Neural Computing and Applications (2021) 33:14741–14751
https://doi.org/10.1007/s00521-021-06113-4

ORIGINAL ARTICLE

Resource-constrained FPGA/DNN co-design

Zhichao Zhang[1] · Abbas Z. Kouzani[1]

Received: 11 November 2020 / Accepted: 5 May 2021 / Published online: 15 May 2021

Abstract
Deep neural networks (DNNs) have demonstrated super performance in most learning tasks. However, a DNN typically contains a large number of parameters and operations, requiring a high-end processing platform for high-speed execution. To address this challenge, hardware-and-software co-design strategies, which involve joint DNN optimization and hardware implementation, can be applied. These strategies reduce the parameters and operations of the DNN, and fit it into a low-resource processing platform. In this paper, a DNN model is used for the analysis of the data captured using an electrochemical method to determine the concentration of a neurotransmitter and the recoding electrode. Next, a DNN miniaturization algorithm is introduced, involving combined pruning and compression, to reduce the DNN resource utilization. Here, the DNN is transformed to have sparse parameters by pruning a percentage of its weights. The Lempel–Ziv–Welch algorithm is then applied to compress the sparse DNN. Next, a DNN overlay is developed, combining the decompression of the DNN parameters and DNN inference, to allow the execution of the DNN on a FPGA on the PYNQ-Z2 board. This approach helps avoid the need for inclusion of a complex quantization algorithm. It compresses the DNN by a factor of 6.18, leading to about 50% reduction in the resource utilization on the FPGA.

Keywords Lempel–Ziv–Welch compression · Deep neural network · Hardware-and-software co-design · Electrochemical sensing · Field-programmable gate array

1 Introduction

The deep neural networks (DNNs) are computing models composed of highly interconnected neurons organized in more than three layers [1]. DNNs have demonstrated excellent data analysis performance in ubiquitous applications such as computer vision [2], speech recognition [3], financial fraud detection [4], bioinformatics [5], drug discovery for the novel COVID-19 [6], among others.

Some emerging applications, such as self-driving vehicles [7], robot-assisted surgery [8], and physical activity tracking [9], require accurate and reliable data analysis at the location of data capture in real-time. Neurochemicals sensing devices based on fast-scan cyclic voltammetry (FSCV) form one such application, where rapid changes in the extracellular concentration of neurotransmitters in the brain are captured electrochemically [10]. Analyzing the FSCV signals helps detect the state of some neurological disease [10, 11], e.g., Parkinson's disease [12], depression [13], etc. Currently, many researchers are focusing on the development of sensing platforms to provide precise recording of the signals acquired using FSCV in response to certain types of neurochemical reactions [14]. For instance, Nasri et al. proposed a sensing system by developing a hybrid COMS-graphene sensor array to improve the sensitivity of measuring the response to a dopamine solution [15]. Guo et al. developed field-effect biochemical imaging sensors with multimodal fibers to record localized pH changes in hippocampus in physiological and pathological conditions [16]. Apart from that, principal components regression (PCR) and partial least squares regression (PLSR) have been the main techniques for FSCV signal analysis [17]. For instance, Kim et al. provided a comparison of the performance between PCR and PLSR and concluded that PLSR should be preferred over PCR in FSCV data analysis [18]. Puthongkham et al. applied an image recognition method of the structural similarity index to distinguish adenosine [19]. However, more advanced

✉ Abbas Z. Kouzani
kouzani@deakin.edu.au

[1] School of Engineering, Deakin University, Geelong, VIC 3216, Australia

machine learning algorithms, e.g., DNN, have not been widely utilized in FSCV data analysis [17]. And more importantly, these researches either focus on building FSCV signal acquisition devices or FSCV data analysis, and we have not seen co-design of both hardware and software to support real-time FSCV data analysis on low-resource devices. Therefore, in this work, we focus on FPGA/DNN co-design for FSCV analysis on resource-constrained mobile platforms, which can further support the next generation of developing wearable FSCV devices for real-time analysis of neurochemicals.

Usually, wearable FSCV devices employ a low-resource processor to operate, thus they need to continuously transmit data to a nearby computer, and receive analysis outcome wirelessly [20]. This approach suffers from large data analysis delay, and low data transmission security and reliability [20, 21]. These issues can be addressed by developing a DNN that can be embedded into the wearable FSCV device [22].

In order to fit a DNN into a low-resource processing platform, some challenges need to be overcome:

1. A DNN typically contains a large number of parameters and operations, which would be difficult to fit into a low-resource processing platform.
2. A DNN usually requires a significant amount of data computation, which would produce slow execution speed and high energy consumption.

According to the literature, a solution that can address these challenges involves joint development of hardware and software aspects, called hardware-and-software co-design [23, 24]. The co-design strategy involves DNN optimization algorithms and hardware mapping approaches [25].

The DNN optimization algorithms miniaturize the DNN model. To deploy a DNN on a low-resource processing platform, the first step is to optimize the DNN by reducing its large number of parameters and operations [26]. Fortunately, many trained DNNs contain a large number of redundant parameters [27]. Accordingly, several algorithms have been developed to reduce these parameters. These algorithms involve pruning [21, 28], compression [29, 30], and quantization [31, 32], etc. For instance, Han et al. [27] developed a pipeline of approaches to reduce the storage requirement of DNNs without loss of analysis accuracy. It includes three steps: iteratively pruning the parameters and re-training, quantizing the parameters, and applying compression on the parameters. In addition, new neural network architectures have been reported in the literature that require less number of learning parameters, e.g., hybrid neural networks using morphological neurons [33, 34].

