Energy-efficient ECG Signals Outlier Detection Hardware using a Sparse Robust Deep Autoencoder

Naoto Soga
Tokyo Institute of Technology
Tokyo Japan
soga@reconf.ict.e.titech.ac.jp

Shimpei Sato
Tokyo Institute of Technology
Tokyo Japan
satos@ict.e.titech.ac.jp

Hiroki Nakahara
Tokyo Institute of Technology
Tokyo Japan
Nakahara@ict.e.titech.ac.jp

Abstract— In recent years, portable electrocardiographs have begun to spread, which enable us to record electrocardiogram (ECG) signals in everyday life. A portable ECG analysis device is needed so that abnormal ECG waves can be detected anywhere. Machine learning techniques, including deep learning, are used in a lot of research to analyze ECG signals since they show more superb performance than conventional methods. However, deep learning models often have too many parameters to implement on mobile hardware. In this research, we propose a method to implement an ECG outlier detector using deep learning techniques in a small builtin device. As a way of detecting outliers, autoencoder, which is based on neural networks, was used. As a learning method, Robust Deep Autoencoder, which can learn unsupervisedly, was adopted. A sparseness technique was applied to the autoencoder, and the trained autoencoder was implemented on a low-end FPGA. Compared with ARM Cortex M3 embedded processor, the proposed hardware result in 159 times better for energy-efficiency improvement.

I. Introduction

An electrocardiogram (ECG) is the record of the electrical activity of the heartbeat. It is widely used to detect heart diseases. Recently, portable electrocardiographs, like Holter electrocardiography, are used in order to record ECG signals even when patients are not in hospitals. To support analyzing enormous ECG data obtained from these devices, software that can automatically analyze ECG signals are used. Recently, machine learning techniques are proposed to analyze ECG automatically. Many of them use a deep convolutional neural network, which is showing high performance in image processing. In this case, because the calculation load increase, the situations these researches suppose are mainly limited as follows.

1). Analyze ECG data after measured by portable devices[10].

2). Portable devices send data to servers that can automatically analyze ECG signals[11].

In the case 1), for ECG signals are analyzed after they are recorded, patients can't know whether abnormal signals are detected while the recording. In the case 2), when connectivity of communication is not ensured, like when the patients are in the car, it is impossible to alert them on the spot that heart diseases develop. In this paper, we propose the small and energy-efficient hardware that can automatically analysis ECG signal on mobile devices so that abnormal waves can be detected at all times.

For the mobile hardware, one way to implement the ECG outlier detector is to use a general-purpose embedded processor. However, it may have unnecessary modules and consume much area and power on a portable device. Also, a general-purpose embedded processor is often over spec for real-time ECG signals processing, which needs to process only from 70 to 90 beats per minutes in general. To minimize the power consumption of the ECG outlier detector embedded on portable hardware, implementing an application-specific circuit is the best way.

As an outlier detection method, we used an autoencoder, one of the neural network models. When we implement the autoencoder on small hardware, it is problematic to store a large number of weight parameters on them. Since the DRAM accesses are energy-costly[9], it is appropriate to store all the parameters in on-chip memory. To reduce the volume of weight parameters small enough to store in on-chip memory, we applied a sparseness technique to the autoencoder. Also, in the paper, we propose an architecture for the sparseness weight autoencoder on an FPGA.

The proposed design flow is shown in Fig.1. First, an autoencoder is trained by ECG data with Robust Deep Autoencoder (RDA). At this time, it is not necessary to prepare target labels beforehand, and outlier values may be included, since RDA can train the autoencoder unsupervised. Next, a sparseness technique is applied to trained autoencoder. Repeat the process of training and applying a sparseness technique while the number of weight parameters reaches the target value. When the pruning process complete, convert the floating-point pa-

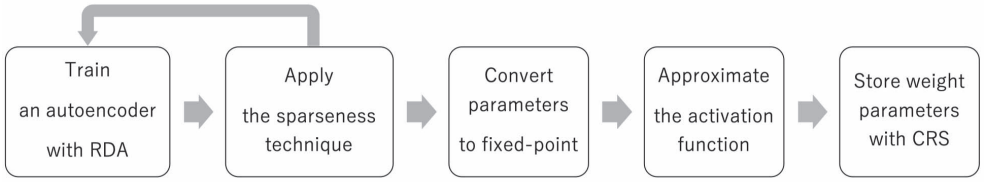

Fig. 1. Proposed design flow

rameters into fixed point one. At the same time, approximate the activate function to the extent that the outlier detection performance does not deteriorate. Finally, obtained weight parameters are converted into Compressed Row Storage format (CRS), which is one of the ways to store a sparse matrix. Then, implement the autoencoder on an FPGA.

