Investigating Approximate Computing to design an energy-efficient deep learning architecture for anomaly detection from ECG signals

DARIO CAPITANI, University Of Twente
Bachelor Student Technical Computer Science, The Netherlands

Cardiovascular diseases make up 32% of global mortality, most of which are detectable through monitoring of physiological signals, such as Electrocardiograms (ECG). Through the use of an Anomaly Detection algorithm, deployed on a device that is continuously monitoring, such diseases can be detected before onset. The requirements for such a device are to be power efficient and small. Existing research covers models that are too large for a wearable device, or reduce the input size. The later allows for smaller models and hardware but leads to a loss of information. Herein, an adapted Multi Layer Perceptron is used taking a whole heart beat as an input, whose architecture is then developed using a Field Programmable Gate Array. From here existing techniques under the paradigm of Approximate Computing are applied, to further reduce hardware and power consumption. This proof of principle paper demonstrates the feasibility of achieving a potential diagnosis while reducing the hardware and power resources.

1 INTRODUCTION

Cardiovascular diseases (CVD) including strokes and other circulatory diseases make up for 32% of the global mortality each year [1]. On top of this, the yearly cost in the European Union due to CVD is estimated to be €169 billion [2]. Such diseases can be detected through physiological signals, such as an ECG before onset [3]. One approach could be to use a wearable device that monitors ECG signals in real-time for the detection of cardiac events like arrhythmias.

The device would have to be sustainable, implying energy and resource efficient, which is achieved by considering the complexity of the model as a bottleneck to hardware implementation. Previous research for an arrhythmia detection task using convolutional neural networks, had the last convolutional layer with 512 filters and a total of $2,097,152$ multiplicative operations [4]. Herein, a Multi-Layer Perceptron (MLP) from [5] is considered, which for 5 arrhythmia classes, had a model size under 4.5KB and consisted solely of dense layers. Development of the architecture was done on a Field Programmable Gate Array (FPGA). When compared to a General Purpose Processor, FPGA's have the potential to perform arithmetic operations in parallel, while being energy and area efficient. FPGA's offer the ability to change the architecture free of charge, enabling the prototyping of Application Specific Integrated Circuits (ASIC), which has well establish power reduction techniques. This process can be combined with the Approximate Computing methodology for FPGA's. Approximations will reduce the range of possible outputs, reducing hardware and power usage, at the cost of potentially reducing the model performance. Therefore a balance between hardware reduction and error introduction should be achieved, while ensuring that performance remains above the specified threshold.

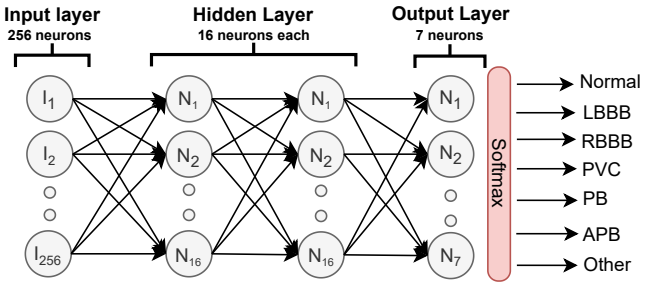


Fig. 1. The final Multi-Layer Perceptron model has an input layer of 256, two hidden layers with 16 neurons each using ReLU activation, and an output layer of 7 classes with a softmax activation.

Section 2 discusses arrhythmia characterization and reviews existing models for arrhythmia detection, approximate computing, and protyping FPGA for ASIC. Section 3 defines the paper's scope, focusing on the model, hardware, and approximate computing. Section 4 provides an overview of the tools, datasets, and preprocessing for the model. Section 5 and Section 6 answer how an MLP is adapted for the task. Section 7 focuses on simulating and implementing the hardware for an FPGA, while Section 8 covers the different types of approximations used. Section 9 presents an overview of the resource, power, and error impacts of the approximations. Finally this paper concludes with Section 10, 11 and 12, where the limitations of the methodology are described, future works are laid out and a summary of the results is made.

2 BACKGROUND

Arrhythmia is characterized as an irregular heart rate or rhythm. This would be either the speed at which the heart beats, or an inconsistent pattern. An ECG records the electrical activity of the heart, which consists of three waves (P, QRS, T) as shown in Fig.2a. Any irregularities with regards to the strength or rhythm of these waves fall under an arrhythmia, which can be fatal. Fig.2b is a type of arrhythmia: notice that while the QRS complex seems normal, the P and T wave are irregular which would warant further investigation.

Three papers were identified among others, which had an objective similar to that of this paper. Namely, developing a hardware architecture for arrhythmia detection in ECG signals, through the use of machine learning, while considering and optimizing for power and resources.

The first paper developed a *Time Convolutional Network* (TCN) for Arrhythmia diagnosis, of 34 different classes, to be deployed on

41st Twente Student Conference on IT, July 5th, 2024, Enschede, Netherlands

a smartphone [6]. This model also made use of a denoising autoencoder, which would down sample and recreate the denoised signal. The noise consisted of motion artifacts, power line interference's or muscle artefacts. For the TCN model, multiple time convolutional layers were placed in parallel. Each layer had different filters of varying kernel sizes, extracting different time/scale-dependent features. By using time convolutional layers, it was able to detect features that manifest over time, such as the interval or differences between heart beats. The model exhibited an overall sensitivity of 96.6%, and used 22.77MB for storage.

