Energy-Efficient ECG Signals Outlier Detection Hardware Using a Sparse Robust Deep Autoencoder

Naoto SOGA\(^{\text{a)}}\), Nonmember, Shimpei SATO\(^{\text{b)}}\), and Hiroki NAKAHARA\(^{\text{c)}}\), Members

SUMMARY  Advancements in portable electrocardiographs have allowed electrocardiogram (ECG) signals to be recorded in everyday life. Machine-learning techniques, including deep learning, have been used in numerous studies to analyze ECG signals because they exhibit superior performance to conventional methods. A mobile ECG analysis device is needed so that abnormal ECG waves can be detected anywhere. Such mobile device requires a real-time performance and low power consumption, however, deep-learning based models often have too many parameters to implement on mobile hardware, its amount of hardware is too large and dissipates much power consumption. We propose a design flow to implement the outlier detector using an autoencoder on a low-end FPGA. To shorten the preparation time of ECG data used in training an autoencoder, an unsupervised learning technique is applied. Additionally, to minimize the volume of the weight parameters, a weight sparseness technique is applied, and all the parameters are converted into fixed-point values. We show that even if the parameters are reduced converted into fixed-point values, the outlier detection performance degradation is only 0.83 points. By reducing the volume of the weight parameters, all the parameters can be stored in on-chip memory. We design the architecture according to the CRS format, which is the well-known data structure of a sparse matrix, minimizing the hardware size and reducing the power consumption. We use weight sharing to further reduce the weight parameter volumes. By using weight sharing, we could reduce the bit width of the memories by 60% while maintaining the outlier detection performance. We implemented the architecture on a Digilent Inc. ZedBoard and compared the results with those for the ARM mobile CPU for a built-in device. The results indicated that our FPGA implementation of the outlier detector was 12 times faster and 106 times more energy-efficient. 

key words:  FPGA, autoencoder, outlier detection, unsupervised training

1. Introduction

An electrocardiogram (ECG) is the record of the electrical activity of the heartbeat. ECGs are widely used to detect heart diseases. Recently, portable electrocardiographs, e.g. Holter monitors, have been used to record ECG signals even when patients are not in hospitals. To support the analysis of large amounts of ECG data obtained from these devices, software that can automatically analyze ECG signals is used. Recently, machine-learning techniques have been proposed for analyzing ECG signals automatically. Many of them use a deep convolutional neural networks, which exhibit high performance for image processing. In this case, because of the large calculation load, these studies are limited to the following situations.

1). ECG data measured using portable devices are analyzed [1].
2). Portable devices send data to servers that can automatically analyze ECG signals [2].

In case 1), for ECG signals that are analyzed after they are recorded, patients cannot know whether abnormal signals are detected during the recording. In case 2), when the connectivity of communication is not ensured, e.g. when the patients are in the car, it is impossible to alert them to the development of heart diseases. In this paper, we propose the small and energy-efficient hardware that can automatically analyze ECG signals 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 a large area and a large amount of power on a portable device. Additionally, a general-purpose embedded processor is often superfluous for real-time ECG signal processing, which generally requires processing only from 70–90 beats per minute. Implementing an application-specific circuit is the best way to minimize the power consumption of the ECG outlier detector embedded in portable hardware.

As an outlier detection method, we use an autoencoder, which is a neural-network model. When we implement the autoencoder on small hardware, storing a large number of weight parameters on device is challenging. Because DRAM accesses are energy-intensive [3], storing all the parameters in on-chip memory is appropriate. We apply a sparseness technique to the autoencoder to reduce the number of weight parameters to a number that can be stored in on-chip memory. 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 using ECG data with a robust deep autoencoder (RDA). It is not necessary to prepare target labels beforehand, and outlier values may be included, because RDA can train the autoencoder unsupervised. Next, a sparseness technique is applied to the trained autoencoder. The training process is repeated, and a sparseness technique is applied, while the number of weight parameters reaches the target value. When the pruning process is complete, the floating-point parameters are converted into fixed-point parameters. The activation function is approximated to the ex-
tent that the outlier detection performance does not deteriorate. Additionally, weight sharing is applied, the parameter volume is reduced. Finally, the obtained weight parameters are converted into a compressed row storage (CRS) format, which is one of the ways to store a sparse matrix. Then, the autoencoder is implemented on an FPGA.

