Analysis and Solution of CNN Accuracy Reduction over Channel Loop Tiling

Yesung Kang[1], Yoonho Park[2], Sunghoon Kim[2], Eunji Kwon[2],
Taeho Lim[3], Sangyun Oh[4], Mingyu Woo[5] and Seokhyeong Kang[2]
[1]Future IT Innovation Laboratory and [2]EE Department, POSTECH, South Korea
[3]SK Hynix Inc., South Korea
[4]CSE Department, UNIST, South Korea
[5]ECE Department, UC San Diego, La Jolla, CA, USA
{yeskang, shkang}@postech.ac.kr

***Abstract*—Owing to the growth of the size of convolutional neural networks (CNNs), quantization and loop tiling (also called loop breaking) are mandatory to implement CNN on an embedded system. However, channel loop tiling of quantized CNNs induces unexpected errors. We explain why channel loop tiling of quantized CNNs induces the unexpected errors, and how the errors affect the accuracy of state-of-the-art CNNs. We also propose a method to recover accuracy under channel tiling by compressing and decompressing the most-significant bits of partial sums. Using the proposed method, we can recover accuracy by 12.3% with only 1% circuit area overhead and an additional 2% of power consumption.**

I. Introduction

Convolution neural networks (CNNs) are widely used in many AI applications, especially in image recognition, detection and segmentation [1]–[3]. The accuracy of CNNs has been improved rapidly, but this improvement has entailed increases in network size, number of computations, and memory usage [4]. Despite many attempts to reduce network size, state-of-the-art CNNs require tens to hundreds of megabytes of memory. One way to alleviate this problem is to use a large on-chip memory. The on-chip memory size of embedded accelerators is increasing, but there is still a huge gap between the required and the available on-chip memory size. Furthermore, large on-chip memory size is not affordable for embedded applications. Also, many hardware accelerators use multiple buffering to reduce latency between memory and computing cores, and this process decreases available on-chip memory in a cycle.

To reduce the required memory size for the CNN implementation, CNNs are quantized [5]–[10]. In general, the precision of CNNs is reduced from 32-bit single-float to 8-bit or 16-bit fixed-point precision. After quantization, if the required on-chip memory size is larger than the available on-chip memory size, CNN loops are tiled into small blocks which can be accommodated in on-chip memory. Loop tiling allows relatively large CNNs to be implemented on relatively small embedded accelerators, but it generates unexpected accuracy degradation.

The computation procedure by which a hardware accelerator generates an output feature map changes depending on whether the channel loop is tiled or not. If the channel loop is not tiled, the accelerator loads input feature maps and filters from external memory for convolution. To prevent overflow of the accumulation, the bit width of the accumulators is set to higher than the bit width of feature maps (e.g., 8-bit for the feature maps and 20-bit for the accumulations.). After the convolution, which consists of several multiplications and accumulations, the results are extracted to the same bit width as the bit width of the output feature map. The extraction of output feature maps leads to a *quantization error*, which is analyzed and controlled during the quantization procedure of CNNs. If the channel loop is tiled (Figure 1), tiles of input feature maps and filters are loaded, and then computed partial sums are stored in external memory. They are reloaded several times until the complete output feature maps are generated. During the process of reading and writing the partial sum, it is rounded and this process generates an additional error, named *channel loop tiling-error*. Because *channel loop tiling-error* deteriorates CNN accuracy, it should be understood and controlled. To the best of our knowledge, this is the first work that analyzes the effects of *channel loop tiling-error* on the accuracy of CNNs.