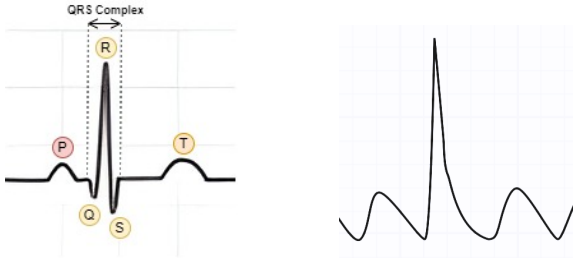
The second paper aimed it's architecture for an ASIC. The developed pipeline consisted of a Finite Impulse Response (FIR) to remove noise, followed by an R-peak detection algorithm to segment the signal into the QRS complex and an MLP with 5 classes of Arrhythmia. The MLP consisted of 3 layers: an input with 100 neurons, a hidden layer with 8 neurons and an output layer of 5 neurons. The paper reports an overall sensitivity of 96.66%. While no mention of the final model size is made, the FIR, R-peak detection algorithm and the model were in total 4.5KB.

For the third paper, the model involves binary classification, into categorizes of normal rhythms or Atrial Fibrillation [7]. In this model, a wavelet transform was used to reduce the dimension of the input ECG to only certain features, which allowed the model to only take as an input the most relevant data. The model consisted of 3 parts; an input of 64 neurons, three hidden layers with 45, 30 and 15 neurons and an output of two classes. The model achieved a sensitivity of 91.84% for normal rhythms and 82.06% for Atrial Fibrillation. The paper did not report the model size, however it consumed 11.098 uW at 25kHz.

All three papers introduce a denoising method, reducing the input to the most relevant information. Although this adds delay, it allows for a smaller model and improved classification in the prescence of noise. Two papers further reduce features to the QRS peak, minimizing model size but losing information about the P and T complexes. Fig. 2b shows that considering only the QRS complex would fail to recognize any irregularity of the P and T complex.

With regards to Approximate Computing on FPGA's, various techniques already exist. By it's definition, the methodology can be introduced in any aspect of the architecture that involves computations. Internal Self Healing (ISH), introduces the idea of canceling out errors, by using errors and their corresponding inverses at different stages [8]. This reduces the area as well as the power consumption of the overall architecture. ISH has been used in the literature, for example to optimize 8-bit multipliers by using sets of 4-bit multipliers [9]. Other techniques not considered by the previously mentioned papers, such ash Loop Perforation, Reduced Precision Computation and Relaxed Synchronization are also used in various fields with resource reduction improvements [10].

A literature review concluded that FPGAs are better suited for prototyping ASIC architectures due to their reconfigurability, despite their higher power consumption from dynamic components [11]. ASIC's have well establish techniques to lower power consumption, while the reconfigurable aspect of FPGA's allow for development of low power hardware [11]. Since ASIC's are hardwired, future FPGA-developed architectures can be transferred to ASICs once optimized, followed by applying ASIC's power reduction techniques to further decrease overall power consumption.



(a) (b)

Fig. 2. Figure 2a is a normal annotated heart beat, while Figure 2b is a heart beat classified as Atrial Flutter

3 PROBLEM STATEMENT

This research paper aims to address the lack of studies regarding the use of complex models running on FPGA's for constrained use. The implemented model should consider the entire spectrum of the ECG waveform, while functioning as a *catch all* model, where it appropriately classifies inputs outside of the trained arrhythmia's. The developed model should then be streamlined, such that it's size and depth are pruned to only what is required, allowing for development of a sustainable and efficient architecture. Subsequently, hardware and power usage should be reduced by implementing approximate computing techniques. The developed architecture will function as a prototype for an ASIC, thus improving the adaptability to a constrained/wearable device. All the while a cardiologist level of sensitivity will have to be maintained. With this in mind, the research hypothesis can be structured as follows:

What is the structure of an ECG anomaly detection algorithm with a cardiologist level sensitivity, which can then be developed into an efficient architecture for a FPGA, followed by using Approximate Computing to further reduce hardware and power requirements for a constrained edge device?

This research hypothesis is investigated by exploring these sub questions:

(1) What is the sensitivity, accuracy and latency required for the final prototype to be used in a medical setting?
(2) How can the layers that make up the model be deployed onto a FPGA?
(3) What *Approximate Computing* technique(s) can be used on the architecture to reduce hardware and power requirements?

4 METHODOLOGY

4.1 Toolflow and Experimental Setup

The model was developed in Python using Keras and Tensorflow. When transitioning to implementation, the fxpmath library was

used to convert model parameters and inputs into fixed-point representation. The model's weights and inputs were extracted into a MAT file for use in MATLAB. Multipliers and adders from [12] were implemented in MATLAB to replicate the model structure. For hardware implementation, Quartus served as the programming environment and ModelSim as the testing environment, using a board from the Cyclone IV E family (specifically the *EP4CE40U1917*). Quartus was used to retrieve resource utilization and power data, with ModelSim simulating realistic switching rates to provide accurate power readings. To get accurate and realistic power readings, ECG data was used as input. For the accurate neuron, 50 instances of the architectures were made in the testbench, while 500 for the approximated architectures, this was due to the fact that in testing the accurate architecture took longer to process data. The test benches were then ran for 101210 and 214660 ns for the accurate and approximate respectively. The VHD files generated were then used in the power analysis tool of Quartus. All processing was done using a 13th Gen Intel(R) Core(TM) i7-13700H.

