the null hypothesis H_0 is rejected without a shadow of a doubt. In the Long Term ST NST 0 dB SNR and the Long Term ST NST -6 dB SNR, all the Bayes factors are $K > 10$. There is strong evidence against the null hypothesis H_0 and the null hypothesis H_0 is rejected.

Reasons for the machine learning pipeline performing better than the other methods are presented below. Baseline wandering noises have frequencies below 5 Hz, of which can be reduced using a highpass filter that removes all signals below 5 Hz. On the other hand, wavelets are basically the templates of the targeted signal that the user wants to detect. To detect a specific signal, the wavelet is convolved with the ECG recording. The positions of the targeted signals can be determined by looking at the local maximums of the convolved signal. Most targeted signals do not have a single frequency. In order to detect targeted signals at different frequencies, the wavelet is first convolved with ECG recording. After convolution, the resulting signal is down-sampled to detect low frequency signals. Convolution in the first step is repeated again to detect lower and lower frequency signals. White Gaussian noise is present in all frequencies. However, electrode contact noise only appears at high frequencies and has a unique shape. Furthermore, the QRS complex only appears at high frequencies and also has a unique shape. The wavelet filter can effectively apply a bandpass filter that eliminates all other frequencies except for the frequencies that the QRS complex resides in. This will remove the white Gaussian noise outside the QRS complex frequency range. Moreover, the wavelet could be chosen to only match the QRS complex and not the other signals, thus removing the electrode contact noise.

The WaveletCNN Autoencoder automatically determines which frequencies to filter and which signals to remove by adjusting the CNN filters. This is in contrast to the classical wavelet filter, where the frequency filters have to be chosen manually and they may not be optimal. Noises also cause fluctuations in the QRS complex amplitudes. Monte Carlo k-NN normalizes the QRS complex amplitudes to 1 mV. It does this by repeatedly scaling the random ECG windows using the mean value of the R peaks. This is in contrast to the traditional normalization techniques, where the ECG signal is only processed once, and it is not detailed enough.

For the ConvLSTM detector, it uses convolutions from the present window and convolutions from past windows to detect QRS complexes. ConvLSTM effectively uses temporal information as well as spatial information. For example, the ConvLSTM detects a QRS complex using the convolutions filters at window $t = 4s$. ConvLSTM knows the QRS complex period is approximately $\Delta t \approx 1s$. It can predict the next QRS complex will occur at approximately $t \approx 5s$. The ConvLSTM can confirm the prediction by executing a convolution on the window at $t = 5s$. This is in contrast to previous papers using CNNs, where the temporal information is not used, and prediction accuracy is inferior.

This paper is consistent with the previous works as the relative performances of the QRS complex algorithms remain the same. Pan-Tompkins [43] algorithm's F_1 score is drastically lower than the other algorithms' F_1 score under lower SNR conditions, because it is the first significant algorithm proposed and improvements were made by other work after its appearance.

3.5 Conclusion

The proposed machine learning pipeline de-noises the ECG signals and detects the QRS complexes. The machine learning pipeline consists of a Butterworth filter, two WaveletCNNs autoencoders, a Monte Carlo k-NN, and a ConvLSTM. The Butterworth filter and the WaveletCNN autoencoders filter out the various noises in ECG signals. Monte Carlo k-NN normalizes the QRS complexes. The ConvLSTM executes the final QRS complex detection and produces the QRS complex detection signal. The MIT-BIH NST, the European ST-T NST, and the Long Term ST NST databases provide the training and testing ECG recordings. Null hypothesis H_0 assumes the other QRS complex algorithms' F_1 scores are higher than the proposed method's F_1 score. It has been demonstrated in Table 3.10 that all the Bayes factor $K > 3.2$, which means there is substantial evidence against the null hypothesis H_0. Therefore, the null hypothesis H_0 is rejected. In conclusion, the proposed machine learning pipeline outperforms existing QRS complex detection methods.

3.6 Limitations and Future Work

The proposed method is trained using the PhysioNet NST. As a result, the proposed method is optimized for the specific noise distribution present in the NST. However,

the machine learning pipeline might have a lower accuracy when faced with real ECG noises as they might have different noise distributions. In the future, an online training version of this machine learning pipeline could be developed in order to adapt to new noises in real time. This would allow the method to detect new QRS complexes beyond the initial training dataset. The QRS complex inverter and the Monte Carlo k-NN run very slowly because the scikit-learn [97] library only uses one CPU. The components could be parallelized in order to run on multiple CPUs. This will decrease the processing time.

Aside from the QRS complex, many other ECG waveforms are also important. For example, the T wave and P wave are very important for determining the ST interval, QT interval, and PR interval. The intervals are useful for classifying ECG diseases such as atrial fibrillation, arrhythmia, and branch blocks. Other biological signals like electroencephalogram (EEG) [98], electromyography (EMG), and photoplethysmogram (PPG) prove useful for diagnosing disorders and diseases. Similar to the ECG signals, the other biological signals are susceptible to noise. In the future, the machine learning pipeline could be used to reduce the noise, perform gain control, and detect the other waveforms.

3.6.1 Ventricular Tachycardia

The machine learning pipeline is unable to detect ventricular tachycardia because the ventricular tachycardia has a very low frequency and the Butterworth highpass filter removes it. Moreover, the beats for ventricular tachycardia are labeled as "!" and they are not used in this study.

Chapter 4

Generating Infrared Gaseous Spectra using Generative Adversarial Networks

The field of spectroscopy excels at the detection of elements or chemical compounds in unknown substances. For example, infrared (IR) absorbance spectroscopy allowed Al-Tameme et al. [99] to determine the chemical composition of Urtica dioica leaves. Zaini et al. [100] determined the chemical concentrations of CO_3 and Al−OH in cement-grade limestone using IR absorbance spectroscopy. Wang et al.'s paper [101] used near IR absorbance spectroscopy for the analysis of alcoholic beverages, non-alcoholic beverages, milk, and oils. Furthermore, Wang et al. [101] classified the types of liquid foods as well as determined the quality of the liquid foods. Ryde et al. [102] estimated the concentrations of H, Fe, Si, Ti, Ca, Mg, and Sc of a red giant using multiple emission spectroscopy bands.

Fourier transforms are popular in the field of spectroscopy. Yang et al. [103] analyzed IR spectra using the Fourier transform. Spectroscopy allowed Yang et al. [103] to retrieve data about the protein structures in aqueous solutions. Fuller and Ogilvie [104] created 2D Fourier transform algorithms for the analysis of spectra. Nesakumar et al. [105] used the Fourier transform and the principal component analysis (PCA) for the detection of moisture content in beetroot.

Partial least squares (PLS) [106] utilizes matrix decomposition to extract features from the absorbance spectra. The extracted features excel at predicting the presences of chemical compounds. For example, Xie et al. [107] used genetic algorithms and

PLS to analyze near IR spectra. The goal of Xie et al.'s paper [107] was to classify the types of soils using near IR spectra. Yin et al. [108] applied genetic algorithms and PLS to terahertz spectroscopy. Yin et al. [108] wanted to identify the types of edible oils found in foods. Zhu et al. [109] used genetic algorithms and PLS to detect the emissions of boron monoxide.

Deep learning has brought many improvements to the field of spectroscopy. A survey paper [110] lists the various deep learning algorithms for spectra quantification and classification. Afara et al. [111] used deep learning for the classification of cartilage integrity in near IR spectroscopy. Liu et al. [112] developed an autoencoder network for near IR spectroscopy feature extraction. After extraction, they classified the qualities of the cigarettes. Gan et al. [113] used PCA together with a multilayer perceptron (MLP) for classifying multiple gasses in near IR spectra. Furthermore, Gan et al. [113] used optimal thresholding to balance the false positive errors and the false negative errors in order to obtain the best performance. Chen et al. [114] implemented convolutional neural networks (CNNs) for the classification of hyperspectral satellite images. A CNN was developed to predict soil properties in Padarian et al.'s paper [115]. They predicted properties such as clay, sand, and pH using multiple 2D CNN layers and max pooling layers. Yuanyuan et al. [116] used an ensemble of CNNs to quantify the concentrations of various gasses and liquids. The ensemble of CNNs performed slightly better than a single CNN. The neural networks mentioned above are very complex and they have many parameters. As a consequence, the networks require large amounts of data in order to avoid overfitting. Traditionally, experimental measurements were used to train the neural networks. However, the collection of experimental data requires large amounts of time. Furthermore, experimental measurements are prone to instrumentation errors and calibration errors.

Line-by-line molecular spectroscopy databases such as High Resolution Transmission molecular absorbance database (HITRAN) [2], Carbon Dioxide Spectroscopic Databank (CDSD-4000) [117], and High-Temperature Molecular Database (HITEMP) [118] were developed to solve these problems. These databases contain spectra of individual molecules, which allows the user to synthesize spectra for any combination of molecules. However, spectra synthesis using the databases comes with significant downsides. The databases do not contain the non linear molecular interactions between the different molecules. This results in invalid spectra for some mixtures. Furthermore, realistic noises are not accurately represented, which creates a discrepancy between the simulated spectra and experimentally measured spectra.

We propose a generative adversarial network (GAN) [119] for the synthesis of C_2H_6, CH_4, CO, H_2O, HBr, HCl, HF, N_2O, and NO spectra. The GAN consists of a generator and a discriminator. The generator creates realistic spectra using the input gas concentrations. It also accounts for the non linear interactions between the molecules. Furthermore, the generator allows the user to change the signal to noise ratios (SNRs) of the synthetic spectra. On the other hand, the discriminator classifies the spectra into two groups: generated spectra and actual spectra. This is done in order to force the generator to synthesize more realistic spectra. The GAN is trained using a zero sum game in order to introduce realistic non-deterministic noises into the spectra.

The paper is organized as follows. Section 4.1 describes the data preparation for the simulated spectra. The MLP and the CNN classification and quantification algorithms for the simulated spectra are explained in Section 4.2. Classification algorithms predict the presences of the gasses in the spectra. On the other hand, quantification algorithms predict the concentrations of the gasses in the spectra. Section 4.3 describes the proposed GAN for the generation of spectra. At the end, Section 4.4 presents a conclusion.

4.1 Data Preparation

The HITRAN [2] database contains many individual gas spectra sampled at 296 K. The database is used to synthesize the simulated 100,000 training, 25,000 cross-validation, and 25,000 testing datasets. Firstly, these individual gas spectra are selected from HITRAN: C_2H_6, CH_4, CO, H_2O, HBr, HCl, HF, N_2O, and NO. The spectra of all 9 gases is shown in Figure 4.1. Secondly, the gas spectra are constrained to have a wavelength range from 1μm to 7μm. The gas spectra are uniformly down sampled to 10,000 pixels. Thirdly, a gas concentration matrix C is randomly generated using a uniform distribution of concentrations ranging from 0μM to 10μM. Fourthly, each gas concentration value c in the gas concentration matrix C is randomly set to 0μM with a probability of $p = 0.5$. Fifthly, the gas spectra are scaled using the gas concentration matrix C. The absorbance $A(\lambda)$ scaling is done using the Beer-Lambert law [120]

$$A(\lambda) = \epsilon(\lambda)lc \tag{4.1}$$

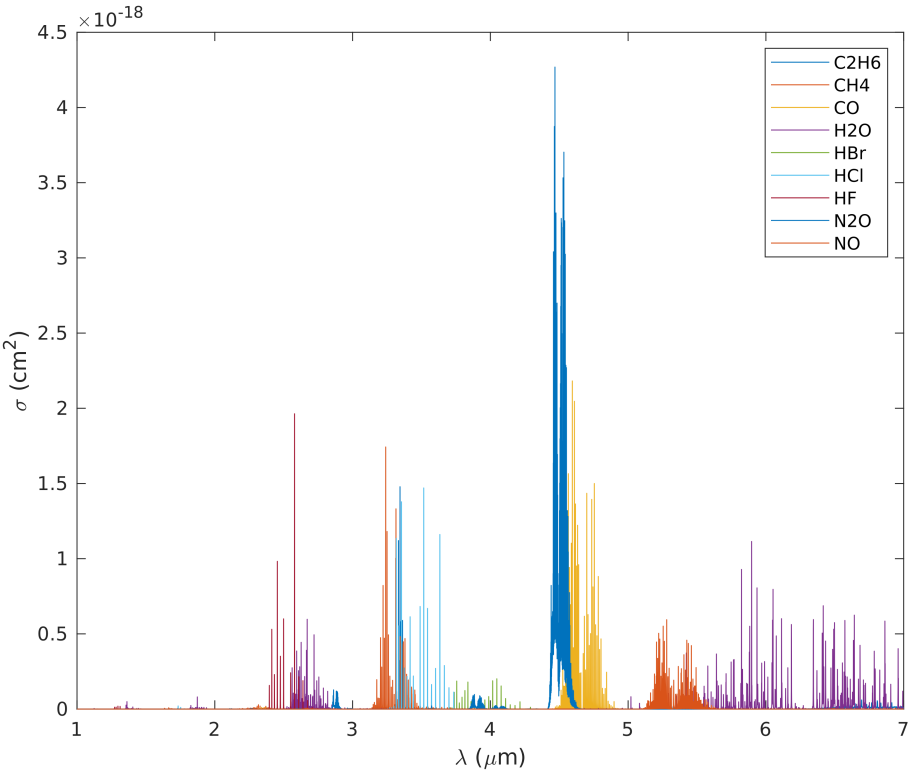

Figure 4.1: 9 gas spectra from HITRAN [2].

where $\epsilon(\lambda)$ is molar attenuation coefficient with respect to wavelength λ. The gas concentration values c are taken from the gas concentration matrix C and the path length is set to $l = 10$ cm. Sixthly, the individual gas spectra are linearly added together in order to get a single spectrum. Noises are added to the absorbance spectra in order to simulate the thermal noise, the light source noise, and the quantum noise found in real spectra analyzers. Specifically, additive white Gaussian noises (AWGN) $g(x|SNR)$

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.2}$$

$$w(x|SNR) = 1 + f(x|\mu = 0, \sigma = 10^{\frac{-SNR}{10}}) \tag{4.3}$$

$$g(x|SNR) = 10\log_{10}(\max(w(x|SNR), 1 \times 10^{-20})) \tag{4.4}$$

are added to the absorbance spectra. For all of the noise distributions, the mean is set to zero $\mu = 0$ and the standard deviation σ depends on the SNR (dB). However, the noises are not added to the concentrations c, path length l, or molar attenuation coefficient $\epsilon(\lambda)$.

4.2 Classification and Quantification Algorithms

In order to verify the spectra synthesis capabilities of the GAN, the MLP and CNN classifiers and quantifiers are developed. For classification, the neural network predicts the presences of the gasses. The algorithms are evaluated using micro recall (R), precision (P), and F_1 score

$$R = \frac{TP}{TP + FN} \tag{4.5}$$

$$P = \frac{TP}{TP + FP} \tag{4.6}$$

$$F_1 = 2\frac{P \cdot R}{P + R} \tag{4.7}$$

where FP represents the false positives, the FN represents the false negatives, and the TP represents the true positives. For quantification, the neural network predicts concentrations of the gasses. The algorithms are evaluated using micro $RMSE(\hat{y}, y)$

$$RMSE(\hat{y}, y) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{4.8}$$

where $\hat{y}_i$ is the predicted concentration and y_i is the actual concentration at gas index i.

4.2.1 Multilayer Perceptron Spectra Classification

Fig. 4.2 shows the structure of the MLP classifier, which takes in the simulated 10,000 pixel 9 gas spectra and predicts the presences of the gases. The MLP classifier follows the basic MLP equation [26]

$$O = A(W^T X + B) \tag{4.9}$$

where x is the input matrix, W is the weight matrix, B is the bias matrix, $A(M)$ is the activation function with respect to matrix M, and O is the output matrix. The first MLP layer has 250 neurons and uses the LeakyReLU activation function $\alpha = 0.001$

$$LeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{4.10}$$

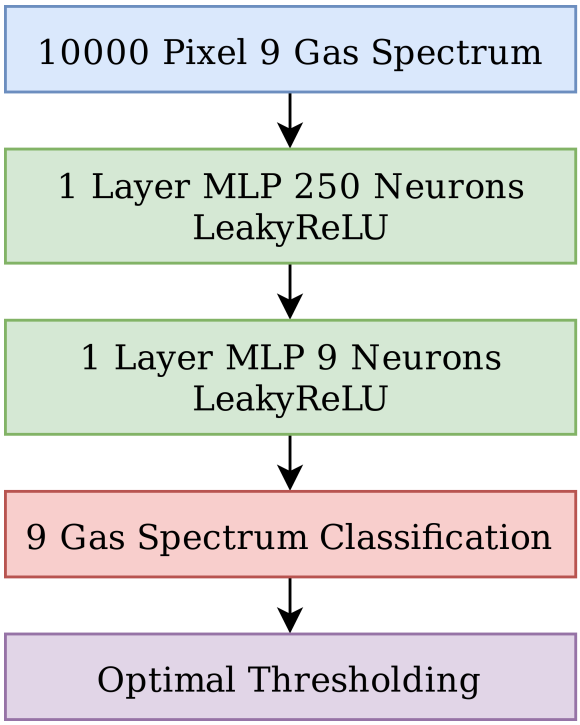

Figure 4.2: Structure of the MLP classifier.

where x is the input matrix and α controls the negative slope. The second MLP layer has 9 neurons and also uses the LeakyReLU activation function. It formats the output of the network. If the model predicts the presence of a gas, then the model outputs $\hat{y} \approx 1$, otherwise the model outputs $\hat{y} \approx 0$. The MLP is trained using the cross-entropy loss function $J(\hat{y}, y)$ with respect to the MLP's prediction $\hat{y}$ and the actual label y

$$S(x) = \frac{1}{1 + e^{-x}} \tag{4.11}$$

$$J(\hat{y}, y) = -\log(S(\hat{y}))(y) - \log(1 - S(\hat{y}))(1 - y) \tag{4.12}$$

where $S(x)$ is the sigmoid function with respect to input matrix x.

Optimal thresholding is applied in order to convert the MLP's continuous values to binary values. In the cross validation set, the threshold function $thres(x)$ is applied to each individual gas.

$$thres(x) = \begin{cases} 1.0 & \text{if } x > t \\ 0.0 & \text{otherwise} \end{cases} \tag{4.13}$$

The individual threshold t is selected such that F_1 score is maximised for each input prediction x. The threshold t is determined using a simple grid search. After finding

the optimal threshold t using the cross validation set, the same threshold t is applied to testing set.