Our main contributions are as follows:

- We proposed the design flow to implement the outlier detector using an autoencoder on a low-end FPGA. To shorten the preparing time of ECG data used in training an autoencoder, Robust Deep Autoencoder, one of the unsupervised learning techniques, was applied. Also, to minimize the volume of weight parameters, a sparseness technique was applied and all the parameters are converted to fixed point values. We show that even if the parameters were reduced and they are calculated in fixed point values, the outlier detection performance degradation is only 0.84%. By reducing the volume of the weight parameters, it was able to store all the parameters in on-chip memory.
- To the best of our knowledge, this work is the first implementation of the outlier detector using the sparseness weight autoencoder on an FPGA. We designed the architecture based on CRS format, which is the well-known data structure of a sparse matrix, to make the hardware size as small as possible and save power consumptions.
- We implemented the autoencoder on the Digilent Inc. ZedBoard and compared with the CPU for a built-in device. The result showed that FPGA implementation of outlier detector is 17.8 times faster and 159 times better energy-efficiency.

II. Related Works

Many of the previous researches of ECG automatic analysis using deep learning techniques is focused on diagnosis classification[10][11]. Though they show very high classification accuracies, they use large CNN models, or apply many preprocessing methods, which both result in high computation cost and is not suited for this research's goal. There are researches focusing on ECG anomaly detection[6][3]. There are also attempts to implement ECG outlier detector on FPGA[8]. In this research, we use unsupervised learning technique to train the autoencoder, while most researches use supervised one. We evaluate how the performance of outlier detector on FPGA changes when the sparseness technique and bit precision modification are applied to the autoecoder and calculation cost is reduced.

Note that this research is mainly focused on the methods to efficiently implement the autoencoder outlier detector on FPGA. How to preprocess signals and extract features to achieve high accuracy rate are not discussed.

III. Robust Deep Autoencoder (RDA)

Outlier detection is the process of detecting extremely deviated data from dataset whose data are almost normal. As a method of outlier detection, an autoencoder is well known. It is one of the neural network models which uses input data as teacher data to the output. When an autoencoder is trained by only normal data, it can't reconstruct data correctly. We can detect outliers by whether the autoencoder can reconstruct data correctly or not. In this case, we have to precisely extract only normal data from dataset, since autoencoder cannot be trained well when there are abnormal data in a training dataset. In order to solve this problem, Robust Deep Autoencoder (RDA)[12] is proposed. RDA is the unsupervised training method that can train autoencoder with data including anomalies.

A. Autoencoder

An autoencoder consists of an encoder, which convert inputs into low-level matrix, and a decoder, which reconstructs the inputs. Let E be an encoder, D be a decoder, X be inputs. Then, outputs $\overline{X}$ can be expressed as follows:

$$\overline{X} = D(E(X)).$$

An autoencoder is trained to minimize the difference between X and $\overline{X}$. This process is equivalent to solve the optimization problem as follows:

$$\min_{D,E} \|X - D(E(X))\|_2,$$

where $\|\cdot\|_2$ is the l_2 norm. In this research, the fully-connected neural network is used in the autoencoder.

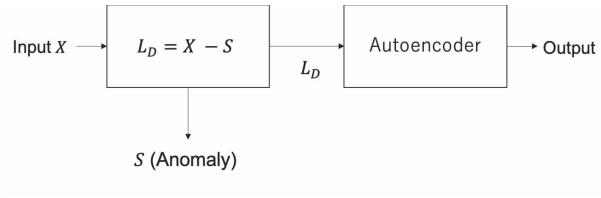

Fig. 2. Training process of RDA

B. Robust Principal Component Analysis

Robust Principal Component Analysis (RPCA) is a generalization of Principal Component Analysis (PCA), which reduces sensitivity to outliers. As shown in Expr.(1), the main idea is to split input data X into the main component L and a sparse matrix S which includes anomalies.

$$X = L + S \tag{1}$$

RPCA is considered as an optimization problem as follows:

$$\min_{L,S} \ \rho(L) + \lambda \cdot count\_nonzero(S), \tag{2}$$
$$s.t. \ \|X - L - S\|_F^2 = 0,$$

where $\rho(L)$ is the rank of L, $count\_nonzero(S)$ is the number of nonzero values of S, and $\|\cdot\|_F$ is the Frobenius norm.