4.2 Dataset

For developing the model used in this paper, as well as testing the hardware, two datasets were used. Taken from *Physiobank*, the first dataset was the *MIT-BIH Arrhythmia Database* [4][13], consisting of 48 hours of ambulatory ECG signals. The dataset had 16 different /beats types, which acted as the main training set. An additional dataset was used, (*A large scale 12-lead electrocardiogram database*) which is made up of 63 different types of /beats. Since most of these are uncommon, the *catch all* class for the final model made use of these uncommon arrhythmias [14][15]. Both dataset used were resampled to 200 samples per second (200Hz).

For training, an R-peak detection algorithm was used, this would index an R-peak for an ECG given a threshold. The ECG is then segmented into an array of 256 values, where the R-peak is at the center. By sampling at 200Hz, it takes 1.27 seconds to sample to 256 values, which is enough time for a whole heartbeat. The overall dataset was divided into training, validation, and testing sets, with splits of 60%, 20%, and 20% respectively. In Appendix B, an overview of all the classes and the representations used in the paper is given.

5 METRIC DEFINITION

To evaluate the model's performance, two metrics are used: accuracy and sensitivity. Accuracy is defined as the rate of overall correctly predicted cases, while sensitivity focuses on only one class, being the rate of correctly predicted cases out of all predication's for that class. Eq.1 and Eq.2 are the formulas to calculate these metrics, with TP and TN being true positive and negative, while FP and FN being false positive and negatives. The minimum sensitivity will follow the previously published average sensitivity of a cardiologist, that is 78.0% [4]. This will result in an overall accuracy above 78.0%.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} \tag{2}$$

During training, a loss function is used to evaluate the model predictions against the actual classes. This implies that a small loss, close to 0, indicates the model is a good predictor. For the MLP in this paper, a *categorical cross entropy* is used, whose function is Eq.3, where y is the true label, $\hat{y}$ is the predicted label and N is the total number of classes.

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{3}$$

The latency is defined as the time taken to process one input of an ECG signals. The maximum admissible latency follows the idea introduced in a previous paper [5]. Consider that at 220 beats per minute, there are 3.67 beats every second, with one beat every 0.272 seconds. If the maximum admissible latency is of 0.272 seconds, it would imply that at most only one heart beat is lost in the worst case scenario, which is acceptable.

6 MODEL

6.1 Multi Layer Perceptron

Selecting and developing the model architecture should take into account the requirements of the end use case, particularly for wearable devices. In [5] an arrhythmia detection task for an ASIC was developed using a MLP, with the final model size below 4.5KB. In a similar manner this paper will adapt an MLP for arrhythmia detection task, while considering the specific classes and subsequent hardware requirements.

The starting MLP had a total of 76,040 parameters (76K), with a shortcut that went from the input to the output layer. The model design was further reduced in size by changing the number of neurons for the first dense layer, going from 256 to 16. This decreased the number of parameters to 6,016 (6K), a total reduction of 92.1%. As can be seen in Fig.3, there is very small difference in sensitivity per class, with an average difference among classes of 1.1%. The final design consisted of removing the shortcut, which reduced the number of parameters to 4.480 (4K). When compared to the previous model with shortcut, on average the sensitivity per class was 1.55% higher without the shortcut layer and 0.98% higher when compared to the 76K model (Fig.3).

The number of classes is then changed for the model with 4K parameters, adding more samples as well as adding a new class called *Other*. An additional dataset was used for this as described in Section 4.2, with the final classes being in Appendix.B. The usage of another dataset implied that the model would be trained on data sampled with different instruments. Hence the data that is used in training and testing is normalized. Normalization occurs at the input layer and follows Eq.4.

$$\text{new input} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{4}$$

Normalizing the input for both training and inference resulted in an increased accuracy per class of 3.28%, except for the *Other* class whose accuracy decreased. Further, by normalizing the inputs to a range between 0 and 1, this ensures the number is representable

when a different/smaller binary representation system is used.

The resulting model consisted of using bias for only the first two layers. As can be seen in Fig.1, two dense layers with 16 neurons each were used, with a ReLU activation function. For the last layer, a softmax function was used for classification. The kernel and bias regulizer for all dense function were L2, with the loss function being *Categorical Cross Entropy* and an Adam optimizer. The epochs and batch size were 256 and 64 respectively, however early stopping was used during training. The final loss and accuracy of the model can be seen in Fig.6.