The processing platform must support the execution of a miniaturized DNN. Field-programmable gate arrays (FPGAs) can efficiently support DNN inference due to their flexible and programmable architectures, power efficiency, and parallel architecture [35]. High-resource FPGAs can achieve high performance; however, they are expensive and power-hungry. Low-resource FPGAs are more suitable for low-cost, battery-operated portable devices as they are cheap and low-power. In this work, a DNN is trained for FSCV data analysis and mapped into a low-resource FPGA device.

A miniaturization approach is proposed in this work to fit the FSCV DNN into a low resource FPGA device. As shown in Fig. 1, the developed framework includes two steps: miniaturization of the DNN with a combined pruning/compression algorithm, and mapping of the miniaturized DNN into a low-resource FPGA device.

DNN Miniaturization: We begin with a trained DNN the analysis of FSCV data to recognize the concentration of the dopamine solution and also the identification of the electrode adopted. Then, a pruning strategy is designed by setting a fixed pruning ratio to filter out small weights per neuron. After that, the pruned DNN is re-trained to recover the accuracy. By iteratively pruning and re-training the DNN, the parameters of the DNN are converted into a sparse representation. Once the recognition accuracy of the DNN is fulfilled under the configured pruning ratio, the parameters are pre-processed into low-precision bytes. Finally, the Lempel–Ziv–Welch (LZW) coding is applied to compress the sparse parameters.

Hardware Implementation: An intellectual property (IP) core is developed involving a decoding module and a DNN module. The decoding module is responsible for converting the compressed parameters into 16-bit floating point precision. Moreover, a DNN module implements the data analysis operation and memory allocation. The hardware implementation is carried out in the Vivado HLS software tool.

The miniaturized DNN is mapped into the FPGA device onboard of the PYNQ-Z2 board without losing recognition accuracy. The contributions of this work are:

1. A DNN model is used for FSCV data analysis to recognize the concentration of a neurotransmitter called dopamine, and determine the electrode used in measuring the neurotransmitter.

2. A DNN miniaturization algorithm is proposed with combined pruning and compression to reduce DNN resource utilization.

3. A DNN overlay is developed, containing decompression and DNN inference to support the compressed DNN inference on the FPGA device onboard of the PYNQ-Z2 board.

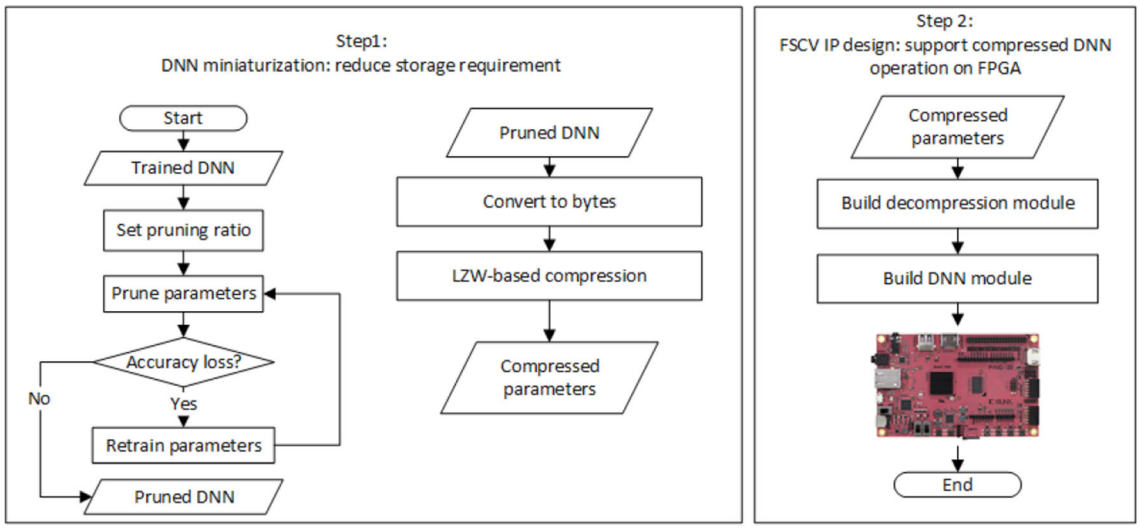

Fig. 1 Framework of our hardware-software co-design approach

The paper is organized as follows. Section 2 reviews and compares some key data compression approaches. Section 3 describes the FSCV data, as well as details of the FSCV DNN. Section 4 presents the DNN miniaturization algorithm and the development of the DNN overlay. The FPGA implementation of the DNN and the development of the customized overlay are given in Section 5. The experimental results are reported in Section 4. Section 7 discusses the performance of the miniaturization algorithm and potential future improvements. Finally, Sect. 8 gives the concluding remarks.

2 Compression algorithms

In this section, we discuss several data compression techniques that could be utilized for DNN compression to reduce the memory requirement. Compared with lossy compression, lossless compression ensures no errors during decompression, thus preferred in applications that are not tolerant of data loss, e.g., text compression. Therefore, we focus on lossless compression approaches [36] for DNN compression.

Huffman coding, compressed sparse columns (CSC) and compressed sparse row (CSR) have been applied to DNN compression [21]. Huffman coding [37] relies on iteratively combining two characters/values that have the smallest weights. The more times the character/value is repeated, the shorter the binary sequence is generated. Han et al. [27] applied Huffman coding on quantized DNN containing repeated values, offering 20 $\sim$ 30% of the reduction in data storage. Differently, the CSC and CSR approaches are particularly suitable for compressing sparse matrix, by storing only the nonzero values and corresponding indexes, currently popular for compressing pruned DNN [21].