The main contributions of this paper are as follows:

- We propose a design flow to implement the outlier detector using an autoencoder on a low-end FPGA. To shorten the preparation time of ECG data used in training an autoencoder, an RDA—an unsupervised learning technique—is applied. Additionally, to minimize the volume of the weight parameters, a sparseness technique is applied, and all the parameters are converted into fixed-point values. We show that even if the parameters are reduced converted into fixed-point values, the outlier detection performance degradation is only 0.84 points. By reducing the volume of the weight parameters, all the parameters can be stored in on-chip memory.

- To the best of our knowledge, this work is the first implementation of an outlier detector using the sparseness weight autoencoder on an FPGA. We design the architecture according to the CRS format, which is the well-known data structure of a sparse matrix, minimizing the hardware size and reducing the power consumption.

- We use weight sharing to further reduce the weight-parameter volumes. By using weight sharing, we can reduce the bit width of the memories by 60% while maintaining the outlier detection performance.

- We implemented the autoencoder on a Digilent Inc. ZedBoard and compared the results with those for the CPU for a built-in device. The results indicated that the FPGA implementation of the outlier detector was 12 times faster and 106 times more energy-efficient.

This paper is built on the past publication [4].

2. Related Works

2.1 Deep Learning ECG Analysis

Much previous studies on ECG automatic analysis using deep-learning techniques have focused on diagnosis classification [1], [2]. Although high classification accuracies were achieved, large convolutional neural network models or apply many preprocessing were applied, which incurred a high computational cost; thus, these approaches were not suitable for achieving the goal of this study. Some studies have focused on ECG anomaly detection [5], [6]. There have also been attempts to implement ECG outlier detector on an FPGA [7]. In this study, we used an unsupervised learning technique to train the autoencoder, whereas a supervised learning technique has been used in most studies.

We evaluated the effects of applying the sparseness technique and bit precision modification to the autoencoder on the performance of the outlier detector on the FPGA.

This study mainly focused on the tiny autoencoder outlier detector on an FPGA. The preprocessing of signals and the extraction of features to achieve a high accuracy were not investigated herein.

2.2 Compression Techniques for Deep-Learning Models

Deep-learning models require several MAC operations, which results in a high computational complexity, a decrease in speed, and an increase in power consumption. It is challenging to use deep learning models on mobile devices, where the power consumption and hardware resources are heavily restricted. To reduce the calculation loads, various compression methods have been proposed, such as distillation, convolutional operation decomposition, the sparseness technique, and quantization [8].

Weight pruning reduces MAC operations by eliminating excessive weight parameters. Because the eliminated weight parameters can be replaced with zero, and the calculation can be skipped, the overall calculation and weight-parameter capacity can be reduced. Additionally, the power consumption can be expected to be reduced by reducing the number of memory accesses [9]. With ImageNet, approximately 90% of the weight parameters can be reduced in AlexNet, and approximately 93% can be reduced in VGG16 without significant accuracy deterioration [10].

We applied weight pruning to the autoencoder for outlier detection. Although supervised training was used to perform weight pruning in most previous studies, we used unsupervised training. To the best of our knowledge, this was the first study in which weight pruning was used for unsupervised autoencoder training.

3. Robust Deep Autoencoder (RDA)

Outlier detection is the process of detecting extreme deviations of data in a dataset whose data are largely normal. The
autoencoder represents a well-known method of outlier detection. It is a neural-network model that uses input data as teacher data to the output. When only normal data are used to train an autoencoder, it cannot reconstruct the data correctly. We can detect outliers by determining whether the autoencoder can reconstruct data correctly. In this case, we must precisely extract only normal data from the dataset, because the autoencoder cannot be trained well when there are abnormal data in a training dataset. To solve this problem, an RDA [11] is proposed. RDA is an unsupervised training method that can train an autoencoder with data, including anomalies.