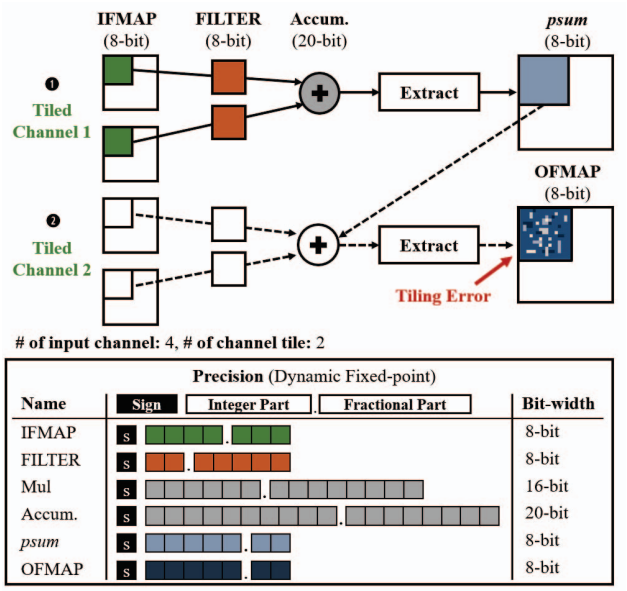

Fig. 1: The CNN computation procedure when channel loop is tiled by two. The precision of the bit width of operations supported by the embedded system. The fractional length varies with the dynamic range of each layer, which is determined by the offline experiment.

The main contributions of our work are:

- We explain the mechanism by which channel loop tiling causes errors, and analyze the effect of the error on the accuracy of CNNs.

978-3-9819263-4-7/DATE20/©2020 EDAA

- We propose a method which reduces *channel loop tiling-error* by using extended partial sums while minimizing circuit and memory overhead by exploiting the bit-level run-length encoding (RLE).
- We improve the accuracy under channel tiling, and quantify the circuit overhead.

The rest of paper is organized as follows: Section II introduces previous works on the loop tiling and the quantization of CNN. Section III addresses the necessity of loop tiling. Section IV presents how channel loop tiling generates error, then shows the effect of *channel loop tiling-error* on CNN accuracy. We also propose the method to recover *channel loop tiling-error* in Section IV. Section V presents the method to minimize circuit and area overhead caused by the accuracy-recovering method. Section VI concludes the paper.

II. Related Works

The quantization of CNN can increase the energy efficiency of the CNN accelerator and reduce the size of CNN. The quantization of CNN is accompanied by reduction in CNN accuracy, so there have been several studies on the change in CNN accuracy due to quantization and the minimization of the accuracy reduction [5]–[10]. Some researchers have analyzed the accuracy of CNN of which filter is quantized with the ternary [5] and the binary precision [6]. Zhou et al. have minimized the accuracy loss by incrementally quantizing the filter with 4-bit, 3-bit and 2-bit precision. Lin et al. have proposed the quantization method for both the filter and the feature maps [8]–[10]. However, they have only focused the quantization without consideration of the change of the computation procedures due to loop tiling.

The data movement of a CNN accelerator is highly related to the energy efficiency and the performance. Data movement is determined by the loop tiling and the loop ordering. Chen et al. have proposed the row-stationary dataflow and implemented it on their hardware platform [11]. The row-stationary data flow has increased energy efficiency by 2.5 times. Many researchers have proposed the loop tiling method to maximize the performance and energy efficiency of CNN computation over FPGA [12]–[14]; the authors have developed a roofline model that has considered CNN memory bandwidth and computational resources, and optimized loop tile size and loop order. By exploiting the roofline model, they have optimized the performance and energy efficiency of CNN. However, they have focused only on the optimization of performance and energy efficiency and have not analyzed the additional accuracy loss caused by the loop tiling.

There have been several researches on the quantization and the loop tiling separately. Studies on quantization have mainly focused on the relationship between bit width and accuracy, while studies on loop tiling have focused on performance and energy efficiency. However, they have not analyzed the interference between loop tiling and CNN quantization; the interference causes additional errors that occur during the accumulation of the partial sums. In contrast to these studies, we analyze the reason for this additional accuracy loss, and propose a solution.

III. Loop Tiling on CNN

A. Implementation procedures of CNN on embedded devices

In this subsection, we introduce the procedures to implement CNNs on a hardware accelerator. Generally, trained CNNs are too large to be mapped directly onto an embedded system, so the CNNs are quantized and tiled.