Optimization of Expr.(2) is NP-Hard problem, since it is a non-convex optimization. To solve this, convex relaxations are applied as follows:

$$\min_{L,S} \ \|L\|_* + \lambda\|S\|_1, \tag{3}$$
$$s.t. \ \|X - L - S\|_F^2 = 0,$$

where $\|\cdot\|_1$ is the l_1 norm, $\|\cdot\|_*$ is the nuclear norm.

C. Robust Deep Autoencoder

In RDA, the nuclear norm in Expr.(3) is replaced with an autoencoder. When an autoencoder is trained by RDA, anomalies S are removed from inputs X, and remained data L_D are learned by an autoencoder, as shown in Fig.2.

Let W be weight parameters in the neural network, b_e and b_d be the biases, and $logit(\cdot)$ be the activation function. We define the encoder E and the decoder D are defined as follows:

$$E_\theta(x) = E_{w,b}(x) = logit(Wx + b_E), \tag{4}$$

$$D_\theta(x) = D_{w,b}(x) = logit(W^T E_{W,b}(x) + b_D). \tag{5}$$

From above expressions, RDA is considered as the optimization problem shown as follows:

$$\min_{\theta} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S\|_1, \tag{6}$$
$$s.t. \ X - L_D - S = 0.$$

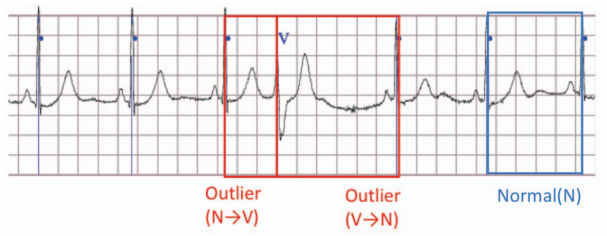

Fig. 3. Examples of ECG signals

The nuclear norm can be regarded as a linear mapping to a low dimension matrix, whereas an autoencoder as a non-linear mapping[12].

Expr.(6) removes anomalies of each element in input data. To remove outliers in inputs, replace the l_1 norm with the $l_{2,1}$ norm, then we have:

$$\|S\|_{2,1} = \sum_{j=1}^{N} \sqrt{\sum_{i=1}^{N} |s_{ij}|^2}$$

Finally, RDA removing outliers is regarded as the optimization problem shown in (7)

$$\min_{\theta,S} \|L_D - D_\theta(E_\theta(L_D))\|_2 + \lambda\|S\|_{2,1}, \tag{7}$$
$$s.t. \ X - L_D - S = 0$$

It removes outliers from input data, and train an autoencoder with only principal data, that is, normal data.

D. ECG Data

As a training data used in RDA, a dataset made from MIT-BIH Arrhythmia Database[7] was used. MIT-BIH Arrhythmia Dataset contains ECG recordings, positions data of R-waves, and diagnosis annotations for each beat. The baseline oscillation was removed from raw data by using the polynomial approximation[2]. Then, the data was cut from the R wave to the next R wave as one cycle, the length was resized to 360, and normalization was performed so that the data had a value of 0 to 1. Among many channels ECG has, lead II was used in this research.

As outliers, premature ventricular contraction (PVCs) was used. The outlier data was collected from the ECG signals which have PVCs (106.dat). Only PVC data were cut from the data. Note that diagnosis annotations were added to each R waves. Therefore, in the case the data between the R wave to the next R wave was defined as a cycle, there are three patterns of outliers; normal-PVC, PVC-normal, PVC-PVC (Fig.3). The normal beats data were collected from different three ECG data (100.dat, 103.dat, 115.dat). Only normal beats were cut from the data. Thus, outlier and normal data collected from four different data were combined and made a data set which has 6073 normal beats, and 332 outliers.