3.1 Autoencoder

An autoencoder consists of an encoder, which converts inputs into a low-level matrix, and a decoder, which reconstructs the inputs. Let \(E\) be an encoder, \(D\) be a decoder, \(X\) be inputs. Then, the output \(\bar{X}\) can be expressed as follows:
\[
\bar{X} = D(E(X)).
\]
An autoencoder is trained to minimize the difference between \(X\) and \(\bar{X}\). This is equivalent to solving the following optimization problem:
\[
\min_{D,E} \|X - D(E(X))\|_2,
\]
where \(\| \cdot \|_2\) represents the \(l_2\) norm. In this study, a fully-connected neural network was used in the autoencoder.

3.2 Robust Principal Component Analysis (RPCA)

RPCA is a generalization of principal component analysis (PCA) that reduces the sensitivity to outliers. As shown in Expr. (1), the main idea is to split the input data \(X\) into the main component \(L\) and a sparse matrix \(S\) which includes anomalies.
\[
X = L + S
\]  
(1)

RPCA is considered an optimization problem, as follows:
\[
\min_{L,S} \rho(L) + \lambda \cdot \text{count}_{\text{nonzero}}(S),
\]
\[
\text{s.t.} \quad \|X - L - S\|_F^2 = 0,
\]
where \(\rho(L)\) represents the rank of \(L\), \(\text{count}_{\text{nonzero}}(S)\) represents the number of nonzero values of \(S\), and \(\| \cdot \|_F\) represents the Frobenius norm.

Optimization of Expr. (2) is an NP-Hard problem because it is a non-convex optimization. To solve this problem, convex relaxations are applied, as follows:
\[
\min_{L,S} \|L\|_1 + \lambda \|S\|_1,
\]
\[
\text{s.t.} \quad \|X - L - S\|_F^2 = 0,
\]
where \(\| \cdot \|_1\) represents the \(l_1\) norm, and \(\| \cdot \|_F\) represents the nuclear norm.

3.3 Robust Deep Autoencoder (RDA)

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

Let \(W\) represent the weight parameters in the neural network, \(b_e\) and \(b_d\) represent the biases, and \(\logit(\cdot)\) represents the activation function. We define the encoder \(E\) and the decoder \(D\) as defined as follows:
\[
E_{\theta}(x) = E_{w,b}(x) = \logit(W^TX + b_e),
\]
\[
D_{\theta}(x) = D_{w,b}(x) = \logit(W^TE_{w,b}(x) + b_d).
\]

Accordingly, the RDA is considered as the following optimization problem:
\[
\min_{\theta} \|L_D - D_{\theta}(E_{\theta}(L_D))\|_2 + \lambda \|S\|_1,
\]
\[
\text{s.t.} \quad X - L_D - S = 0.
\]

The nuclear norm can be regarded as a linear mapping to a low-dimension matrix, whereas an autoencoder is a non-linear mapping [11].

Expr. (6) removes the anomalies of each element in the input data. To remove outliers in the inputs, the \(l_1\) norm is replaced with the \(l_{2,1}\) norm. Then, we have
\[
\|S\|_{l_{2,1}} = \sum_{j=1}^{N} \sqrt{\sum_{i=1}^{N} |x_{ij}|^2}
\]

Finally, the RDA removes outliers, in accordance with the optimization problem given by Expr. (7).
\[
\min_{\theta, S} \|L_D - D_{\theta}(E_{\theta}(L_D))\|_2 + \lambda \|S\|_{l_{2,1}},
\]
\[
\text{s.t.} \quad X - L_D - S = 0
\]

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

3.4 ECG Data

As a training data for RDA, a dataset extracted from the MIT-BIH Arrhythmia Database [12] was used. The MIT-BIH Arrhythmia Dataset contains ECG recordings, position data of R-waves, and diagnosis annotations for each heartbeat. The baseline oscillation was removed from the raw data by using a median filter, and a low-pass filter removed the noise. Then, 180 samples of filtered data around the R-waves were cut out, as shown in Fig. 3, and normalization was performed so that the data had a values in the range of

\[
\text{Input: } X \rightarrow L_D = X - S \rightarrow \text{Autoencoder} \rightarrow \text{Output}
\]

\[
\text{S (Anomaly)}
\]

