Resource Constrained Cellular Neural Networks for Real-time Obstacle Detection using FPGAs

Xiaowei Xu[†*], Tianchen Wang[*], Qing Lu[*], and Yiyu Shi[*]
[*]University of Notre Dame, South Bend, IN, USA
[†]Huazhong University of Science and Technology, Wuhan, China
yshi4@nd.edu

***Abstract*—Due to the fast growing industry of smart cars and autonomous driving, advanced driver assistance systems (ADAS) with its applications have attracted a lot of attention. As a crucial part of ADAS, obstacle detection has been challenge due to the real-tme and resource-constraint requirements. Cellular neural network (CeNN) has been popular for obstacle detection, however suffers from high computation complexity. In this paper we propose a compressed CeNN framework for real-time ADAS obstacle detection in embedded FPGAs. Particularly, parameter quantizaion is adopted. Parameter quantization quantizes the numbers in CeNN templates to powers of two, so that complex and expensive multiplications can be converted to simple and cheap shift operations, which only require a minimum number of registers and LEs. Experimental results on FPGAs show that our approach can significantly improve the resource utilization, and as a direct consequence a speedup up to 7.8x can be achieved with no performance loss compared with the state-of-the-art implementations.**

I. Introduction

With the fast-growing smart vehicle industry, advanced driver assistance system (ADAS) has attracted a lot of attention in recent years, in which obstacle detection is one of the most significant modules. However, most commercially available solutions for obstacle detection, e.g., LIDAR [1] and cellular neural networks (CeNN), are expensive and computationally intensive [5]. While embedded systems on vehicles are typically constrained by computation capacity and energy. Considering the safety and security requirement, the real-time processing of obstacle detection on vehicles is preferred.

A very powerful tool for obstacle detection is cellular neural network (CeNN), which can achieve very high accuracy through proper training. It should be noted that CeNNs are popular in image processing areas such as classification [3], segmentation [4], while convolutional Neural Networks (CNNs) are most powerful in classification related tasks. However, due to the complex nature of segmentation and other image process tasks and the associated real-time requirements in many applications, hardware implementations of CeNNs have remained an active research topic in the literature.

As analog implementation of CeNNs are typically bounded by limited data precision and small supported image size [14], digital implementations of CeNNs have been proposed [17], where data is quantized with approximation. Tens to hundreds of iterations are needed in the discretized process and as a result, the computational complexity of digital CeNNs is very high. For example, to process an image of 1920x1080 pixels requires 4-8 Giga operations (for 3×3 templates and 50-100 iterations), which needs to be done in a timely manner for real-time image processing.

To tackle the computation challenge, CeNN accelerations on digital platforms such as ASICs [8][9], GPUs [13] and FPGAs [2][12] [10][18][19] [11] have been explored, with FPGA among the most popular choices due to its high flexibility and low time-to-market. The work [2] presented a baseline design with several applications, while the study [12] took advantage of reconfigurable computing for CeNNs. Recently, the CeNN implementation for binary images was demonstrated [11]. Expandable and pipelined implementations were achieved on multiple FPGAs [10]. Taking advantage of the structure in [10], the work [18] implemented a high throughput real-time video streaming system, which is further improved to be a complete system for video processing [19]. All the three works share the same architecture for CeNN computation. Due to the large number of multiplications needed in CeNNs, the limited number of embedded multipliers in an FPGA become the bottleneck for further improvement. For example, in work [10] 95%-100% of the embedded multipliers are used. On the other hand, it is interesting to note that the utilization rates of LEs and registers are only 5% and 2%, respectively, which is natural to expect as not many logic operations are needed. However, in a mainstream FPGA, LEs and registers count for significantly larger portion of the total programmable resources than embedded multipliers. For example, LEs and registers occupy 95.4% of the core area while embedded multipliers only 4.6% for a EP3LS340 FPGA [16]. Such an unbalanced resource utilization apparently cannot attain the best possible speed of the CeNN being implemented, and an improved strategy is strongly desired.