Apart from that, Run-length encoding (RLE) offers a strategy that replaces multiple consecutive duplications with a count of these duplications. Compared with CSC, RLE focuses on reducing the space for storing the duplications. We have noticed that the pruned DNN often contains consecutive zeroes. This inspired us to test the effectiveness of RLE on DNN compression [38]. Lempel–Ziv–Welch (LZW) [39] is another lossless compression algorithm that can effectively compress repeated values. It focuses on encoding the data in 8-bit without requiring additional storage of the dictionary table. However, to the best of our knowledge, LZW has not been utilized for compressing the DNNs with sparse parameters.

In order to select the most suitable compression approach for developing the miniaturization algorithm in this work, we compared the performance of these approaches on a pruned DNN developed in our earlier work [40] with 166,102 parameters in 32-floating points format, involving approximately 70% of zeroes, typically repeated zeroes. Table 1 presents a comparison of these compression approaches.

As shown in Table 1, RLE offers simple encoding/decoding, however, gives the lowest compression ratio. The higher compression ratios are achieved by Huffman coding and LZW, benefited from the low-precision representation that could increase the redundancy of data. Compared with the other three algorithms, LZW achieved the highest compression ratio with the minimum time complexity. Compared with LZW, Huffman coding is quite slow for compressing a large dataset since it requires a scan of the entire dataset to identify the unique characters and count the repeated time of each character. Besides, it requires

Table 1 Main features of Huffman, CSC, RLE, and LZW algorithms

Algorithms	Huffman	CSC	RLE	LZW
Advantages	Binary format	Focuses on compressing the sparse matrix	1. Simple algorithm 2. Particularly suitable for compressing data with consecutive values	1. No need to prior information of the entire dataset 2. Fast encoding (and decoding) that compresses (and decompresses) data per single pass 3. No need to store dictionary
Disadvantages	1. Slow, and computationally complicated encoding 2. Need extra storage of dictionary 3. Slow decoding, requiring looking up dictionary per symbol	Could not deal with the nonzero duplications	May not be able to give the highest compression ratio	Needs additional process of converting unsigned int8 to 32-bit floating points
Encoding format	Binary stream	32-bit floating point	32-bit floating point	Unsigned int16
Input format	Char	32-bit floating point	32-bit floating point	Unsigned int8
Time complexity	$O(nlog^n)$	$O(n)$	$O(n)$	$O(n)$
Compression ratio	2.02	1.58	1.33	2.21

additional space and time to store and look up the dictionary table. In contrast, the LZW algorithm dynamically generates the dictionary during encoding, without a scan of the entire dataset. According to the above analysis, we have identified that LZW is more suitable for compressing our developed DNN with sparse parameters due to its high compression ratio and fast execution.

3 FSCV dataset and the DNN

3.1 FSCV Dataset

In this work, the FSCV data are collected at the Neural Engineering Laboratory of the Mayo Clinic by five carbon fiber electrodes fabricated (namely electrode 1, …, electrode 5) by following the procedures reported in [11]. WINCS Harmoni and WINCSware were utilized to capture in-vitro FSCV voltammograms [41, 42]. Each electrode is plugged into five concentrations of dopamine solutions (0, 0.5, 1, 1.5, 2 μM). Due to unavoidable slight variations of carbon fiber electrodes during manufacture, each electrode performs differently in detecting a certain concentration of the dopamine solution. Accordingly, the dataset contains the FSCV signals captured by 5 different electrodes within 5 concentration levels of dopamine solutions. In total, 1250 FSCV signals are collected. The sampling rate of the FSCV measurements is 100 kHz. Each FSCV signal consists of 850 sample points, represented in a 32-bit floating-point format. The captured signals are subtracted from the background, averaged, and filtered as described in ref. [11]. Figure 2 shows a sample signal from the FSCV dataset. From Fig. 2, the FSCV signal does not involve high-frequency noise. It is an electrochemical signal that reflects dopamine concentrations. The signal is deterministic since there is no uncertainty with respect to its value at a sample point of time. The peak dopamine oxidation currents are recognizable, which maintains the sensitivity to dopamine at high scan rates.

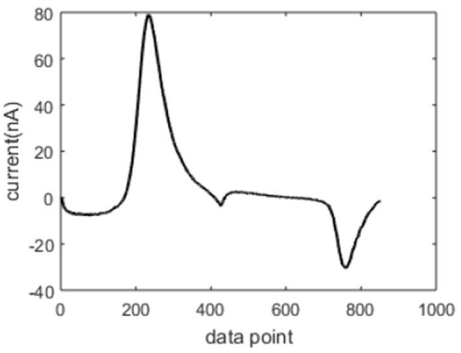

Fig. 2 A sample FSCV signal

3.2 DNN

We developed a 10-layer DNN to recognize the concentration level of the dopamine solution as well as the plugged electrode that the FSCV signal was acquired from. Since there are 850 sample points in each input FSCV signal, the input layer of the DNN contains 850 neurons. Eight hidden layers are constructed with each of them containing 100 neurons. The output layer has two neurons that correspond to the concentration level of the dopamine solution and the FSCV electrode, respectively. The DNN contains 166,102 parameters. The equation of the DNN is:

$$\begin{aligned} output(l,j) &= f_{activation}\left(bias(l,j) + \sum_{i=1}^{i=N_l} input(l,i) \times weights(l,i,j)\right) \\ input(l,i) &= \begin{cases} signal(i), if l = 1 \\ output(l-1,i), if l > 1 \end{cases} \\ & i,j \in Z, 1 \le i < N_l, 1 < j \le N_l, \\ & N \in neurons\_layer \\ & l \in Z, 1 \le l \le num\_layer \end{aligned} \tag{1}$$

where the neurons_layer indicates the number of neurons per layer, i.e., {850, 100, 100, 100, 100, 100, 100, 100, 100, 2}, and the num_layer represents the number of the layers, i.e., num_layer = 10. The rectifier linear unit (ReLu) function described by Eq. (2) is applied to the input layer. The rest of the DNN, i.e., the hidden layers and output layer, uses scaled exponential linear unit (SeLu) function, represented in Eq. (3).