**Fig. 2** Training process of RDA.
0–1 [13]. Among the many ECG channels, lead II was used in this study.

Premature ventricular contractions were used as outliers. Outlier and normal data were collected randomly from patients who did not have severe diseases, such as myocardial infarction, or who did not use pacemakers (100.dat, 101.dat, 103.dat, 106.dat, 113.dat, 114.dat, 115.dat, 116.dat, and 119.dat). Thus, we created a dataset with 6073 normal heartbeats and 332 outliers.

4. Lightweight Autoencoder

4.1 Weight Pruning for RDA

To minimize the number of weight parameters, the sparseness technique was applied. The sparseness technique eliminates unnecessary weight to reduce the number of weight parameters. Typically, the unnecessary weights are close to 0, and they are handled as 0. When the neural network is implemented on hardware, the memory consumption should be reduced, as there is no need to store the “zero” parameters. The sparseness technique is defined by Expr. (8), where \( w \) represents the weight parameter, \( Th \) represents the threshold value, \( w_{sp} \) represents the weight parameter after pruning, and \(| \cdot |\) represents the absolute value. Then, we have

\[
\begin{align*}
    w_{sp} &= \begin{cases} 
    0 & (|w| \leq Th) \\
    w & (|w| > Th).
    \end{cases}
\end{align*}
\]

In this study, the weight parameters in a trained autoencoder whose absolute values were below the threshold value were replaced with 0. Subsequently, the autoencoder was retrained. This process was repeated several times [14]. When trained repeatedly, neural networks may be overfitted. Thus, the number of iterations and the learning rate were gradually reduced, and the F1 score was kept as high as possible. In training using the RDA, the matrix \( S \) (Fig. 2, and Expr. (7)) should be stored. When the training is executed repeatedly, the matrix \( S \) from the previous training is used again in the next training.

4.2 Fixed-Point Quantized Model

When the RDA trains the autoencoder, a 32-bit floating-point value is used. 32-bit floating-point value is used. However, a floating-point calculation has a high computational cost, increasing the hardware resources and computational time. In this study, we converted all the parameters into fixed-point values to implement the autoencoder efficiently on a low-end FPGA.

The conversion into fixed-point values was performed after the floating-point model was obtained. We did not apply additional training for the fixed-point model. The bit width of the integer part was set according to the range of the trained parameters. We carefully set the bit width of the decimal part, because there is a tradeoff between the bit width and the model performance.

4.3 Weight Sharing

The bit width of the array that stores the weight parameters (\( data \) in Fig. 4) depends on the bit width of the maximum value stored in the array. The bit width of the weight parameters should be kept as large as possible to maintain the performance of the outlier detector, which increases the parameter volume. Therefore, we use weight sharing, which maintains the bit width of the weight parameter by clustering parameters and storing only the clustered indices. We can reduce the weight-parameter capacity by using the same values for the weight parameters belonging to the same cluster. The K-means method was used for clustering.

Although retraining was applied after clustering in a previous study [9], we only apply clustering to the trained parameters and use the centroids’ values for each cluster. As indicated by \( w_{list} \) in Fig. 4, the weight values used in the calculation are stored in the other array, and only the addresses for the \( w_{list} \) are stored in the weight-parameter matrix.

In this study, we first set the bit width \( n \) of the \( data \) array. When the bit width is \( n \), the weight parameters can have \( 2^n \) different values. Thus, the number of clusters \( k \) is set as \( 2^n \). The bit width of \( w_{list} \) is set within the rage for which the outlier detection performance does not change. This bit width does not significantly affect the hardware resource size, because \( w_{list} \) stores only \( k \) values, and \( k \) is far smaller than the total number of weight parameters. We optimize a tradeoff between the number of clusters \( k \) (= \( 2^n \)) and the outlier detection performance.