In this paper we present a compressed CeNN framework for real-time ADAS obstacle detection. Particularly, we systematically put forward powers-of-two based incremental quantization of CeNNs for efficient hardware implementation. The incremental quantization contains iterative procedures including parameter partition, parameter quantization, and re-training. We propose five different strategies including random strategy, pruning inspired strategy, weighted pruning inspired strategy, nearest neighbor strategy, and weighted nearest neighbor strategy. Experimental results show that our approach can achieve a speedup up to 7.8x with no performance loss compared with the state-of-the-art FPGA solutions for CeNNs.

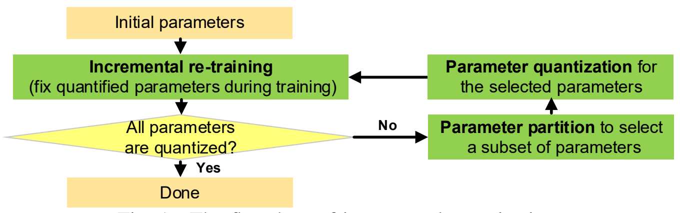
Initial parameters
Incremental re-training
(fix quantified parameters during training)
Parameter quantization for the selected parameters
All parameters are quantized?
No
Parameter partition to select a subset of parameters
Yes
Done

Fig. 1. The flowchart of incremental quantization.

II. Compressed CeNN Framework and Hardware Implementation

A. Incremental Quantization

The proposed incremental quantization framework is an iterative process as shown in Figure 1. Each iteration completes three tasks: parameter partition, parameter quantization, and incremental re-training. We assume that as a starting point, we have all parameters in the original templates before quantization well trained.

1) Parameter partition: This task selects a subset of parameters not yet quantized (un-quantized parameters) to perform quantization. Two knobs exist in this task: parameter priority and batch size. For the first knob, the pruning-inspired (PI) strategy has been well explored in quantization of CNNs [20], based on the consideration that weights with larger magnitudes contribute more to the result and thus should be quantized first. However, the parameters in CeNNs have some unique characteristics that some parameters are coupled. In order to tackle the problem, we propose a nearest neighbor (NN) strategy and a weighting method for the first knob. The combined weighted nearest neighbor algorithm takes the number that a parameter appears in the template, defined as its repetition quantity (rq) as the reciprocal of the weight, and uses the difference between the parameter and its nearest power-of-two as distance to perform a weighted NN algorithm (WNN). Other combinations such as weighted pruning-inspired (WPI) strategy adopt the same weighting method but with PI to form WPI. A total of five strategies PI, WPI, NN (WNN with all weights set to 1), WNN and a random strategy (RAN) are compared in the experimental section.

For the second knob, batch size is the number of parameters selected in each iteration, which will affect re-training speed and quality. We propose to use two batch sizes, constant and log-scale. The former selects the same number of parameters in each iteration, while the latter picks a fixed percentage from the remaining un-quantized parameters, rounded to the nearest integer. Compared with constant batch size, log-scale batch size quantizes more parameters in the first several iterations and fewer towards the end.

2) Parameter quantization: Suppose the quantization set is designed as depicted in Equation 1, where k and m indicate the range of quantization. The corresponding bit width bw is calculated as shown in Equation 2, where the extra one bit is the sign bit.

$$qs = \{\pm(2^k, ., 2^p, ., 2^m), 0\}, \quad k \le p \le m, \quad p, k, m \in \mathbb{Z}. \quad (1)$$

$$bw = Ceiling[log_2(2 \times (m-k+1)+1)]+1. \quad (2)$$

With the quantization set, a parameter $uq(i)$ is quantized as shown in Equation 3. When the absolute value of a parameter is smaller than 2^{-k-1}, it will become zero after quantization and get pruned. Lower bit width can prune more parameters, at the cost of accuracy loss.

$$uq(i) = \begin{cases} 2^p & \text{if } 3 \times 2^{p-2} \le |uq(i)| < 3 \times 2^{p-1}; \\ & \quad k \le p \le m; \\ 2^m & \text{if } |uq(i)| \ge 2^m; \\ 0 & \text{if } |uq(i)| < 2^{-k-1}. \end{cases} \quad (3)$$