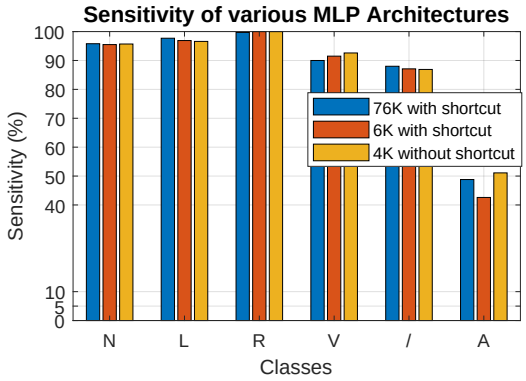


Fig. 3. Various MLP architectures described in Section 6.1, with their respective class sensitivity

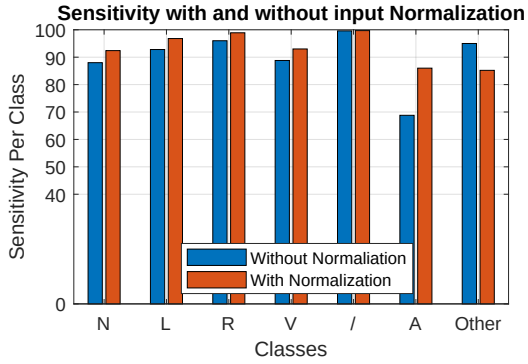


Fig. 4. The effects on class sensitivity when using a normalized and denormalized input.

7 HARDWARE

7.1 Fixed Point Representation

The MLP developed in Section 6.1 is only made up of dense layers followed by activation functions. While the activation functions will be covered in Section 7.2, using only dense layers limits the operations to only multiplication and addition. To represent inputs and parameters in hardware, a fixed point binary representation is used.

In a fixed point system, a number is split into three parts: sign, integer and fraction. The decimal point is fixed into position, X

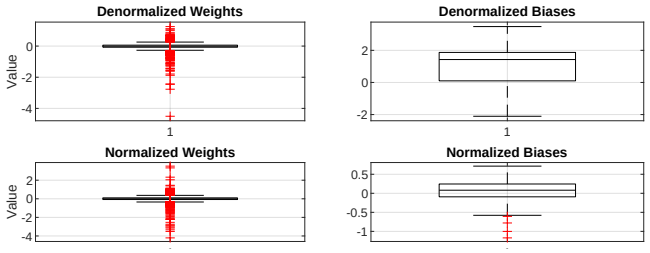


Fig. 5. The distribution of weights and biases for normalized and denormalized inputs. Normalizing the inputs reduces the range of the biases, while slightly increasing the range of the weights.

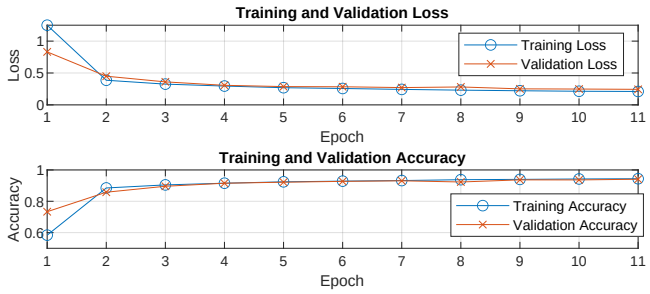


Fig. 6. Training metrics, loss and overall accuracy, for the final MLP architecture.

amount of bits are assigned to the integer part and Y amount to the fraction part and the most significant bit (MSB) is taken as the sign. Using a python script and the fxpmath module, the weights, biases and inputs are recalculated into their fixed point representation. Starting from 32 and going down to 3 bits, every combination of integer to fraction was calculated. A sample of 100 ECG's was used with each combination and the total accuracy excluding the Other class was recorded. This is because the Other class functions as the catch all, meaning it will have a high sensitivity.

Fig.7 shows the results of the average accuracy per combination. From it various conclusions can be made; 9 bits for the fraction is the minimum amount to keep the accuracy above 92%, whereas 3 bits for the integer part will have an average accuracy above 92%, overall 12 bits are required to satisfy these two conditions.

The final implementation aims to ensure that the model can be generalized to other datasets while maintaining the same accuracy and sensitivity, and also serving as a preliminary implementation. Therefore, a fixed-point representation of 16 bits, with 5 bits for the integer part, 10 bits for the fractional part, and 1 bit for the sign were taken forward.

7.1.1 Arithmetic Operations. Multiplication is implemented as follows; multiplying two 16-bit numbers will result in a 32 bit number. Initially, the decimal point is at the 10th bit for both numbers, the 32bit number will have it's decimal point at the 20th bit. The 10 bits before the decimal point are taken as the fraction part and the 5 bits after the decimal point, as the integer with the 32nd bit acting as the sign. Hardware implementation can be recursive by splitting