$$f_{\text{activation}}(x) = max(0,x) \tag{2}$$

$$f_{\text{activation}}(x) = \lambda \begin{cases} x \, if \, x > 0 \\ \alpha(e^x - 1) if \, x \le 0 \end{cases} \tag{3}$$

The tenfold cross-validation and hold-out validation are configured to avoid overfitting. First of all, the tenfold cross-validation algorithm is applied. To train, tune, and test the DNN, we divided the dataset into training set, validation set, and test set. Twenty percent of the FSCV signals are utilized for testing the performance of the DNN. The rest 80% of FSCV signals are divided into training set (900 signals) and validation set (100 signals). In addition, within each iteration, 50 epochs of training are required. During each epoch, hold-out validation is applied to keep 20% of the 900 signals for the sub-validation within an epoch. Both hold-out validation and cross-validation approaches are helpful to train and tune the DNN. The loss function is defined by Eq. (4).

$$loss = \frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y)^2, \tag{4}$$

where $\hat{y}$ and y represent the recognition result and the truth, respectively. The stochastic gradient descent optimizer [43] is applied for training, and the learning rate is set to 0.01. Also, the dropout rate of 0.2 is set for the hidden layers during the training.

Besides, the output results from the DNN are represented in the floating-point format. This is unlikely to be identical to the corresponding integers of labels. Accordingly, Eq. (5) is used to quantize the output results to the closest label.

$$\begin{aligned} y\_predict(output) &= \begin{cases} -1, output < 0 \\ label_l, 0 \le output < 2.25 \\ -1, output \ge 2.25 \\ NaN, output = 0.25, 0.75, 1.25, 1.75 \end{cases} \\ & lable \in \{0, 0.5, 1, 1.5, 2\}, \\ & l \in \{\, l | f(l) = \min(|output - label_l|)\}, \\ & l = 1, 2, 3, 4, 5. \end{aligned} \tag{5}$$

4 4. DNN miniaturization

To reduce resource utilization, in this section, we introduce our proposed DNN miniaturization algorithm with combined pruning and compression. Besides, the corresponding decompression algorithm is provided. A description of the proposed DNN miniaturization and decompression algorithm are shown in Figs. 3 and 4, respectively.

First of all, the DNN contains a large number of small weights, which may not have much influence on the output of the DNN. Therefore, the straightforward way of reducing the parameters is to identify and remove these small values. However, by going through the literature, we have noticed that the challenge is often in setting the threshold. In this work, rather than setting a fixed threshold for the parameters of all DNN layers, we identify a certain percentage of small weights that connect to each output neuron. We have noticed that the higher the pruning ratio, the more training time needed to recover the recognition accuracy. Moreover, in our experiment, we have found that the recognition results of our DNN are more sensitive to changing the weights of the output layer. Accordingly, the pruning ratio of 90% is specified to remove 90% of the weights connecting to each neuron in the input layer and hidden layers, whereas 40% of the pruning ratio is set for the output layer. The pruning ratios configured in this work are specified in consideration of the pruning efficiency within an acceptable training time. Ninety iterations of pruning and re-training are required to achieve 100% recognition accuracy in this work. By operating the pruning

```
Input: 32-bit floating point layer weights W={w0,w1,w2,...wL-1}
Output: Compressed weights W'={w0', w1', w2'... wL-1'}

Initialization: acc=0, size=256

while (acc<100%):
do{
   train DNN
   for l in range(0,L):
        if (l<L):
           set pruning ratio r=0.90
        else:
           set pruning ratio r=0.40
        end
        for each neuron n:
            sort(wi,n)
            thresh=wi,n [r*len(wi,n)]
            wi,n (wi,n <thresh)=0
        end
   end
   Input FSCV signals
   Calculate recognition accuracy acc with updated weights W
}

for l in range(0,L):
    Convert Wl into a 1-D array of bytes
    T=[]
    for i in range(size):
        T[i]=i
    P=null
    C=T[0]
    for n in range(0,len(Wl)):
        if (T.find(P+C)=-1):
            P=P+C
        else:
            W'l,n=P
            T.append(P)
            P=C
            if (n==len(Wl)-1):
               C=null
               else:
                  C=T[n+1]
            end
        end
    end
end
```

Fig. 3 Algorithm: DNN miniaturization with combined pruning and compression

procedure, the parameters of the DNN have been converted into sparse matrices.

After that, a pre-processing procedure is conducted to convert these 32-bit floating point represented sparse matrices into bytes. Then, we employ the LZW algorithm to compress the pruned DNN, thus leading to the efficiency of storing the sparse matrixes. To begin with, a dictionary table is initialized by filling in all possible individual bytes. During encoding, the dictionary table is extended dynamically. Accordingly, the encoded data represent the index

```
Input: Compressed weights W'={w0', w1', w2'... wL-1'}
Output: Decompressed weights W={w0,w1,w2,...wL-1}

for i in range(size):
        T[i]=i
for l in range(len(W')):
    P=W'l [0]
    Wl[0]=T[P]
    for i in range(1,len(W'l)):
        Q= W'l [i]
        if Q <len(T):
           Wl[i]=T[Q]
        else:
           Wl[i]=T[P]+C
        end
        C= T[Q][0]
        T.append(P+C)
        P=Q
    end
end
```

Fig. 4 Algorithm: DNN decompression

that indicates the location in the dictionary table where stores the longest byte-sequence that matches the input byte-sequence. In the dictionary table, this longest matched byte-sequence is recorded temporally. By concatenating the current longest matched byte-sequence with the next input byte, a new sequence is generated, and that it is attached to the dictionary table. The concatenation of the bytes is conducted using shift operation. The encoding enables a long byte-sequence simply replaced with its index, therefore, saving storage resources.