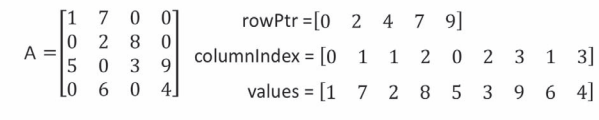

Fig. 4. Compressed Row Strage (CRS)

IV. Lightweight Autoencoder

In order to minimize the number of weight parameters, the sparseness technique was applied. The sparseness technique eliminates the unnecessary weight to reduce the number of weight parameters. Typically, unnecessary weight is close to 0, and they are handled as 0. When the neural network is implemented on hardware, the volume of memories should be reduced, since there is no need to store the "zero" parameters. The sparseness technique is defined as Expr.(8). Let w be the weight parameters, Th be the threshold value, w_{sp} be the weight parameters after pruning, and $|\cdot|$ be the absolute value. Then we have,

$$w_{sp} = \begin{cases} 0 & (|w| \leq Th) \\ w & (|w| > Th). \end{cases} \tag{8}$$

In this research, the weight parameters in a trained autoencoder whose absolute values are below the threshold value are replaced with 0. After that, the autoencoder is retrained. This process is repeated for several times[4]. When trained repeatedly, neural networks may be overfitted. To avoid this, the number of iterations and the learning rate were gradually reduced, and the F1 score was kept as high as possible. While training using RDA, the matrix S (Fig. 2, and Expr.(7)) should be stored. When the training is executed repeatedly, the matrix S in the previous training is succeeded to the next training.

V. Hardware Implimentation

A. Sparse Weight Representation and Computation

Weight parameters of the autoencoder which the sparseness technique is applied considered as a sparse matrix whose elements are almost zero. In this research, Compressed Row Storage (CRS)[5] was used. In the CRS, as shown in Fig.4, sparse matrix data is stored in three arrays. *values* is the value of the non-zero elements in a sparse matrix, *columnIndex* is the entry for each of non-zero elements, and *rowPtr* stores the index of the first element in the row. The indexes of rows and columns of non-zero elements correspond to the index of the input and the output of the autoencoder. For example, if the index of a weight parameter is (i, j), the index of the input node is "i", and the output node is "j".

In this research, the weight parameter W used in the encoder is transposed and used in the decoder, as shown in the Expr.(4) and (5). When transposing the matrix, the (i, j) component $(W)_{ij}$ of the original matrix W becomes $(W^T)_{ji}$ which replaced the indexes i and j in the transposed row W^T.

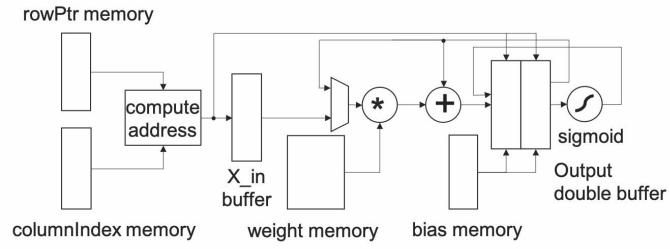

Fig. 5. Architecture overview

The calculation procedure of the encoder implemented on an FPGA is shown in Fig.1. Let $N1$ be the number of input nodes, $N2$ be the number of output nodes, and $Ndata$ be the number of non-zero elements of weight parameters.

Practically, the calculation of Algorithm1 is repeated by the number of the encoder layers. The output *out* is stored in the buffer and used as the output to the next layer. In the decoder, the weight parameter used in the encoder is transposed. This can be realized by swapping the input and output of the node, practically, swapping *input_index* and *output_index*.

The architecture of an autoencoder used in this research is shown in Fig.5. Parameters *weight*, *rowPtr*, *columnIndex*, and *bias* are stored in memories.

B. Prototype Implemetation by an FPGA

When weight parameters and biases are implemented with floating points, floating point addition and multiplication will be executed. These floating-point calculations are time-consuming computation and greatly affect the calculation execution speed. Also, hardware resource consumptions, such as DSP blocks, may increase. In this research, all parameters are converted into a fixed point and implemented on an FPGA.

The sigmoid function also may consume a lot of hardware resources, since it has divisions and index calculations. It is approximated as the expression follows:

$$y = \begin{cases} 0 & (x \leq -4) \\ 0.05x + 0.2 & (-4 < x \leq -2) \\ 0.2x + 0.5 & (-2 < x \leq 2) \\ 0.05x + 0.8 & (2 < x \leq 4) \\ 1 & (4 < x). \end{cases}$$

VI. Experimental Results

A. Evaluation Method

The F1 score is used for the evaluation of the outlier detection. It is the harmonic mean of the recall and the