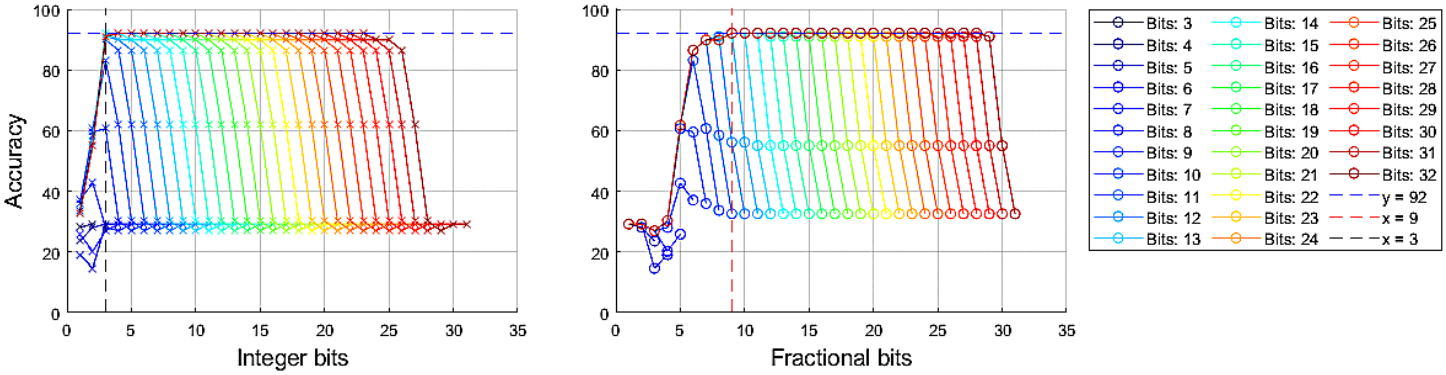


Fig. 7. Average accuracy excluding *Other* class, with varying the number of integer or fraction bits. Each color indicates the total number of bits excluding the sign bit.

and multiplying parts, then summing the shifted results [9][8]. This paper uses an existing accurate 16x16 bit multiplier [12].

On paper adding two 16-bit numbers can return at most a 17 bit number. Using 16 bits in two's complement, the representable range goes from -32768 to 32767. When the 17th bit is present, it indicates that the result is out of the 16 bit range, depending on the sign of the starting operands, the maximum or minimum value is set. An existing addition implementation is used [12].

7.2 Neuron

The neurons are the blocks that make up a dense layer, as seen in Fig.1 there are a total of 16, 16 and 7 neurons per layer. In total there are four parts to each neuron, a multiplication between the input and the weight, addition between the multiplication result and the previous multiplications and an addition of the bias. The final part, as seen in Fig.8 is the activation function, where the hidden layers make use of ReLU and the output layer uses SoftMax. ReLU is a simple comparison, if the input is greater than 0 then the result is passed. For a SoftMax function Eq.5 is used, where the output is the probability of class i and x_i is the result of neuron i.

$$o_i = \frac{e^{x_i}}{\sum_{j=1}^{7} e^{x_j}} \tag{5}$$

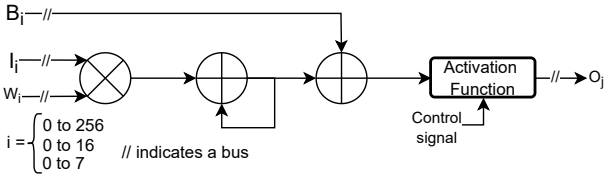


Fig. 8. Neuron architecture for the Multi Layer Perceptron used in this paper

With in mind the goal of reducing hardware resources, the most computationally intensive operations are implemented, where approximations can then be applied. Existing papers and implementation such as [16] and [17], present a way to reduce SoftMax into a precomputed look up table and a series of adders and dividers. In the end, 6 additions and 7 division take place. For the dense layers, when the numbers of SoftMax operation are take into account with Table 1, overall dense layers make up 99.9% of all the operations in the MLP model.

MLP Total Operations		
	Multiplication	**Addition**
Dense 1	4096	4096
Dense 2	256	256
Dense 3	112	105

Table 1. Number of operations per dense layer in MLP

7.3 Hardware Simulation

Using the code developed in [12] for the 16x16 bit multipliers and adders, Fig.8 is replicated in MATLAB. For the activation functions (ReLU and SoftMax) standard MATLAB code is used. The aim is to ensure that the simulation of the hardware returns the correct predictions for a 16 bit architecture. Once the simulated architecture is found to return good results, it will be implemented into hardware, with the subsequent step of applying approximations.

During implementation, the architecture was compared with the 64 bit implementation in python as it is more precise. For the 64 bit implementation the initial weights and inputs are set to 16 bit, multiplication and addition operations happened at 64 bits floating point. This implied that the result was a 32 bit number during multiplication, when compared to the 16 bit architecture which would result in 16 bit number. This 16 bit difference would affect the summation operations, as results between the two architecture could initially differ by a factor of one, then increasing in the following summations. The carry in, is an optional input to adders, which would add one the result when it is set to '1'. It was found that alternating the carry for each consecutive summation operations between 0 and 1 would reduce this difference. It is important to note that, when 1000 random ECG samples were used on the architecture that used carry and the architecture that had no carry, both architectures resulted in an overall accuracy above 92%.

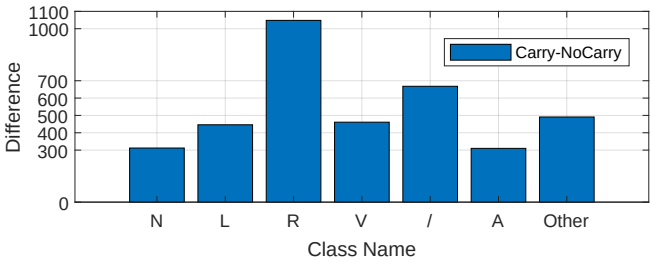


Fig. 9. Comparing the difference in average class error between using a carry and no carry architecture. 1000 random samples were used, where both architectures resulted in an average accuracy above 92%

7.4 Hardware Implementation

To implement the operations described in Fig.8, the designs takes 6 different signals: *input, weight, bias* and the remaining are control signals. The control signals are made up of the *reset*, *clock* and *start*, whose functions are to reset all the registers, provide a reference signal and start the arithmetic operations. The design involves using a state machine made up of two states, the first one is to set the carry, while the second sets the new summed value for the following summation. While the addition and multiplication will occur independently, from here the inputs have to be synchronized with the clock. In hardware overflow could only occur when the operands of an addition operation have the same sign. The result would be a number of the opposite sign, in which case the result is set to the maximum value for the operands' sign.