The dictionary table can be regenerated according to the sequence of the compressed data during decompression. As shown in Fig. 4, during the compression, the initialization of the dictionary table does not rely on the original data distribution. Moreover, the dictionary table can be rebuilt by the sequence of the compressed data. As introduced earlier, the decompressed data are the indices in the dictionary table. The dictionary table gets updated while decoding an individual index in the sequence. Similarly to the encoding procedure, a new byte-sequence is generated by concatenating the currently decoded byte-sequence and the first character of the next decoded byte-sequence and is attached to the dictionary table. In this way, the data can be decoded by simply looking up the dictionary table. And, the dictionary gets updated dynamically.

5 FPGA implementation of DNN overlay

In this work, we implement the miniaturized DNN on the PYNQ-Z2 board, a SoC-based platform that contains the XC7Z020-1CLG400C (Artix-7) FPGA. The PYNQ-Z2

platform is composed of the programmable system (PS) and programmable logic (PL) blocks. The PS partition consists of a 650 MHz dual-core ARM® Cortex™-A9 microcontroller, memory interfaces, and other fixed peripherals. The integrated PL partition of Artix-7 FPGA includes around 630 KB of block RAM, 220 DSP slices, as well as 13 K logic slices. Moreover, 512 MB of DDR3 (with the 16-bit bus at 1050 Mbps) together with 16 MB flash is attached to the PS partition. Besides, the MicroSD slot is available onboard.

Our aim is to design a custom DNN overlay to support the miniaturized DNN execution on the PL partition of the PYNQ-Z2 board for the application of real-time FSCV signals analysis. We expect to achieve fast execution speed and low-energy consumption using as few FPGA resources as possible. The overlay is a specific programmable FPGA design that is developed using Vivado. The core function of the overlay is determined by the developed IP, which can be defined using Vivado HLS. To support the miniaturized DNN execution, our developed IP core oversees decompression as well as DNN execution. The developed IP is shown in Fig. 5.

As shown in Fig. 5, the FSCV IP contains five modules/functions including the storage of compressed parameters, a converter, the decompression function, and the DNN operation. First of all, to achieve the fast speed and low-energy analysis, it is preferred to avoid transferring the DNN parameters from DDR3. Therefore, the miniaturized DNN parameters are considered to be stored directly into the IP. Secondly, we develop the decompression module described in Sect. 4. Next, a converter is built to convert the returns of the decompression module from bytes into16-bit floating points. After that, the DNN module gets initialized with the decompressed parameters. Besides, we attach the "axis_stream" interface to the developed FSCV IP to communicate with the PS partition, i.e., reading the input FSCV signal and returning the recognition result from the DNN analysis. Once the DNN module has received an FSCV signal, it can execute the neural network and return the recognition results.

The next step is to export the developed FSCV IP core to Vivado for block design. Before that, we design a bench test program to ensure the developed IP core returns the correct recognition results. Once the IP core has been tested, it can be then synthesized into Verilog. Moreover, the register-transfer level (RTL) simulation is provided to examine whether the translated Verilog program matches the C + + code. Finally, the RTL abstraction is generated and then exported.

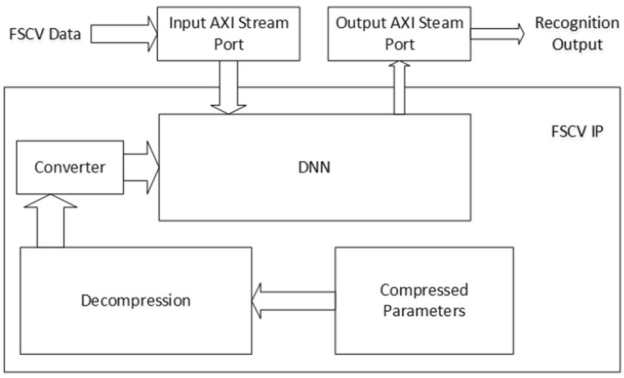

Fig. 5 Developed FSCV IP core

6 Experiment and results

6.1 DNN development

In our experiment, the DNN is initially trained and tested on a computer with Intel Core i5-45,705 CPU (at the clock frequency of 2.90 GHz) using the Tensorflow and Keras backend. As mentioned in Sect. 3.2, the tenfold cross-validation was applied during training to help tune the DNN. Accordingly, in this experiment, the average validation accuracy of 98.7% was achieved. After that, the tuned DNN was trained on the entire training dataset while hold-out validation was adopted. The loss and accuracy during training and hold-out validation are shown in Figs. 6 and 7, respectively. Besides, the recognition accuracy on the test set is 96%. The effectiveness of the DNN, and its comparison with the most common FSCV data analysis methods of PCR and PLSR have been discussed in our previous work [40].