The quantization of CNN entails two procedures: (1) determining the bit width of feature maps and filters; and (2) choosing the precision. Because each hardware accelerator has a different bit width for CNN convolution, the bit width should be determined by considering the target accelerator. For example, if the accelerator supports 8-bit multiplication, the bit width of input feature maps and filters, BW_I and BW_F, should be quantized to 8-bit. Because the output feature maps are the input feature maps of the following layer, the bit width of output feature maps, BW_O, is usually equal to BW_I.

To quantize CNN using dynamic fixed point [10], we should choose the fractional length of each layer. In dynamic fixed point, feature maps and filters are represented as follows:

$$(-1)^s \cdot 2^{-FL} \sum_{i=0}^{BW-2} 2^i \cdot x_i \tag{1}$$

, where *s* denotes sign, *FL* the fractional length, *x* the mantissa bits. To avoid overflow or saturation, the fractional lengths should be precisely configured. The fractional length is determined by the following equation:

$$FL = BW - \lceil log_2(max\ x + 1) \rceil . \tag{2}$$

Using the equation, we choose the fractional length of the filter of each layer. Because the magnitude of feature maps changes dynamically, the distribution of the feature maps is analyzed during the inference of the evaluation set. Then, we determine the fractional length using the maximum value of feature maps within the range of 3σ.

If the required memory size of a CNN layer is still larger than on-chip memory size after quantization, we must appropriately split each layer of CNNs by considering both the size of the quantized CNNs and the memory capacity of the accelerators. The convolution layer consists of six for-loops. Each layer takes C input feature maps of width W and height H. Input feature maps are convolved with M filters which have a size of $C \times K \times K$, where K is the height and the width of a filter. As a result of convolution, M output feature maps are generated. Each design parameter of six loops can be tiled independently, so the design space for loop tiling has six dimensions. Considering that the filter size of state-of-the-art CNNs is small (usually three or five), design space can be reduced to four dimensions, M, C, W and H loop. The four loops are appropriately tiled by considering the on-chip memory size of the target CNN accelerators.

B. Necessity of channel loop tiling

Before discussing the effect of the *channel loop tiling-error*, we address whether the *channel loop tiling-error* is inevitable or not. Our basic assumption is that the size of a CNN layer is bigger than on-chip memory size so that loop tiling is mandatory (considering mobile or embedded applications, the assumption seems to be reasonable). The CNN loops are tiled by considering both the computational speed of the CNN

accelerator and the bandwidth between off-chip and on-chip memory. In real implementation of large CNN layers, *channel loop tiling-error* is avoidable in the following two cases.

1) *Exclusion of channel loop tiling*. In this case, other CNN loops (M, W, H) are tiled except for channel loops. When the number of input channels is small as in the first few CNN layers, the computational speed of CNN hardware would not be degraded. However, in deeper layers, the number of channels grows to a few hundred, so not tiling the channel loops is very inefficient and greatly affects performance. Also, the loop tiling of H and W requires the load of T_H rows and T_W columns of the input feature map [15]. Because the feature map is usually stored in row-major or column-major format, the tiling of H and W causes inefficient memory accesses. Tiling over rows and columns of the input feature map requires accesses to discontinuous memory addresses. The discontinuous memory accesses make it hard to use the burst mode of the memory interface so that the bandwidth of memory decreases. Hence, tiling over W and H of the input feature maps should be minimized. Therefore, it is inefficient to map CNN onto embedded system by exploiting the output channel tiling and the very limited W and the H tiling.

2) *Storage of the $psum$ without extraction*. Another approach to eliminate *channel loop tiling-error* is to store full-bit $psum$ without extraction. This approach requires a larger size of on-chip memory for the $psum$ tile so that a relatively small size of on-chip memory is assigned to save the input feature map and the filter. If the available memory size for the storage of the input feature map and the filter decreases, the reuse of the input feature map and the filter also decreases. The decrease of the data reuse increases the data transfer between on-chip and off-chip memory. Also, to store a large bit of $psum$ requires a memory interface that is independent of the one used to handle the output feature map.

It is very inefficient to implement CNN on the accelerator without channel loop tiling to prevent degradation of accuracy. Also, storing a large bit width without extracting the $psum$ is very expensive. Therefore, it is inevitable to use the channel loop tiling for mapping CNN onto embedded system with a small on-chip memory.