To test the final architecture, a test bench is written in VHDL. The algorithm used for testing is provided in Appendix C. Using only *one* neuron the inputs are fed in a consecutive manner. The output for each neuron is stored in the appropriate file for the layer. The output of one layer is then used as the input for the next layer. A comparison between hardware and software version is made. This is done by comparing the neuron outputs before the SoftMax activation function. Doing so ensures that the implemented hardware is as accurate as the software simulation. For ten random inputs it was found that the average discrepancy between hardware and software implementation was of 8.12% (265).

8 APPROXIMATIONS

As dense layers account for 99.9% of operations, approximating multiplications and additions will lead to the best results. From Table 1, layer one makes up 91.7% of multiplications and 91.9% of additions. Following [9], the most power-efficient 4x4 multiplier is used recursively in layer one, which introduces the most error. For layers two and three, a less efficient 4x4 multiplier is used to balance this introduction of error. Specifications for these multipliers, R_{1311} and R_{433A}, are found in [9]. For addition, the most area-efficient 1x1 adders are used, varying the number of bits approximated to minimize error. Summation in layer one approximates 2 bits, while layers two and three approximate 4 bits. Bias additions approximate 8 bits in all layers.

This architecture was first tested in MATLAB, to ensure that the approximations do not lower the sensitivity per class below the defined threshold. Subsequently it was implemented in VHDL. Using the same process as described in Section 7.4, the hardware was tested. Using ModelSim and Quartus, resource utilization and power consumption was recorded. The mentioned design was then tested with 1000 random samples.

9 RESULTS

On average the approximated implementation scored lower than the accurate version, with an average decrease of 2.14%, whilst for the *Other* class an increase of 3% occurred. Further approximations were not done as the *A class*, would fall beneath the set threshold. From here, the error introduced by the approximations was calculated by taking the difference for each neuron output before the softmax function. Tab.2 shows the average error for 1000 samples.

Fig. 10. The predictive accuracy per class between Approximated and Accurate in software equivalent

Class	Difference	Average	STD	Percentage
N	818.85	4464	1919	42.67
L	1035.821	5421	2138	48.44
R	1719.786	8696	2543	67.63
V	859.552	5955	2015	42.64
/	1653.759	11621	1482	111.61
A	613.997	3427	1161	52.90
Other	1357.8	6767	2852	47.60

Table 2. Per class, the difference indicates the average introduced error from the approximated architecture. The *Average* is the mean value of when a class is the correct prediction. The standard deviation (STD) indicates the amount that correct prediction varies from their mean class. The percentage is the relative introduced error between the Difference and STD column.

Table 2 shows increased variability per class due to the introduced error. For classes *R* and /, while the variability introduced was larger, the mean value per correct prediction was on average large compared to their standard deviation. This explains why there was a smaller decrease in their sensitivity compared to the other classes.

Both approximated architectures made use of fewer Look Up Table (LUT), due to the multipliers which reduce the output range [9]. This is evident by layer 1 making use of the least amount of arithmetic LUT, which reflects the usage of the most power efficient multiplier. The total reduction in pins for both architectures can be

Resource	Accurate	Layer 1	Layer 2 and 3
Combinational LUT	838	727	766
Arhitmetic LUT	377	352	377
Normal LUT	460	374	389
Total Register	35	35	35
I/O pin	89	83	83

Table 3. The total resources, the approximated components of layer 1, 2 and 3 are defined in Section 8.

Component	Total Power	Block Dyn.	Routing Dyn.
Multiplier	0.46	0.27	0.19
Adder Bias	0.03	0.02	0.01
Adder Sum	0.10	0.05	0.05

Table 4. Power readings (mW) for the accurate neuron, these readings include the power per component plus the power for subcomponents. Recorded at a clock frequency of 100 MHz, with the ECG as input where 50 instances were generated in the testbench.

Component	Total Power	Block Dyn.	Routing Dyn.
Multiplier	0.09	0.04	0.05
Adder Bias	0.00	0.00	0.00
Adder Sum	0.01	0.01	0.00

Table 5. Power readings (mW) for the approximated neuron in layer 1, these readings include the power per component plus the power for subcomponents. The multiplier would be a 16x16 made up of multiple R_{1311} multipliers. A reading of 0.00 indicates the power for that component is below the threshold for Quartus. Recorded at a clock frequency of 100 MHz, with the ECG as input where 500 instances were generated in the testbench.

Component	Total Power	Block Dyn.	Routing Dyn.
Multiplier	0.10	0.04	0.06
Adder Bias	0.00	0.00	0.00
Adder Sum	0.01	0.01	0.00

Table 6. Power readings (mW) for the approximated neuron in layer 2/3, these readings include the power per component plus the power for subcomponents. The multiplier would be a 16x16 made up of multiple R_{433A} multipliers. A reading of 0.00 indicates the power for that component is below the threshold for Quartus. Recorded at a clock frequency of 100 MHz, with the ECG as input where 500 instances were generated in the testbench.

attributed to the approximated adder, which was picked due to it's area reduction.