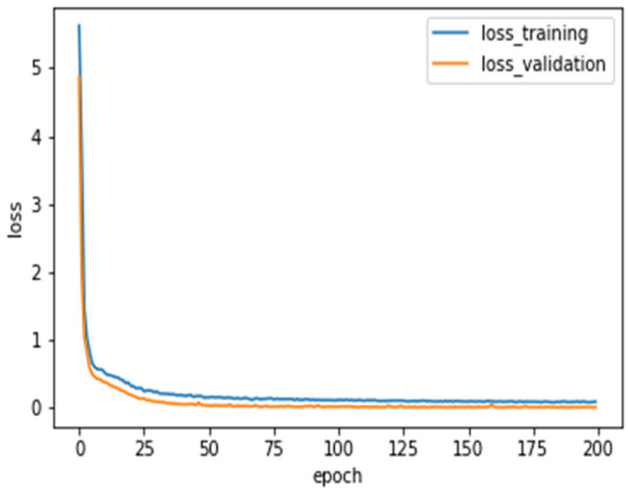

Fig. 6 Loss during training and hold-out validation on the entire training set

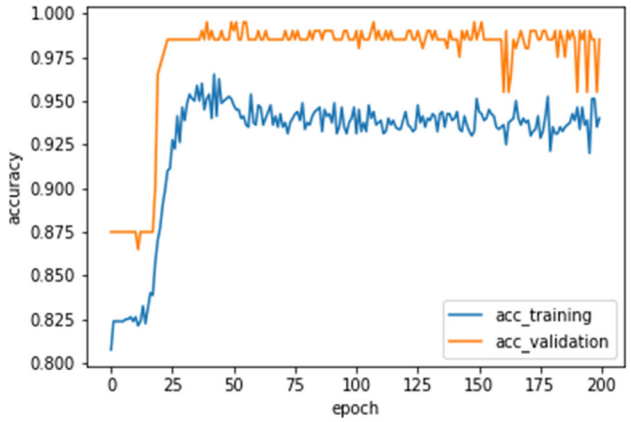

Fig. 7 Accuracy of training and hold-out validation on the entire training set

6.2 DNN miniaturization

In our experiment, we used the miniaturization algorithm introduced in Sect. 4, and obtained the results from the two main steps of the DNN miniaturization, i.e., DNN pruning and compression.

6.2.1 Pruning and recognition accuracy

After pruning and retraining, 90% of the weights are converted to zeroes, and that we successfully reduced the overall nonzero parameters (including weights and bias) by 9.96 times approximately, from the total number of 166,102 to 16,682. Figure 8 shows the trends of the loss and accuracy during the last iteration, i.e., the last 50 epochs of training. From Fig. 8, we can see that both loss and accuracy remain stable in the last iteration. The loss keeps around 0.05, and that the accuracy stays around 94%. These results demonstrate that the DNN is not overfitted and it has got converged as expected.

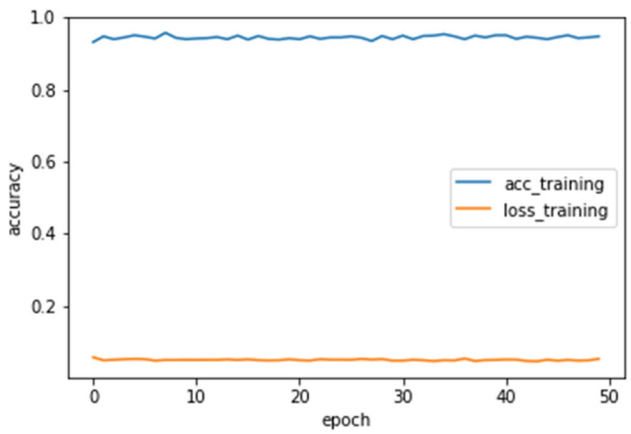

Fig. 8 Trends of the loss and accuracy during the last iteration

The accuracy on the test dataset of each iteration of training is shown in Fig. 9. As shown in Fig. 9, in the beginning, the test accuracy has significantly dropped to less than 0.1 after removing 90% of the parameters. Fortunately, the test accuracy gets improved steadily with the training iteration increasing. Finally, the pruned DNN achieves 100% test accuracy with 90 iterations of training. The test accuracy remains stable in the last 10 iterations of training. Thus, the sparse interpretation of the pruned DNN parameters is generated without losing the recognition accuracy.

6.2.2 Compression ratio

Once the DNN has been pruned and represented by a sparse matrix, the compression procedure introduced in Sect. 4 is executed. By compressing the pruned DNN, we finally reduce the storage of the entire DNN by 6.18 times, from 664,408 bytes to 107,512 bytes. Table 2 gives the final compression ratio of each layer after compression. The combination of pruning and LZW allows the DNN parameters to be stored in bytes (8-bit) rather than 32-bit floating-point format on the FPGA, avoiding extra complex quantization algorithms.

6.3 DNN overlay

By following the description given in Sect. 5, the FSCV overlay is developed and programmed into the PYNQ-Z2 board. The miniaturized DNN was integrated into the overlay. Table 3 provides the resource utilization on the PYNQ-Z2, consumed by the developed overlay. As shown in the table, the DNN consumes 47% of the BRAM, 12% of the DSP, 4% of the FF, and 14% of the LUT. The developed overlay running on the PYNQ-Z2 platform achieves execution time of 17 ms and energy consumption

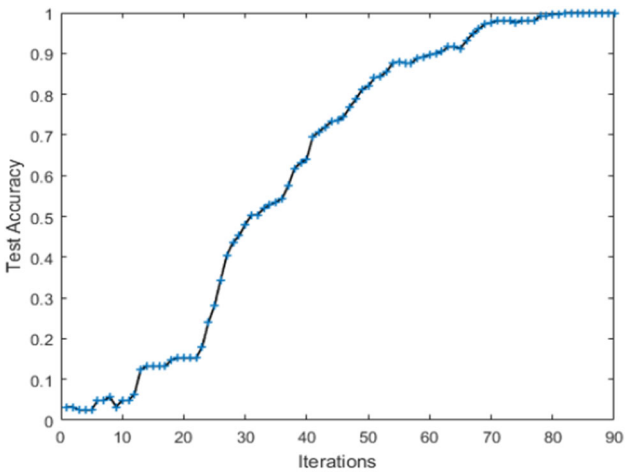

Fig. 9 Accuracy on the test dataset after each iteration of training

Table 2 Number of bytes in each layer

Layer	1	2	3	4	5	6	7	8	9	10
Number of bytes	53,699	6567	6653	6638	6617	6608	6662	6638	6654	776
Original number of bytes	340,400	40,400	40,400	40,400	40,400	40,400	40,400	40,400	40,400	808
Compression ratio	6.34	6.15	6.07	6.09	6.10	6.11	6.06	6.09	6.07	1.04

