Enhanced Automatic DNN Acceleration

Mengyun Liu
Department of Electrical and Computer Engineering
Duke University
Durham, NC, USA
Email: mengyun.liu@duke.edu

Rana Elnaggar
Department of Electrical and Computer Engineering
Duke University
Durham, NC, USA
Email: rana.elnaggar@duke.edu

Naman Jain
Department of Computer Science
Duke University
Durham, NC, USA
Email: naman.jain@duke.edu

***Abstract*—Deep neural networks (DNN) are currently used in many applications like autonomous driving that requires real-time inference with high classification accuracy. The most time-consuming operations of convolution neural networks are operations performed in the convolution layers. FPGAs are attractive DNN acceleration platforms. FPGAs utilize DSP slices to perform multiplication operations. However, the number of DSP slices are limited in FPGASs. Therefore, we propose an efficient encoding scheme that reduces the number of the multiplication operations required in the convolution layer by 30% compared to prior work without reducing the classification accuracy. We also propose to enhance the currently available automatic DNN synthesis tool called "DNNWEAVER" with our proposed encoding approach.**

I. Introduction

Deep Neural Networks (DNNs) are used widely in the field of image classification and the convolutional layers end up taking a large proportion of the execution time [1]. When surveying through the various image classification techniques, it was found that most of them use bit-parallel computation with 16-bit fixed-point or quantized 8-bit encoding. The bit-parallel computing is better than bit-serial computing for the aforementioned encoding schemes. However, there is a major problem: In these encoding schemes, there may be a lot of zeroes in the input data vectors. When there is any computation where multiplication with a zero is involved, the final result of that computation is zero. But still the computation takes place and the result of this computation is useless and wastes computation power. So, it is important to come up with a different encoding system that avoids the multiplication of zeros without affecting the accuracy of the computation.

A DNN accelerator, Pragmatic [2], came up with a different kind of encoding scheme where they only record the positions of the ones in the 16-bit encoding scheme and store that. It completely ignores the zeroes as they just lead to additional computation. They use bit-serial computation as bit-parallel will not be possible in such a case. However, the problem lies with some specific cases. In case of image classification, we see that there are a lot of cases where the number of ones is a lot and as we are using bit-serial computation, the performance decreases. Another problem is that each neuron has different number of essential bits and hence a synchronization mechanism needs to be used. So, there is a need for an encoding mechanism that fixes these problems without reducing the classification accuracy. We propose an encoding scheme called "Bit-Complementary encoding". This scheme records the position of bits with a value of '1' ('0') if the number of bits with a value of '1' ('0') is less than the number of bits with a value of '0' ('1'). However, this scheme does not solve the synchronization problem. Therefore, we propose optimum approximation encoding method which uses fixed number of essential bits for an approximated expression. However, this technique suffers from high time complexity. Hence, we propose another approximation method which is known as fast approximation. This technique only expresses the significant essential bits.

In the case of Deep Neural Networks (DNNs), the main focus is to allot a high fraction of the execution time to computational requirements. Applications in this area are growing as a lot of fields are recognizing the importance and advantages of using DNNs. Using Field Programmable Gate Arrays (FPGAs) for DNNs is a viable choice as they provide the function of programmability which is very useful from the perspective of a researcher. As programmability allows the construction of various types of circuits, FPGAs are being used in a lot of areas and in a number of ways.

DNNWEAVER [3] is a software, which when given a (DNN, FPGA) pair, produces a synthesizable accelerator by means of hand-optimized design templates. Initially, a given high-level DNN specification is converted to an instruction set architecture (ISA), which is represented in a macro dataflow graph of the DNN. The main aim of DNNWEAVER is to maximize data reuse and best utilize target FPGAs memory by scheduling the DNN operations in the best way possible. What is obtained at the end is a synthesizable accelerator, which can be customized and offers high increase in the performance and efficiency for the target FPGA while not compromising the functions of the DNN. With the help of this frameworks output, programmers do not have to spend extra time on designing the hardware.