IV. Channel Loop Tiling-Errors

A. Cause of channel loop tiling-error

During the aforementioned procedures for the implementation of CNNs, the error can occur because of the interaction between the quantization and channel tiling. In this paper, we focus on the CNN accuracy reduction from the *channel loop tiling-error*. In Figure 2, we suppose that the channel loop is split into three tiles. max_{accum} and max_{quant} are the largest numbers that can be represented with bit width of BW_{accum} and BW_{quant}. After the bias is loaded, the PE starts convolution, and a partial sum is accumulated. When the convolution of the first tile is completed, the PE saves the partial sum in the output buffer after rounding. In the second tile and third tile, the stored partial sum is loaded and then the remaining process is similar to the first tile. During the extraction and reload of a partial sum, $psum$, two types of errors can occur: an **(1) exceeding error** and a **(2) rounding error**.

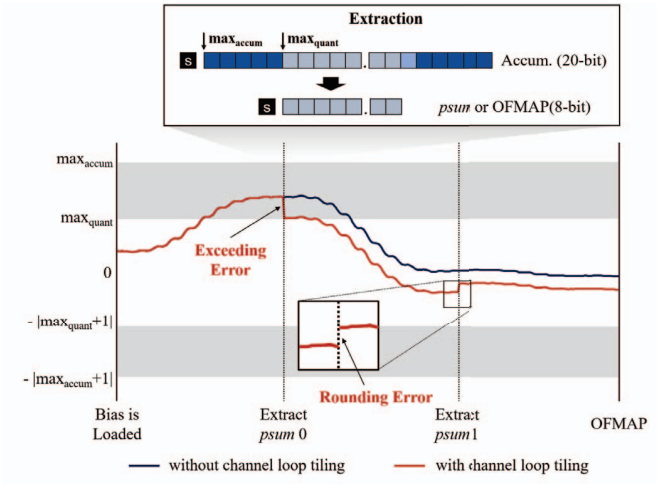

Fig. 2: Two types of *channel loop tiling-errors*. The values of partial sums (y-axis) are accumulated during three channel loops. If a value of partial sum stored to memory is larger than max_{quant}, exceeding error occurs. The round of the partial sum generates a rounding error.

The exceeding error occurs when $psum$ is larger than max_{quant}. At the end of the first tile, $psum$ is rounded to max_{quant}, the process yields an exceeding error. Exceeding errors cannot be predicted during the quantization procedure because the final output feature map can be in the range $[-(max_{quant}+1), max_{quant}]$. Although exceeding errors are rare, they are significant when they occur, so they can degrade the accuracy of CNNs.

A rounding error is smaller than half of the minimum precision of BW_O. Each rounding error is relatively small, but they occur in almost all outputs and channel tiles. In addition, the accumulated rounding error is much larger than the quantization error. Table I shows the frequency and quantity of exceeding errors and rounding errors which occur during the calculation of a certain layer of *AlexNet*. For the experiment, we have tiled all channels of the layer separately.

TABLE I: Frequency and quantity of the exceeding error and the rounding error of the various bit extensions (additional bits on integer and fractional parts).

Extension		Exceeding Error			Rounding Error		
IP	FP	*Freq*[1]	*Avg*[2]	*Exp*[3]	*Freq*	*Avg*	*Exp*
+0	+0	0.011	10.8	1.18	98.2	0.018	18.1
+1	+0	0.000	2.7	0.00	98.2	0.018	18.1
+0	+1	0.011	10.6	1.16	97.8	0.009	9.5
+0	+2	0.011	10.6	1.15	97.0	0.005	4.9

The numbers of exceeding errors and rounding errors can be reduced by increasing the bit widths of the integer and fractional parts (Table I). We have considered three extension cases; one bit extension for integer part, and one bit and two bit extension in fractional part. As shown in Table I, the extension of the bit width of the integer part reduces both the probability and the significance of exceeding errors. On the other hand, The extending bit width of the fractional part can reduce both of the probability and the significance of rounding errors.