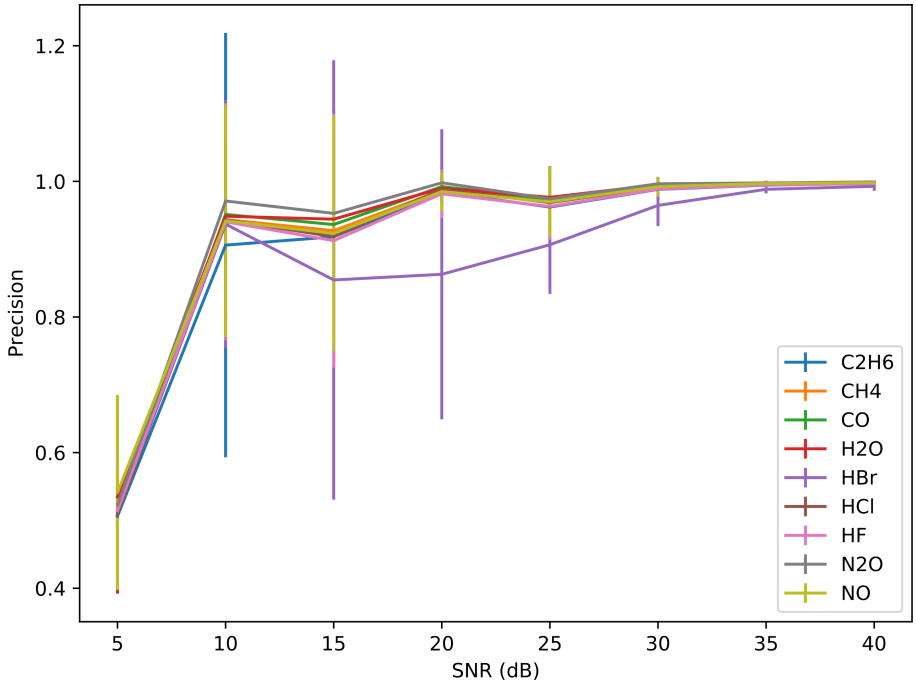

Figure 4.3: Precision of the MLP classification with 2σ error bar.

Fig. 4.3-4.6 show the performances of the MLP classifier. As the SNR increases from 5 dB to 40 dB, the micro averaged F_1 score and the micro averaged precision increases. This is due to the model being able to predict the presences of the gasses better as the noises are decreasing. Furthermore, all of the individual gasses' F_1 score are increasing except for the HBr gas. In the lower SNRs, the MLP struggles to detect HBr because it has the smallest absorbance spectra and it is overpowered by the noise floor. In the higher SNRs, the MLP does detect HBr relatively well but it is still overshadowed by the other gasses.

4.2.2 Multilayer Perceptron Spectra Quantification

The MLP quantifier is constructed from the MLP classifier, of which the optimal thresholding is removed. Furthermore, the MLP quantifier predicts the gas concentrations using the simulated spectra instead of the presences of the gasses. The model is also trained using the RMSE loss function shown in Eqn. 4.8 instead of the cross-entropy loss function.

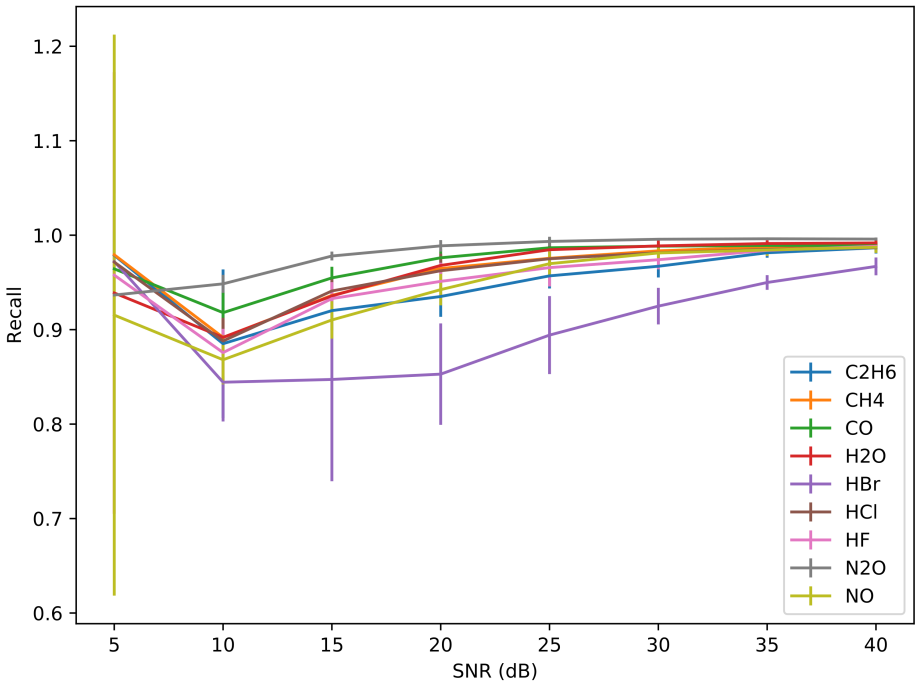

Figure 4.4: Recall of the MLP classification with 2σ error bar.

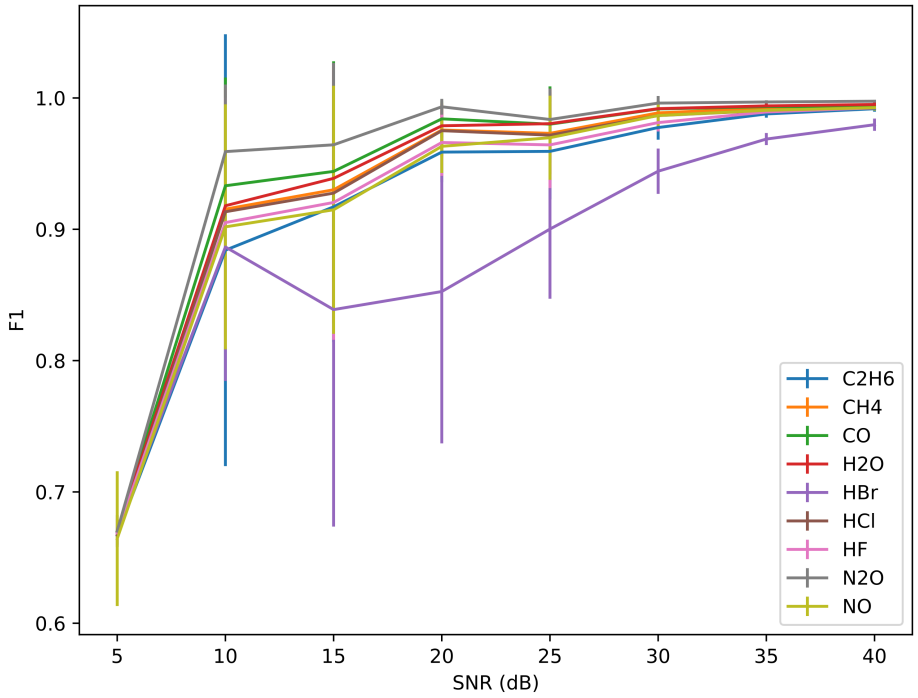

Figure 4.5: F_1 score of the MLP classification with 2σ error bar.

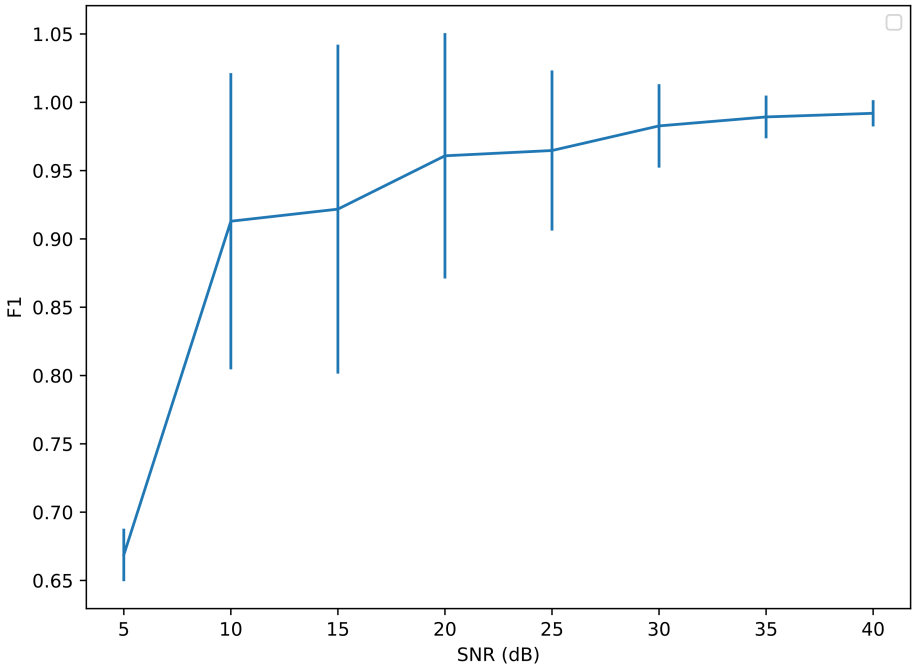

Figure 4.6: Micro averaged F_1 score of the MLP classification with 2σ error bar.

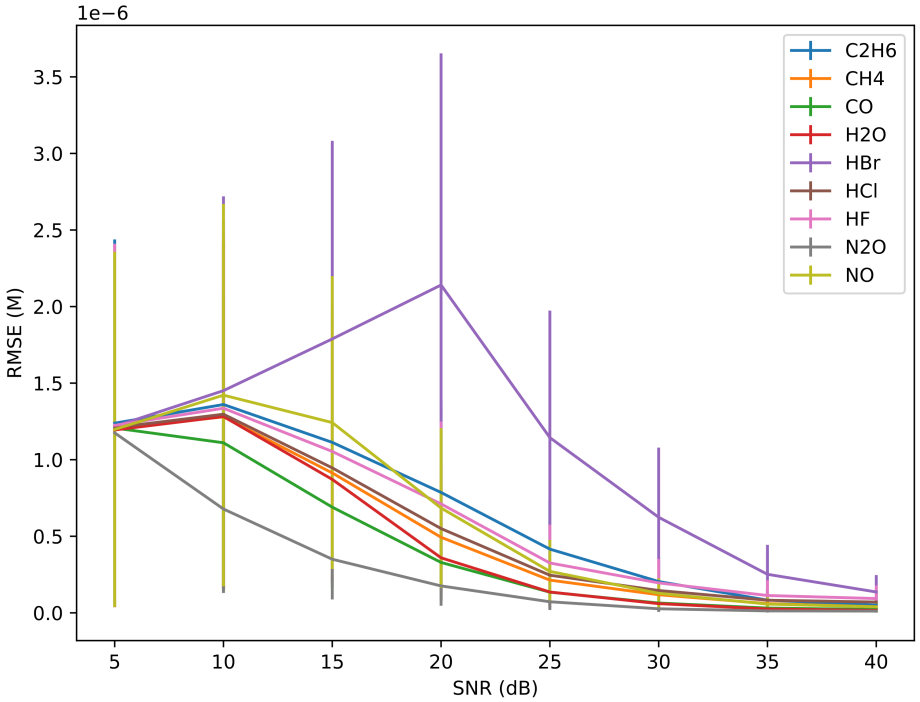

Figure 4.7: RMSE of the MLP quantification with 2σ error bar.

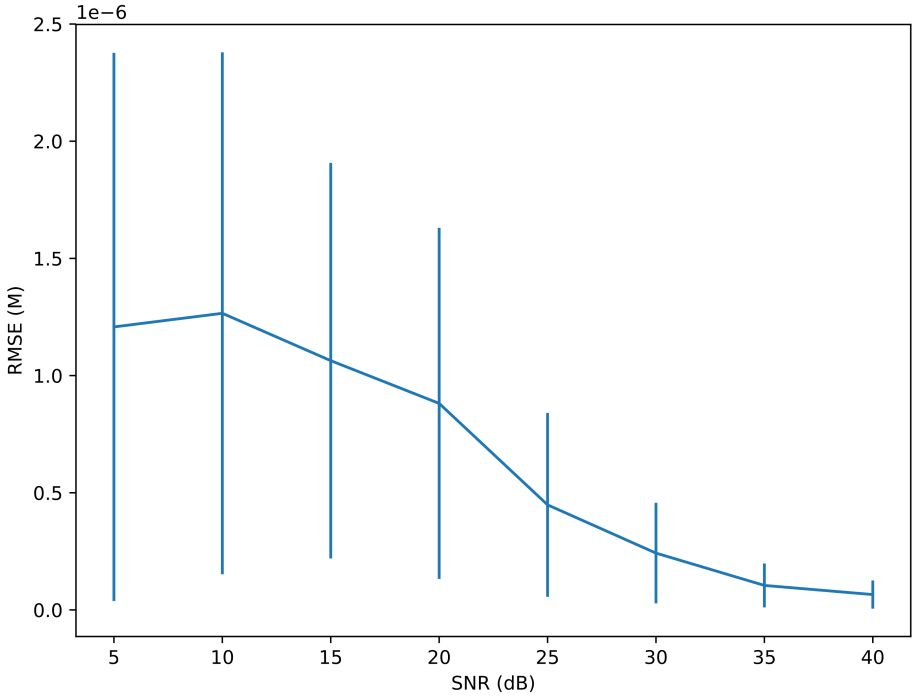

Figure 4.8: Micro averaged RMSE of the MLP quantification with 2σ error bar.

Fig. 4.7 and Fig. 4.8 show the performances of the MLP quantifier. As the SNR increases, the RMSE decreases. This is caused by the MLP being able to detect the gas concentrations more accurately at lower noise levels. The standard deviation of the RMSE decreases as the SNR increases, which means the MLP predicts the gas concentration with higher certainty. HBr has a higher RMSE than the other gasses because it has the smallest absorbance spectra.

4.2.3 Convolutional Neural Network Spectra Classification

Fig. 4.9 shows the structure of the CNN classifier, which classifies the presences of the gasses using the simulated 10,000 pixel spectra. The CNN uses the standard CNN equation [93]

$$V^i_{y,z} = \sum_{j=1}^{N}\sum_{k=1}^{M} W^i_{j,k} X^i_{j+y-1,k+z-1} \tag{4.14}$$

$$O^i = A(V^i + B^i) \tag{4.15}$$

where X is the input matrix, W is the CNN kernel, B is the bias matrix, $A(M)$ is the activation function with respect to matrix M, and O is the output matrix. The CNN

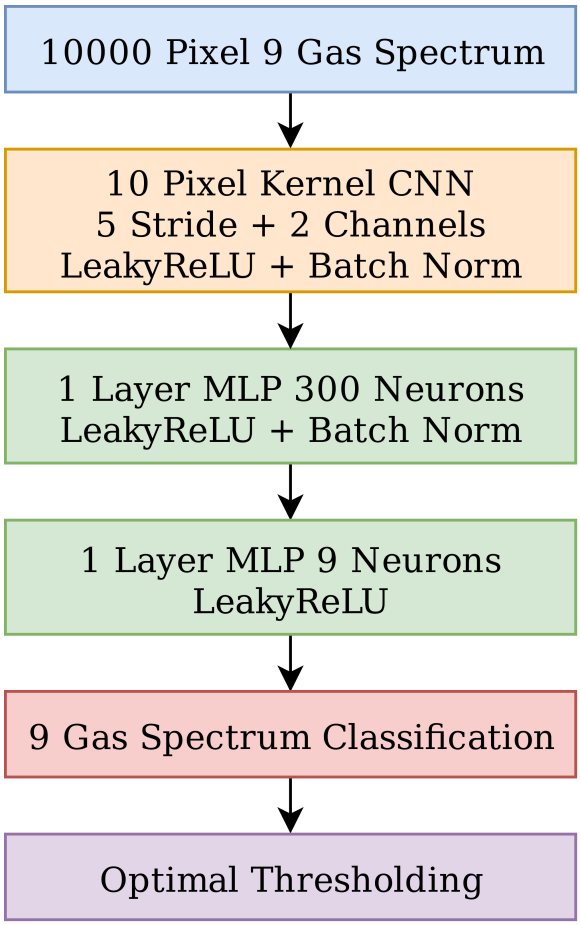

Figure 4.9: Structure of the CNN classifier.

layer uses the 10 pixel kernel with a stride of 5 pixels, which means the kernel moves in steps of 5 pixels. Furthermore, the CNN layer uses the LeakyReLU activation function and has 2 channels to detect 2 different signals. The batch normalization function

$$BN(x) = \frac{x - \mu_x}{\sigma_x} \tag{4.16}$$

is used to accelerate the training phase, where x is the input matrix. μ_x is the mean of x and σ_x is the standard deviation of x. The outputs from the CNN layer are flattened and are sent to the MLP layers. The first MLP layer uses 300 neurons and the LeakyReLU activation function. The second MLP layer uses 9 neurons and the LeakyReLU activation function, of which it predicts the presences of the gasses. The CNN classifier is trained using the cross entropy loss function shown in Eqn. 4.12. The optimal thresholding follows the same procedure as described in the MLP

classifier.

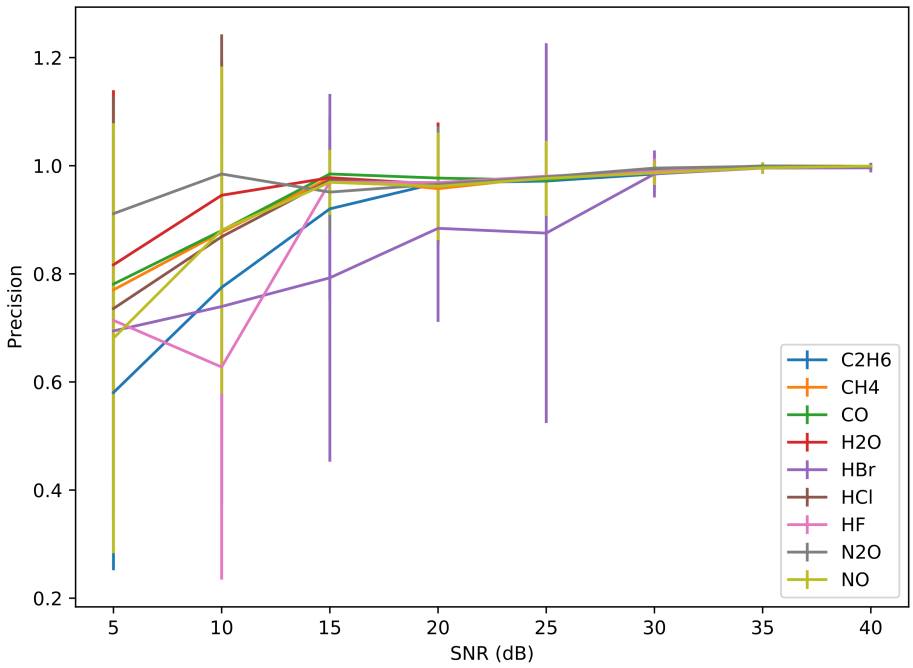

Figure 4.10: Precision of the CNN classification with 2σ error bar.