Multiplication operations in convolution layers utilize DSP slices. The number of DSP slices on FPGA is limited. However, DNNWEAVER does not utilize any optimization algorithm to optimize the use of DSP slices or reduce the number op operations performed in the convolution layers. Therefore, we propose to enhance DNNWEAVER with our

proposed encoding method. The contributions of this work are as follows:

- We propose an efficient encoding scheme that uses only representative bits in the input for classification without affecting the overall classification accuracy.
- We successfully replicate the DNNWEAVER platform.
- We propose to integrate our encoding scheme within DNNWEAVER framework.

The remainder of the paper is organized as follows. Section II describes related prior work. Section III describes our proposed encoding scheme, the steps for replicating DNNWEAVER and how we intend to integrate our encoding scheme in DNNWEAVER framework. In Section IV, we describe our experimental setup and show the simulation results. In Section V, we discuss the insights about our proposed encoding methods. Finally, we conclude the paper and propose future work in Section VI.

II. Related Work and Technical Background

A. Deep Neural Network

The most basic part of a typical neural network is a neuron which generates a set of activations. Deep Learning is responsible for producing accurate results but consists of lots of layers of these neurons. On the other hand shallow neural networks consist of a few consecutive non-linear layers. Deep Neural networks are highly required for Unsupervised learning although they are very productive in Supervised learning too. Applications of DNNs are increasing for Reinforcement Learning. Pattern recognition and classification is one of the best uses of DNNs. Feed-forward neural networks (FNNs) and recurrent neural networks (RNNs) are types of DNNs. However, RNNs are better than FNNs in terms of computation power. RNNs are very efficient when it comes to learning programs that support processing of parallel data which used to be a big deal before RNNs [4]. An unprecedented process known as automatic feature extraction is executed by DNNs and it does not require any external help from the users/ scientists [5]. This technique is a huge benefit of using DNNs as it reduces human involvement substantially. Also, this leads to easier processing of big data. In case of unsupervised learning, DNN tries to restructure the input data and learns as much as possible in order to guess better.

B. Automatic DNN acceleration

When considering the processing part of a neural network, high model accuracy, better throughput and increased energy efficiency are very important. Usually, bigger the network more is the model accuracy. In most cases, the throughput and model accuracy are inversely proportional with some non-linear factor. By compressing the models with various techniques, the input can be lowered in size, reducing the storage requirements and hence, the network has less data to work on. The peak performance of an FPGA chip can be improved by increasing the working frequency and decreasing the area of the computation units. In a lot of cases, by decreasing the number of bits or number of weights, the model accuracy decreases [6]. There are broadly two types of quantization methods used for model compression, linear quantization and non-linear quantization. Linear quantization rounds off to the nearest fixed-point representation of each weight and neuron, which obviously results in loss of data and accuracy. In one of the papers [7], to evade an exponential design space, the most efficient solution of a network is selected on a per layer basis. In case of non-linear quantization, there is a look-up table which maps the quantized code to its relevant value. In one of the papers [8], weights are assigned to all the items in the look-up table by using a specific hash function. In a follow-up to that paper [9], the author allots weights to the table values by taking a trained model and clustering its weights. This results in compression of weights of CNN models to 4-bits without any loss of accuracy.

C. Encoding Methods for DNN computing acceleration

The most basic way of representing data in DNNs is the 16-bit fixed-point. The positions of 0s and 1s represent the powers of 2 which when multiplied by the number at that position contribute to the final number in the decimal form. Another representation is 8-bit quantized numbers. In a fairly recent DNN accelerator, STR [10], the proposed encoding allows each and every layer to select its own precision which in turn improves performance. In this technique, execution time is directly proportional to the number of bits required by every layer of the network. The network was built upon DaDN. Now, DaDN uses 16-bit fixed-point representation but STR uses less than or equal to 16 for every layer, hence, improving performance. STR uses serial-parallel multiplication and the length of the serial input defines time of computation. However, this results in more computational time and hence, a large amount of parallel computation is also done to mitigate this latency and increase the throughput. After STR another DNN accelerator, Pragmatic [2], altered the encoding scheme and focused on only recording the positions of the ones in other basic encoding schemes like 16-bit fixed-point and 8-bit quantized, and store that. The zeroes are disregarded as they are there due to lack of explicitness and excess of precision and they do not contribute to the output. As it focuses only on bits individually instead of the set of bits as a whole number, it is bound to use bit-serial computation, which in normal cases is slower than bit-parallel computation. But this type of encoding ends up decreasing the performance in case there are a lot of ones. In that case, it is almost equivalent to bit-serial computation with normal encoding, which is obviously more expensive than bit-parallel.