[1] *Freq* is a frequency of errors, (# of error)/(# of total $psum$)×100%.
[2] *Avg* is an average error, (sum of error)/(# of error).
[3] *Exp* is an expected value of error, (sum of error)/(# of total $psum$)×1,000.

Tiling Condition

Layer	Channel	Number of Channel Loop Tiling
1	3	3
2	96	2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 96
3	384	2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 384
4	384	2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 384
5	256	2, 4, 6, 8, 16, 32, 64, 128, 256

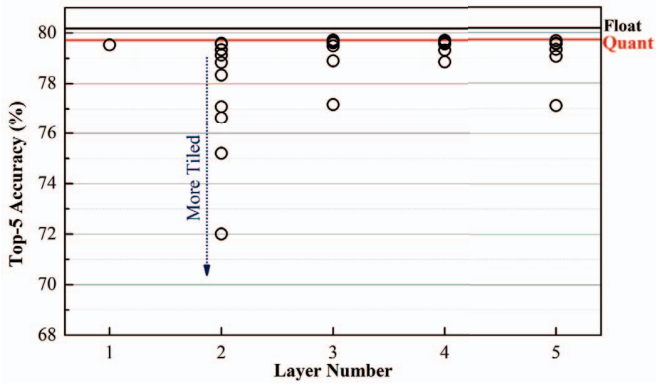

Fig. 3: The effects of *channel loop tiling-error* on Top-5 accuracy of *AlexNet*. Each layer is tiled without tiling channel loops of the other layers. Black lines: floating-point convolution results, red: 8-bit quantized convolution results.

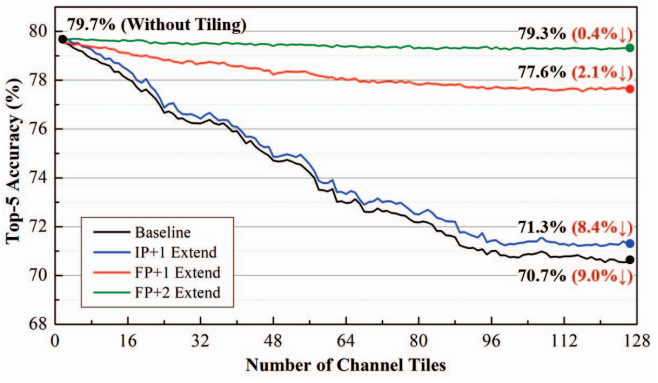

Fig. 4: The influence of *channel loop tiling-error*s from the multiple layers and the accuracy recovery by the extension of the bit widths of the integer part (IP) or the fractional part (FP) of the $psum$. Top-5 accuracy is measured by the inference of 50,000 ImageNet dataset using *AlexNet*.

B. Reduction and recovery of accuracy of channel loop tiled CNN

We have investigated how *channel loop tiling-error* affects the accuracy of CNNs. To evaluate accuracy loss of channel loop tiled CNNs, we implement CNNs in C and CUDA programming by referencing darknet [16]. To mimic channel loop tiling, we quantize partial sums, input feature maps and filters for every channel tile convolution. CNNs are pre-trained using the *ImageNet* training set [17] and evaluated using 50,000 validation sets.

We have analyzed that how *channel loop tiling-error* from a single layer affects the Top-5 accuracy of *AlexNet*. For the experiment, we have swiped the number of channel tiles to all divisors of channel size, C, for each layer (e.g., if C of a layer is eight, we break the input channel in one, two, four, and eight tiles). In Figure 3, the accuracy drops as the number of channel tiles increases in all layers due to the increase of *channel loop tiling-error*. The first layer shows the highest