Algorithm 1 The algorithm for the encoder

INPUT:$X \in R^{N1}, data \in R^{Ndata}$,
$bias \in R^{N2}, rowPtr \in R^{N1+1}, columnIndex \in R^{Ndata}$
1.Initialize the value of *out* with *bias*
for i in 0 to N2 **do**
  $out[i] \leftarrow bias[i]$
end for
2.Compute the indexes of the input and output node, and execute the multiplication
for i in 0 to N2 **do**
  for j in rowPtr[i] to rowPtr[i+1] **do**
    $input\_index \leftarrow columnIndex[j]$
    $output\_index \leftarrow i$
    $w \leftarrow data[j]$
    $x \leftarrow X[input\_index]$
    $out[output\_index]$
    $\leftarrow out[output\_index] + w \times x$
  end for
end for
3.Calculate the activation function and output the result
for i in 0 to N2 **do**
  $out[i] \leftarrow logit(out[i])$
end for

precision. Note that in this research, positive data is "outlier", since the ability to detect outliers is evaluated. To evaluate the result data, cross-validation is used.

B. Training using RDA

The layer sizes of the autoencoder are 360,200,and 100. From the used ECG dataset, input and output size are set to 360. The initial values of weight parameters are set by random numbers following the normal distribution, and biases are set by random numbers following the uniform distribution. The hyperparameter λ(shown in Expr.(7)) in RDA is set to the value which shows the highest F1 score when detecting outliers while training. Also, the sparseness technique was applied while training the autoencoder using RDA.

C. Implementation on an FPGA

The autoencoder trained by RDA is used to detect ECG outlier data. The mean square error (MSE) of inputs and outputs is calculated, and if the value exceeds the certain threshold, the data is judged as an outlier. This threshold is set to the value of the smallest MSE among ECG wave data which is judged as an outlier while training.

D. Environment Setup

The RDA was implemented with Chainer Deep learning Framework 4.0.0[1]. Then, the autoencoder trained by RDA was implemented on Xilinx ZedBoard Zynq-7000 Development Boardby using Xilinx SDSoC 2018.2. The timing constraint is set to 100MHz. Also, Arduino DUE was used for comparison. The autoencoder with sparseness weight and fixed-point parameters are implemented with software and executed on ARM CortexM3 CPU, which is mounted on Arduino DUE board.

TABLE I
COMPARISON OF RECOGNITION ACCURACY

	not pruned	pruned
precision	0.958	0.950
recall	0.958	0.952
F1 score	0.957	0.949

TABLE II
THE NUMBER OF WEIGHT PARAMETERS

	not pruned	pruned
layer 1	72000	4450
layer 2	20000	578
total	92000	5028
ratio	1.00	0.05

E. Comparison of Sparseness with Non-Sparseness (Dense) Autoencoder

Table I shows precisions, recalls, and F1 scores of the predictions by autoencoders. Autoencoders the sparseness technique was applied to and not were compared. Table II shows the comparison of the number of weight parameters. As for a dense weight autoencoder, it has 92000 parameters, and the F1 score was 0.957. When the sparseness technique was applied, 94% of the number of weight parameters were eliminated. In this case, the F1 score was 0.949, which is only 0.84% below the score of the dense autoencoder.

F. Comparison of F1 Scores for Different Parameters

Table III compares precisions, recalls, and F1 scores of the predictions of implemented autoencoder. The F1 score didn't change when parameters are stored with the 32-bit floating point, 16-bit floating point, and fixed-point numbers. When the parameters are stored with fixed-point numbers, the bit precision of calculations in the prediction process is set to 16-bit float to keep the F1 score. In the case implemented with fixed-point numbers, the volume of weight parameters was reduced compared to floating-point ones, since the bit width of parameters can be changed for each memory.

G. Hardware Resource consumption

Table IV shows the hardware resource consumption when the autoencoder is implemented on an FPGA. Reducing the number of weight parameters by applying the sparseness technique, the consumption of BRAM was greatly reduced, and all the parameters of autoencoder was able to be stored in on-chip memory (BRAM). By converting parameters into fixed-point numbers, the BRAM consumptions were more reduced. From now on,

TABLE III
THE WAY OF STORING WEIGHT PARAMETERS.("FIXED POINT" IS ARBITRARY PRECISION FOR EACH LAYER)

	Sparseness			Dense
Bit precision	32bit float	16bit float	fixed point	32bit float
Ratio of weight parameters	0.05			1.00
Precision	0.950	0.950	0.950	0.958
Recall	0.952	0.952	0.952	0.957
F1 score	0.949	0.949	0.949	0.957
Non-zero parameters[kbyte]	20.1	10.1	7.78	368.0
Ratio	0.0546	0.0274	0.0211	1.0000