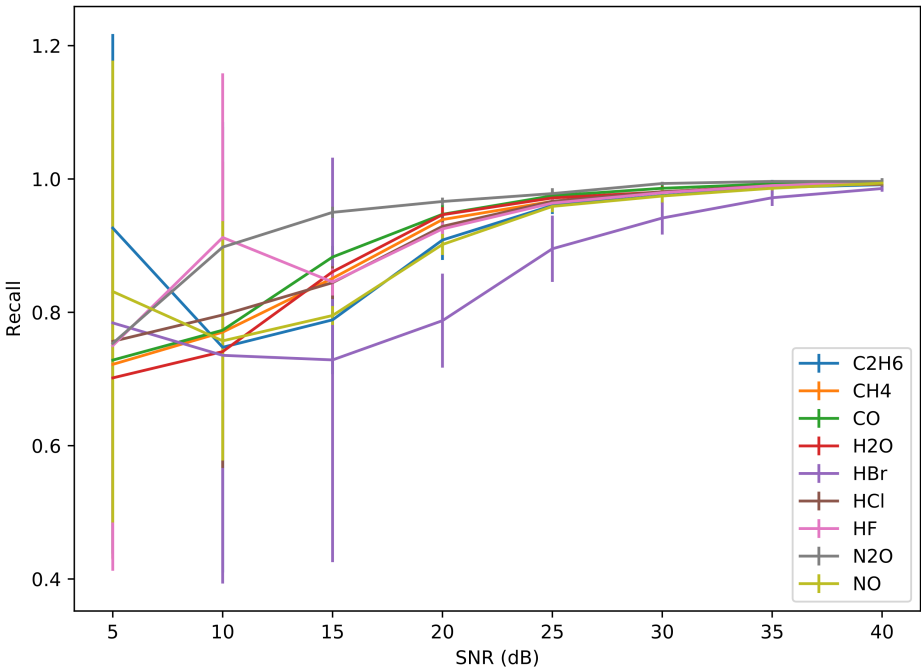

Figure 4.11: Recall of the CNN classification with 2σ error bar.

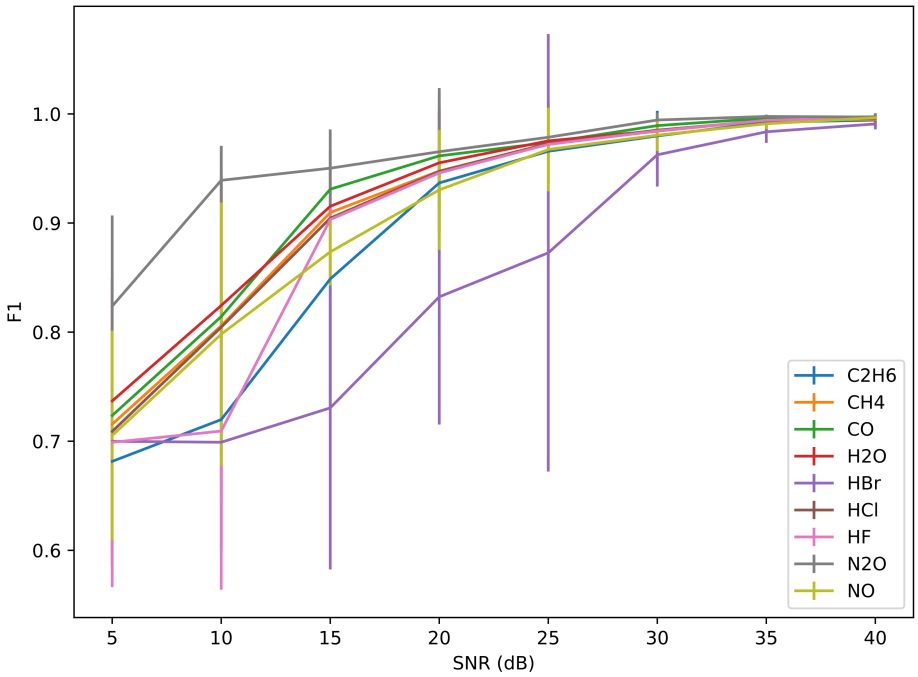

Figure 4.12: F_1 score of the CNN classification with 2σ error bar.

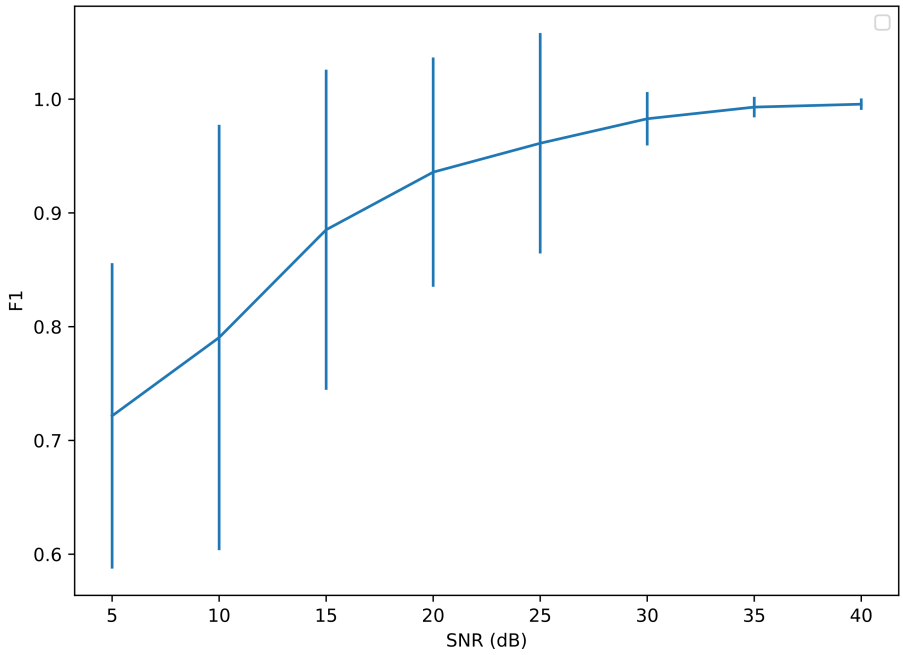

Figure 4.13: Micro averaged F_1 score of the CNN classification with 2σ error bar.

Fig. 4.10-4.13 show the performances of the CNN classifier, of which the CNN classifier performs similar to the MLP classifier. As the SNR increases, the F_1 score

increases due to the CNN being able to the detect the presences of the gasses better. For the 5 dB SNR, the CNN performs considerably better than the MLP due to the CNN having superior noise filtering capabilities over the MLP.

4.2.4 Convolutional Neural Network Spectra Quantification

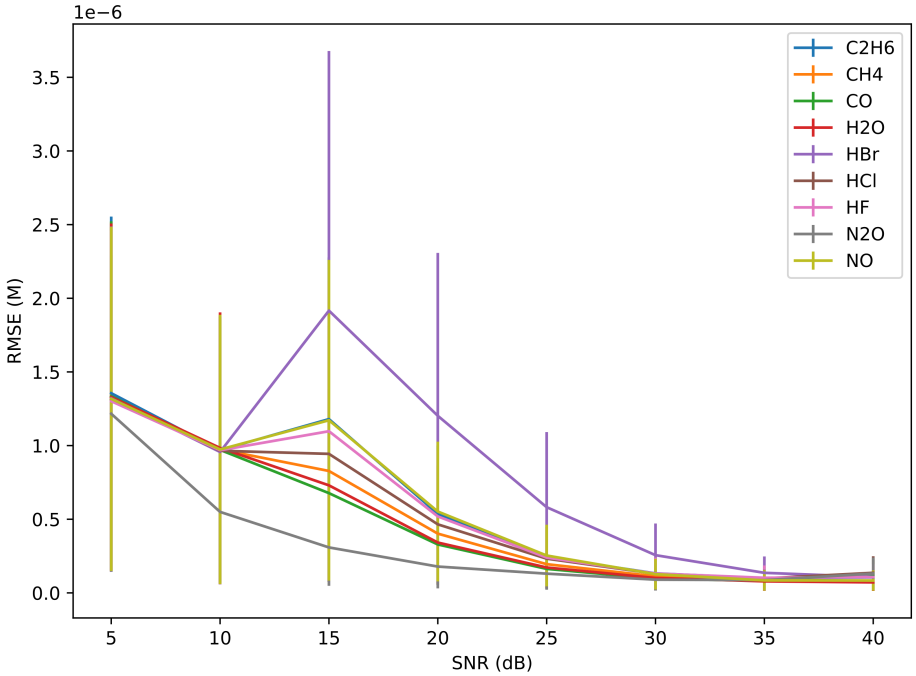

Figure 4.14: RMSE of the CNN quantification with 2σ error bar.

The CNN quantifier is constructed from the CNN classifier, of which the optimal thresholding is removed. The CNN quantifier predicts the concentrations of the gasses using the simulated 9 gas spectra. Fig. 4.14 and Fig. 4.15 show the performances of the CNN quantifier. For the SNRs ranging from 5 dB to 25 dB, the CNN performs better than the MLP because the CNN has a lower RMSE than the MLP. For the SNRs ranging from 30 dB to 40 dB, the MLP performs the same as the CNN because both RMSEs are equivalent.

4.3 Proposed Generative Adversarial Network

In this paper, we propose a GAN for the synthesis of the 9 gas spectra. The GAN is trained using the simulated spectra from HITRAN. Fig. 4.16 and Fig. 4.17 show

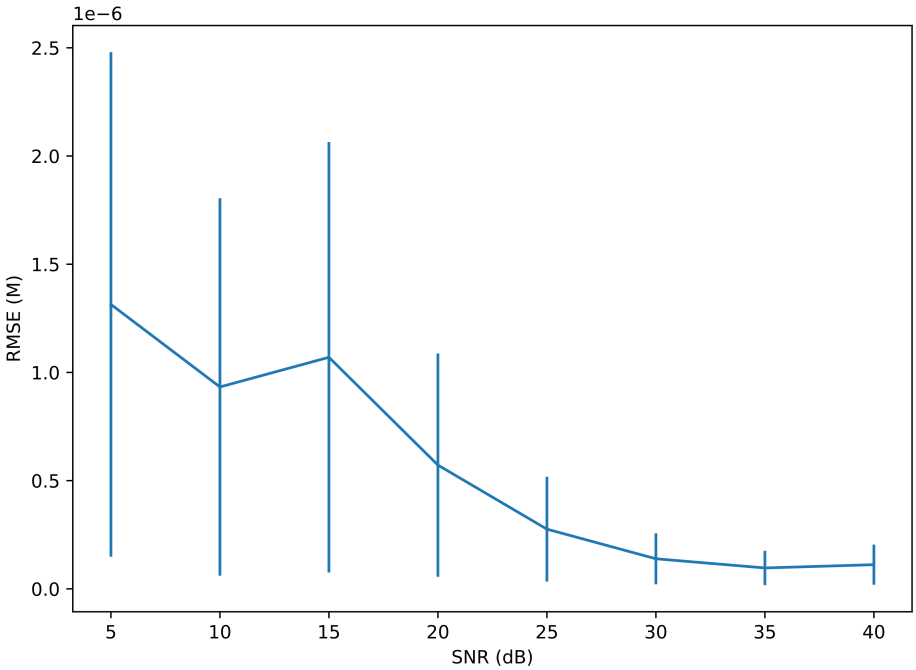

Figure 4.15: Micro averaged RMSE of the CNN quantification with 2σ error bar.

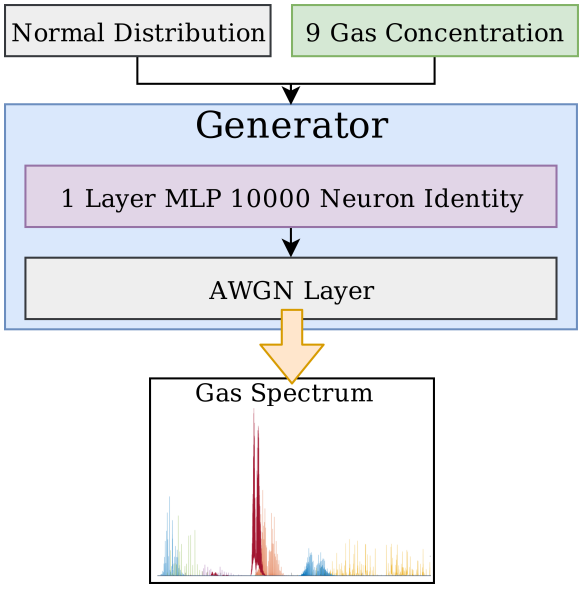

Figure 4.16: Structure of the GAN generator.

the generator and the discriminator of the GAN respectively. The generator takes in the 9 gas concentrations and the normal distribution in order to create the synthetic spectra. The synthetic spectra contains the superposition of the individual gas spectra as well as the simulated noise. The discriminator differentiates between the

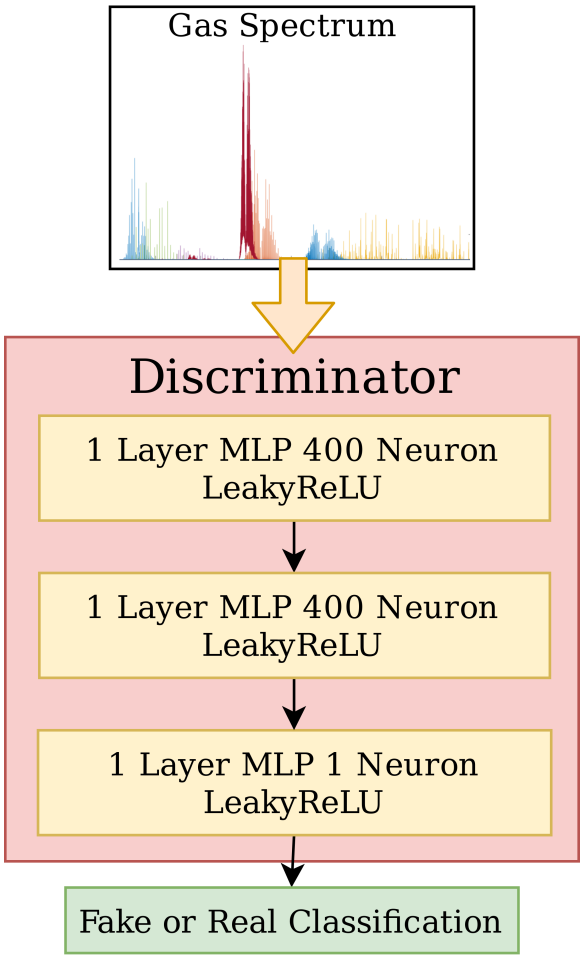

Figure 4.17: Structure of the GAN discriminator.

fake spectra produced by the generator and the actual spectra from HITRAN. This essentially forces the generator to improve upon the realism of synthetic spectra. The generator and the discriminator are trained in a zero-sum game, where both of them compete against each other.

4.3.1 Hyper-parameter Tuning

Table 4.1 shows the hyperparameter tuning of the generator. Firstly, various activation functions are tested. The identity function is selected because it has the lowest RMSE. This makes sense as the spectra is a superposition of individual spectra. Secondly, the optimal number of layers is found to be 1 layer. There are zero nonlinear effects in the spectra. Therefore, only 1 layer is needed in order to generate the linear spectra. Thirdly, the bias matrix is not needed because the RMSE does not change

Table 4.1: Generator Hyper-parameter Tuning.

RMSE	Activation Function	Layers	Hidden Neurons	Bias
1.54159	ReLU	3	200	TRUE
0.86128	Tanh	3	200	TRUE
0.05729	Identity	3	200	TRUE
0.04121	LeakyReLU	3	200	TRUE
0.04121	LeakyReLU	3	200	TRUE
0.02376	LeakyReLU	2	100	TRUE
0.01961	LeakyReLU	1	0	TRUE
0.01961	LeakyReLU	1	0	TRUE
0.01735	Identity	1	0	TRUE
0.01735	Identity	1	0	FALSE

Note: Cross validation 9 gas spectra 30 dB SNR.

when it is removed. Fourthly, the output layer has 10,000 neurons because it needs to generate 10,000 pixel spectra.

Table 4.2: Discriminator Hyper-parameter Tuning.

Cross Entropy	Activation Function	Layers	Hidden Neurons
1.5423	LeakyReLU	3	500
1.5752	ReLU	3	500
1.5509	Sigmoid	3	500
1.9562	Identity	3	500
1.5652	LeakyReLU	4	750
1.5423	LeakyReLU	3	500
1.7481	LeakyReLU	2	250
1.5423	LeakyReLU	3	500
1.4529	LeakyReLU	3	600
1.4032	LeakyReLU	3	700
1.3763	LeakyReLU	3	800

Note: Cross validation 9 gas spectra 30 dB SNR.

Table 4.2 shows the hyperparameter tuning of the discriminator. Firstly, the activation functions are varied. LeakyReLU is selected as it has the lowest cross entropy error. Secondly, the number of layers is changed. The optimal number of layers is found to be 3 because it has the lowest cross entropy error. Thirdly, the

number of hidden neurons is varied until the optimal amount is found to be 800.

4.3.2 Generator Description

The generator takes in the 9 gas concentrations and produces clean spectra using the 1 layer MLP. The 1 layer MLP has the identity activation function and does not use any bias matrix. After that, the AWGN layer adds noise to the clean spectra using the input normal distribution. During training, only the standard deviation σ parameter of the AWGN layer is changed. This is done to mimic the noises in the experimental measurements. In the end, realistic spectra are generated.

4.3.3 Discriminator Description

The purpose of the discriminator is to differentiate between the fake spectra synthesized by the generator and the real spectra from HITRAN. The discriminator uses 2 MLP layers, each having 400 neurons. Moreover, the MLP layers use the LeakyReLU activation function. The final MLP layer executes the fake or real classification. If the discriminator predicts the spectra is real, then it outputs $\hat{y} \approx 1$, otherwise it outputs $\hat{y} \approx 0$.