III. Design Description and Technology Contributions

A. Encoding-related

1) Bit-Complementary Encoding: The existing encoding methods that reviewed in Section II-C has two major challenges: (1) not suitable for the binary expressions with lots

of bits equal '1'. (2) synchronization mechanism is needed. In order to address these challenges, a bit-complementary encoding method was proposed in this project. The main idea of the proposed bit-complementary encoding method is to handle the binary expressions with more '0's and the expressions with more '1's in different way. When the binary expression of a neuron input has a small number of bits equal '1', we would recognize these bits equal '1's as the essential bits, and record only their positions, namely 1-based expression. This is the same mechanism used in the bit-pragmatic encoding [2]. When the binary expression of a neuron input has more bits equal '1' than the bits equal '0', instead of recording the positions of bits equal '1', we take bit-complementary of the original binary expression, and then record the position of bits equal '1' to save computation operations. This is the same as recording the position of '0's in the original expression, hence we call it 0-based expression. For example, $132_{(10)} = 10000100_{(2)}$, its binary expression contains only two '1's, so we record the expression as $1:(7,2)$. But for $254_{(10)} = 11111110_{(2)}$, the number of bits equal '1' is much larger than the number of bits equal '0'. By applying the proposed bit-complementary encoding, we record the position of '0's as $0:(0)$. The complementary expression is obtained by taking bit-complementary for each bit from the original expression. In this way, $254_{(10)} = 11111110_{(2)} = 11111111_{(2)} - 00000001_{(2)}$. Therefore, we need only to record the position of 1 in the complementary expression $00000001_{(2)}$.

In order to obtain the same computing results, two more subtraction operations are required for 0-based expressions. For example, $254_{(10)} = 11111110_{(2)} = 11111111_{(2)} - 00000001_{(2)} = 100000000_{(2)} - 1_{(2)} - 00000001_{(2)}$, where the last operand $00000001_{(2)}$ is the 0-based expression, and the first two operands are needed to get the correct results when using 0-based expression. For this case, seven shift and add operations are needed when the bit-pragmatic encoding method is used. However, only three shift and subtract operations are needed when we choose 0-based expression in the proposed bit-complementary encoding method.

2) Optimum approximation of bit-complementary encoding: The proposed bit-complementary encoding are useful for addressing the challenge of having many '1's in the binary expression. However, by applying this method, it is still impossible for us to ensure that each neuron input has the same number of essential bits. Inspired by the fact that CNN has a strong capability to tolerate some input differences, we further propose an approximation of the bit-complementary encoding. The main idea of this approximated encoding method is to use a fixed number of essential bits to record an approximated expression. In this way, no synchronization mechanism is needed since all neurons are recorded with a fixed number of essential bits. For example, $248_{(10)} = 11111000_{(2)}$, since there are more '1's than '0's in the binary expression, we carry out the complementary encoding: $11111000_{(2)} = 11111111_{(2)} - 00000111_{(2)}$. Based on the design, we use 0-based expression, which records the position of '1's in the complementary expression $00000111_{(2)}$. However, it still has three essential bits. If we are required to use no more than one essential bits in a 0-based expression, approximation is needed. In stead of recording $00000111_{(2)}$, we record its approximation $00001000_{(2)}$, which contains only one essential bit with a difference equal 1. The difference is small in this example, and can be tolerated by our application.

In order to analyze the feasibility of this approximation, we proved that for any N-bit expression, when we are required to use no more than $m1$ bits in a 1-based expression or use no more than $m0$ bits in a 0-based expression, we can always find an optimum approximation which generate a difference no larger than $2^{N-m1-m0-1}-1$. For a typical parameter configuration, $N = 8, m1 = 3, m0 = 1$, (m1 = m0+2 because 0-based expressions require two more subtract operations during computing), the maximum difference is 7, compare with the maximum value that a 8-bit expression can denote, $7/255 = 2.7\%$. For another typical parameter configuration, $N = 16, m1 = 4, m0 = 2$, the maximum difference over the maximum value is only $511/65535 = 0.8\%$. The ratio of difference is small and acceptable in DNN applications. However, the optimum approximation can be obtained only by carrying out a brutal force search algorithm, and the time complexity is extremely high, i.e., $O(2^{(n-m1-m0)})$.