Dynamic power indicates consumption when active, which consists of: *routing power* from the interconnection of logic blocks and *block power* from active components. The readings in Table 4, 5 and 6 are the various dynamic readings for each of the neuron, recorded at a clock frequency of 100 MHz. These readings include the power for each component plus sub components, which can consist of registers, logic gates, memory elements and so on. Readings for individual components could return a value of 0.00 mW, that is a recorded value being below a certain threshold. This does not imply that said components have no energy consumption. By comparing the accurate against the approximated architectures it can be seen that the energy consumption for the multipliers decreased by 0.27 mW, while for the adders there was as decrease of 0.11 mW. The biggest change was in block power for the multipliers of 0.19 mW, while a difference of 0.08 mW for routing power was observed. This is to be expected as both approximated multipliers were chosen due to their power efficiency. The difference in power readings for the adders is seen to be 0.05 mW for block power and 0.06 mW for routing power. As 16x16 recursive multipliers are used, which are made up of four different 8x8 multipliers, during analysis it was seen that the 8x8 multiplier which took the lower end of the operands had a higher power consumption than the other three. For the accurate multiplier this was recorded at 0.28 mW, while for the approximated neurons it was at 0.06 and 0.07 mW. The reason for a higher power reading at this specific multiplier, is that it took as input the lower end of the operands, which had the highest concentration of 1.

10 LIMITATIONS

One limitation of the research described in this paper is the method of multiplication for fixed point numbers. This ignored the most significant bits of the fraction. Considering the lower 10 bits could potentially improve rounding error. Also the discrepancy between hardware and software implementation is of 8.12%, this value is negligible when compared to the standard deviation of Table 2, but may affect results when using a larger amount of classes. This difference seems to be caused by the hardware implementation of setting the carry to either 0 or 1. Future work should aim to reduce this discrepancy to 0.0%. Another limitation is the small samples used for Fig.7 or Fig.10, caused by the limited processing power available. For instance, Fig.7 required approximately 2000 minutes to process. Additionally, the types of classes used to train the model were purely based on the abundance of data for that class.

11 FUTURE WORK

For the task of arrhythmia detection, one area of work could be implementing a second model in the pipeline with the intentions of detecting variations in heart beat over time. One approach would be using a Time Convolutional Network or a Long short-term memory model. Another approach could be implementing a buffer that stores the past X recordings or information regarding the past X data, from which a comparison is made. Overall implementing an additional model would improve the usability and effectiveness of the device in a medical setting.

For the model developed in this paper, when considering Table 5, the majority of the weights are found close to zero. This could indicate that pruning is a viable operation for weights that are small, which would bring about a further decrease in model size. Another improvement could be the dataset used in training. Research should be done to develop a method to preprocess ECG from various datasets, into usable a format for model training. This would allow for rare arrhythmias with few data samples to be used in the

model. As the approximations used in this paper are for a 4x4 multipliers and the adders consider only one bit addition, greater benefit could be derived by using a 16x16 approximated multipliers, as well as considering other methods to approximate adders. Another area of research could be the use of highly approximated multipliers for certain parts of the inputs compared to others. This is because the approximations used in this paper assume a uniform distribution, when in reality more bits are assigned to the fraction part as these carry a higher value, compared to 5 bits for integer. This is further reinforced by Section 9, where the multiplier that takes the lower end of a number consumed more power than the other multipliers.

12 CONCLUSION

This study began by defining three sub-questions. The first sub-question was regarded the development of an appropriate model with a defined threshold for accuracy, sensitivity and latency. The developed model was a 4 layered Multilayer Perceptron which took the whole ECG of a heart beat as an input for arrhythmia detection, achieving a final accuracy of 94%, with per class sensitivity above the defined threshold of 78% and a model size of 8.75KB. The second sub-question addressed the implementation of the model onto a FPGA. As an MLP consisted of solely dense layers and activation functions, it was found that dense layers made up 99.9% of all operations, therefore implementation focused on dense layers. The components that make up a dense layer are the neurons, which consists of one multiplier and two adders, whose architecture are developed on a FPGA. The last sub-question investigated the usage of approximate computing to reduced hardware and power usage. Using energy efficient multipliers and area efficient adders, on average, the sensitivity per class decreased by 2.14% with class *A* being on the minimum threshold of 78%. Two approximated neurons were made, with the first one being found only in layer 1 and the second neuron for layer 2 and 3. Overall both architectures made use of 100 less LUT, where the reduction was in either the arithmetic or normal LUT. The dynamic power of both the multipliers and adders were recorded at a clock frequency of 100 MHz, the accurate neuron had a total dynamic power of 0.59 mW, while the summed total of both the approximated neurons was 0.21 mW. Using the same clock frequency of 100 MHz, an ECG input would be processed within 0.00021791 seconds. This time however does not include the time required to fetch intermediate layer output. Even so this time falls below the latency threshold of 1.27 seconds. Overall the results of this paper seem to indicate the feasibility of using a simple 4 layer based MLP for the task of arrhythmia detection in a whole heart beat ECG, providing a transition of the design in resource and power efficient manner.