TABLE IV
HARDWARE CONSUMPTION (() DONATES THE UTILIZATION RATIO. "SPEED" IS THE NUMBER OF WAVE FORMS (R WAVE TO R WAVE) THAT CAN BE PROCESSED PER SECOND)

	16bit float	fixed point
BRAM	19.5(13.93%)	17(12.14%)
DSP	10 (4.55%)	8 (3.64%)
FF	9245(8.69%)	9225(8.67%)
LUT	5863(11.02%)	6375(11.98%)
Speed [waves/s]	1274	2745

the parameters of the autoencoder is implemented with fixed point one.

H. Power Consumption

Table V shows the power consumption and the total power consumption of the FPGA and the CPU. The autoencoder with sparseness weight and fixed-point parameters are implemented. Here, total power consumption is the power consumption multiplied by the execution time shown in TableV. The autoencoder implemented on an FPGA can operate with 11% power consumption compared to ARM Cortex M3 CPU. In addition, the total power consumption is 0.63% of that of CPU (159 times better as for the energy consumption), since the FPGA can execute the prediction much faster than CPU, and it has limited processing units for the autoencoder.

VII. CONCLUSION

In this research, an autoencoder is trained unsupervised by using Robust Deep Autoencoder. The sparseness technique was also applied and the number of weight parameters was reduced. The volume of weight parameters was reduced by 97.9% while suppressing the decrease of the F1 score to only 0.84%. The trained autoencoder was implemented on Xillinx ZedBoard Zynq-7000 Development Board. By using CRS format, adjusting bit precisions, and approximating the activate function, the hardware resource consumption was small enough to be implemented on the low-end FPGA. Also, compared to the CPU implementation, the autoencoder implemented on an FPGA can operate with 11% power consumption. Since the architecture has minimum processing units possible to perform the outlier detection using an autoencoder, the proposed sparseness hardware is 159 times better as for its energy consumption, compared to the CPU implementation. It can be said that it was able to design with high power efficiency and suitable for mounting on mobile hardware.

TABLE V
POWER CONSUMPTION

	FPGA	CPU
Power consumption[W]	0.072	0.65
Execution time[ms]	467	8330
Total power consumption[J]	0.034	5.400

ACKNOWLEDGEMENTS

This research is supported in part by the Grants in Aid for Scientistic Research of JSPS, and the New Energy and Industrial Technology Development Organization (NEDO). Also, thanks to the Xilinx University Program (XUP), Intel University Program, and the NVIDIA Corp. ' s support.

REFERENCES

[1] Chainer v4.0.0. https://docs.chainer.org/en/v4.0.0/.

[2] Peak analysis. https://jp.mathworks.com/help/signal/examples/peak-analysis.html?lang=en.

[3] A. Gharaviri, M. Teshnehlab, and H. A. Moghaddam. Pvc arrhythmia detection using neural networks. In *2007 5th International Symposium on Image and Signal Processing and Analysis*, pages 234–237, Sep. 2007.

[4] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015.

[5] Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer. Parallel programming for fpgas. *CoRR*, abs/1805.03648, 2018.

[6] Hela Lassoued and Ketata Raouf. Artificial neural network classifier for heartbeat arrhythmia detection. 03 2017.

[7] MIT-BIH arrhythmia database. https://physionet.org/physiobank/database/mitdb/.

[8] D. J. M. Moss, Z. Zhang, N. J. Fraser, and P. H. W. Leong. An fpga-based spectral anomaly detection system. In *2014 International Conference on Field-Programmable Technology (FPT)*, pages 175–182, Dec 2014.

[9] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel S. Emer, Stephen W. Keckler, and William J. Dally. SCNN: an accelerator for compressed-sparse convolutional neural networks. *CoRR*, abs/1708.04485, 2017.

[10] Pranav Rajpurkar, Awni Y. Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y. Ng. Cardiologist-level arrhythmia detection with convolutional neural networks. *CoRR*, abs/1707.01836, 2017.

[11] A. Walinjkar and J. Woods. Personalized wearable systems for real-time ecg classification and healthcare interoperability: Real-time ecg classification and fhir interoperability. In *2017 Internet Technologies and Applications (ITA)*, pages 9–14, Sept 2017.

[12] Chong Zhou and Randy C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 665–674, New York, NY, USA, 2017. ACM.