3) Fast approximation of bit-complementary encoding: In order to implement the encoding in practical, a fast approximation approach is proposed. The main idea of this fast approximation is strait-forward, which is to record only the most significant essential bits. For an 8-bit expression $11111000_{(2)}$, suppose that we are required to use no more than three essential bits in 1-based expression, and use no more than one essential bit in 0-based expression. For 1-based expression, instead of recording the positions of all the five bits equal '1', we will record only the three-most significant bits that are '1', in this way 1111100011100000, and the difference is $11000_{(2)} = 24_{(10)}$. For 0-based expression 00000111, instead of recording all the three bits equal '0', we will record only the most significant bit as 00000100, hence the difference is $11_{(2)} = 3_{(10)}$.

In order to verify the usefulness of this fast approximation technique, we compare the differences caused by the optimum approximation and the fast approximation. The results in TABLE I are evaluated using three typical settings. As shown in the table, the fast approximation generate same maximum differences as the optimum approximation method. However, by carrying out the fast approximation method among all the possible expressions, the average difference is around 30% higher than the optimum approximation. The experimental results in Section IV proves that these differences are acceptable. However, the time complexity of the fast approximation is $O(m1 + m0)$, which reduce dramatically from the optimum approximation, and hence can be easily realized in practical.

B. FPGA-related

FPGAs are attractive solutions for CNN acceleration due to the improved speed up they provide at relatively low power consumption. However, the synthesis of the different CNN models on different types of FPGA boards to accelerate them is challenging. Therefore, different automatic CNN acceleration synthesis frameworks have been proposed [3] [11]. However, the available automatic CNN accelerator synthesis frameworks do not include optimization algorithms to further speed up the CNN and reduce its energy consumption. Therefore, we propose to enhance DNNWeaver automatic FPGA acceleration synthesis to include input encoding method to reduce the number of required calculations. This will lead to overall energy consumption and DSP slices usage reduction. In addition, this will lead to overall improved speed up in convolution layer computations. We propose to integrate our proposed input encoding within the DNNWeaver flow. DNNweaver automatic synthesis flow is shown in Fig.1. As shown in Fig.1, during the design planner phase, DNNWEAVER uses hardware design templates that represent the operations that are performed in every layer of the convolution neural network. The processing element blocks (PEs)—highlighted in red colour—perform multiplication operations in the convolution layers.

Currently multiplication in convolution layers includes all the bits in the input data vector. Currently, the multiplication is performed using the multiplier accumulator Xilinx IP core [12]. We propose to implement the proposed encoding algorithm in the PE blocks in DNNWEAVER flow. Moreover, we propose to utilize the multiplier accumulator Xilinx IP core to multiply encoded inputs—with only the most essential bits—by the weights of each layer.

DNNWEAVER is implemented on FPGA SoC board. FPGA SoC integrates arm cores (processing system) and programmable logic fabric. The CNN accelerator is implemented on the programmable logic and the software that passes in the data input vectors and weights to be classified by the accelerator operates on the arm core. A simplified diagram to show the interactions within the FPGA SoC board is shown in Fig. 2. Fig. 2 shows that the processing system and the programmable logic share some memory regions for data exchange. Therefore, the software running on the arm core (processing system) writes the data to the shared memory, then when a specific memory location is written, the accelerator implemented in the programmable logic starts its operations. The results can be written back to the shared memory and accessed by the arm core for final classification evaluation. Therefore, to replicate DNNWEAVER platform, we need to run the controlling software successfully on the arm core within the Zynq FPGA SoC and be able to communicate with the implemented accelerator. Therefore, the second part of this project is focused on trying to run the accelerator successfully on the Zynq FPGA SoC board.