4.3.4 Training Process

The GAN is trained in 3 phases. In the first phase, the generator's weights are varied, while the discriminator's weights are held constant. The weights in the AWGN layer are also held constant. The RMSE loss function is used to train the generator

$$RMSE_{generator}(\hat{A}, A) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(A_i - \hat{A}_i)^2} \tag{4.17}$$

where A is the actual absorbance spectra and $\hat{A}$ is the predicted absorbance spectra.

In the second phase, the discriminator's weights are varied, while the generator's weights are held constant. The discriminator is trained using the cross entropy loss function

$$J_{discr}(\hat{y}, y) = -\log(S(\hat{y}))(y) - \log(1 - S(\hat{y}))(1 - y) \tag{4.18}$$

where y are the actual labels and $\hat{y}$ are the predicted labels.

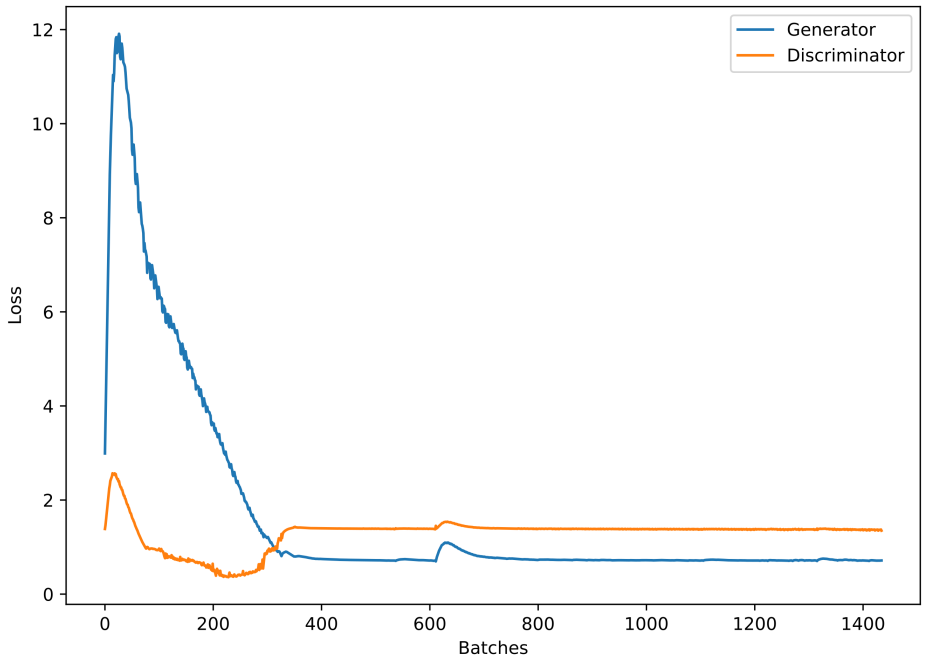

Figure 4.18: The generator's training curve and the discriminator's training curve. 9 gas spectra 30 dB SNR.

For the final phase, discriminator's weights and the generator's weights are both varied at the same time. Both networks are trained using

$$J_{discr}(\hat{y}, y) = -\log(S(\hat{y}))(y) - \log(1 - S(\hat{y}))(1 - y) \tag{4.19}$$

$$J_{gener}(\hat{y}, y) = -\log(S(\hat{y}))(1 - y) - \log(1 - S(\hat{y}))(y) \tag{4.20}$$

opposing objective functions. Fig. 4.18 shows the generator and discriminator in a zero sum game. At first, the discriminator outperforms the generator due to the discriminator having prior knowledge of the generator. After a few batches, the generator gets better at producing spectra. Subsequently, the generator's loss decreases, while the discriminator's loss increases. Then the generator outperforms the discriminator and the two networks stabilize in a Nash equilibrium. At the end, the generator is able to produce realistic spectra.

4.3.5 Learning Curve

Fig. 4.19 shows the MLP quantifier's learning curve. The MLP quantifier reaches a RMSE $\approx 1.4 \times 10^{-7}$ M at 100,000 actual spectra samples. On the other hand, the

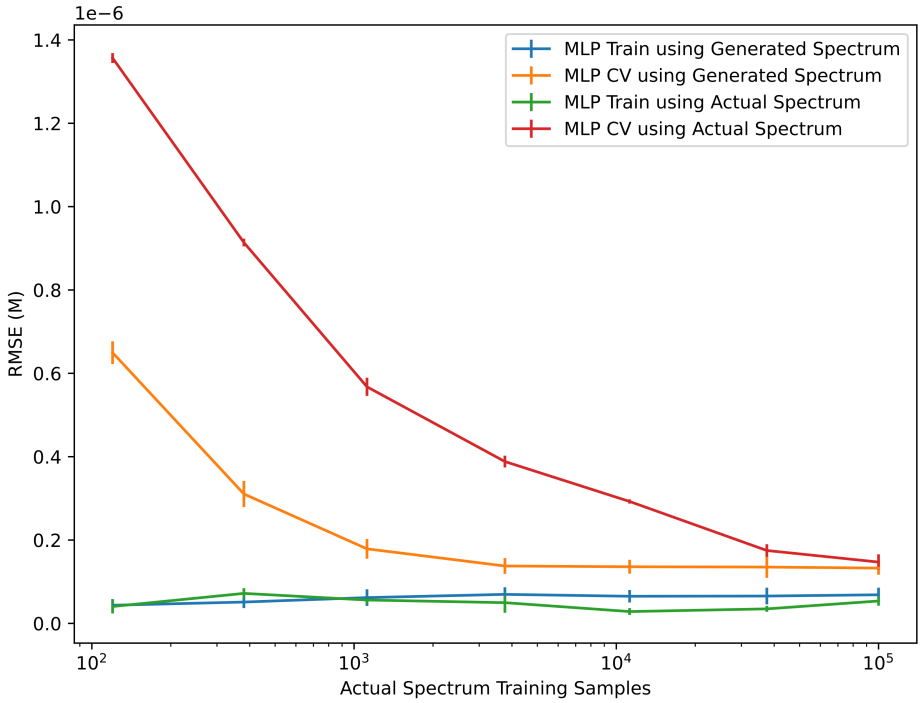

Figure 4.19: MLP quantifier's learning curve using the generated spectra and the actual spectra. 30 dB SNR 1x10 fold learning curve.

GAN is trained using 3760 actual spectra samples and generates 125,000 synthetic spectra samples. Upon training the MLP quantifier with 125,000 synthetic spectra samples, it reaches a RMSE $\approx 1.4 \times 10^{-7}$ M. This proves the GAN allows the MLP quantifier to converge with a fewer number of actual samples.

Fig. 4.20 shows the learning curve for PLS quantification. The PLS quantifier requires 100,000 actual spectra samples for it to converge and to get a RMSE $\approx 2 \times 10^{-7}$ M. On the other hand, the GAN is trained using 120 actual spectra samples and generates 125,000 synthetic spectra samples. Upon training the PLS quantifier with 125000 synthetic spectra samples, it converges to RMSE $\approx 2 \times 10^{-7}$ M. This proves the GAN generates very realistic synthetic spetrums that are indistinguishable from the actual spectra. The actual spectra needed by the PLS quantifier could be substituted for generated spectra. As a result, the PLS quantifier converges faster when using the generated spectra.

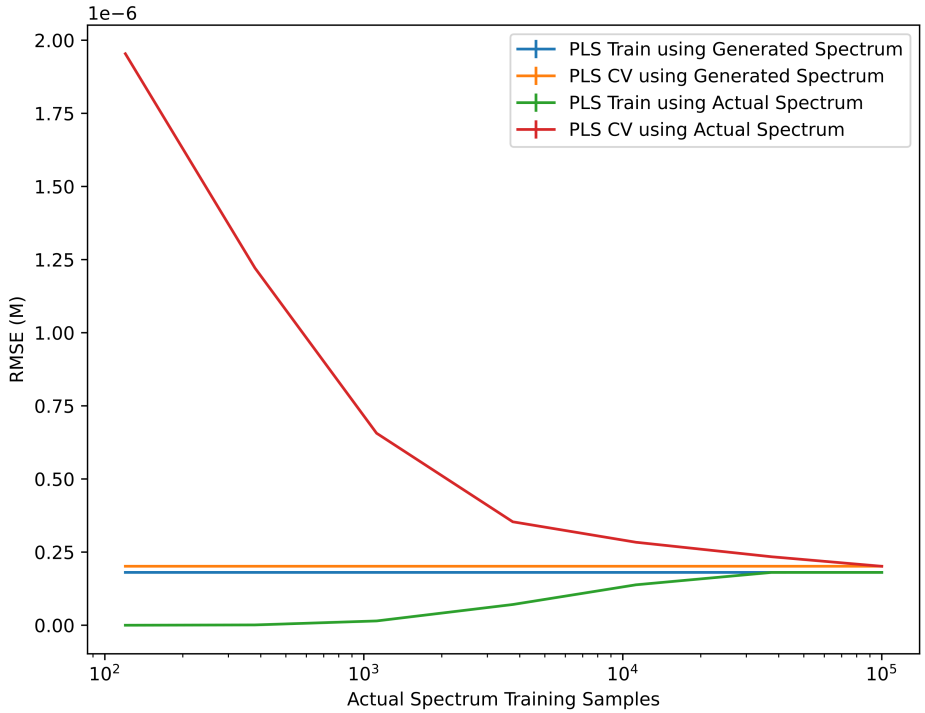

Figure 4.20: PLS quantifier's learning curve using the generated spectra and the actual spectra. 30 dB SNR 1x10 fold learning curve.

4.3.6 Verification

Fig. 4.21 and Fig. 4.22 compare the generated spectra to the actual spectra. The GAN recreates the peaks of spectra with high accuracy because the peaks of the spectra are placed at the correct positions. Moreover, the amplitudes of the generated peaks match the amplitudes of the actual peaks. However, the noise levels of the generated spectra are lower than the actual spectra. This is due to the GAN underestimating the noise level on the training samples.

4.4 Conclusion

IR spectra of various gas mixtures are created using the HITRAN database. For each gas mixture, the individual spectra's absorbance is proportional to the gas's concentration. MLP and CNN classifiers are developed to detect the presences of gases. As shown in Fig. 4.6 and in Fig. 4.13, the MLP classifier performs similarly to the CNN classifier. Moreover, the classifiers achieves good performance when $SNR \geq 20$ dB because the $F_1 \geq 0.9$. MLP and CNN quantifiers are created to detect

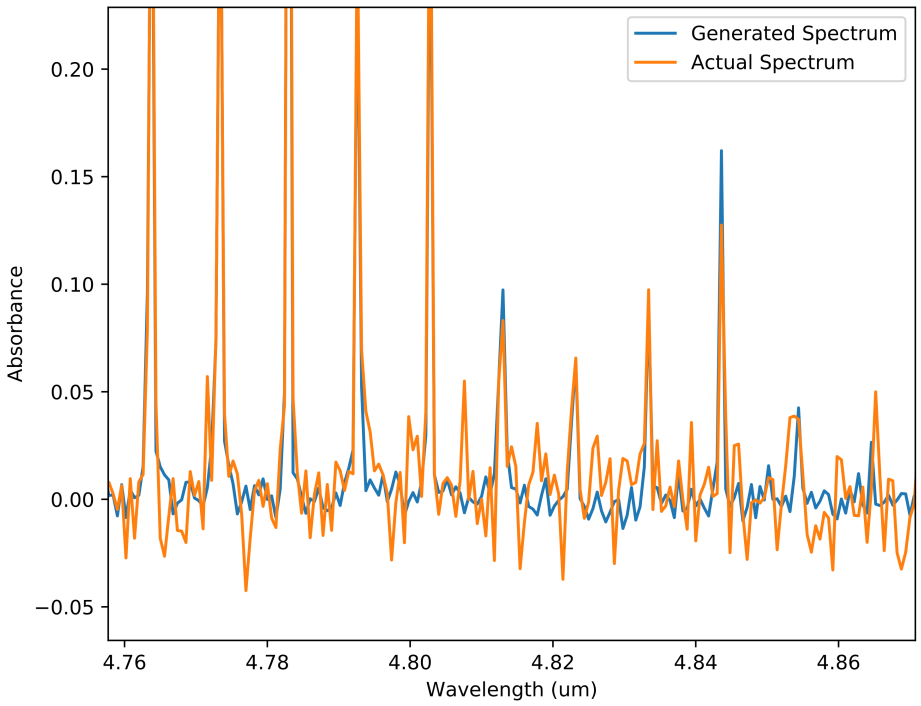

Figure 4.21: Generated spectra and the actual spectra.

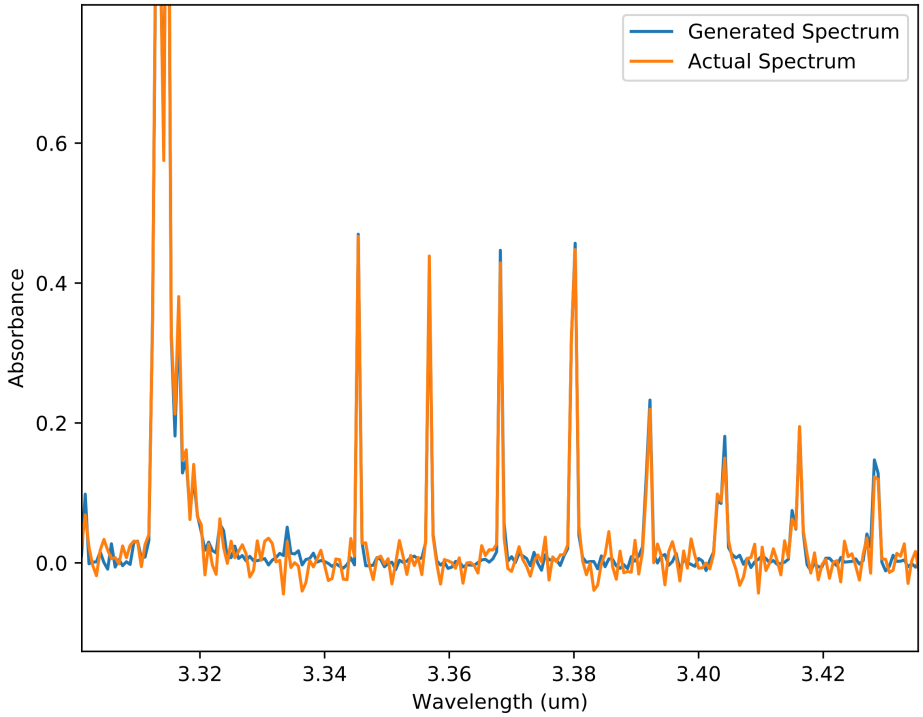

Figure 4.22: Generated spectra and the actual spectra.

the concentrations of the gasses. Fig. 4.8 and Fig. 4.15 display the quantification performances of the MLP and CNN. Both methods yield a RMSE $\approx 1.4 \times 10^{-7}$ when the $SNR \geq 30$ dB. Fig. 4.19 and Fig. 4.20 show the learning curves for the generated spectra and the synthetic spectra. Using the actual spectra samples, the MLP and PLS quantifiers converge at 100,000 actual samples. On the other hand, the GAN takes in the actual spectra samples and generates large amounts of synthetic spectra samples. In the synthetic case, the MLP quantifier converges at 3760 actual samples, while the PLS quantifier converges at 120 actual samples. As a result, the MLP and PLS quantifiers converge at a faster rate and with fewer actual spectra samples, when using the GAN.

Chapter 5

Hardware/Software Codesign for Training/Testing Neural Networks on Multiple Field Programmable Gate Arrays

Neural networks excel at a variety of tasks. For example, speech recognition, noise filtering, and text prediction are easily solved using neural networks. However, there are many downsides to neural networks. Neural networks require large training datasets and testing datasets. Moreover, neural networks require very powerful processors to execute the matrix operations. As result, the processors' computational powers limit the sizes and speeds of the neural networks.

Central processing units (CPUs) are an obvious choice to train and to test neural networks. CPUs are general purpose processors that can execute any task. In spite of the CPUs' flexibility, CPUs are very inefficient at computing neural networks because the CPUs are not optimized for matrix operations. On the other hand, co-processors such as Nvidia Tesla [121] and Intel Xeon Phi [122] are better at processing neural networks, when compared to CPUs. The co-processors employ a large array of specialized processors, of which allow matrix operations to be massively speed up. Despite co-processors being very efficient, they have memory bandwidth limitations. Most co-processors use PCIe 3.0 x16 to retrieve data, which is limited to 16 GB/s. This bandwidth limitation bottlenecks the neural networks' computations. Furthermore, co-processors require a CPU for control, which adds costs and latencies.

FPGAs are a possible solution to the problems presented above. FPGAs have better memory bandwidth/cost ratios, when compared to co-processors. Moreover, FPGAs do not require a CPU for control. The FPGAs' flexibility allows the FPGAs to adapt to different types of neural networks. Therefore, FPGAs are a cost efficient solution for processing neural networks. The literature contains numerous examples of FPGA frameworks optimized towards neural networks. The paper, "SpWA: an efficient sparse winograd convolutional neural networks accelerator on FPGAs" [123], shows a CNN implemented in Vivado HLS. Another paper, "Runtime Programmable and Memory Bandwidth Optimized FPGA-Based Coprocessor for Deep Convolutional Neural Network" [124], proposes a re-programmable DCNN accelerator using FSM based processors. The paper [124] also uses advanced caching techniques to minimize the load times of the data. A similar paper, "Hardware/Software Codesign for Convolutional Neural Networks Exploiting Dynamic Partial Reconfiguration on PYNQ" [125], shows the codesign of a CNN on the Xilinx ZYNQ. In the paper [125], the ARM cores load data from the RAMs, while the FPGA executes the CNN's matrix operations.

This paper proposes an FPGA solution to the problems above. The solution consists of an assembler and a VHDL design. The assembler takes in neural network assembly codes and produces microcodes. The microcodes are flashed onto a cluster of FPGAs. The cluster of FPGAs executes multiple neural networks in parallel, which accelerates the training and testing phases. Furthermore, the cluster of FPGAs allows for a greater memory bandwidth. The cluster of FPGAs overcomes the memory bandwidth limitations of individual FPGAs.

5.0.1 Multi-Layer Perceptions

Let X_i = data input vector of layer i
Let W_i = weight matrix of the layer i
Let B_i = bias vector of the layer i
Let $A(V)$ = activation function with respect to matrix V
Let O_i = layer output vector of layer i

$$O_i = A(W_i^T X_i + B_i) \tag{5.1}$$

$$ReLU(x) = max(0, x) \tag{5.2}$$

Multi-layer perceptions (MLPs) [26] are a type of neural network. MLPs have an input layer, multiple hidden layers, and an output layer. The input data enters a layer through the input vector X_i. Then the input vector X_i is multiplied by the weights W_i^T. After matrix multiplication, biases B_i are added to the $W_i^T X_i$. After that, the result passes through the activation function $A(V)$ and produces the layer's output O_i. There are many types of activation functions used in neural networks. For example, Eqn. 5.2 shows the ReLU activation function. The ReLU activation function sets all negative numbers to zero. Overall, the input data goes through many layers until the final result is produced at the output layer.