3) Incremental Re-training Algorithm: Usually, re-training algorithm is an optimal problem as shown in Equation 4, where P is the set of all the parameters. In incremental re-training algorithm, the optimal problem is revised as shown in Equation 5, where U and Q are the sets of un-quantized and quantized parameters, respectively. a_i and b_i are the lower and upper bounds for both P_i and U_i, respectively. Note that $P = Q \cup U$, and $U \cap Q = \emptyset$. In each iteration, a subset of U will be quantized and added to Q.

$$f = min\ obj(P), \ s.t.\ P_i \in [a_i, b_i], 0 \le i \le |P|. \quad (4)$$

$$f = min\ obj(U,Q), s.t. U_i \in [a_i, b_i], 0 \le i \le |U|. \quad (5)$$

Q will be fixed during the re-training process and only U is used for space searching. After multiple iterations, all the required parameters are quantized. It should be noted that the bias in CeNN computation is not required to be quantized as it is not involved in multiplication. Therefore, another re-training iteration is required for the optimal bias when all the required parameters are quantized.

B. Efficient Hardware Implementations

We base our work on the state-of-the-art FPGA CeNN implementations [10][18][19], which is expandable, highly parallel and pipelined. The basic element of the architecture is the stage module which handles all the processes in one iteration of CeNN computation for $1 \le i \le M$, $1 \le j \le N$. Multiple stages are connected sequentially for multiple iterations to form a layer, which processes the input in a pipelined manner. Furthermore, multiple layers can be connected sequentially for more complex processing or be distributed in parallel for a higher throughput. Note that First In First Out (FIFO) are used between adjacent stages to store the temporary results of each stage (or each iteration), and they are configured as single-input multiple-output memories. Please refer to FPGA implementations in [10][18] for more details.

Our efficient hardware implementation focuses on the optimization of the stage design as shown in Figure 2. Two optimizations are performed: multiplication simplification and data movement optimization. First, with incremental quantization, simplification can be achieved by replacing multiplications with shift operations. The detailed hardware implementation will be discussed in the following section. Second, when FPGA resource is extremely limited (e.g. for low-end FPGAs), data movement optimization can be performed utilizing

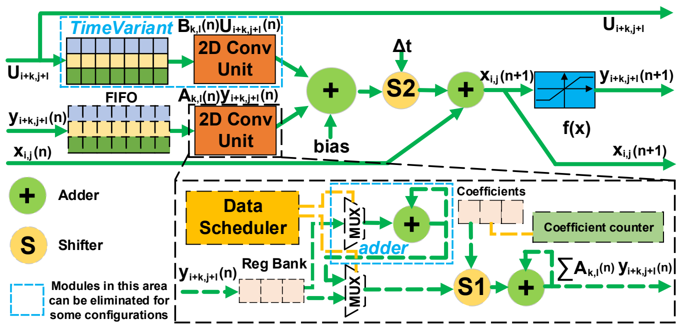

Fig. 2. Architecture of the optimized stage design.

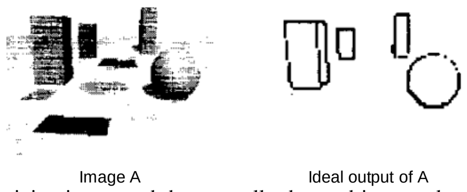

Fig. 3. Training image and the manually detected image taken from [5].

the sparsity and repetition in CeNN templates. In fact, in many applications CeNN templates naturally involves zero or repeated parameters. With incremental quantization, more zeros are yielded leading to higher sparsity and the small quantization set introduces a larger number of repetitions. Data movement optimization can minimize the number of computations needed.

The optimized stage can be configured for both time-invariant templates and time-variant templates. Note that the FPGA implementation [18] is dedicated to CeNN with time-invariant templates, while [10] is for time-variant. The $TimeVariant$ part in Figure 2 is specific for time-variant templates, and can be eliminated in the configuration for time-invariant ones.