IV. Experimental Procedures and Results

A. Setup of DnnWeaver and expected results

In order to integrate our proposed encoding method with DNNWeaver, we first replicate DNNWeaver to establish the baseline of our framework and comparisons and then start to add our proposed modules. DNNWeaver requires a Xilinx Zynq ZC7020 FPGA SoC board. In our experiment, we connected the Zynq FPGA SoC with a host PC that interfaces with the Zynq FPGA SoC board through a serial port. We boot Petalinux version 2017.14 [13] on the FPGA SoC board. Next, we transfer the software that operates on the arm core and load the input and weight vectors into the shared memory. Next, we run the software and observe the output of CNN classification printed on the host screen. Our experimental framework setting along with the displayed output on the screen are shown in Fig. 3. We have not implemented the proposed encoding scheme in hardware yet. We plan to do it during the summer.

It is also important to note that we have tried to run the controlling software the Zynq FPGA SoC board using Xilinx SDK. However, cross-compilation of the controlling software to run on the arm cores through Xilinx SDK is challenging. Moreover, tcf agent needs to be running on the arm cores. To activate the tcf agent, we had to rebuild petalinux from scratch using Xilinx Petalinux SDK. Therefore, we recommend the method of booting Petalinux on the arm core and directly run the controlling software as shown in Fig. 3.

B. Encoding Evaluations

We evaluate our encoding scheme in terms of the number of operations required and the change in accuracy when encoding is used. We use VGG16 CNN network and use it to classify 570 images of the validation dataset of Imagenet dataset. We use the pre-trained VGG16 in Keras. We load all the weights of the pre-trained networks and use it to evaluate the output of each layer and then apply the encoding scheme and pass the output of the encoding scheme as an input to the next convolution layer in the network; see Fig. 4.

Experiment 1: Required Number of Operations

We implemented our proposed encoding schemes and the pragmatic encoding scheme in python and compare the number of required operations per layer when each method is used. The results are shown in Fig. 5. The results show that our proposed encoding scheme requires less number of operations compared to pragmatic encoding at all the different convolution layers. The average reduction in needed operation is 30% compared to pragmatic encoding method.

Experiment 2: Effect of Encoding Scheme on Accuracy

We investigate the change in overall classification accuracy when optimum bit-complementary encoding and its fast approximation are performed on the input to a single convolution layer, two convolution layers and all convolution layers in the network. The results are shown in Fig.6. The results show that our proposed encoding method as well as its fast approximation preserve the classification accuracy of

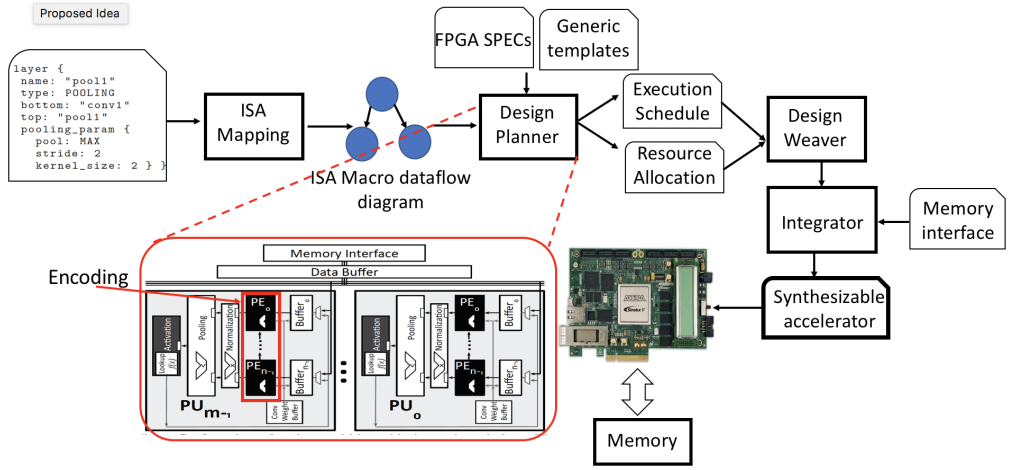

Fig. 1 DNNWeaver Synthesis Flow

TABLE I Comparison between the optimum approximation and the fast approximation

	Configuration	N=8, m1=3, m0=1	N=16, m1=3, m0=1	N=16, m1=4, m0=2
Optimum Approximation	Maximum diff	7	2047	511
	Average diff	1.55	436.02	102.60
Fast Approximation	Maximum diff	7	2047	511
	Average diff	1.96	615.80	144.94

