NN Compactor: Minimizing Memory and Logic Resources for Small Neural Networks

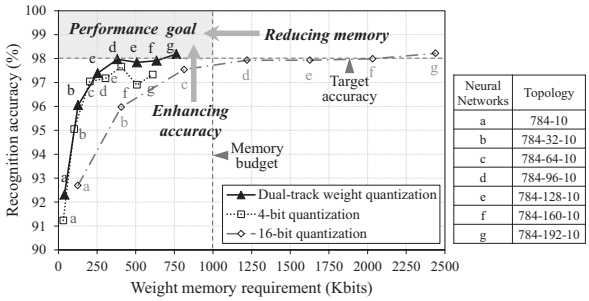
Seongmin Hong[1], Inho Lee[1], and Yongjun Park[2]
[1]Dept. of Electronic and Electrical Engineering, Hongik University, Seoul, Korea
[2]Dept. of Computer Science, Hanyang University, Seoul, Korea
{seongminhong, inholee}@mail.hongik.ac.kr, yongjunpark@hanyang.ac.kr

***Abstract*—Special neural accelerators are an appealing hardware platform for machine learning systems because they provide both high performance and energy efficiency. Although various neural accelerators have recently been introduced, they are difficult to adapt to embedded platforms because current neural accelerators require high memory capacity and bandwidth for the fast preparation of synaptic weights. Embedded platforms are often unable to meet these memory requirements because of their limited resources. In FPGA-based IoT (internet of things) systems, the problem becomes even worse since computation units generated from logic blocks cannot be fully utilized due to the small size of block memory. In order to overcome this problem, we propose a novel dual-track quantization technique to reduce synaptic weight width based on the magnitude of the value while minimizing accuracy loss. In this value-adaptive technique, large and small value weights are quantized differently. In this paper, we present a fully automatic framework called *NN Compactor* that generates a compact neural accelerator by minimizing the memory requirements of synaptic weights through dual-track quantization and minimizing the logic requirements of PUs with minimum recognition accuracy loss. For the three widely used datasets of MNIST, CNAE-9, and Forest, experimental results demonstrate that our compact neural accelerator achieves an average performance improvement of $6.4\times$ over a baseline embedded system using minimal resources with minimal accuracy loss.**

***Keywords*—Neural networks; Accelerator; Automation; Quantization**

I. Introduction

Recently, neural networks have been successfully researched for various machine-learning algorithms, such as computer vision [13], speech recognition [6], and image classification [9]. GPUs are currently the most successful hardware solution for utilizing neural networks in these applications. However, because the massive amount of data transfer between neurons has become the main bottleneck for the low performance of GPU implementations, various ASIC/FPGA-based special accelerators have been proposed to enhance data transfer efficiency between memory and processing units [11, 12]. While most of these neural accelerators target large neural networks running on server class machines, the increasing demand to adapt neural networks for small Internet of Things (IoT) devices, such as smart watches and drones, has pushed solution designers to add neural accelerators to embedded solutions. In contrast to server-class neural accelerators, neural accelerators for small neural networks have relatively tighter constraints to achieve target performance levels and must be designed more carefully to meet the tight cost and energy efficiency requirements of resource-constrained IoT systems.



Neural Networks	Topology
a	784-10
b	784-32-10
c	784-64-10
d	784-96-10
e	784-128-10
f	784-160-10
g	784-192-10

Fig. 1. Weight memory requirement and recognition accuracy exploration for various neural accelerators in different quantization levels with MNIST dataset

To deploy neural accelerators in IoT solutions, one of the most challenging obstacles is their large memory requirement. To mitigate this capacity problem, 16-bit fixed-point value based neural accelerators [5, 10] have been widely adopted because weight quantization from 32-bit floating-point values to 16-bit fixed-point values has almost no impact on final accuracy. However, total memory capacity must be decreased further for highly resource-constrained IoT systems. Figure 1 illustrates the relationship between the weight memory requirement and recognition accuracy for neural accelerators in different network topologies for the MNIST dataset. In order to meet both accuracy and memory size requirements, a new quantization technique to minimize memory size is required. Another problem is to determine the optimal number of processing units (PUs). The maximum number of available PUs within the logic area budget is not always necessary because performance may become saturated at some number of PUs less than the maximum number. Therefore, the number of PUs should be carefully selected to achieve the highest power and area efficiency.