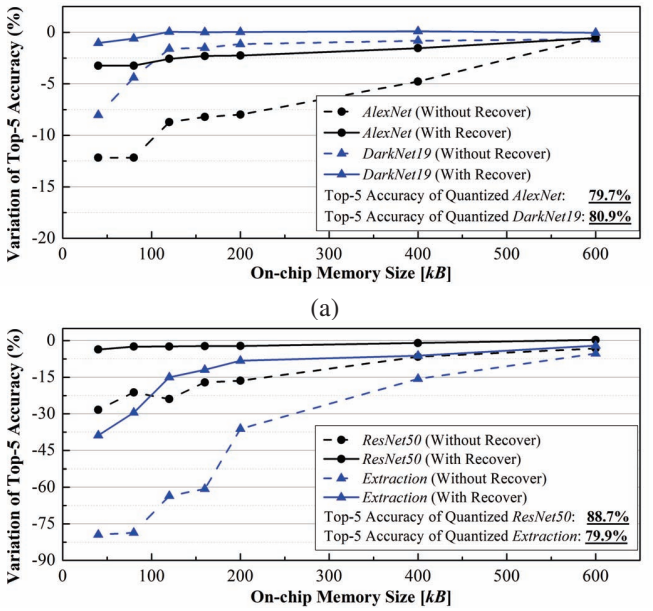

Fig. 5: The variation of accuracy of relatively (a) thin CNNs (*AlexNet* and *DarkNet19*) and (b) deep CNNs (*ResNet50* and *Extraction*) according to on-chip memory size. The number of convolution layers of each CNN is 5, 8, 50 and 21, respectively.

accuracy. This is because the number of input channels is only three, so that the extraction of partial sums occurs at most two times for each activation. Considering that the number of channels of the second layer is smaller than that of the 3rd, 4th, and 5th layers, the second layer is the most vulnerable to *Channel loop tiling-error*. Because errors occurred in the second layer would propagate and accumulate in later layers so that the accumulated errors would greatly affect accuracy.

Then, we have analyzed the influence of the *channel loop tiling-errors* simultaneously occurred in multiple layers. For the experiment, we have changed the number of channel tiles of all layer simultaneously from one to 127. In Figure 4, dividing the channel loops into 127 tiles reduces the accuracy by 9% to 70.7%: the trend is a result of the exceeding and rounding errors. We have plotted the same graph by extending the bit widths of the integer and fractional parts. With the extension of the integer part, the accuracy decreases by 8.4% to 71.3%. If we extend 1-bit or 2-bit of fractional part, the accuracy is reduced by only 2.1% or 0.4%, resulting 77.6% or 79.3%, respectively. Comparing the accuracy of extending the bit width with that of the baseline, the 1-bit extension of the integer part and the 1-bit and the 2-bit extension of the fractional part can recover 0.6%, 6.9% and 8.6%, respectively. Among three extensions, the 1-bit extension of the fractional part shows the highest the accuracy recovery per each bit extension. We have chosen the 1-bit extension of the fractional part to recover *channel loop tiling-error*.

To analyze the accuracy reduction due to channel loop tiling in a real CNN accelerator, we simulate the seven different available on-chip memory sizes: 40, 80, 120, 160, 200, 400 and 600 kB. We have assumed that the feature maps and the filters are double-buffered to overlap the computation with data transfer. For each available memory size, we have appropriately tiled four loops, W, H, C and M loops, so that the size of on-chip memory needed to store the ifmap,

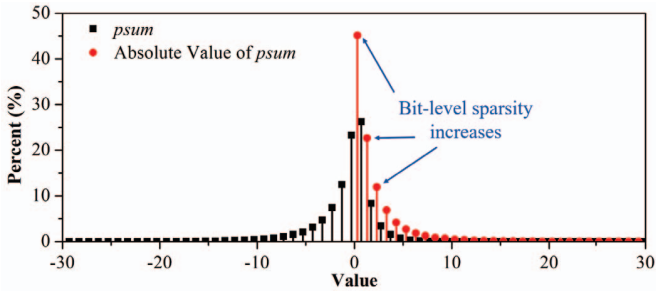

Fig. 6: Histogram of the $psum$ and the absolute value of the $psum$.