5. Hardware Implementation

5.1 Sparse Weight Representation and Computation

The weight parameters of the autoencoder in which the
Algorithm 1: The algorithm for the sparse autoencoder

INPUT: X ∈ R^{N_1}, data ∈ R^{N_{data}}, bias ∈ R^{N_2}, rowPtr ∈ R^{N_{row}+1}, columnIndex ∈ R^{N_{data}}

1. Initialize the value of out with bias
2. Compute the indexes of the input and output node, and execute the multiplication
   for i in 0 to N2 do
   \( out[i] \leftarrow bias[i] \)
   end for
   for j in rowPtr[i] to rowPtr[i+1] do
     input_index ← columnIndex[j]
     output_index ← i
     w ← data[j]
     x ← X[input_index]
     out[output_index] ← out[output_index] + w × x
   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

The structure of the autoencoder is presented in Table 1. According to the data size of the ECG dataset sample, input and output sizes were set as 180. The initial values of the weight parameters were set as random numbers following a normal distribution, and random numbers set biases following a uniform distribution. The hyperparameter \( \lambda \) (given by Expr. (7)) in the RDA was set to the value exhibiting the highest F1 score during the detection of outliers in the training. Additionally, the sparseness technique was applied while training the autoencoder using the RDA.

6. Experimental Results and Discussion

6.1 Evaluation Method

The F1 score was used for the evaluation of outlier detection. It is the harmonic mean of the recall and precision. In this study, positive data were “outliers,” as the ability to detect outliers was evaluated. Cross-validation was used to evaluate the data.

6.2 Training Using RDA

The structure of the autoencoder is presented in Table 1. The F1 score was used for the evaluation of outlier detection. It is the harmonic mean of the recall and precision. In this study, positive data were “outliers,” as the ability to detect outliers was evaluated. Cross-validation was used to evaluate the data.

6.3 Outlier Detection Using an Autoencoder

The autoencoder trained by the RDA was used to detect the
Table 2  Sparse ratio and F1 score.

| layer1 & layer4 | layer2 & layer3 |
|-----------------|----------------|
|                 | sparse ratio (%) | 80 | 90 | 95 |
|                 | 80              | 0.970 | 0.968 | 0.893 |
|                 | 90              | 0.971 | 0.968 | 0.852 |
|                 | 95              | 0.876 | 0.835 | 0.787 |

Table 3  Comparison of recognition accuracy.

|                   | not pruned | pruned |
|-------------------|------------|--------|
| Precision         | 0.954      | 0.938  |
| Recall            | 0.982      | 1.0    |
| F1 score          | 0.967      | 0.968  |
| # of weight params| 46000      | 4600   |
| Ratio of weight params | 1.0     | 0.1    |

ECG outlier data. The mean squared error (MSE) was calculated for the inputs and outputs, and if the value exceeded a certain threshold, the datum was considered to be an outlier. This threshold was set as the value of the smallest MSE among the ECG wave data, which was judged as an outlier during the training.

6.4 Environment Setup

The RDA was implemented using Chainer Deep Learning Framework 6.0.0[16]. The autoencoder trained by the RDA was implemented on a Xilinx ZedBoard Zynq-7000 Development Board using Xilinx SDSoC 2018.2. The timing constraint was set as 100 MHz. An Arduino Due was used for comparison. The autoencoder with sparseness weight and fixed-point parameters is implemented with software and executed on an ARM CortexM3 CPU, which is mounted on the Arduino Due board. Since the main purpose of the proposed hardware is to detect outliers in a portable ECG monitor, which has a very limited power supply, we chose ARM CortexM3, a CPU for embedded systems, as a counterpart.

6.5 Comparison of Sparse and Non-Sparse Autoencoders

Table 2 shows the tradeoff between the sparse ratio and the F1 score. The sparse ratios for layer1 & layer4, and layer2 & layer3 are set as the same value for simplicity. The F1 score did not decrease until the sparse ratio reaches 90%. When the sparse ratio is set above 90%, the F1 score decreased significantly. Thus, we set the sparse ratio to 90% for all the layers in the autoencoder.