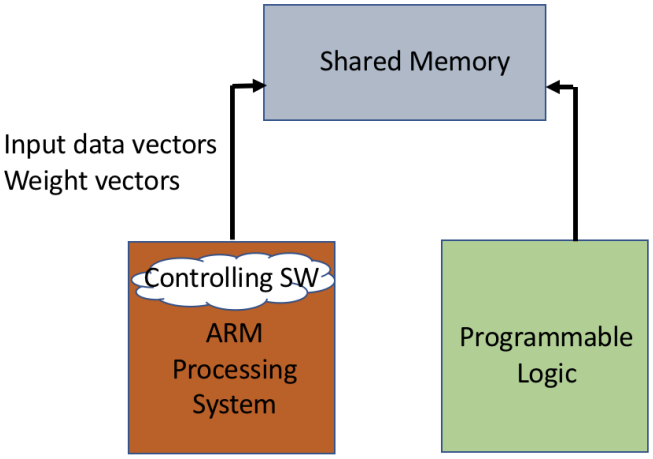

Fig. 2 Simplified illustration of the interaction between arm core and FPGA programmable logic

Imagenet using VGG16. It is important to note that "4-2" and "3-1" settings represent the number of positions we will record for the essential bits if they have a value of '1' or '0'. For example, "4-2" means that when we record the position of '1's, we only record the position of the 4-most significant bits which are '1's. Similarly, when we record the position of '0's, we only record the position of the 2-most significant bits which are '0's. The reported results of the optimum complementary encoding is performed using "8-6" settings.

Therefore, we propose as a future work to apply the encoding method to weights as well as input vectors and evaluate the effect of encoding the network weights on the classification accuracy.

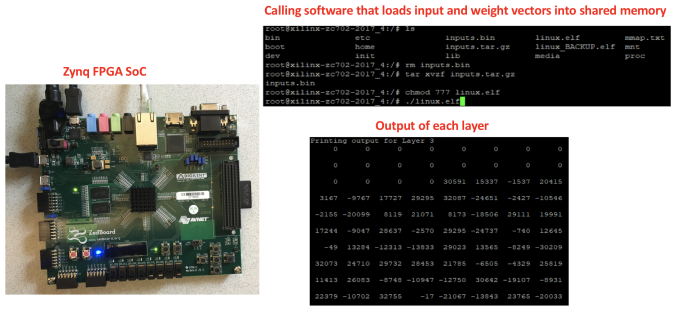

Fig. 3 Simplified illustration of the interaction between arm core and FPGA programmable logic

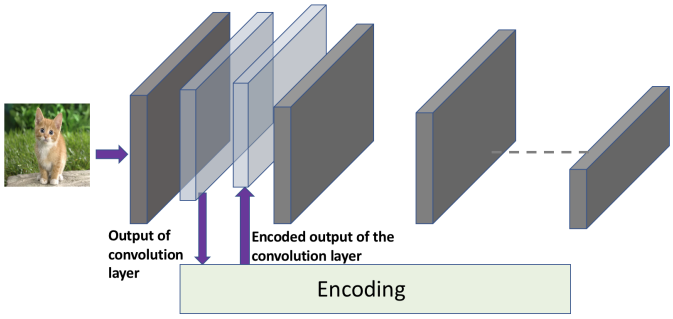

Fig. 4 Encoding Integration within VGG16 flow

V. Insights

This project explore the potential of acceleration of DNN computing with complementary encoding method and making approximations. Studies have found that among the computing of DNN, convolutional layers take most of the