1) Shifter Module: In Figure 2, shifter $S1$ is for multiplications in CeNNs and $S2$ is for discrete approximation involved with Δt in CeNN computation. Usually Δt is very small, and the hardware implementation of $S2$ in this paper is designed to support Δt=2^s, where $-7 \leq s \leq 0$, $s \in \mathbb{Z}$. Note that when Δt is configured to 2^0 or 1, the computation is transformed to discrete CeNN [6].

2) Data Scheduler Module: Data scheduler module exploits the sparsity and repetition of parameters in CeNN templates. Ignoring multiplications with zeros will give a significant improvement in efficiency as a majority of templates have zero values.

III. Experiments

In this section, we first evaluate the performance of various incremental quantization strategies discussed in Section II for obstacle detection. Then we implement the quantized CeNNs on FPGAs and compare their speed with state-of-the-art works.

A. Performance Evaluation

1) Experimental Setup: For incremental quantization, a total of 10 incremental quantization strategies are evaluated: five partition strategies (RAN, PI, WPI, NN (WNN with all weights set to 1), and WNN) in combination with two batch

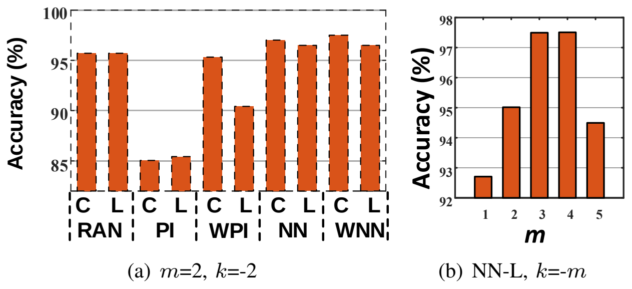

Fig. 4. Performance comparison between templates with various (a) strategies and (b) quantization sizes m for obstacle detection.

sizes (constant and log-scale). For compact presentation, we use postfix -C and -L to denote constant and log-scale batch sizes, respectively. For constant batch size, we set the size to 20% of the total parameters. While for log-scale batch size, we set it to half of the remaining un-quantized parameters. We discuss five quantization set sizes with m =0, 1, 2, 3, 4 and k = $-m$ (see Equation 1).

We also adopts the objective function as accuracy to evaluate the quality of output images. The pattern structures of the 3×3 templates A and B are as follows: $A = \{a_0, a_0, a_0; a_0, a_1, a_0;\ a_0, a_0, a_0\}$, and $B = \{a_2, a_2, a_2; a_2, a_3, a_2; a_2, a_2, a_2\}$. The training dataset is from [5] as shown in Figure 3, which is the of the same configuration with the work [15]. For test dataset, totally 40 test images are selected from Hlevkin test images collection [7].

$$obj = accuracy = \sum_{i=1}^{t} abs(output_i - IdealOutput_i)/area. \tag{6}$$

2) Results and Discussion: We fix the quantization size using m = 2 and k = $-m$, and evaluate all 10 incremental quantization frameworks. The results are depicted in Figure 4(a). From the figure we can observe that the quantized templates achieve similar accuracy compared with the original template without quantization. The lowest accuracy is about 15% lower than that with the original templates. Interestingly, the highest accuracy is achieved with WNN-C strategy, which is only 3% lower than that of the original templates. Note that generally PI strategy achieves the best performance for CNNs [20]. However, WNN strategy obtains the best performance for CeNN, and NN strategy also obtains a comparable performance. Furthermore, NN and WNN strategy are much stable than PI as NN and WNN can achieve almost the same accuracy for constant and logscale batch sizes while PI not. Even random strategy can have a better accuracy than PI in some configurations. In terms of batch size, constant seems to perform better than log-scale in most cases. The impact of batch sizes is presented in Figure 4(b) with the optimal partition WNN-C. The quantization set size has an interesting relationship with the performance. First, even when the quantization set is only of three values (-1, 0, 1), the quantized template can still achieve high accuracy. Second, there exists an optimal m which gives the best performance and m=3 for obstacle detection. Further increasing m will not provide any performance gain.