Table 3 represents the precisions, recall, and F1 scores for the predictions of the autoencoders. Autoencoders with and without the sparseness technique were compared. The dense-weight autoencoder had 46000 parameters, and the F1 score was 0.967. When the sparseness technique was applied, 90% of the number of weight parameters were eliminated. In this case, the F1 score was 0.968, which was 0.1% higher than the score of the dense autoencoder. The performance of sparse deep-learning models is sometimes equivalent to or higher than that of dense ones because the excessive parameters are removed.

6.6 Fixed-Point Implementation

The autoencoder uses 32-bit floating-point values during training. When the floating-point values are converted into fixed-point values, there is a tradeoff between the bit width of the decimal part and the model performance. We must carefully select the bit width to avoid reducing the recognition accuracy.

Figure 7 shows the F1 score with changes in the bit width of the decimal part of the weight parameters. All the results were obtained using a 90% sparse model, as mentioned in Chapter 6.5. Because the weight parameters were within the range of −3 to 3, the bit width of the integer part was set as 2 bits, and 1 bit was used as a sign bit. The remaining bits were used for the decimal part.

When the bit width was 10 or 11 bits, the F1 score was 0.989, which was identical to the score for the 90% sparse model. When the bit width was <10 bits, the F1 score gradually decreased. When the bit width was 9 bits, the F1 score was 0.957, which was 1% lower than that for the dense model. For the FPGA implementation of the fixed-point model, a 10-bit fixed-point value was used.

6.7 Weight Sharing

Figure 8 shows the F1 score when weight sharing was applied to the 10-bit fixed-point autoencoder. The bit width of the data array (see Fig. 4) to n bits and the weight parameters were divided into k (= 2^n) clusters. When k was 16 (= 2^4), the F1 score was 0.959, which was only 0.83% lower than that for the dense model. In this case, the weight parameters could be stored as 4-bit values, and the parameter volume was reduced by 60%. Even when k was 8 (= 2^3), the F1 score was 0.944, which was 2.3% lower than the dense-model score.

In contrast to the deep-learning models for the other domains, e.g. image classification, the autoencoder needs a certain number of bit width to maintain its performance, as it must reconstruct the input data. It is not easy to significantly
reduce the bit width of the weight parameters. By using weight sharing, we can easily reduce the bit width of the parameters without additional training.

In this study, we used the $k = 16$ weight-sharing model for the FPGA implementation.

6.8 Comparison of F1 Scores for Different Parameters

Table 4 represents the precisions, recalls, and F1 scores for the predictions of the implemented autoencoder. Non-zero parameters denotes the total parameter volume size, that is, rowPtr, columnIndex, data, and for weight-sharing model, $w\_list$. When 90% of the weight parameters are eliminated, this does not imply that the parameter volume is reduced by 90%, because of the address indices of the CRS format. By reducing the bit width via the conversion of floating-point values into fixed-point values, the weight-parameter volume is reduced by 53.8% compared with the floating-point model. Moreover, when weight sharing is applied, the weight-parameter volume is reduced by 30.3% compared with the fixed-point model. Thus, the parameter volume size is only 4.13% of that of the dense and floating-point model. The parameter volume size of the sparse and weight-sharing model is sufficiently small to store in on-chip memories on a low-end FPGA.

6.9 Hardware Resource Consumption and Speed

Table 5 shows the hardware resource consumption when the autoencoder is implemented on an FPGA. By reducing the number of weight parameters via applying the sparseness technique, the use of BRAM was significantly reduced. All the parameters of the autoencoder can be stored in on-chip memory. By converting parameters into fixed-point numbers, the BRAM consumption was reduced. Additionally, the number of DSP blocks was reduced in the fixed-point model. When weight sharing is applied, BRAM, FF, and LUT were further reduced.

With regard to the computational speed, the fixed-point model is 2.9 times faster than the floating-point model. In the case of the weight sharing model, even if there is an overhead for reading $w\_list$, the speed is almost the same as the fixed-point model with no weight sharing. The reading process is executed in pipelines while other computations are running.