Table 3 Utilization of resources table

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	–	–	–	–	–
Expression	–	–	0	79	–
FIFO	–	–	–	–	–
Instance	133	28	4413	7155	0
Memory	1	–	2	14	0
Multiplexer	–	–	–	230	–
Register	–	–	193	–	–
Total	134	28	4608	7478	0
Available	280	220	106,400	53,200	0
Utilization (%)	47	12	4	14	0

of 1.25 W. In our earlier work [40], we fitted the DNN into the FPGA on the PYNQ-Z2 board for the first time, which consumed 96% of the BRAM utilization. Compared with that, the developed overlay in this work reduces the use of more than half of the storage space on the FPGA on the PYNQ-Z2 board.

The decompression is executed once per layer before the DNN calculation, i.e., multiply-and-addition. The decompression algorithm is executed 10 times since the DNN contains 10 layers in this work. The time complexity of the decompression is O(n). The DNN overlay maintains 17 ms under 1.25 W, which is almost the same level compared with our earlier work [40], i.e., 13 ms under 1.4 W. The decompression requires approximately 4 ms to decompress the parameters on the PYNQ-Z2 board. This shows that the decompression module designed in the DNN overlay does not cost significant delay as well as energy. Although the compression can reduce the storage requirement of the DNN parameters, we have observed a limitation that it does not simplify or accelerate the DNN operations. Further parallelism is needed to improve the speed and energy consumption in our future work.

7 Conclusion

In this paper, we introduced a DNN miniaturization algorithm, and implemented it on a low-resource FPGA to allow real-time FSCV data analysis. The key advantage of the proposed miniaturization algorithm is the combination of the pruning with the LZW compression. The pruning contributes to the formation of sparse matrices of parameters filled with approximately 90% zeroes, which can be compressed effectively by using LZW. Moreover, the miniaturization algorithm benefits from fast execution as well as no extra resources for storing the dictionary. Besides, our framework does not require quantization together with the associated ASIC platform, and it has been implemented on the FPGA of the PYNQ-Z2 platform. The compression ratio of 6.18 achieved by the miniaturization algorithm in storage level leads to the success of reducing more than half of the on-chip resources of FPGA. In addition, the development of the DNN miniaturization strategy supported customized overlay maintains the rapid execution speed of 17 ms using low energy consumption of 1.25 W.

Acknowledgements The authors would like to thank Dr. Yoonbae Oh and Dr. Kevin E. Bennet of Mayo Clinic for providing the FSCV dataset used.

References

1. Goodfellow I, Bengio Y, Courville A, Bengio Y (2016) Deep learning, vol 1. MIT press Cambridge,
2. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018. pp 4510–4520
3. Martinez AMC, Gerlach L, Payá-Vayá G, Hermansky H, Ooster J, Meyer BT (2019) DNN-based performance measures for