TABLE I
SPEED AND RESOURCE UTILIZATION COMPARISONS OF THE STATE-OF-THE-ART WORK [19] AND OURS WITH NINE MULTIPLIERS/SHIFTERS IN 2D CONVOLUTION MODULE. THE NUMBERS IN THE BRACKETS ARE THE RESOURCE UTILIZATION RATE.

IMPLEMENTATION	VC7VX-980T	VC7VX-585T	STRATIX V E	STRATIX V GS
# OF STAGES	352	179	233	291
LEs($\times 10^3$)	780(80%)	465(80%)	718(80%)	524(80%)
REGISTERS($\times 10^3$)	170(17%)	93(16%)	133(15%)	128(19%)
EMBEDDED MULT.	3600(100%)	1260(100%)	704(100%)	3926(100%)
SPEEDUP	**2.3x**	**3.3x**	**7.8x**	**1.7x**

B. Speed Evaluation Using FPGAs

In previous section we have evaluated the performance of our compressed CeNN framework in terms of accuracy. In this section we will evaluate its speed when implemented in FPGAs. For a fair comparison with existing works [10][18][19], we adopt the same configurations of stages and try to place the maximum possible number of stages utilizing our quantized templates. Note that all the three works share the same architecture for CeNN computation. The performance of the implementation is evaluated by equivalent computing capacity which is the product of number of stages and the computing capacity of each stage. The proposed efficient hardware implementation is implemented on an XC4LX25 FPGA. The data width of the input, state, and output (u, x, and y) is configured to be 18 bits. The widely-used template size 3×3 is adopted. Note that general CeNN is adopted for the FPGA implementation, and delayed CeNN is not considered here. Time-variant templates are configured. In the implementation, multiplication is achieved with embedded multipliers (more specifically, DSP48 modules on XC4LX25 FPGAs) at first, and shifters are used when there are no more available embedded multipliers. Considering the routability of FPGAs, the utilization rate of LEs and registers are constrained to be no higher than 80%. Note that since different quantization frameworks only affects the performance and do not show significant difference in hardware resource utilization, in this part of experiments we simply use WNN-L with m=5 and k=-5, and other frameworks should yield almost identical speed.

We select four high-end FPGAs from Altera and Xilinx with about 500,000 to 1,000,000 LEs, and there are nine multipliers in each 2D convolution module. As shown in Table I, our implementations can achieve a speedup of 1.7x-7.8x. Note that the resource consumption of LEs and registers are almost the same for all the implementations, and the speedup varies with the number of embedded multipliers, or more specifically, the ratio of LEs to embedded multipliers. A high ratio of LEs to embedded multipliers means more LEs can be used to implement shifters resulting with a high speedup. The highest speedup of 7.8x is due to the fact that the Stratix V E FPGA has the highest rate of LEs to embedded multipliers.

IV. CONCLUSIONS

In this paper, we propose a compressed CeNN framework for computation reduction in CeNNs for obstacle detection. Particularly, we present powers-of-two based incremental quantization. The incremental quantization adopts an iterative procedure including parameter partition, parameter quantization, and re-training to produce templates with values being powers of two. We propose a few quantization strategies based on the unique CeNN computation patterns. Thus, multiplications are transformed to shift operations, which are much more resource-efficient than general embedded multipliers.

Experimental results show that the proposed framework can achieve similar performance compared with that using original templates without optimization, and the implementation with incremental quantization can achieve a speedup up to 7.8x compared with the state-of-the-art FPGA implementations.

REFERENCES

[1] A. N. Catapang and M. Ramos. Obstacle detection using a 2d lidar system for an autonomous vehicle. In *Control System, Computing and Engineering (ICCSCE), 2016 6th IEEE International Conference on*, pages 441–445. IEEE, 2016.