5.1 Design Overview and Requirements

The goal of the project is to accelerate multiple neural networks using multiple FPGAs. The targeted FPGA boards must use Xilinx's 7 Series FPGAs. All the FPGA boards must be identical. Moreover, the FPGA boards must have onboard flash, RAM, and system buses. Fig. 5.1 shows the neural network processor and assembler. The Matrix Assembler is a high level optimizing assembler, which parses the neural network assembly codes. The Matrix Assembler parses as many neural network assembly codes as the user wants. After parsing the assembly codes, the Matrix Assembler optimizes the assembly codes and neural network processors. Then the Matrix Assembler generates the VHDL codes and the microcodes. The VHDL codes contain the structure of the Matrix Machine. The Matrix Machine consists of multiple Mini Vector Machines. Each Mini Vector Machine computes a vector operation using a single DSP. The DSPs are set to process 16 bit signed integers. 16 bit precision is enough for most of the neural network applications. When the Mini Vector Machines are put together, the Mini Vector Machines perform matrix operations. The Mini Vector Machines allow the Matrix Machine to adapt to different sizes of matrices. Subsequently, the VHDL codes are synthesized into the bit-streams using the Xilinx's Vivado Design Suite. After generating the bit-streams, the bit-streams are flashed to the onboard flash. The onboard flash then loads the bit-stream onto the FPGA. The system buses transfer the neural network data and microcode from the control server to the onboard RAM. The onboard RAM acts as a buffer for the FPGA. The microcodes schedule the execution of the Matrix Machine by coordinating the individual Mini Vector Machines.

For the functional requirements, the Matrix Machine must train and test MLPs.

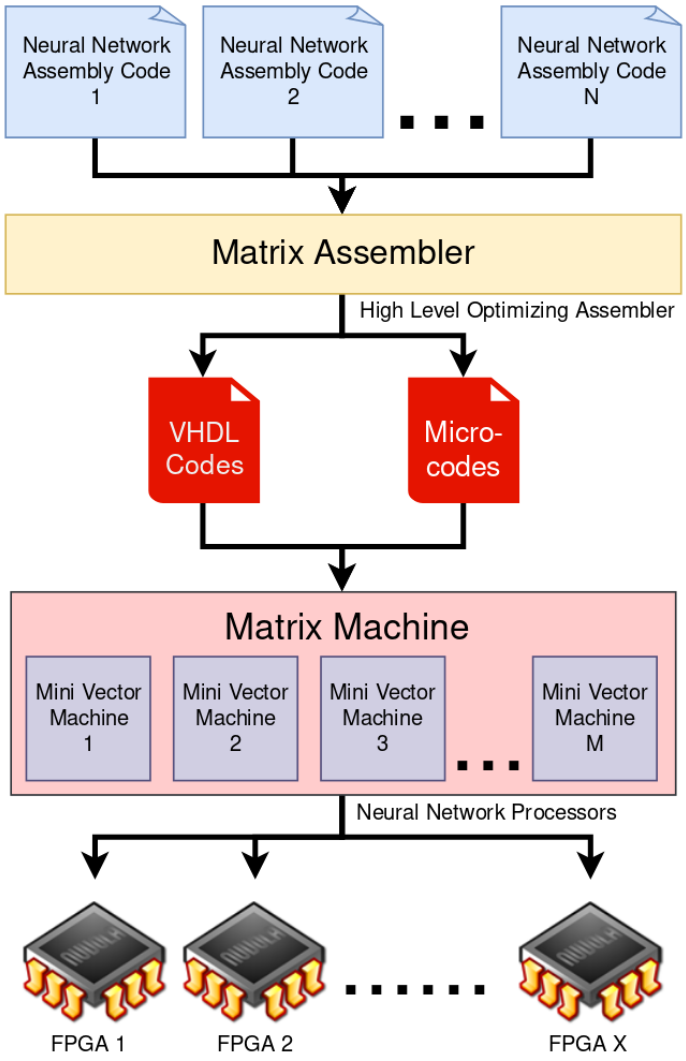

Figure 5.1: Overview of the neural network processor and assembler.

The Matrix Machine must calculate the forward passes of the MLPs. After calculating the forward passes, the loss functions' gradients must be calculated using the back-propagation algorithm. The gradients are then used to update the weights of the MLPs. In order to be flexible, the VHDL design must be generalized to run any type of MLP. Firstly, the Matrix Assembler must handle any number of MLPs regardless of the number of FPGAs. Secondly, the Matrix Machine must handle matrices of any size and shape. The input matrices, the weight matrices, and the bias matrices could be as big as the user wants. Thirdly, the Matrix Machine must be able to dynamically load different MLPs at runtime. In other terms, the Matrix Machine must be able to switch between different MLPs without regenerating the bit-stream. Fourthly, the

Matrix Machine must scale to any number of LUTs, BRAMs, and DSPs. If the Matrix Assembler detects the FPGA has a high number of DSPs, then the Matrix Assembler generates more Mini Vector Machines to take advantage of the DSPs. If the Matrix Assembler detects the FPGA has a low number of DSPs, then the Matrix Assembler reduces the number of Mini Vector Machines. Lastly, the Matrix Machine must scale to any number of FPGAs. If the number of MLPs is greater than the number of FPGAs, then the MLPs are processed sequentially. If the number of MLPs is less than the number of FPGAs, then the MLPs are divided and are processed in parallel. If the number of MLPs is equal the number of FPGAs, then the Matrix Assembler maps 1 MLP to 1 FPGA.

5.2 Matrix Assembler: High Level Optimizing Assembler

The Matrix Assembler takes in neural network assembly codes and produces instructions and VHDL codes. At runtime, the instructions are decoded into microcodes. The decoding is done to reduce the size of the instruction cache. Moreover, the Matrix Assembler controls the number of processor groups and the types of processors using the VHDL codes. As a result, the Matrix Assembler is able to optimize the VHDL codes for a specific FPGA.

5.2.1 Assembly Codes

Assembly	ARG0	ARG1	ARG2	ARG3	ARG4	Description
INPUT	OUTMAT	SIZEN	SIZEM	NONE	NONE	Loads an N X M data matrix
WEIGHT	OUTMAT	SIZEN	SIZEM	NONE	NONE	Loads an N X M weight matrix
BIAS	OUTVEC	SIZEN	NONE	NONE	NONE	Loads a bias vector with size N
ACT	OUTVEC	SIZEN	NONE	NONE	NONE	Loads an activation lookup table with size N
MLP	OUTMAT	INMAT	INMAT	INVEC	INVEC	Executes a MLP layer
OUTPUT	INMAT	NONE	NONE	NONE	NONE	Stores data matrix

Table 5.1: Neural network assembly codes.

Table 5.1 shows the neural network assembly codes. INPUT code specifies the input matrix to the neural network. WEIGHT, BIAS, ACT, and MLP codes define the structure of a single layer. The OUTPUT code controls the output matrix of the neural network.

5.2.2 Instruction Set Architecture

Instruction	Op code	Description
VECTOR_DOT_PRODUCT	000	Vector dot product
VECTOR_SUMMATION	001	Vector summation
VECTOR_ADDITION	010	Vector addition
VECTOR_SUBTRACTION	011	Vector subtraction
ELEMENT_MULTIPLICATION	100	Element wise multiplication
ACTIVATION_FUNCTION	101	Apply activation function to vectors
NOP	110	No operation

Table 5.2: Instruction set architecture.

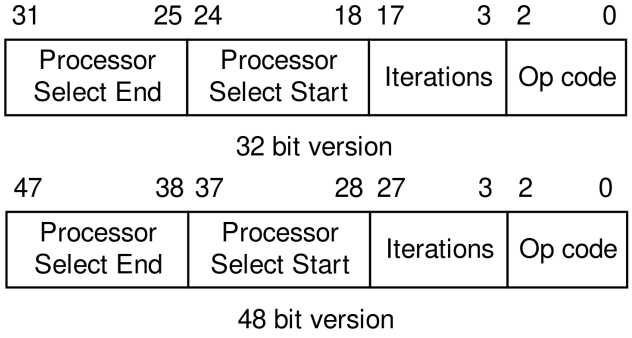

Figure 5.2: Instruction set architecture bit arrangement.

The Matrix Assembler translates the assembly codes to the instructions. Table 5.2 shows the list of instructions. Matrix multiplication is achieved by using multiple vector dot operations. Moreover, matrix addition is achieved using by multiple vector additions. Fig. 5.2 shows the bit arrangement for the instruction architecture. The operation code controls the type of operation, while the number of iterations controls the number of loops. Moreover, the operation code is applied to the processors designated by the processor select start and the processor select end. For the 32 bit version, the instructions only control a maximum of 128 processor groups. For the 48 bit version, the instructions only control a maximum of 1024 processor groups.

5.2.3 Microcode

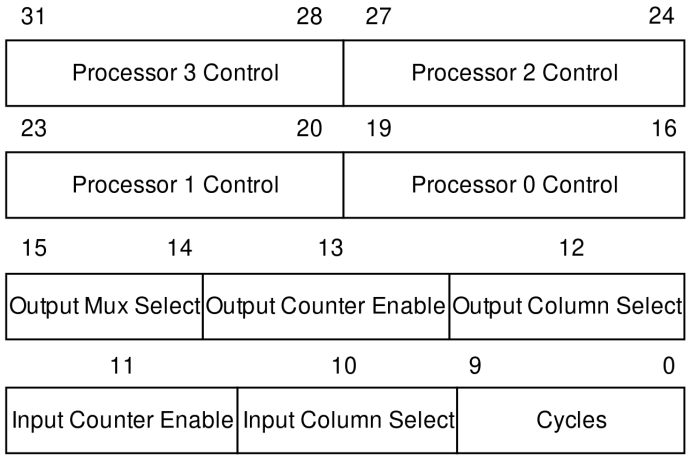

Figure 5.3: Microcode bit arrangement.

The Matrix Assembler also translates the instructions to microcode. Fig. 5.3 shows the 32 bit microcode. Each microcode controls 4 MVMs. The MVMs are arranged in groups of 4 because the 4:1 multiplexer is the most efficient multiplexer. The 4:1 multiplexer uses the least amount of LUTs and has the lowest latency. microcode(9..0) controls the number of cycles in a microcode. The number of cycles allows the Matrix Assembler to execute a given microcode for any length of time. microcode(10) controls the selection of the input columns. If input column 0 is selected, then the input data is written to column 0. If input column 1 is selected, then the input data is written to column 1. microcode(11) controls the activation of the input counter. If the input counter is enabled, then the input counter increments at every cycle. The input counter's value is feed into the input addresses of the individual MVMs. microcode(12) controls the selection of the output columns. microcode(13) controls the activation of the output counter. microcode(15..14) controls the selection of the output 4:1 multiplexer. The output 4:1 multiplexer controls the output of the processor group. microcode(31..16) contains 4 processor control signals. Each processor control signal is mapped to a MVM input processor control signal.

5.2.4 Resource Allocation

Let N_{DDR} = the number of 32 bit DDR RAM channels

Let CLK_{DDR} = the DDR RAM clock in MHz

Component	LUTs	FFs	RAMB18Ks	DSPs
MVM_PG	495	1642	8	4
ACTPRO_PG	447	1406	12	0

Table 5.3: Processor group resource usages.

Let CLK_{FPGA} = the FPGA clock in MHz
Let LUT_{FPGA} = the number of leftover DSPs on the FPGA
Let LUT_{ACTPRO_PG} = the number of DSPs used by the ACTPRO_PG
Let FF_{FPGA} = the number of leftover FFs on the FPGA
Let FF_{ACTPRO_PG} = the number of FFs used by the ACTPRO_PG
Let $BRAM_{FPGA}$ = the number of leftover block RAMs on the FPGA
Let $BRAM_{ACTPRO\_PG}$ = the number of block RAMs used by the ACTPRO_PG
Let N_{MVM_PG} = the optimal number of Mini Vector Machine processor groups
Let N_{ACTPRO_PG} = the optimal number of Activation processor groups

$$N_{MVM\_PG} = \frac{N_{DDR} CLK_{DDR}}{CLK_{FPGA}} \tag{5.3}$$

$$N_{ACTPRO\_PG} = \min\left(\frac{LUT_{FPGA}}{LUT_{ACTPRO\_PG}}, \frac{FF_{FPGA}}{FF_{ACTPRO\_PG}}, \frac{BRAM_{FPGA}}{BRAM_{ACTPRO\_PG}}\right) \tag{5.4}$$

The Matrix Assembler determines the optimal number of processor groups in order to fully utilize the FPGA's resources. Eqn. 5.3 shows the equation for the optimal number of Mini Vector Machine processor groups N_{MVM_PG}. The number of Mini Vector Machine processor groups N_{MVM_PG} is only limited by the number of DDR RAM channels N_{DDR}. Furthermore, Table 5.3 shows the resource usages of each processor group. The optimal number of Activation processor groups N_{ACTPRO_PG} is calculated using Table. 5.3 and Eqn. 5.4.

5.3 Matrix Machine: Neural Network Processors

Fig. 5.4 shows the Matrix Machine. The Matrix Machine contains a global controller that coordinates multiple processor groups. The global controller first decodes the instructions into microcodes. Then the global controller writes microcodes and data to a circular FIFO. The FIFO's purpose is to distribute the microcodes and data to each processor group. The FIFO also collects outputs of each processor group. Moreover, the FIFO reduces the propagation delay of the signals. Each processor

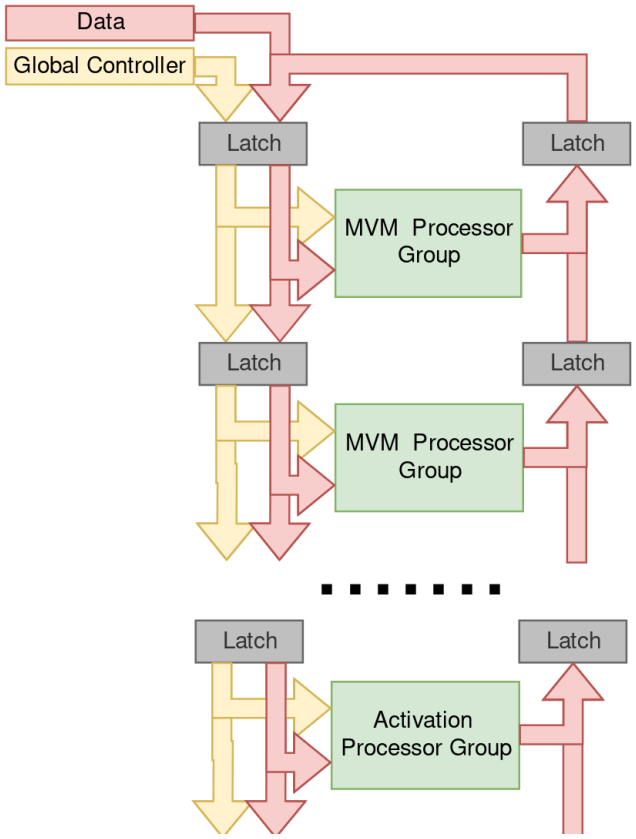

Figure 5.4: Matrix Machine.

group has a local controller, which receives microcodes from the global controller. The local controller's purpose is to cache the microcodes. The microcode cache reduces the number of load operations and minimizes the propagation delay. Moreover, each processor group consists of Mini Vector Machines (MVMs) or Activation Processors (ACTPROs). The Mini Vector Machines execute vector operations, while the Activation Processors execute activation functions.

5.3.1 Processor Groups

Table 5.4 shows the ports of the MVM processor group. The group control port starts and stops the executions of the processor group. Furthermore, the microcode input port is written to the microcode cache. The microcode cache is used to minimize the load penalties. After writing the microcodes, the microcodes are decoded and are sent to the individual processors. The microcode controls the counters, number of cycles, and the type of operation. Each processor group has two input data ports and

Signal	Direction	Description
CLK	IN	Clock
group_control(1..0)	IN	Control signal for execution
microcode(31..0)	IN	Microcode input
input_data0(15..0)	IN	Input data port 0
input_data1(15..0)	IN	Input data port 1
output_data0(15..0)	OUT	Output data port 0
output_data1(15..0)	OUT	Output data port 1

Table 5.4: Mini Vector Machine processor group ports.

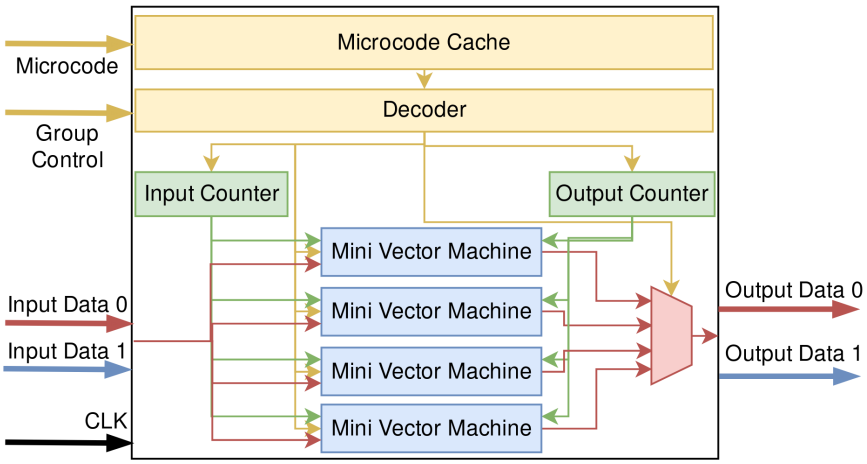

Figure 5.5: MVM processor group.

two output data ports. Each input port receives a 16 bit integer. The output ports transmit 16 bit integers.

The structure of the MVM processor group is presented in Fig. 5.5. The MVM processor group consists of 4 processors joined together by 1 x 4:1 multiplexer, 1 x microcode cache, and 1 x local controller. The processors are arranged in groups of 4 because the 4:1 multiplexer is the most efficient multiplexer. Each MVM processor group uses 495 LUTs, 1642 FFs, 4 x DSP48E1, and 8 x RAMB18Ks in total. The microcode cache stores 16 microcodes in total. The 8 bit input counter is used to select the input addresses of the MVMs. The input counter allows the MVMs to load the vectors column-wise. Column-wise vector loading enables the MVMs to cache the column vectors in order to minimize the load penalties. The 8 bit output counter is used to store vectors column-wise. The output counters are designed to mirror the input counters. The output multiplexer is used to select the outputs of the MVMs.