REFERENCES

[1] June 2021. URL: https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[2] José Leal et al. "Economic burden of cardiovascular diseases in the enlarged European Union". In: *European Heart Journal* 27.13 (Feb. 2006), pp. 1610–1619. DOI: 10.1093/eurheartj/ehi733.

[3] Nurul Hikmah Kamaruddin, M. Murugappan, and Mohammad Iqbal Omar. "Early prediction of Cardiovascular Diseases using ECG signal: Review". In: (2012), pp. 48–53. DOI: 10.1109/SCOReD.2012.6518609.

[4] Awni Y. Hannun et al. "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network". In: *Nature Medicine* 25.1 (Jan. 2019), pp. 65–69. DOI: https://doi.org/10.1038/s41591-018-0268-3. URL: https://www.nature.com/articles/s41591-018-0268-3.

[5] Chen Zhang et al. "A Low-Power ECG Processor ASIC Based on an Artificial Neural Network for Arrhythmia Detection". In: *Applied Sciences* 13.17 (2023). ISSN: 2076-3417. URL: https://www.mdpi.com/2076-3417/13/17/9591.

[6] Jun Luo et al. "A Smartphone-Based M-Health Monitoring System for Arrhythmia Diagnosis". In: *Biosensors* 14.4 (2024). ISSN: 2079-6374. URL: https://www.mdpi.com/2079-6374/14/4/201.

[7] Rushik Parmar et al. "Design of DNN-Based Low-Power VLSI Architecture to Classify Atrial Fibrillation for Wearable Devices". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* PP (Mar. 2023), pp. 1–11. DOI: 10.1109/TVLSI.2023.3236530.

[8] Gillani Abbas et al. "MACISH: Designing Approximate MAC Accelerators with Internal-Self-Healing". In: *IEEE Access* (May 2019). DOI: 10.1109/ACCESS.2019.2920335.

[9] R.H. van der Wijk. "Designing 8-bit approximate multipliers for FPGA using internal self-healing". In: (Mar. 2023). URL: http://essay.utwente.nl/94789/.

[10] Ankur Agrawal et al. "Approximate computing: Challenges and opportunities". In: (2016), pp. 1–8. DOI: 10.1109/ICRC.2016.7738674.

[11] Amara Amara, Frédéric Amiel, and Thomas Ea. "FPGA vs. ASIC for low power applications". In: *Microelectronics Journal* 37.8 (2006), pp. 669–677. ISSN: 0026-2692. DOI: https://doi.org/10.1016/j.mejo.2005.11.003. URL: https://www.sciencedirect.com/science/article/pii/S0026269205003927.

[12] Muhammad Shafique et al. *Implementation Approximate Computing Library*. Mar. 2022.

[13] Ary L. Goldberger et al. "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals". In: *Circulation* 101.23 (2000). [Online], e215–e220.

[14] J. Zheng, H. Guo, and H. Chu. *A large scale 12-lead electrocardiogram database for arrhythmia study (version 1.0.0)*. https://doi.org/10.13026/wgex-er52. PhysioNet. 2022.

[15] J. Zheng et al. "Optimal Multi-Stage Arrhythmia Classification Approach". In: *Scientific Reports* 10 (2020).

[16] Mohamed Amr and contributors. *Verilog implementation of Softmax function*. https://github.com/maomran/softmax. Accessed: 2024-06-14. 2024. URL: https://github.com/maomran/softmax.

[17] Francesco Di Franco et al. "An Hardware Implementation of the Softmax Function on FPGA". In: (July 2020). VICOSYSTEMS S.r.l V.le Odorico da Pordenone, 33, Catania, CT, Italy. Department of Computer, Control, and Management Engineering, Sapienza University of Rome, Via Ariosto 25, Roma, Italy. ISSN: 1613-0073. URL: http://ceur-ws.org.

13 APPENDIX

A AI USAGE

During the preparation of this work the author used ChatGPT in order to preprocess ECG and binary files. After using this tool/service, the author takes full responsibility for the content of the work.

B CLASSES

Class	Name
N	Normal Beat
L	Left bundle branch block beat
R	Right bundle branch block beat
V	Premature ventricular contraction
/	Paced beat
A	Atrial premature beat

Table 7. The various classes used in the model with their respective

C NEURON TESTBENCH

Algorithm 1 Testbench for one Neuron

```
Input: NEURON, input, weight, L1, L2, L3, CK
weights = open(weight)
for i = 0 to 2 do
    if i == 0 then
        inputs = open(input, read)
        outputs = open(L1, write)
        forLoop = 15
    else if i == 1 then
        inputs = open(L1, read)
        outputs = open(L2, write)
        forLoop = 15
    else if i == 2 then
        inputs = open(L2, read)
        outputs = open(L3, write)
        forLoop = 7
    end if
    for j = 0 to forLoop do
        NEURON.rst = 1
        wait for 2× CK
        NEURON.rst = 0
        while inputs is not None do
            NEURON.input = inputs
            NEURON.weight, NEURON.bias = weights
            NEURON.STR = 1
            wait for CK
            NEURON.STR = 0
            wait for 2× CK
            if NEURON.output < 0 or i == 2 then
                write(output, NEURON.output)
            else write(output, '0')
            end if
        end while
    end for
end for
```