6.10 Power Consumption

Table 6 shows the power consumption and the total power consumption of the FPGA and the CPU. Note that they are not the results of overall power consumption. The results are on FPGA (CPU) only. An autoencoder with sparseness weight, fixed-point, and weight-sharing parameters was implemented. Here, total power consumption was defined as the power consumption multiplied by the execution time, as shown in Table 6. The processing time for the test dataset (which included 1282 ECG waves) was measured. The autoencoder implemented on an FPGA operated with 11% of the power consumption of the ARM Cortex M3 CPU. Moreover, the inference speed was 11.7 times higher than that of the CPU implementation. The total power consumption was 0.94% of that of the CPU (106 times better with regard to energy consumption) because the FPGA executed the prediction significantly faster than the CPU and had limited processing units for the autoencoder.

6.11 Sparse Weight Parameter Volume

Table 7 represents the parameter volume size for the sparse
autoencoder. When weight sharing was applied, the volume of the values was reduced by 60% via the bit-width reduction. However, the other parts of the CRS format remained unchanged, because the weight sharing did not affect the connections between neurons. In the case of the weight-sharing model, 60% of the parameter volume was occupied by the columnIndex, which stored the input neurons’ index in the autoencoder. It is difficult to reduce the size of columnIndex, because the number of nonzero parameters bounds the number of columnIndex, and the number of input neurons bounds the bit width. In this study, the maximum number of input neurons was 180, thus, the bit width of the columnIndex was set as 8 bits. We must modify the autoencoder model or apply some constraints during training to reduce the volume. A potential approach is the use of other neural-network structures such as locally connected layers, which would limit the range of the input neuron area.

7. Conclusion

In this paper, we implemented an ECG outlier detector using an autoencoder based on neural networks. As a training method, we used a robust deep autoencoder that can conduct unsupervised learning. We proposed a design flow to implement the outlier detector using an autoencoder on a low-end FPGA. To minimize the volume of the weight parameters, a weight sparseness technique was applied, and all the parameters were converted into fixed-point values. We showed that even if the parameters are reduced and converted into fixed-point values, the outlier detection performance degradation was only 0.83 points. By reducing the volume of the weight parameters, all the parameters could be stored in on-chip memory. To the best of our knowledge, this work is the first implementation of an outlier detector using the sparseness weight autoencoder on an FPGA. We designed the architecture according to the CRS format, which is the well-known data structure of a sparse matrix, minimizing the hardware size and reducing the power consumption. We used weight sharing to further reduce the weight-parameter volumes. By using weight sharing, we can reduce the bit width of the memories by 60% while maintaining the outlier detection performance. We implemented the autoencoder on a Digilent Inc. ZedBoard and compared the results with those for the CPU for a built-in device. The results indicated that the FPGA implementation of the outlier detector was 12 times faster and 106 times more energy-efficient. Thus, we showed that our proposed scheme is suitable for mobile ECG devices.

Acknowledgements

This research is supported in part by the Grants in Aid for Scientific Research of JSPS, Industry-academia collaborative R&D programs centre of innovation (COI) program, Core Research for Evolutional Science and Technology (CREST), and the New Energy and Industrial Technol- ogy Development Organization (NEDO). Thanks to NVIDIA support and Xilinx University Program (XUP). Reviewers’ comments improved the paper.

References

[1] P. Rajpurkar, A.Y. Hannun, M. Huangpanahi, C. Bourn, and A.Y. Ng, “Cardiologist-level arrhythmia detection with convoutional neural networks,” CoRR, vol.abs/1707.01836, 2017.

[2] A. Walinjkar and J. Woods, “Personalized wearable systems for real-time ecg classification and healthcare interoperability: Real-time ECG classification and FHIR interoperability,” 2017 Internet Technologies and Applications (ITA), pp.9–14, Sept. 2017.

[3] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J.S. Emer, S.W. Keckler, and W.J. Dally, “SCNN: an accelerator for compressed-sparse convolutional neural networks,” SIGARCH Comput. Archit. News, vol.45, no.2, pp.27–40, 2017.