Let T_{cycle} = the period of a cycle in seconds

Let N_{bits} = the number of bits per element

Let N_e = the number of elements per processor

Let N_{proc} = the number of processors per group

Let N_I = the number of iterations

Let C_{STALL} = the number of stall cycles per iteration

Let C_{LOAD} = the number of load cycles per iteration

Let C_{RUN} = the number of run cycles per iteration

Let C_{STORE} = the number of store cycles per iteration

Let $T_{RUN}(N_I)$ = the total number of run cycles for a given number of iterations N_I

Let $T_{all}(N_I)$ = the total number of cycles for a given number of iterations N_I

Let $E(N_I)$ = the efficiency for a given number of iterations N_I

Let $P(N_I)$ = the processing rate in $\frac{elements}{s}$ for a given number of iterations N_I

Let $R(N_I)$ = the data throughput in Mb/s for a given number of iterations N_I

$$T_{RUN}(N_I) = N_{proc} \cdot N_I \cdot C_{RUN} \tag{5.5}$$

$$T_{all}(N_I) = N_{proc} \cdot ((N_I + N_{proc}^2 - 1) \cdot (C_{LOAD}) + N_I \cdot (C_{RUN} + C_{STORE} + C_{STALL})) \tag{5.6}$$

$$E(N_I) = \frac{T_{RUN}(N_I)}{T_{all}(N_I)} \tag{5.7}$$

$$P(N_I) = \frac{N_{proc}^2 \cdot N_I \cdot N_e}{T_{all}(N_I) \cdot T_{cycle}} \tag{5.8}$$

$$R(N_I) = P(N_I) \cdot N_{bits} \cdot 1 \times 10^{-6} \tag{5.9}$$

For vector addition and $N_I = 1024$ iterations, the total number of run cycles and total number of cycles are calculated below. Also, the efficiency and processing rate are calculated.

$T_{RUN}(1024) = 4 \cdot 1024 \cdot 519 = 2125824$

$T_{all}(1024) = 4 \cdot ((1024 + 4^2 - 1) \cdot (256) + (1024) \cdot (519 + 256 + 0)) = 4238336$

$E(1024) = \frac{T_{RUN}(1024)}{T_{all}(1024)} = \frac{2125824}{4238336} = 0.501$

$P(1024) = \frac{4^2 \cdot 1024 \cdot 1024}{4238336 \cdot 10 \times 10^{-9} s} = 3.95 \times 10^8 \frac{elements}{s}$

$R(1024) = 3.95 \times 10^8 \frac{elements}{s} \cdot 16bits \cdot 1 \times 10^{-6} = 6320 \frac{Mb}{s}$

For vector dot product and $N_I = 1024$ iterations, the total number of run cycles and total number of cycles are calculated below. Also, the efficiency and processing rate are calculated.

$T_{RUN}(1024) = 4 \cdot 1024 \cdot 519 = 2125824$

$T_{all}(1024) = 4 \cdot ((1024 + 4^2 - 1) \cdot (256) + (1024) \cdot (519 + 0 + 248) + 256) = 4206592$

$E(1024) = \frac{T_{RUN}(1024)}{T_{all}(1024)} = \frac{2125824}{4206592} = 0.505$

$P(1024) = \frac{4^2 \cdot 1024 \cdot 1024}{4206592 \cdot 10 \times 10^{-9} s} = 3.99 \times 10^8 \frac{elements}{s}$

$R(1024) = 3.99 \times 10^8 \frac{elements}{s} \cdot 16bits \cdot 1 \times 10^{-6} = 6384 \frac{Mb}{s}$

For the activation function and $N_I = 1024$ iterations, the total number of run cycles and total number of cycles are calculated below. Also, the efficiency and processing rate are calculated.

$T_{RUN}(1024) = 4 \cdot 1024 \cdot 517 = 2117632$

$T_{all}(1024) = 4 \cdot ((1024 + 4) \cdot (512) + (1024) \cdot (517 + 256 + 0)) = 5271552$

$E(1024) = \frac{T_{RUN}(1024)}{T_{all}(1024)} = \frac{2117632}{5271552} = 0.401$

$P(1024) = \frac{4^2 \cdot 1024 \cdot 1024}{5271552 \cdot 10 \times 10^{-9} s} = 3.18 \times 10^8 \frac{elements}{s}$

$R(1024) = 3.18 \times 10^8 \frac{elements}{s} \cdot 16bits \cdot 1 \times 10^{-6} = 5088 \frac{Mb}{s}$

The processor groups have high efficiency as the efficiency approaches 50% for vector operations. Moreover, each processor group processes elements at a rate of $> 5000 \frac{Mb}{s}$, which is $\frac{1}{5}$ the bandwidth of a 32 bit DDR2 RAM.

5.3.2 Mini Vector Machines

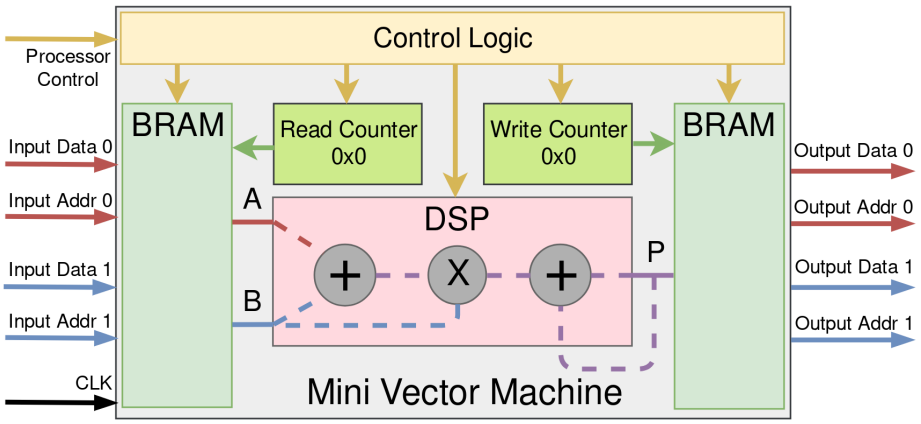

Figure 5.6: The structure of the Mini Vector Machine.

Signal	Direction	Description
CLK	IN	Clock
processor_control(2..0)	IN	Operation code
processor_control(3)	IN	Right BRAM MSB select
input_data0(15..0)	IN	Input data port 0
input_addr0(15..0)	IN	Input address port 0
input_data1(15..0)	IN	Input data port 1
input_addr1(15..0)	IN	Input address port 1
output_data0(15..0)	OUT	Output data port 0
output_addr0(15..0)	OUT	Output address port 0
output_data1(15..0)	OUT	Output data port 1
output_addr1(15..0)	OUT	Output address port 1

Table 5.5: Mini Vector Machine ports.

processor_control(2..0)	Operation name	Operation description
000	MVM_RESET	Reset all registers
001	MVM_READ	BRAM read
010	MVM_WRITE	BRAM write
011	MVM_VEC_DOT	Vector dot product using BRAM
100	MVM_VEC_SUM	Vector summation using BRAM
101	MVM_VEC_ADD	Vector addition using BRAM
110	MVM_VEC_SUB	Vector subtraction using BRAM
111	MVM_ELEM_MUTLI	Element wise multiplication

Table 5.6: Mini Vector Machine processor control.

The Mini Vector Machine's purpose is to execute vector operations. Tab. 5.5 shows the Mini Vector Machine's ports. The Mini Vector Machine uses clocks of 100MHz, 100MHz, 300MHz, and 500MHz for Spartan-7, Artix-7, Kintex-7, and Virtex-7 respectively. The processor control signal is shown in Tab. 5.6. The processor control signal allows the Mini Vector Machine to run vector dot product, vector summation, vector addition, and vector subtraction. Moreover, the processor control signal manages the BRAMs' reading and writing. The Mini Vector Machine has 2 input ports and 1 output port. The input ports have input data lines and input address lines. The input ports allow vectors to be written to the left BRAM. The output port has a output data line and a output address line. The output port allows vectors to be read from the right BRAM.

Fig. 5.6 shows the structure of the Mini Vector Machine. The Mini Vector Machine

consists of 1 x DSP48E1, 2 x BRAM, 2 x counter, and control logic. The control logic requires 50 LUTs and 210 FFs. Each BRAM (RAMB18E1) [126] [127] stores 1024 x 16 bit signed value. Furthermore, each BRAM has two read/write ports. The left BRAM's dual outputs are feed to the dual inputs of the DSP48E1. Then the DSP48E1 [128] performs arithmetic on the DSP48E1's inputs. After computing the values, the DSP48E1 outputs a 48 bit signed result. Subsequently, the 48 bit signed integer is truncated into a 16 bit signed integer. The DSP48E1's single output is connected to the right BRAM's port 0. The right BRAM's port 0 is always set to write DSP48E1's output, while port 1 is always set to read the right BRAM's data.

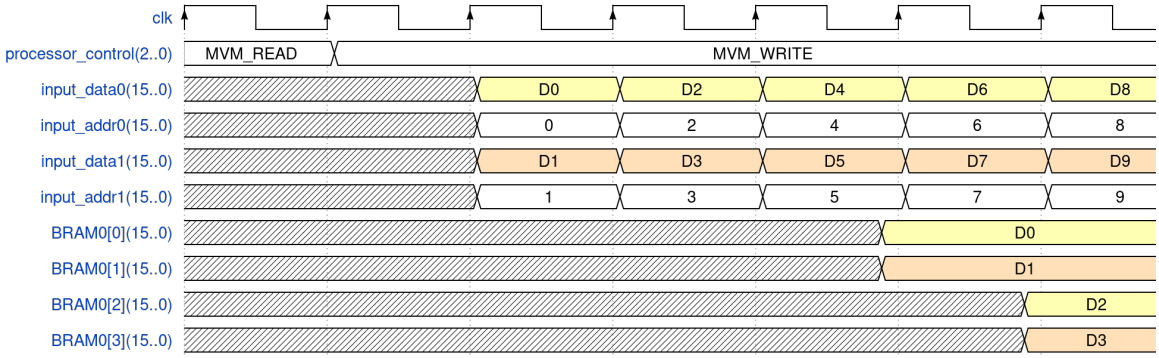

Figure 5.7: Mini Vector Machine's write timing diagram.

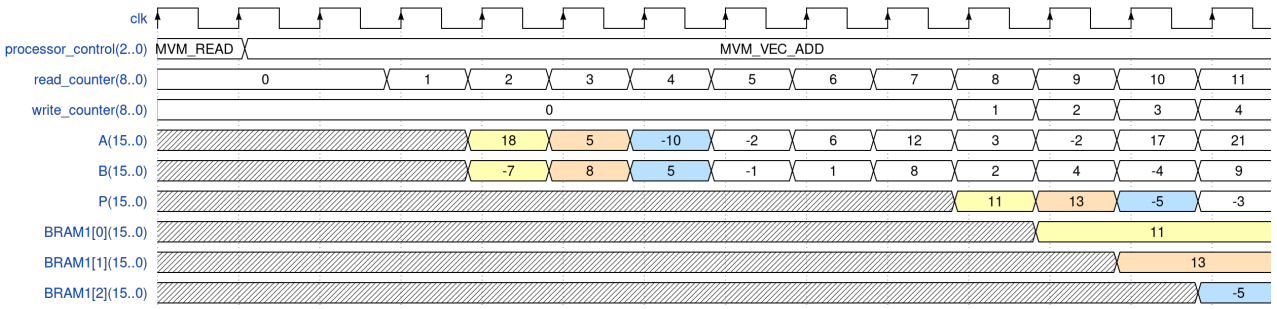

Figure 5.8: Mini Vector Machine's vector addition.

Fig. 5.7 shows the writing timing diagram of the Mini Vector Machine. Mini Vector Machine starts with the MVM_READ state, where Mini Vector Machine is halted. Then the Mini Vector Machine's state transitions to MVM_WRITE. In the MVM_WRITE state, the Mini Vector Machine executes the setup phase of the left BRAM in the 1st cycle. In the 2nd cycle, the left BRAM writes input_data0 and input_data1 in parallel using the addresses given by input_addr0 and input_addr1.

Input_data0 and input_data1 each have a 16 bit signed integer. Moreover, the left BRAM takes 1 cycle to write the input data pairs into the columns.

Once the left BRAM is full, the Mini Vector Machine executes the vector operations. Fig. 5.8 shows the Mini Vector Machine's vector addition. The 1st cycle is used for the setup phase of the DSP48E1, BRAMs, read counter, and write counter. In the 2nd cycle, the left BRAM is read using the read counter. At the same time, the read counter is incremented. In the 3rd cycle, the DSP48E1's A and B ports are feed with the left BRAM's data. The DSP48E1 is configured as a 6 stage pipeline. At the 8th cycle, the DSP48E1's P port outputs the result. In the 8th cycle, the write counter increments. In the 9th cycle, the right BRAM writes the result using the write counter.

5.3.3 Activation Processors

processor_control(1..0)	Operation name	Operation description
00	ACTPRO_READ	Read BRAM
01	ACTPRO_WRITE_ACT	Write activation function to BRAM
10	ACTPRO_WRITE_DATA	Write input data to BRAM
11	ACTPRO_RUN	Bit shift and activation function

Table 5.7: Activation Processor operations.

The Activation Processor performs bit shifts and executes the activation function. The Activation Processor's ports are similar to the Mini Vector Machine's ports shown in Table 5.5. The only difference is the size of the processor control signal. Table 5.7 shows the list of controls for the Activation Processor.

Fig. 5.9 shows the structure of the Activation Processor. Activation Processor consists of 3 x BRAM, 2 x counter, and 1 x control logic. The control logic requires 70 LUTs and 210 FFs. The left BRAM is connected to the dual bit shifts. Each bit shifter applies a 7 bit shift to the right. After the dual bit shifts, the values are used as addresses to look-up the results for the activation functions. Each look-up table uses 1 BRAM resource. Moreover, the look-up tables are able to store the activation functions as well as the derivatives of the activation functions. At the end, the results are written to the right BRAM.

Fig. 5.10 shows the Activation Processor executing the ReLU function. In the 1st cycle of ACTPRO_RUN, the control logic sets up the pipeline. At the 2nd cycle,

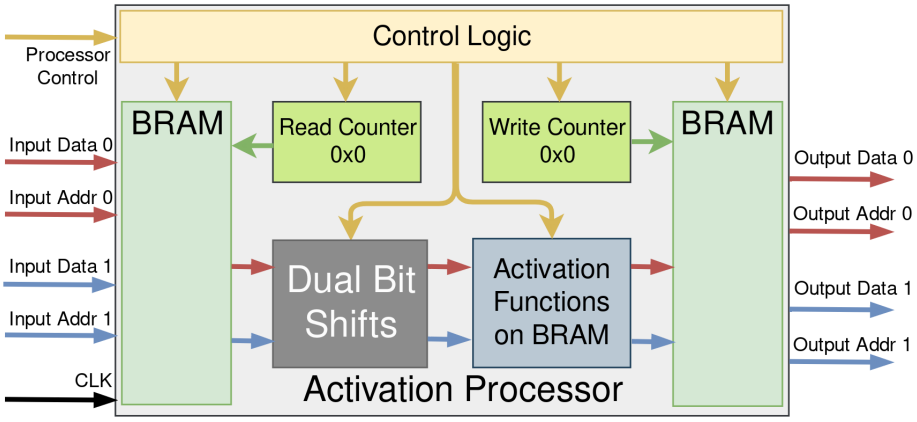

Figure 5.9: The structure of the Activation Processor.

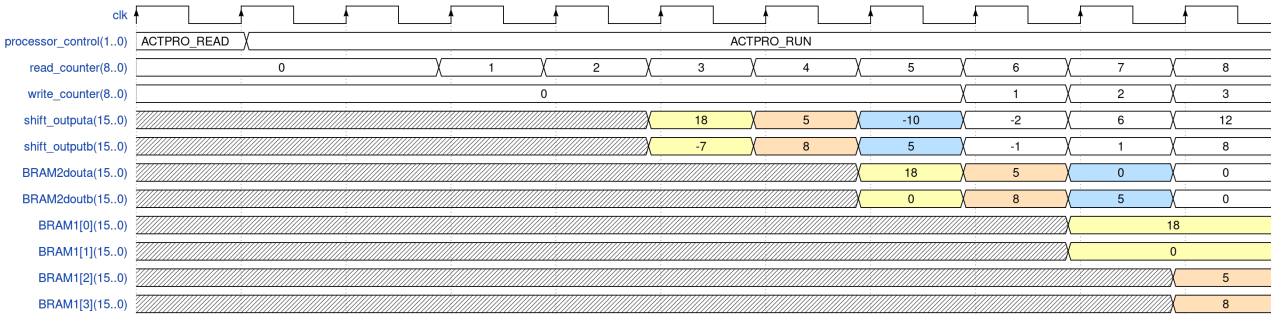

Figure 5.10: The Activation processor executing the ReLU function.

the control logic reads the left BRAM using the read counter. At the same time, the read counter is incremented. In the 3rd cycle, the Activation Processor shifts the 2 x 16 bit integer. In the 5th cycle, result of the activation function is retrieved. In the 6th cycle, the write counter is incremented. In the 7th cycle, the result is written to the right BRAM using the write counter.

5.4 Performance/Cost Evaluation

Let N_{DDR} = number of DDR RAM channels

Let CLK_{DDR} = DDR bus clock in MHz

Let N_{bits} = number of bits on the DDR RAM bus

Let C_{FPGA} = the cost of the FPGA in CAD

Let R = DDR throughput in $\frac{Mb}{s}$

Let F = DDR throughput to cost ratio in $\frac{Mb}{s \cdot CAD}$

FPGA	IO pins	DDR channels	DDR Bus Clock (MHz)	Cost (CAD)	DDR/Cost (Mb/s/CAD)
XC7S50-1	250	2	333.33	75.94	561.84
XC7S75-1	400	4	333.33	134.46	634.63
XC7S100-1	400	4	333.33	163.73	521.17
XC7S50-2	250	2	400	95.11	538.32
XC7S75-2	**400**	**4**	**400**	**147.95**	**692.12**
XC7S100-2	400	4	400	198.12	516.85
XC7A75T-1	300	3	333.33	213.27	300.08
XC7A100T-1	300	3	333.33	234.6	272.80
XC7A200T-1	500	5	333.33	381.95	279.26

Table 5.8: Performance/Cost evaluation of FPGAs.

$$R = CLK_{DDR} \cdot 2 \cdot N_{bits} \cdot N_{DDR} \tag{5.10}$$

$$F = \frac{R}{C_{FPGA}} \tag{5.11}$$

The main limiting factor in the FPGAs' performances is the DDR throughput R. Table 5.8 [129] [130] [131] shows the performance/cost evaluation of FPGAs. Only the Spartan-7 and Artix-7 families were considered because they have the highest performance/cost ratio. Firstly, the FPGAs' DDR throughputs R were calculated using the Eqn. 5.10. Secondly, the performance/cost ratios F were calculated using the costs of the FPGAs and Eqn. 5.11. Finally, Spartan-7 XC7S75-2 was selected as the best FPGA because the XC7S75-2 has the highest performance/cost ratio. Moreover, a cluster of FPGAs could be built using the XC7S75-2. The cluster would outperform a standalone FPGA because the cluster has a higher number of DDR channels.