To solve these problems, we present a design of *NN (Neural Network) Compactor*, a framework for automatically creating compact neural accelerators for small neural networks on target devices by minimizing the memory requirement of synaptic weights and logic requirement of PUs with minimum recognition accuracy loss. The major contributions of this paper can be summarized as follows:

- We propose a novel dual-track quantization technique that reduces weight width to 5 bits. This enables neural accelerators to use more weights within an FPGA's limited on-chip memory size.
- We design a scalable accelerator, and the number of PUs is automatically determined through design space exploration.

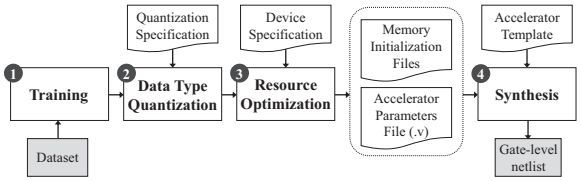


Fig. 2. Overview of NN compactor which is consisted of four stages

- We provide a fully automatic framework that generates an optimized accelerator for target neural networks.

Performance and accuracy evaluation is performed using six different networks on three datasets (MNIST [3], CNAE-9 [1], and Forest [2]), with a target FPGA platform [7]. The neural accelerator generated by NN compactor achieves an average performance improvement of $6.4\times$ over traditional embedded cores with minimal accuracy loss and minimal resource utilization.

II. Overview

NN compactor consists of four stages: training, data type quantization, resource optimization, and synthesis processes as shown in Figure 2. Three processes from training to resource optimization are performed using a modified TensorFlow and the synthesis process can be performed using various synthesis tools.

Training: Various neural networks for target applications can be implemented by utilizing various training methods via TensorFlow. The input for this process can be a target application dataset, such as MNIST, CNAE-9, or Forest, and the outputs for the next process are weights, biases, and a target neural network topology.

Data Type Quantization: This process performs quantization of synaptic weights from the original 32-bit floating-point data type to one of three more compact data types: 16-bit fixed-point, 4-bit fixed-point, or 5-bit custom. The 5-bit custom data type values are generated from 8-bit fixed-point data values using a dual-track quantization technique that will be discussed in Section IV.

Resource Optimization: This process finds the minimum number of PUs required to achieve the target performance, which will be discussed in Section V. This process then produces memory initialization files and an accelerator parameter file based on resource allocation.

Synthesis: In this process, a real gate-level netlist is generated for the target accelerator, which would be an FPGA or ASIC. Two types of input files from the previous processes are required, as shown in Figure 2. Memory initialization files are the sets of synaptic weights and biases (.mif files on Altera FPGAs). An accelerator parameter file contains various parameters for the target accelerator configuration including the number of PUs and the data type of the weights. This process synthesizes the design into a gate-level netlist as the final output.

III. NN Compactor Architecture

Figure 3 presents an architectural overview of a target neural accelerator generated by NN compactor. It consists of a finite state machine (FSM), a controller, an input buffer, an output buffer, processing units, weight memories, bias memories, and neuron memories. All modules are controlled by a controller in the form of an FSM. Weight memories are specially designed to support multiple data types, including the proposed 5-bit custom data type. Therefore, the accelerator requires additional flag memory blocks and decoders. PUs contain a multiplier, an accumulator, an activation function block, and a forwarding block, as shown in Figure 4. The multiplier and the accumulator are used to generate the inner products of weights and neuron data, and a rectified linear unit (ReLU) is used as an activation function. The PUs are pipelined and additional forwarding blocks are implemented to avoid data hazards. In the PUs, the data type of the neurons and biases is 16-bit fixed-point (integer (I): 6-bit, fraction (F): 10-bit), but the data type of the weights can vary based on the quantization specification during the data type quantization process. When using the 5-bit custom weights, a decoded 8-bit fixed-point data type is used as the input for the multiplier.