[4] N. Soga, S. Sato, and H. Nakahara, “Energy-efficient ecg signals outlier detection hardware using a sparse robust deep autoencoder,” The 22nd Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI), 2019.

[5] H. Lassoued and K. Raouf, “Artificial neural network classifier for heartbeat arrhythmia detection,” 03 2017.

[6] A. Gharaviri, M. Teshehlab, and H.A. Moghaddam, “PVC arrhythmia detection using neural networks,” 2007 5th International Symposium on Image and Signal Processing and Analysis, pp.234–237, Sept. 2007.

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

[8] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani, “A comprehensive survey on model compression and acceleration,” Artif. Intell. Rev., 2020.

[9] S. Han, H. Mao, and W.J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” 2015, cite arxiv:1501.00499Comment: Published as a conference paper at ICLR 2016 (oral).

[10] S. Han, J. Pool, J. Tran, and W.J. Dally, “Learning both weights and connections for efficient neural networks,” CoRR, vol.abs/1506.02626, 2015.

[11] C. Zhou and R.C. Paffenroth, “Anomaly detection with robust deep autoencoders,” Proc. 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’17, New York, NY, USA, pp.665–674, ACM, 2017.

[12] “MIT-BIH arrhythmia database.” https://physionet.org/physiobank/database/mitdb/.

[13] P. de Chazal, M. O’Dwyer, and R.B. Reilly, “Automatic classification of heartbeats using ECG morphology and heartbeat interval features,” IEEE Trans. Biomed. Eng., vol.51, no.7, pp.1196–1206, 2004.

[14] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in Advances in Neural Information Processing Systems 28, ed. C. Cortes, N.D. Lawrence, D.D. Lee, M. Sugiyama, and R. Garnett, pp.1135–1143, Curran Associates, Inc., 2015.

[15] R. Kastner, J. Matai, and S. Neundorffer, “Parallel programming
Naoto Soga received the B.E. degree in engineering from Tokyo Institute of Technology, Tokyo, Japan, in 2019. He is currently a Master Student with the Department of Information and Communications Engineering of Tokyo Institute of Technology. He received the SASIMI Best Paper Award in 2019. His current research interests include Deep Neural Network and FPGA.

Shimpei Sato received the B.S., M.S., and Ph. D. degrees in engineering from Tokyo Institute of Technology (Tokyo Tech), in 2007, 2009, and 2014. He is currently an Assistant Professor with the Department of Information and Communications Engineering of Tokyo Tech. From 2014 to 2016, he worked in High performance computing area as a post doctoral researcher, where he investigated an application performance analysis/tuning method. His current research interests include approximate computing realization by architecture design and circuit design and their applications.

Hiroki Nakahara received the B.E., M.E., and Ph.D. degrees in computer science from Kyushu Institute of Technology, Fukuoka, Japan, in 2003, 2005, and 2007, respectively. He has held research/faculty positions at Kyushu Institute of Technology, Fukuoka, Japan, Kagoshima University, Kagoshima, Japan, and Ehime University, Ehime, Japan. Now, he is an associate professor at Tokyo Institute of Technology, Japan. He was the Workshop Chairman for the International Workshop on Post-Binary ULSI Systems (ULSIWS) in 2014, 2015, 2016 and 2017, respectively. He served the Program Chairman for the International Symposium on 8th Highly-Efficient Accelerators and Reconfigurable Technologies (HEART) in 2017. He received the 8th IEEE/ACM MEMOCODE Design Contest 1st Place Award in 2010, the SASIMI Outstanding Paper Award in 2010, IPSJ Yamashita SIG Research Award in 2011, the 1st FIT Funai Best Paper Award in 2012, the 7th IEEE MCMSoC-13 Best Paper Award in 2013, and the ISMVL2013 Kenneth C. Smith Early Career Award in 2014, respectively. His research interests include logic synthesis, reconfigurable architecture, digital signal processing, embedded systems, and machine learning. He is a member of the IEEE, the ACM, and the IEICE.