5.5 Design Verification

The individual components such as the BRAM and DSP design were created in Vivado block design. After wrapping the block design into VHDL modules, the individual components were place into the VHDL design of Mini Vector Machines and Activation Processors. The individual processors were tested using Vivado unit testing.

Subsequently, the Matrix Machine was synthesized on Vivado and was flashed onto NEXYS 4 DDR development board. NEXYS 4 DDR was able to run the Matrix Machine at 100 MHz.

5.6 Advantages and Disadvantages of FPGAs

The advantages of the FPGAs are the flexibility, the low cost, the low power, the low latencies in non linear memory access, and the low latencies in integer computations. FPGAs are more flexible than GPUs because FPGAs can be bought as individual ICs. The types of FPGAs, RAMs, flash memories, network ports could be selected specifically to solve a specific problem. On the other hand, GPUs are mainly designed for gaming and video rendering. Most FPGAs do not require any CPUs and the FPGAs can run stand alone. Moreover, the FPGAs can accept commands directly from the network transceivers. In the GPU case, the GPUs requires CPUs, CPUs' RAMs, and CPUs' motherboards. The CPUs need to have high clock frequencies in order to maintain the memory transfer bandwidth to the GPUs. This adds significant cost to the GPUs. FPGAs have distributed LUT based RAM that has very low latencies and is optimized for non linear memory access. Unlike the FPGAs, the caches in GPUs are optimized for linear memory access. The DSPs in the FPGAs are optimized for integer operations. Moreover, the LUTs in FPGAs can be used to execute many integer operations. As the integer operations are much simpler than floating point operations and the clock frequency of the FPGA is much lower, the FPGAs uses less power.

The disadvantages of FPGAs are the high latencies of floating point computations, high latencies of serial tasks, low clock frequency, and the difficulty of programming and debugging. FPGAs have higher latencies in floating point operations because none of the individual FPGA elements are able to execute floating point operations by default. Numerous DSPs and LUTs need to be routed into a complex pipeline for floating point operations and this creates large latencies. The individual processors in GPUs are optimized for floating point operations and they have lower latencies. Due to the FPGAs having lower clock frequencies, the FPGAs are not suited for serial tasks such as loading data from HDDs and writing to an atomic database. The main difficulty of FPGAs is the programming and debugging. Programming languages such as VHDL and Verilog are hard to understand for novice programmers. VHDL and Verilog are built around parallelism, concurrency, and timing diagrams. Timing

diagrams need to be very precise and correct. An error of 1 cycle due to timing jitters or changes in temperatures could cause catastrophic errors. Debugging is very hard because a soft processor is needed. The soft processor latches on to the signals at specific time intervals and translates the signals to a serial protocol format. The serial data is then read by the JTAG debugger. The act of debugging may cause errors in the timing diagram because the soft processor may contain the critical path.

5.7 Conclusion

Neural networks prove to be extremely useful. However, neural networks require a lot of computational power. Moreover, neural networks need a large memory bandwidth to load the data. FPGAs were selected to solve the problems because FPGAs have a high memory bandwidth/cost ratio. Spartan-7 XC7S75-2 was selected because XC7S75-2 has the best bandwidth/cost ratio out of the Xilinx's 7 series FPGAs. Moreover, the Matrix Assembler was implemented to optimize the design of the Matrix Machine. The Matrix Assembler takes in neural network assembly codes and produces microcodes and VHDL codes. The VHDL codes form the structure of the Matrix Machine. The Matrix Machine has multiple Mini Vector Machines that execute vector operations. The Mini Vector Machines allow neural network acceleration. Furthermore, the microcodes were used to schedule the executions of the Mini Vector Machines. The microcodes allow the FPGAs to switch between neural networks without reloading the bitstream.

Chapter 6

Conclusion

This thesis covered ECG analysis, spectra analysis, and FPGA VHDL design. In ECG, the QRS complex signal is used for calculating the heart rate. It is also used for measuring the PR interval, ST interval, and QT interval. Moreover, the QRS complex is used for the segmentation of ECG signals. Using the heart rate, ECG intervals, and ECG segments, the diagnoses of various heart diseases could be made. For example, arrhythmia is diagnosed by examining irregular heart rates, ST intervals, and QT intervals. In recent years, many wearable ECG devices have appeared on the market. The wearable ECG devices introduce many noises such as motion artifact noise and electrode contact noise due to the movements of the patients. Furthermore, the analog to digital converters and the low noise amplifiers found in the wearable ECG devices are of lower quality. This causes many quantization errors and introduces many amplification noises.

Chapter 2 proposes a CNN-LSTM for the detection of QRS complex in noisy environments. Assuming most wearable ECG devices only have a single channel, the first CNN layer takes in the single channel ECG signal and the gradient of single channel ECG signal. The first CNN layer uses a 91x2 kernel with 4 channels to reduce the ECG noises. After that, the second CNN layer collapses the 4 channels into 1 channel using another 91x2 kernel. The second CNN layer also extracts visual features from the ECG signals. Subsequently, the 2 LSTM layers, each layer having 200 neurons, extract time information from the ECG signals. Finally, the 3 MLP layers, each having 200 neurons, format the output of the QRS complex detection. The CNN-LSTM is tested on the MIT-BIH arrhythmia database [75, 76] and the European ST-T database [77]. Noise is added to the ECG recordings using the PhysioToolkit Noise Stress Test [78]. Tables 2.2-2.5 show the results of the 1x10 fold testing on the MIT-

BIH NST and the European ST-T NST databases. For both databases, the proposed CNN-LSTM outperforms GQRS [47], Pan and Tompkins [43], Wavedet [48], Xiang et al.'s CNN [65], and Chandra et al.'s CNN [68] in terms of F_1 score.

Chapter 3 proposes an upgraded version of the CNN-LSTM found in Chapter 2. Due to the limitations of the CNN-LSTM, a new machine learning pipeline is created for the detection of QRS complexes in noisy ECG signals. The machine learning pipeline consists of a Butterworth filter, two wavelet convolutional neural networks (WaveletCNNs) autoencoders, an optional QRS complex inverter, a Monte Carlo k-nearest neighbours (k-NN), and a convolutional long short-term memory (ConvLSTM). WaveletCNN autoencoders filter out electrode contact noise, instrumentation noise, and motion artifact noise by using the advantages of wavelet filters and convolutional neural networks. The QRS complex inverter flips inverted QRS complexes. Monte Carlo k-NN performs automatic gain control on the ECG signals in order to normalize it. The ConvLSTM executes the final QRS complex detection by using the power of a convolutional neural network and a long short-term memory. It has been demonstrated in Table 3.10 that all the Bayes factor $K > 3.2$, which means there is substantial evidence against the null hypothesis H_0. Therefore, the null hypothesis H_0 is rejected. As a result, the proposed machine learning pipeline outperforms existing QRS complex detection methods.

Spectra analysis is useful for the detection of chemical compounds in unknown substances. Given IR spectra of gas mixtures, the concentrations of the gases can be determined. This is useful for the detection of hazardous gases such as CO, CO_2, HF, and HBr. Many researchers have used PLS and MLP quantifiers for spectra analysis. However, PLS and MLP quantifiers require large training datasets and testing datasets. Line-by-line molecular spectroscopy databases such as High Resolution Transmission molecular absorbance database (HITRAN) [2], Carbon Dioxide Spectroscopic Databank (CDSD-4000) [117], and High-Temperature Molecular Database (HITEMP) [118] were developed to solve these problems. These databases contain spectra of individual molecules, which allows the user to synthesize spectra for any combination of molecules. However, spectra synthesis using the databases comes with significant downsides. The databases do not contain the non linear molecular interactions between the different molecules. This results in invalid spectra for some mixtures. Furthermore, realistic noises are not accurately represented, which creates a discrepancy between the simulated spectra and experimentally measured spectra.

Chapter 4 proposes a generative adversarial network (GAN) [119] for the synthesis

of C_2H_6, CH_4, CO, H_2O, HBr, HCl, HF, N_2O, and NO spectra. The GAN consists of a generator and a discriminator. The generator creates realistic spectra using the input gas concentrations. It also accounts for the non linear interactions between the molecules. Furthermore, the generator allows the user to change the signal to noise ratios (SNRs) of the synthetic spectra. On the other hand, the discriminator classifies the spectra into two groups: generated spectra and actual spectra. This is done in order to force the generator to synthesize more realistic spectra. The GAN is trained using a zero sum game in order to introduce realistic non-deterministic noises into the spectra. Fig. 4.19 and Fig. 4.20 show the learning curves for the generated spectra and the synthetic spectra. Using the actual spectra samples, the MLP and PLS quantifiers converge at 100,000 actual samples. On the other hand, the GAN takes in the actual spectra samples and generates large amounts of synthetic spectra samples. In the synthetic case, the MLP quantifier converges at 3760 actual samples, while the PLS quantifier converges at 120 actual samples. As a result, the MLP and PLS quantifiers converge at a faster rate and with fewer actual spectra samples, when using the GAN.

The de facto way to train and test neural networks is by using GPUs, supported by CPUs. The data is usually transferred from the CPU's RAM to the GPU's RAM via PCIe bus. Furthermore, the GPU is directly controlled by the CPU. However, the PCIe bus has limited bandwidth and it bottlenecks the execution of neural networks. The situation is even worse for clusters of GPUs because they need to communicate over optical fiber. The electronic to photonic conversion process in optical fiber transceivers cause large latencies. FPGAs might be a solution to this problem. FPGAs operate independently from CPUs because they can pull data directly from hard drives or from RAM banks. Also, FPGAs can directly communicate with other FPGAs using the traces on the PCB. This eliminates the PCIe bus bottleneck.

Chapter 5 details a VHDL design that allows FPGAs to train and test any neural network. The Matrix Assembler is a high level optimizing assembler, which parses the neural network assembly codes. The Matrix Assembler parses as many neural network assembly codes as the user wants. After parsing the assembly codes, the Matrix Assembler optimizes the assembly codes and neural network processors. Then the Matrix Assembler generates the VHDL codes and the microcodes. The VHDL codes contain the structure of the Matrix Machine. The Matrix Machine consists of multiple Mini Vector Machines. Each Mini Vector Machine computes a vector operation using a single DSP. The DSPs are set to process 16 bit signed integers.

16 bit precision is enough for most of the neural network applications. When the Mini Vector Machines are put together, the Mini Vector Machines perform matrix operations. The Mini Vector Machines allow the Matrix Machine to adapt to different sizes of matrices. Subsequently, the VHDL codes are synthesized into the bit-streams using the Xilinx's Vivado Design Suite. After generating the bit-streams, the bit-streams are flashed to the onboard flash. The onboard flash then loads the bit-stream onto the FPGA. The system buses transfer the neural network data and microcode from the control server to the onboard RAM. The onboard RAM acts as a buffer for the FPGA. The microcodes schedule the execution of the Matrix Machine by coordinating the individual Mini Vector Machines.

Chapter 7

Publications

7.1 Published Papers

Yuen, Brosnan, Xiaodai Dong, and Tao Lu. "Inter-Patient CNN-LSTM for QRS Complex Detection in Noisy ECG Signals." IEEE Access 7 (2019): 169359-169370.

Yuen, Brosnan, Xiaodai Dong, and Tao Lu. "Detecting Noisy ECG QRS Complexes using WaveletCNN Autoencoder and ConvLSTM." IEEE Access 8 (2020): 143802-143817.

Yuen, Brosnan, and Mihai Sima. "Low Cost Radiation Hardened Software and Hardware Implementation for CubeSats." The Arbutus Review 9.1 (2018): 46-62.

Hoang, Minh Tu, Brosnan Yuen , et al. "Recurrent neural networks for accurate RSSI indoor localization." IEEE Internet of Things Journal 6.6 (2019): 10639-10651.

Hoang, Minh Tu, Brosnan Yuen , et al. "Semi-Sequential Probabilistic Model For Indoor Localization Enhancement." IEEE Sensors Journal 20.11 (2020): 6160-6169.

Gan, Luyun, Brosnan Yuen, and Tao Lu. "Multi-label classification with optimal thresholding for multi-composition spectroscopic analysis." Machine Learning and Knowledge Extraction 1.4 (2019): 1084-1099.

Hoang, Minh Tu, Yizhou Zhu , Brosnan Yuen , et al. "A soft range limited k-nearest

neighbors algorithm for indoor localization enhancement." IEEE Sensors Journal 18.24 (2018): 10208-10216.

Bibliography

[1] R. E. Kass and A. E. Raftery, "Bayes factors," *Journal of the american statistical association*, vol. 90, no. 430, pp. 773–795, 1995.

[2] L. S. Rothman, I. E. Gordon, Y. Babikov, A. Barbe, D. C. Benner, P. F. Bernath, M. Birk, L. Bizzocchi, V. Boudon, L. R. Brown *et al.*, "The HITRAN 2012 Molecular Spectroscopic Database," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 130, pp. 4–50, 2013.

[3] A. Guez and J. Selinsky, "Neurocontroller design via supervised and unsupervised learning," *Journal of Intelligent and Robotic Systems*, vol. 2, no. 2-3, pp. 307–335, 1989.

[4] M. E. Celebi and K. Aydin, *Unsupervised learning algorithms*. Springer, 2016, vol. 9.

[5] A. Odena, "Semi-supervised learning with generative adversarial networks," *arXiv preprint arXiv:1606.01583*, 2016.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] D. Teets and K. Whitehead, "The discovery of Ceres: How Gauss became famous," *Mathematics Magazine*, vol. 72, no. 2, pp. 83–93, 1999.

[8] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[9] P. J. Huber, "Robust estimation of a location parameter," in *Breakthroughs in statistics*. Springer, 1992, pp. 492–518.

[10] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[11] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[12] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[13] W. Price, "Global optimization by controlled random search," *Journal of Optimization Theory and Applications*, vol. 40, no. 3, pp. 333–348, 1983.

[14] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[15] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[16] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[17] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[18] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 2. IEEE, 1999, pp. 1470–1477.

[19] L. Bottou, "Stochastic gradient learning in neural networks," *Proceedings of Neuro-Nimes*, vol. 91, no. 8, p. 12, 1991.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[22] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[23] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International Journal of Neural Systems*, vol. 19, no. 04, pp. 295–308, 2009.

[24] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[25] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[26] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[27] F. Richards, "A flexible growth function for empirical use," *Journal of experimental Botany*, vol. 10, no. 2, pp. 290–301, 1959.

[28] B. L. Kalman and S. C. Kwasny, "Why tanh: choosing a sigmoidal function," in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 4. IEEE, 1992, pp. 578–581.

[29] G. E. Hinton and Z. Ghahramani, "Generative models for discovering sparse distributed representations," *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 352, no. 1358, pp. 1177–1190, 1997.

[30] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[31] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[32] H. Zheng, Z. Yang, W. Liu, J. Liang, and Y. Li, "Improving deep neural networks using softplus units," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–4.

[33] E. J. Hartman, J. D. Keeler, and J. M. Kowalski, "Layered neural networks with Gaussian hidden units as universal approximations," *Neural Computation*, vol. 2, no. 2, pp. 210–215, 1990.

[34] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, no. 2, pp. 246–257, 1991.

[35] D. Misra, "Mish: A Self Regularized Non-Monotonic Neural Activation Function," *arXiv preprint arXiv:1908.08681*, 2019.

[36] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[37] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[38] Y. V. Bodyanskiy, A. Tyshchenko, and A. Deineko, "An evolving radial basis neural network with adaptive learning of its parameters and architecture," *Automatic Control and Computer Sciences*, vol. 49, no. 5, pp. 255–260, 2015.

[39] S. Qian, H. Liu, C. Liu, S. Wu, and H. San Wong, "Adaptive activation functions in convolutional neural networks," *Neurocomputing*, vol. 272, pp. 204–212, 2018.

[40] P. Campolucci, F. Capperelli, S. Guarnieri, F. Piazza, and A. Uncini, "Neural networks with adaptive spline activation function," in *Proceedings of 8th Mediterranean Electrotechnical Conference on Industrial Applications in Power Systems, Computer Science and Telecommunications (MELECON 96)*, vol. 3. IEEE, 1996, pp. 1442–1445.

[41] B.-U. Kohler, C. Hennig, and R. Orglmeister, "The principles of software QRS detection," *IEEE Eng. in Med. and Biol. Magazine*, vol. 21, no. 1, pp. 42–57, 2002.

[42] G. M. Friesen, T. C. Jannett, M. A. Jadallah, S. L. Yates, S. R. Quint, and H. T. Nagle, "A comparison of the noise sensitivity of nine QRS detection algorithms," *IEEE Trans. Biomed. Eng.*, vol. 37, no. 1, pp. 85–98, 1990.

[43] J. Pan and W. J. Tompkins, "A real-time QRS detection algorithm," *IEEE Trans. Biomed. Eng*, vol. 32, no. 3, pp. 230–236, 1985.

[44] R. Gupta, M. Mitra, K. Mondal, and S. Bhowmick, "A derivative-based approach for QT-segment feature extraction in digitized ECG record," *2011 Second Intl. Conf. on Emerging Applications of Informat. Technol.*, pp. 63–66, 2011.

[45] N. M. Arzeno, C.-S. Poon, and Z.-D. Deng, "Quantitative analysis of QRS detection algorithms based on the first derivative of the ECG," *2006 Int. Conf. IEEE Eng. Med. Biol. Soc.*, pp. 1788–1791, 2006.

[46] N. M. Arzeno, Z.-D. Deng, and C.-S. Poon, "Analysis of first-derivative based QRS detection algorithms," *IEEE Trans. Biomed. Eng.*, vol. 55, no. 2, pp. 478–484, 2008.