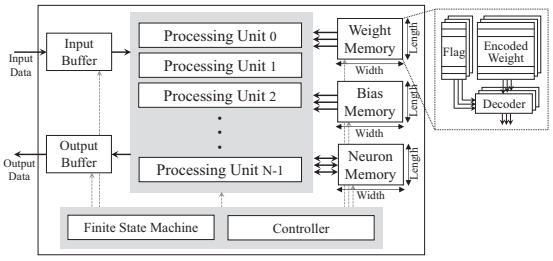


Fig. 3. Overall architecture of the neural accelerator

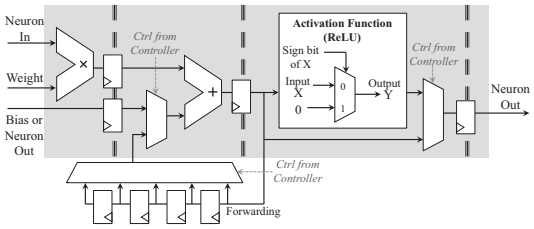


Fig. 4. Block diagram of processing unit (PU)

IV. Data Type Quantization

Figure 5 presents the synaptic weight distributions for different quantization levels of a neural network trained on the MNIST dataset. Because most weights in the neural network have near zero values (less than 0.25), as shown in Figure 5(a), NN compactor focuses on minimizing the quantization error for small weight values. Quantization to the 16-bit fixed-point data type (I: 6-bit, F: 10-bit) is the most common technique used in neural networks [5, 10] because it maintains similar precision to the 32-bit floating-point type, as shown in Figure 5(b). Although this quantization has low accuracy loss, it still requires amounts of large memory. Quantization to the 4-bit fixed-point data type (I: 2-bit, F: 2-bit) can be used as an alternative method for further memory requirement reduction. However, it shows high quantization noise because all small values less than 0.25 are grouped into the three values of $\{$ -0.25, 0, 0.25$\}$, as shown in Figure 5(c).

In order to mitigate this high quantization error by ensuring the precision of small values and maintaining the numeric

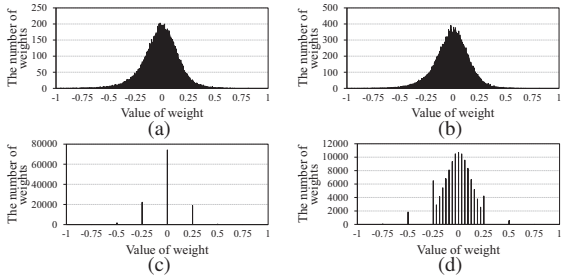


Fig. 5. Weight distribution of neural networks with MNIST for each quantization: (a) 32-bit floating-point, (b) 16-bit fixed-point, (c) 4-bit fixed-point, and (d) the proposed dual-track weight quantization (5-bit)

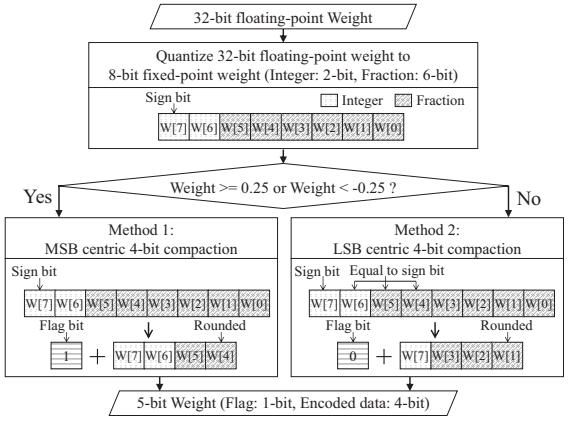


Fig. 6. Algorithm flow of the proposed dual-track weight quantization that reduces weight width into 5-bit

range to cover large values, NN compactor utilizes a novel value-adaptive dual-track quantization technique to encode 8-bit fixed-point values into 5-bit values. In this technique, the original 32-bit floating-point value is first quantized into an 8-bit fixed-point value (I: 2-bit, F: 6-bit), as illustrated in Figure 6. NN compactor then converts the 8-bit value into a 5-bit value using different bit encoding configurations (MSB/LSB centric) depending on the magnitude of the value. For large numbers, the four MSBs and an additional {flag bit: 1} are combined into the target 5-bit value (MSB centric compaction in Figure 6). For small numbers, the MSB and three bits of {W[3-1]} and a {flag bit: 0} are selected because the {W[6-4]} values are equal to the sign bit for small numbers (LSB centric compaction in Figure 6). Using this dual-track quantization technique, the quantization error for small numbers between -0.25 and 0.25 can be minimized while maintaining coverage for large numbers, as shown in Figure 5(d).