predicting error rates in automatic speech recognition and optimizing hearing aid parameters. Speech Commun 106:44–56
4. Zhang X, Han Y, Xu W, Wang Q (2021) HOBA: a novel feature engineering methodology for credit card fraud detection with a deep learning architecture. Inf Sci 557:302–316
5. Li Y, Huang C, Ding L, Li Z, Pan Y, Gao X (2019) Deep learning in bioinformatics: introduction, application, and perspective in the big data era. Methods 166:4–21
6. Zhou Y, Hou Y, Shen J, Huang Y, Martin W, Cheng F (2020) Network-based drug repurposing for novel coronavirus 2019-nCoV/SARS-CoV-2. Cell Discov 6(1):1–18
7. Behrendt K, Novak L, Botros R A deep learning approach to traffic lights: Detection, tracking, and classification. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017. IEEE, pp 1370–1377
8. Shvets AA, Rakhlin A, Kalinin AA, Iglovikov VI Automatic instrument segmentation in robot-assisted surgery using deep learning. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 2018. IEEE, pp 624–628
9. Wang Z, Yang Z, Dong T (2017) A review of wearable technologies for elderly care that can accurately track indoor position, recognize physical activities and monitor vital signs in real time. Sensors 17(2):341
10. Adams SD, Doeven EH, Tye SJ, Bennet KE, Berk M, Kouzani AZ (2019) TinyFSCV: FSCV for the Masses. IEEE Transactions on Neural Systems and Rehabilitation Engineering
11. Oh Y, Park C, Kim DH, Shin H, Kang YM, DeWaele M, Lee J, Min H-K, Blaha CD, Bennet KE (2016) Monitoring in vivo changes in tonic extracellular dopamine level by charge-balancing multiple waveform fast-scan cyclic voltammetry. Anal Chem 88(22):10962–10970
12. Struzyna LA, Browne KD, Brodnik ZD, Burrell JC, Harris JP, Chen HI, Wolf JA, Panzer KV, Lim J, Duda JE (2018) Tissue engineered nigrostriatal pathway for treatment of Parkinson's disease. J Tissue Eng Regen Med 12(7):1702–1716
13. Srejic LR, Wood KM, Zeqja A, Hashemi P, Hutchison WD (2016) Modulation of serotonin dynamics in the dorsal raphe nucleus via high frequency medial prefrontal cortex stimulation. Neurobiol Dis 94:129–138
14. Purcell EK, Becker MF, Guo Y, Hara SA, Ludwig KA, McKinney CJ, Monroe EM, Rechenberg R, Rusinek CA, Saxena A (2021) Next-generation diamond electrodes for neurochemical sensing: challenges and opportunities. Micromachines 12(2):128
15. Nasri B, Wu T, Alharbi A, You K-D, Gupta M, Sebastian SP, Kiani R, Shahrjerdi D (2017) Hybrid CMOS-graphene sensor array for subsecond dopamine detection. IEEE Trans Biomed Circuits Syst 11(6):1192–1203
16. Guo Y, Werner CF, Handa S, Wang M, Ohshiro T, Mushiake H, Yoshinobu T (2021) Miniature multiplexed label-free pH probe in vivo. Biosens Bioelectron 174:112870
17. Puthongkham P, Venton BJ (2020) Recent advances in fast-scan cyclic voltammetry. Analyst 145(4):1087–1102
18. Kim J, Oh Y, Park C, Kang YM, Shin H, Kim IY, Jang DP (2019) Comparison study of partial least squares regression analysis and principal component analysis in fast-scan cyclic voltammetry. Int J Electrochem Sci 14:5924–5937
19. Puthongkham P, Rocha J, Borgus JR, Ganesana M, Wang Y, Chang Y, Gahlmann A, Venton BJ (2020) Structural similarity image analysis for detection of adenosine and dopamine in fast-scan cyclic voltammetry color plots. Anal Chem 92(15):10485–10494
20. Falconi C, Mandal S (2019) Interface electronics: state-of-the-art, opportunities and needs. Sensors Actuators A: Phys
21. Sze V, Chen Y-H, Yang T-J, Emer JS (2017) Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE 105(12):2295-2329
22. Verhelst M, Moons B (2017) Embedded deep neural network processing: algorithmic and processor techniques bring deep learning to iot and edge devices. IEEE Solid-State Circuits Mag 9(4):55–65
23. Zhang Z, Kouzani AZ (2020) Implementation of DNNs on IoT devices. Neural Comput Appl 32(5):1327–1356
24. Abdelfattah MS, Dudziak Ł, Chau T, Lee R, Kim H, Lane ND Best of both worlds: automl codesign of a cnn and its hardware accelerator. In: 2020 57th ACM/IEEE Design Automation Conference (DAC), 2020. IEEE, pp 1–6
25. Reagen B, Whatmough P, Adolf R, Rama S, Lee H, Lee SK, Hernández-Lobato JM, Wei G-Y, Brooks D Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016. IEEE, pp 267–278
26. Iandola F, Keutzer K Small neural nets are beautiful: enabling embedded systems with small deep-neural-network architectures. In: Proceedings of the Twelfth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion, 2017. ACM, p 1
27. Han S, Mao H, Dally WJ (2015) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:151000149
28. Ma X, Guo F-M, Niu W, Lin X, Tang J, Ma K, Ren B, Wang Y Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In: Proceedings of the AAAI Conference on Artificial Intelligence, 2020. vol 04. pp 5117–5124
29. Nan K, Liu S, Du J, Liu H (2019) Deep model compression for mobile platforms: a survey. Tsinghua Sci Technol 24(6):677–693
30. Li Z, Wang Z, Xu L, Dong Q, Liu B, Su C-I, Chu W-T, Tsou G, Chih Y-D, Chang T-YJ (2020) RRAM-DNN: an RRAM and model-compression empowered all-weights-on-chip DNN accelerator. IEEE J Solid-State Circuits
31. Guo Y (2018) A survey on methods and theories of quantized neural networks. arXiv preprint arXiv:180804752
32. Nicodemo N, Naithani G, Drossos K, Virtanen T, Saletti R Memory requirement reduction of deep neural networks for field programmable gate arrays using low-bit quantization of parameters. In: 2020 28th European Signal Processing Conference (EUSIPCO), 2021. IEEE, pp 466–470
33. Hernández G, Zamora E, Sossa H, Téllez G, Furlán F (2020) Hybrid neural networks for big data classification. Neurocomputing 390:327–340
34. Gómez-Flores W, Sossa H (2021) Smooth dendrite morphological neurons. Neural Netw 136:40–53
35. Guo K, Zeng S, Yu J, Wang Y, Yang H (2019) [DL] A survey of FPGA-based neural network inference accelerators. ACM Transactions Reconfig Technol Syst (TRETS) 12(1):1–26
36. Uthayakumar J, Vengattaraman T, Dhavachelvan P (2021) A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. J King Saud Univ Comput Inf Sci 33(2):119–140
37. Moffat A (2019) Huffman coding. ACM Comput Surv (CSUR) 52(4):1–35
38. Capon J (1959) A probabilistic model for run-length coding of pictures. IRE Transactions Information Theor 5(4):157–163
39. PM N, Chezian RMA (2013) Survey on lossless dictionary based datacompression algorithms. Int J Sci Eng Technol Res 2(2):256–261
40. Zhang Z, Yoonbae Oh, Adams SD, Bennet KE, Kouzani AZ (2020) An FSCV deep neural network: development, pruning,

and acceleration on an FPGA. IEEE J Biomed Informatics. https://doi.org/10.1109/JBHI.2020.3037366
41. Lee KH, Lujan JL, Trevathan JK, Ross EK, Bartoletta JJ, Park HO, Paek SB, Nicolai EN, Lee JH, Min H-K (2017) WINCS Harmoni: Closed-loop dynamic neurochemical control of therapeutic interventions. Sci Rep 7:46675
42. Kimble CJ, Johnson DM, Winter BA, Whitlock SV, Kressin KR, Horne AE, Robinson JC, Bledsoe JM, Tye SJ, Chang S-Y Wireless instantaneous neurotransmitter concentration sensing system (WINCS) for intraoperative neurochemical monitoring. In: 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2009. IEEE, pp 4856–4859
43. Bottou L (2012) Stochastic gradient descent tricks. Neural networks: Tricks of the trade. Springer, New York, pp 421–436