[47] Physionet. (2018) GQRS, QRS detector and post-processor. [Online]. Available: https://www.physionet.org/physiotools/wag/gqrs-1.htm

[48] J. P. Martinez, R. Almeida, S. Olmos, A. P. Rocha, and P. Laguna, "A wavelet-based ECG delineator: evaluation on standard databases," *IEEE Trans. Biomed. Eng.*, vol. 51, no. 4, pp. 570–581, 2004.

[49] S. Kadambe, R. Murray, and G. F. Boudreaux-Bartels, "Wavelet transform-based QRS complex detector," *IEEE Trans. Biomed. Eng.*, vol. 46, no. 7, pp. 838–848, 1999.

[50] K. Mourad and B. R. Fethi, "Efficient automatic detection of QRS complexes in ECG signal based on reverse biorthogonal wavelet decomposition and nonlinear filtering," *Measurement*, vol. 94, pp. 663–670, 2016.

[51] C. Li, C. Zheng, and C. Tai, "Detection of ECG characteristic points using wavelet transforms," *IEEE Trans. Biomed. Eng.*, vol. 42, no. 1, pp. 21–28, 1995.

[52] F. Bouaziz, D. Boutana, and M. Benidir, "Multiresolution wavelet-based QRS complex detection algorithm suited to several abnormal morphologies," *IET Signal Processing*, vol. 8, no. 7, pp. 774–782, 2014.

[53] U. D. Ulusar, R. Govindan, J. D. Wilson, C. L. Lowery, H. Preissl, and H. Eswaran, "Adaptive rule based fetal QRS complex detection using Hilbert transform," *2009 Int. Conf. IEEE Eng. Med. Biol. Soc.*, pp. 4666–4669, 2009.

[54] D. S. Benitez, P. Gaydecki, A. Zaidi, and A. Fitzpatrick, "A new QRS detection algorithm based on the Hilbert transform," *Comput. Cardiology*, pp. 379–382, 2000.

[55] F. I. de Oliveira and P. U. Cortez, "A QRS detection based on Hilbert transform and wavelet bases," *Proc. of the 2004 14th IEEE Signal Processing Soc. Workshop Machine Learning for Signal Processing*, pp. 481–489, 2004.

[56] P. Hamilton and W. Tompkins, "Adaptive matched filtering for QRS detection," *Eng. in Med. and Biol. Soc., 1988. Proc. of the Annu. Intl. Conf. of the IEEE*, pp. 147–148, 1988.

[57] D. T. Kaplan, "Simultaneous QRS detection and feature extraction using simple matched filter basis functions," *Proc. Comput. Cardiology*, pp. 503–506, 1990.

[58] D. Craven, B. McGinley, L. Kilmartin, M. Glavin, and E. Jones, "Adaptive dictionary reconstruction for compressed sensing of ECG signals," *IEEE Journal of Biomed. and Health Informatics*, vol. 21, no. 3, pp. 645–654, 2017.

[59] M. Balouchestani, K. Raahemifar, and S. Krishnan, "High-resolution QRS detection algorithm for wireless ECG systems based on compressed sensing theory," *Circuits and Systems (MWSCAS), 2013 IEEE 56th Intl. Midwest Symp.*, pp. 1326–1329, 2013.

[60] G. Goovaerts, S. Padhy, B. Vandenberk, C. Varon, R. Willems, and S. Van Huffel, "A machine learning approach for detection and quantification of QRS fragmentation," *IEEE Journal of Biomed. and Health Informatics*, 2018.

[61] S. Mehta and N. Lingayat, "SVM-based algorithm for recognition of QRS complexes in electrocardiogram," *IRBM*, vol. 29, no. 5, pp. 310–317, 2008.

[62] S. S. Mehta and N. S. Lingayat, "Identification of QRS complexes in 12-lead electrocardiogram," *Expert Systems with Applications*, vol. 36, no. 1, pp. 820–828, 2009.

[63] Q. Xue, Y. H. Hu, and W. J. Tompkins, "Neural-network-based adaptive matched filtering for QRS detection," *IEEE Trans. Biomed. Eng.*, vol. 39, no. 4, pp. 317–329, 1992.

[64] B. Abibullaev and H. D. Seo, "A new QRS detection method using wavelets and artificial neural networks," *Journal of Med. Syst.*, vol. 35, no. 4, pp. 683–691, 2011.

[65] Y. Xiang, Z. Lin, and J. Meng, "Automatic QRS complex detection using two-level convolutional neural network," *Biomed. Eng. OnLine*, vol. 17, no. 1, p. 13, Jan 2018. [Online]. Available: https://doi.org/10.1186/s12938-018-0441-4

[66] J. Camps, A. McCarthy, B. Rodrıguez, and A. Mincholé, "Deep learning based QRS multilead delineator in electrocardiogram signals," *Comput. Cardiology*, 2018.

[67] K. Arbateni and A. Bennia, "Sigmoidal radial basis function ANN for QRS complex detection," *Neurocomputing*, vol. 145, pp. 438–450, 2014.

[68] B. S. Chandra, C. S. Sastry, and S. Jana, "Robust heartbeat detection from multimodal data via CNN-based generalizable information fusion," *IEEE Trans. Biomed. Eng.*, vol. 66, no. 3, pp. 710–717, 2018.

[69] N. Ravanshad, H. Rezaee-Dehsorkh, R. Lotfi, and Y. Lian, "A level-crossing based QRS-detection algorithm for wearable ECG sensors," *IEEE Journal of Biomed. and Health Informatics*, vol. 18, no. 1, pp. 183–192, 2014.

[70] F. Zhang and Y. Lian, "Electrocardiogram QRS detection using multiscale filtering based on mathematical morphology," *2007 29th Annu. Intl. Conf. IEEE Eng. Med. Biol. Soc.*, pp. 3196–3199, 2007.

[71] P. Phukpattaranont, "QRS detection algorithm based on the quadratic filter," *Expert Systems with Applications*, vol. 42, no. 11, pp. 4867–4877, 2015.

[72] M. Zihlmann, D. Perekrestenko, and M. Tschannen, "Convolutional recurrent neural networks for electrocardiogram classification," *Comput. Cardiology*, pp. 1–4, 2017.

[73] T. J. Jun, H. M. Nguyen, D. Kang, D. Kim, D. Kim, and Y.-H. Kim, "ECG arrhythmia classification using a 2-D convolutional neural network," *arXiv preprint arXiv:1804.06812*, 2018.

[74] P. Rajpurkar, A. Y. Hannun, M. Haghpanahi, C. Bourn, and A. Y. Ng, "Cardiologist-level arrhythmia detection with convolutional neural networks," *arXiv preprint arXiv:1707.01836*, 2017.

[75] G. B. Moody and R. G. Mark, "The impact of the MIT-BIH arrhythmia database," *IEEE Eng. in Med. and Biol. Magazine*, vol. 20, no. 3, pp. 45–50, 2001.

[76] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals." *Circulation*, vol. 101, no. 23, pp. E215–E220, 2000.

[77] A. Taddei, G. Distante, M. Emdin, P. Pisani, G. Moody, C. Zeelenberg, and C. Marchesi, "The European ST-T database: standard for evaluating systems for the analysis of ST-T changes in ambulatory electrocardiography," *European Heart Journal*, vol. 13, no. 9, pp. 1164–1172, 1992.

[78] G. B. Moody, W. Muldrow, and R. G. Mark, "A noise stress test for arrhythmia detectors," *Comput. Cardiology*, vol. 11, no. 3, pp. 381–384, 1984.

[79] A. L. Goldberger, Z. D. Goldberger, and A. Shvilkin, *Goldberger's clinical electrocardiography 9th edition: a simplified approach.* Elsevier Health Sciences, 2017.

[80] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[81] R. S. Andersen, A. Peimankar, and S. Puthusserypady, "A deep learning approach for real-time detection of atrial fibrillation," *Expert Systems with Applications*, vol. 115, pp. 465–473, 2019.

[82] D. Verma and S. Agarwal, "Cardiac arrhythmia detection from single-lead ecg using cnn and lstm assisted by oversampling," in *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2018, pp. 14–17.

[83] B. Pourbabaee, M. Roshtkhari, and K. Khorasani, "Deep convolutional neural networks and learning ECG features for screening paroxysmal atrial fibrillation patients," *IEEE Trans. Syst., Man, Cybern.: Syst.*, vol. 48, no. 12, pp. 2095–2104, 2018.

[84] Physionet. (2018) QRS detection and waveform boundary recognition using ecgpuwave. [Online]. Available: https://www.physionet.org/physiotools/ecgpuwave/

[85] W. Zong, G. Moody, and D. Jiang, "A robust open-source algorithm to detect onset and duration of QRS complexes," *Comput. Cardiology*, pp. 737–740, 2003.

[86] M. Jia, F. Li, J. Wu, Z. Chen, and Y. Pu, "Robust QRS detection using high-resolution wavelet packet decomposition and time-attention convolutional neural network," *IEEE Access*, vol. 8, pp. 16 979–16 988, 2020.

[87] C.-C. Lin, H.-Y. Chang, Y.-H. Huang, and C.-Y. Yeh, "A novel wavelet-based algorithm for detection of QRS complex," *Applied Sciences*, vol. 9, no. 10, p. 2142, 2019.

[88] M. B. Hossain, S. K. Bashar, A. J. Walkey, D. D. McManus, and K. H. Chon, "An accurate QRS complex and P wave detection in ECG signals using complete ensemble empirical mode decomposition with adaptive noise approach," *IEEE Access*, vol. 7, pp. 128 869–128 880, 2019.

[89] H. Limaye and V. Deshmukh, "ECG noise sources and various noise removal techniques: a survey," *Intl. Journal of Application or Innovation in Eng. & Management*, vol. 5, no. 2, pp. 86–92, 2016.

[90] F. Jager, A. Taddei, G. B. Moody, M. Emdin, G. Antolič, R. Dorn, A. Smrdel, C. Marchesi, and R. G. Mark, "Long-term ST database: a reference for the development and evaluation of automated ischaemia detectors and for the study of the dynamics of myocardial ischaemia," *Medical and Biological Engineering and Computing*, vol. 41, no. 2, pp. 172–182, 2003.

[91] S. Butterworth *et al.*, "On the theory of filter amplifiers," *Wireless Engineer*, vol. 7, no. 6, pp. 536–541, 1930.

[92] C. E. Heil and D. F. Walnut, "Continuous and discrete wavelet transforms," *SIAM Review*, vol. 31, no. 4, pp. 628–666, 1989.

[93] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Inform. Processing Syst. 25 (NIPS 2012)*, pp. 1097–1105, 2012.

[94] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in Neural Information Processing Systems*, 2015, pp. 802–810.

[95] B. Yuen, X. Dong, and T. Lu, "Inter-patient CNN-LSTM for QRS complex detection in noisy ECG signals," *IEEE Access*, vol. 7, pp. 169 359–169 370, 2019.

[96] J. O. Berger and L. R. Pericchi, "The intrinsic Bayes factor for model selection and prediction," *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 109–122, 1996.

[97] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[98] R. K. Chikara and L.-W. Ko, "Neural activities classification of human inhibitory control using hierarchical model," *Sensors*, vol. 19, no. 17, p. 3791, 2019.

[99] H. J. Al-Tameme, M. Y. Hadi, and I. H. Hameed, "Phytochemical analysis of Urtica dioica leaves by Fourier-transform infrared spectroscopy and gas chromatography-mass spectrometry," *Journal of Pharmacognosy and Phytotherapy*, vol. 7, no. 10, pp. 238–252, 2015. [Online]. Available: https://doi.org/10.5897/JPP2015.0361

[100] N. Zaini, F. V. D. Meer, F. V. Ruitenbeek, B. D. Smeth, F. Amri, and C. Lievens, "An alternative quality control technique for mineral chemistry analysis of portland cement-grade limestone using shortwave infrared spectroscopy," *Remote Sensing*, vol. 8, no. 11, p. 950, 2016. [Online]. Available: https://www.mdpi.com/2072-4292/8/11/950

[101] L. Wang, D. W. Sun, H. Pu, and J. H. Cheng, "Quality analysis, classification, and authentication of liquid foods by near-infrared spectroscopy: a review of recent research developments," *Critical Reviews in Food Science and Nutrition*, vol. 57, no. 7, pp. 1524–1538, 2017. [Online]. Available: https://doi.org/10.1080/10408398.2015.1115954

[102] N. Ryde, T. K. Fritz, R. M. Rich, B. Thorsbro, M. Schultheis, L. Origlia, and S. Chatzopoulos, "Detailed abundance analysis of a metal-poor giant in the galactic center," *The Astrophysical Journal*, vol. 831, no. 1, p. 40, oct 2016. [Online]. Available: https://doi.org/10.3847/0004-637x/831/1/40

[103] H. Yang, S. Yang, J. Kong, A. Dong, and S. Yu, "Obtaining information about protein secondary structures in aqueous solution using Fourier transform IR spectroscopy," *Nature Protocols*, vol. 10, no. 3, p. 382, 2015.

[104] F. D. Fuller and J. P. Ogilvie, "Experimental implementations of two-dimensional Fourier transform electronic spectroscopy," *Annual Review of Physical Chemistry*, vol. 66, pp. 667–690, 2015.

[105] N. Nesakumar, C. Baskar, S. Kesavan, J. B. B. Rayappan, and S. Alwarappan, "Analysis of moisture content in beetroot using Fourier transform infrared spectroscopy and by principal component analysis," *Scientific Reports*, vol. 8, no. 1, p. 7996, 2018.

[106] P. Geladi and B. R. Kowalski, "Partial least-squares regression: a tutorial," *Analytica Chimica Acta*, vol. 185, pp. 1–17, 1986.

[107] H. Xie, J. Zhao, Q. Wang, Y. Sui, J. Wang, X. Yang, X. Zhang, and C. Liang, "Soil type recognition as improved by genetic algorithm-based variable selection using near infrared spectroscopy and partial least squares discriminant analysis," *Scientific Reports*, vol. 5, p. 10930, 2015.

[108] M. Yin, S. Tang, and M. Tong, "Identification of edible oils using terahertz spectroscopy combined with genetic algorithm and partial least squares discriminant analysis," *Analytical Methods*, vol. 8, no. 13, pp. 2794–2798, 2016.

[109] Z. Zhu, J. Li, Y. Guo, X. Cheng, Y. Tang, L. Guo, X. Li, Y. Lu, and X. Zeng, "Accuracy improvement of boron by molecular emission with a genetic algorithm and partial least squares regression model in laser-induced breakdown spectroscopy," *Journal of Analytical Atomic Spectrometry*, vol. 33, no. 2, pp. 205–209, 2018.

[110] J. Yang, J. Xu, X. Zhang, C. Wu, T. Lin, and Y. Ying, "Deep learning for vibrational spectral analysis: Recent progress and a practical guide," *Analytica Chimica Acta*, 2019.

[111] I. O. Afara, J. K. Sarin, S. Ojanen, M. Finnila, W. Herzog, S. Saarakkala, R. K. Korhonen, and J. Töyräs, "Deep learning classification of cartilage integrity using near infrared spectroscopy," in *Microscopy Histopathology and Analytics*. Optical Society of America, 2018, p. JTu3A.27.

[112] T. Liu, Z. Li, C. Yu, and Y. Qin, "NIRS feature extraction based on deep auto-encoder neural network," *Infrared Physics & Technology*, vol. 87, pp. 124–128, 2017.

[113] L. Gan, B. Yuen, and T. Lu, "Multi-label classification with optimal thresholding for multi-composition spectroscopic analysis," *arXiv e-prints*, p. arXiv:1906.10242, Jun 2019.

[114] Y. Chen, H. Jiang, C. Li, X. Jia, and P. Ghamisi, "Deep feature extraction and classification of hyperspectral images based on convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 10, pp. 6232–6251, 2016.

[115] J. Padarian, B. Minasny, and A. McBratney, "Using deep learning to predict soil properties from regional spectral data," *Geoderma Regional*, vol. 16, p. e00198, 2019.

[116] C. Yuanyuan and W. Zhibin, "Quantitative analysis modeling of infrared spectroscopy based on ensemble convolutional neural networks," *Chemometrics and Intelligent Laboratory Systems*, vol. 181, pp. 1–10, 2018.

[117] S. Tashkun and V. Perevalov, "CDSD-4000: High-resolution, high-temperature carbon dioxide spectroscopic databank," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 112, no. 9, pp. 1403–1410, 2011.

[118] L. Rothman, I. Gordon, R. Barber, H. Dothe, R. Gamache, A. Goldman, V. Perevalov, S. Tashkun, and J. Tennyson, "HITEMP, the high-temperature molecular spectroscopic database," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 111, no. 15, pp. 2139–2150, 2010.

[119] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[120] P. Bouguer, *Essai d'optique sur la gradation de la lumière*. chez Claude Jombert, ruë S. Jacques, au coin de la ruë des Mathurins, à l'Image Notre-Dame, 1729.

[121] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[122] L. Jin, Z. Wang, R. Gu, C. Yuan, and Y. Huang, "Training large scale deep neural networks on the Intel Xeon Phi many-core coprocessor," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2014, pp. 1622–1630.

[123] L. Lu and Y. Liang, "SpWA: an efficient sparse winograd convolutional neural networks accelerator on FPGAs," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 135.

[124] N. Shah, P. Chaudhari, and K. Varghese, "Runtime Programmable and Memory Bandwidth Optimized FPGA-Based Coprocessor for Deep Convolutional Neural Network," *IEEE Transactions on Neural Networks and Learning Systems*, no. 99, pp. 1–13, 2018.

[125] F. Kästner, B. Janßen, F. Kautz, M. Hübner, and G. Corradi, "Hardware/Software Codesign for Convolutional Neural Networks Exploiting Dynamic Partial Reconfiguration on PYNQ," in *2018 IEEE International Parallel*

and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2018, pp. 154–161.

[126] Xilinx, "UG473: 7 Series FPGAs Memory Resources," https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf, 2016.

[127] ——, "PG058: Block Memory Generator v8.3," https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_3/pg058-blk-mem-gen.pdf, 2017.

[128] ——, "UG479: 7 Series FPGAs DSP48E1," https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf, 2018.

[129] ——, "7 Series FPGAs Data Sheet: Overview," https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf, 2018.

[130] ——, "Spartan-7 FPGAs Data Sheet: DC and AC Switching Characteristics," https://www.xilinx.com/support/documentation/data_sheets/ds189-spartan-7-data-sheet.pdf, 2018.

[131] ——, "Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics," https://www.xilinx.com/support/documentation/data_sheets/ds181_Artix_7_Data_Sheet.pdf, 2018.