V. Resource Optimization

NN compactor utilizes a resource optimization algorithm to determine the optimal number of PUs. Algorithm 1 uses a simple approach to estimate system performance when reducing the number of PUs and find the minimum number of PUs required to achieve 90% of maximum performance. It first finds the maximum available number of PUs (NumPU, line 1) based on the target hardware specification (FPGA/ASIC) and the maximum potential performance (MinCycle, line 2). It then finds the target PU number (NumPU, line 9) by iteratively

Algorithm 1 Find the optimal number of processing units

Input: *NNmodel*, *TARGETspec*

Output: *NumPU*

```
1: NumPU = MaxNumPU(TARGETspec);
2: MinCycle = GetExecutionCycle(NNmodel, NumPU);
3: NormSpeed = 1;
4: while (NormSpeed > 0.9 ) do
5:     NumPU = NumPU / 2;
6:     Cycle = GetExecutionCycle(NNmodel, NumPU);
7:     NormSpeed = MinCycle / Cycle;
8: end while
9: NumPU = NumPU * 2;
```

estimating the performance with reduced numbers of PUs (line 4-9).

VI. Experimental Results

A. Methodology

To evaluate the performance of the neural networks, we first implemented the networks using a TensorFlow framework [4] and three datasets for training: MNIST [3], CNAE-9 [1], and Forest [2]. The NN compactor accelerator templates are written in the Verilog/SystemVerilog HDL languages and synthesized using Altera Quartus Prime 16.0 [8] for evaluation. To measure performance and accuracy, timing simulations using Quartus were performed and the accelerator design was successfully verified on an Altera Cyclone V FPGA device (5CSEMA5F31C6) running at 100 MHz on a DE1-SoC board [7]. We compare the performance of three platforms: embedded CPUs, FPGAs, and regular CPUs. We used a Quad-core ARM Cortex-A53@1.2GHz and Quad-core Intel i7-4790@3.6GHz for the target embedded CPU and regular CPU, respectively. To measure and compare their execution times, we used the average results from 100 runs.

B. Accuracy Comparison

To investigate the accuracy variation caused by quantization, we performed multiple experiments with different quantization levels. Table I lists the accuracy and weight memory requirements of each quantization level normalized to the baseline 32-bit floating-point value result. The baseline accuracy is determined by the TensorFlow framework with 32-bit floating-point weights. Accuracy variation is defined by the accuracy differences between the target quantization techniques and the baseline. To measure accuracy, we ran our NN compactor framework 10 times and implemented an accelerator 10 times for each quantization technique. We trained the neural networks on three datasets with three different topologies: 784-128-128-10 (MNIST), 856-128-128-9 (CNAE-9), and 54-128-512-128-7 (Forest).

For each dataset, the accelerator with 16-bit fixed-point quantization achieved the best accuracy but also required the highest memory capacity. In contrast, 4-bit fixed-point quantization required the lowest memory capacity and achieved the lowest accuracy. Based on these results, 16-and 4-bit quantization techniques cannot satisfy both high accuracy and low memory consumption requirements. However, the dual-track quantization based accelerator successfully achieved high accuracy with low memory capacity consumption as shown in Table I.

TABLE I. The accuracy and weight memory requirement of neural networks using different quantization with each dataset

Dataset	Baseline accuracy (%)	Quantization	Accuracy variation(%)			Weight memory requirement (Kbits)
			Mean	Max	Min	
MNIST	98.055	16-bit	0.00	0.02	-0.04	1,888(100%)
		Dual-track	-0.04	0.03	-0.13	590(31.3%)
		4-bit	-1.63	-0.73	-3.09	472(25.0%)
CNAE-9	96.778	16-bit	0.00	0.00	0.00	2,034(100%)
		Dual-track	-0.22	0.00	-0.56	636(31.3%)
		4-bit	-0.28	0.56	-1.11	508(25.0%)
Forest	69.696	16-bit	-2.14	0.89	-7.87	2,222(100%)
		Dual-track	-4.14	1.45	-10.63	694(31.3%)
		4-bit	-20.16	-8.54	-29.79	556(25.0%)

TABLE II. The topology of each neural network

Dataset	Networks scale	Topology
MNIST	MNIST(S)	784-64-10
	MNIST(L)	784-128-128-10
CNAE-9	CNAE-9(S)	856-64-9
	CNAE-9(L)	856-128-128-9
Forest	Forest(S)	54-128-128-7
	Forest(L)	54-128-512-128-7

C. Performance and Resource Utilization

To compare the performance of the neural networks, we consider two types of networks for each application: large networks and small networks. Large type networks provide the best accuracy and small type networks provide fair accuracy with minimum neurons. Based on these concepts, we implemented six neural networks for the three datasets as shown in Table II. Figure 7 presents the speedups achieved by FPGA-based accelerators generated by NN compactor and regular CPUs compared to the performance of the embedded CPU. The proposed accelerator achieves an average $6.4\times$ speedup over the embedded CPU. Although the performance of our accelerator is approximately half of the performance of the regular CPU, the energy efficiency of our accelerator will be much better than the regular CPU because the power consumption of the regular CPU is much higher.