[2] H.-C. Chen, Y.-C. Hung, C.-K. Chen, T.-L. Liao, and C.-K. Chen. Image-processing algorithms realized by discrete-time cellular neural networks and their circuit implementations. *Chaos, Solitons & Fractals*, 29(5):1100–1108, 2006.

[3] F. Dohler, F. Mormann, B. Weber, C. E. Elger, and K. Lehnertz. A cellular neural network based method for classification of magnetic resonance images: towards an automated detection of hippocampal sclerosis. *Journal of neuroscience methods*, 170(2):324–331, 2008.

[4] M. Duraisamy and F. M. M. Jane. Cellular neural network based medical image segmentation using artificial bee colony algorithm. In *Green Computing Communication and Electrical Engineering (ICGCCEE), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[5] D. Feiden and R. Tetzlaff. Obstacle detection in planar worlds using cellular neural networks. In *Cellular Neural Networks and Their Applications, 2002.(CNNA 2002). Proceedings of the 2002 7th IEEE International Workshop on*, pages 383–390. IEEE, 2002.

[6] H. Harrer and J. A. Nossek. Discrete-time cellular neural networks. *International Journal of Circuit Theory and Applications*, 20(5):453–467, 1992.

[7] Hlevkin. http://www.hlevkin.com/06testimages.htm, 2017.

[8] S. Lee, M. Kim, K. Kim, J.-Y. Kim, and H.-J. Yoo. 24-gops 4.5-mm^2 digital cellular neural network for rapid visual attention in an object-recognition soc. *IEEE transactions on neural networks*, 22(1):64–73, 2011.

[9] D. Manatunga, H. Kim, and S. Mukhopadhyay. Sp-cnn: A scalable and programmable cnn-based accelerator. *IEEE Micro*, 35(5):42–50, 2015.

[10] J. J. Martnez, J. Garrigs, J. Toledo, and J. M. Ferrndez. An efficient and expandable hardware implementation of multilayer cellular neural networks. *Neurocomputing*, 114:54–62, 2013.

[11] J. Muller, R. Wittig, J. Muller, and R. Tetzlaff. An improved cellular nonlinear network architecture for binary and greyscale image processing. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2016.

[12] R. Porter, J. Frigo, A. Conti, N. Harvey, G. Kenyon, and M. Gokhale. A reconfigurable computing framework for multi-scale cellular image processing. *Microprocessors and Microsystems*, 31(8):546–563, 2007.

[13] S. Potluri, A. Fasih, L. K. Vutukuru, F. Al Machot, and K. Kyamakya. Cnn based high performance computing for real time image processing on gpu. In *INDS & 16th ISTET, 2011 Joint 3rd Int'l Workshop on*, pages 1–7. IEEE, 2011.

[14] A. Rodrguez-Vzquez, G. Lin-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galn, F. Jimnez-Garrido, R. Domnguez-Castro, and S. E. Meana. Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(5):851–863, 2004.

[15] R. Rouhi, M. Jafari, S. Kasaei, and P. Keshavarzian. Benign and malignant breast tumors classification based on region growing and cnn segmentation. *Expert Systems with Applications*, 42(3):990–1002, 2015.

[16] H. Wong, V. Betz, and J. Rose. Comparing fpga vs. custom cmos and the impact on processor microarchitecture. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 5–14. ACM, 2011.

[17] X. Xiaowei, L. Qing, W. Tianchen, L. Jinglan, Z. Cheng, H. Sharon, and S. Yiyu. Empowering mobile telemedicine with compressed cellular neural networks. In *Proc. of IEEE/ACM ICCAD*. IEEE/ACM, 2017.

[18] N. Yildiz, E. Cesur, K. Kayaer, V. Tavsanoglu, and M. Alpay. Architecture of a fully pipelined real-time cellular neural network emulator. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(1):130–138, 2015.

[19] N. Yildiz, E. Cesur, and V. Tavsanoglu. On the way to a third generation real-time cellular neural network processor. *CNNA 2016*, 2016.

[20] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *5th International Conference on Learning Representations*, 2017.