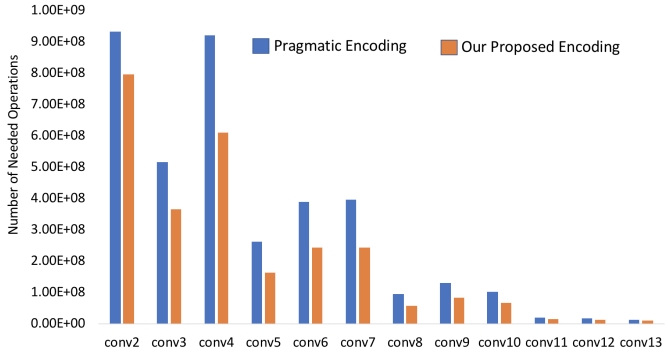

Fig. 5 Comparison of our proposed encoding scheme and pragmatic encoding

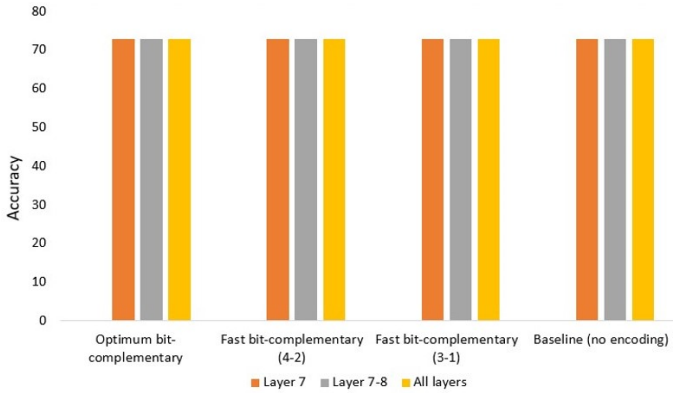

Fig. 6 Comparison of our proposed encoding scheme and pragmatic encoding

computing time. Hence lots of studies have been done to accelerate the computing of convolution. The parallelism of the covolutional layer enable us to realize bit-serial computing, which feed in only one bit of input from neurons in each cycle. We utilize the complementary characteristic of binary expressions, and record only either the positions of '1's or the positions of '0's, depending on which value occurs less commonly. In this way, we only need to record and calculate for at most half of the bits. Therefore, compared with the original bit-serial computing, at least two times of speedup can be achieved. Compared with the existing bit-pragmatic computing, the proposed method achieve the same performance when the number of bits equal '1' is small, while our method outperforms when the number of bits equal '1' is large.

However, subtract operations are needed for processing 0-based expressions. New hardware designs are needed to replace the adder trees in original processing elements. The time consumption of encoding process has not been discussed in this report, but the encoding process can be pipelined, and will not affect the throughput. Because of the robustness of the DNN, experimental results show that approximation does not hurt the accuracy of recognition. However, in the experimental parts, we show only the results compared with using 16-bit pragmatic computing, but some studies have proposed to use 8-bit quantized expression instead. Therefore, in order to better prove the usefulness of the proposed encoding method, more experiments are needed to evaluate the speedup and accuracy change comparing among the applications that use 8-bit quantized expressions.

VI. Conclusion and Future Work

In conclusion, we propose a new encoding scheme that reduces the number of required multiplication operations in the convolution layer without affecting the overall classification accuracy. In order to integrate our proposed encoding method to DNNWEAVER's design flow, we successfully replicated DNNWEAVER platform.

In the future, we intend to apply our encoding scheme on hardware. Moreover, we propose to apply our proposed encoding method to weight vectors as well as input data vectors. In addition, we plan to investigate the effect of our proposed encoding scheme on hybrid neural networks and in neural networks that use low-precision numeric computations. Although DNNWEAVER presents an automatic synthesis platform for CNN on FPGAs, the current version of DNNWEAVER is strictly dependant on Zynq FPGA SoC. Thus, the automatic synthesis of CNN on different types of FPGA boards is challenging. Therefore, we propose to modify the flow of DNNWEAVER to be independent of Zynq SoC.

References

[1] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *International conference on artificial neural networks*, pp. 281–290, Springer, 2014.

[2] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 382–394, ACM, 2017.

[3] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pp. 1–12, IEEE, 2016.

[4] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[5] D. Team, "Deeplearning4j: Open-source distributed deep learning for the jvm," *Apache Software Foundation License*, vol. 2, 2016.

[6] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of fpga based neural network accelerator," *arXiv preprint arXiv:1712.08934*, 2017.

[7] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 26–35, ACM, 2016.

[8] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, pp. 2285–2294, 2015.

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[10] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on*, pp. 1–12, IEEE, 2016.

[11] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J.-s. Seo, "Alamo: Fpga acceleration of deep learning algorithms with a modularized rtl compiler," *Integration*, 2018.

[12] "The multiplier accumulator xilinx ip core." https://www.xilinx.com/products/intellectual-property/multiplier_accumulator.html. Accessed: 2018-04-29.

[13] "Petalinux." http://www.wiki.xilinx.com/Zynq2017.4%20Release. Accessed: 2018-04-29.