Table III lists the FPGA resource utilization of the proposed accelerators for each neural network. In this table, all the neural accelerators consume less than 57% of the LUTs since 64 or 128 PUs are chosen as the optimal number of PUs to guarantee more than 90% of maximum performance. The accelerators also consume less than 24% of total block RAM by significantly reducing memory requirements by using the dual-track quantization technique.

VII. Related Works

There have been various studies on designing neural accelerators. Minerva [11] and DnnWeaver [12] are the most similar systems to ours in terms of framework design, but there are some key differences compared to our NN compactor. Minerva is a framework that utilizes an automated co-design approach across algorithms and accelerators for deep neural networks. DnnWeaver provides a framework that automatically generates neural accelerators for deep neural networks. However, neither of these systems utilize a novel weight quantization technique. Therefore, NN compactor is a more appropriate solution for small neural networks compared to Minerva and DnnWeaver.

VIII. Conclusion

We presented the NN compactor system, which automatically generates accelerators for small neural networks. To efficiently generate accelerators, NN compactor has four processes of training, data type quantization, resource optimization, and synthesis. The data type quantization and resource optimization processes perform memory and logic resource minimization to facilitate a compact neural accelerator using dual-track synaptic weight minimization and minimum PU number search techniques. Experimental results demonstrated that the neural accelerators with dual-track weight quantization generated by NN compactor guarantee similar accuracy to the 16-bit fixed-point accelerators with significantly less resource consumption. In addition, the accelerators generated by NN compactor achieve an average $6.4\times$ speedup over embedded CPUs.

TABLE III. The FPGA utilization for each neural network

Networks	LUTs	Registers	BRAM(bits)	DSP
MNIST(S)	9,294(29%)	13,067(10%)	409,600(10%)	64(74%)
MNIST(L)	18,356(57%)	26,180(20%)	900,864(22%)	87(100%)
CNAE-9(S)	9,290(29%)	13,062(10%)	409,600(10%)	64(74%)
CNAE-9(L)	18,338(57%)	26,262(20%)	900,864(22%)	87(100%)
Forest(S)	18,326(57%)	26,124(20%)	409,344(10%)	87(100%)
Forest(L)	18,331(57%)	26,240(20%)	982,784(24%)	87(100%)

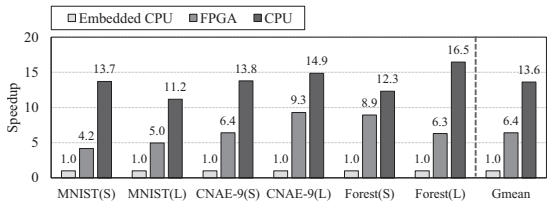


Fig. 7. The speedup comparison of embedded CPUs, FPGA-based accelerators, and CPUs

IX. Acknowledgments

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP)(No.NRF-2015R1C1A1A01053844), ICT R&D program of MSIP/IITP (No.2017-0-00142), and the R&D program of MOTIE/KEIT (No.10077609).

References

[1] Cnae-9 dataset. https://archive.ics.uci.edu/ml/datasets/CNAE-9.
[2] Forest covertype dataset. https://archive.ics.uci.edu/ml/datasets/covertype.
[3] The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/.
[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015.
[5] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 269–284, New York, NY, USA, 2014. ACM.
[6] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.
[7] Intel-FPGA. Altera de1-soc board. https://www.altera.com/solutions/partners/partner-profile/terasic-inc-/board/cyclone-v-se-device-family—de1-soc-board.html.
[8] Intel-FPGA. Altera quartus prime. https://www.altera.com/downloads/download-center.html.
[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
[10] J. Kung, D. Kim, and S. Mukhopadhyay. A power-aware digital feedforward neural network platform with backpropagation driven approximate synapses. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 85–90, July 2015.
[11] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernandez-Lobato, G. Y. Wei, and D. Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 267–278, June 2016.
[12] 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 *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, Oct 2016.
[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.