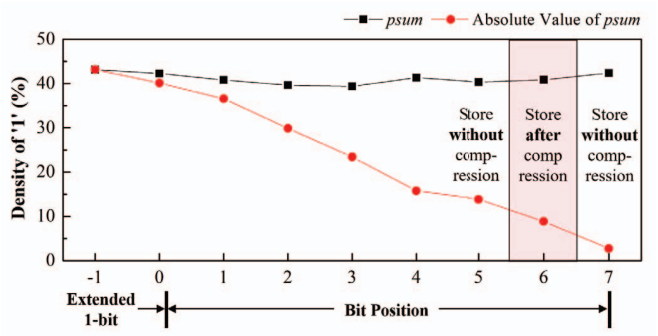

Fig. 7: Density of '1' in each bit position. The absolute values of MSBs are sparse, and the sparse MSBs have an advantage in compression.

ofmap and filter does not exceed the available memory size. Considering that the tiling of W and H loops decreases the bandwidth for transferring the feature maps, We have tiled the C and M loops with priority over the W and H loops. The average number of channel tiles is summarized in Table II.

TABLE II: The average number of channel loop tiles for different available on-chip memory size.

Memory Size [kB]	*AlexNet*	*DarkNet19*	*ResNet50*	*Extraction*
40	224.6	133.1	446.4	429.9
80	224.6	83.6	369.3	335.9
120	158.8	54.8	339.5	210.4
160	57.0	29.3	240.4	170.6
200	36.2	25.8	232.5	125.8
400	15.0	11.9	112.1	58.5
600	4.6	6.5	31.3	21.7

We have measured the accuracy of CNNs for each tiling condition. Figure 5 shows the variation of the accuracy of relatively (a) shallow and (b) deep CNN. As shown in Figure 5, the decrease of on-chip memory size induces *channel loop tiling-error* and reduces accuracy of CNN. The accuracy degradation is relatively large in *ResNet50* and *Extraction*, because these CNNs are very large, so the number of channel tiles is high. When the on-chip memory size is $200kB$, Top-5 accuracy of *AlexNet*, *DarkNet19*, *ResNet50* and *Extraction* decrease by 8.0%, 1.2%, 15.7% and 36.2%, respectively. On average, 15.3% of Top-5 accuracy decreases due to *channel loop tiling-error*. By extending one bit of fractional part, we can recover Top-5 accuracy of *AlexNet* by 5.7%, *DarkNet19* by 1.2%, *ResNet50* by 14.3%, and *Extraction* by 27.9%. On average, 12.3% of Top-5 accuracy can be recovered by the extension one bit of fractional part.

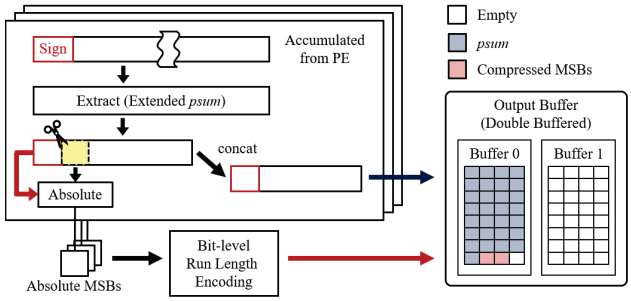

Fig. 8: The operating principle of the proposed *channel loop tiling-error* recovery circuit.

V. Proposed Method to Recover Channel Loop Tiling-error

A. Proposed channel loop tiling-error recovery circuit

The processing element (PE) in a CNN accelerator generates a tile of $psum$ from the convolution of a tile of input feature map and a tile of filter. The $psum$ generated from PEs has larger bit width than that of the output feature map. For the consistency between the bit width of the output feature map and that of the $psum$, $psum$ is extracted to the same bit width as the bit width of the output feature map before being saved in external memory. During the extraction of $psum$, the *channel loop tiling-error*s occur. As analyzed in previous section, if we extend 1-bit in the part, we can recover CNN accuracy. The increase in the bit width of $psum$ requires additional memory space and bandwidth. Moreover, bus widths of memory interfaces are usually multiples of eight bits, so dynamically adjusting bit width is a complex procedure, and circuit overhead is very large.

To compress the partial sums, we exploit a bit-level run-length encoding (RLE) and decoding circuit. RLE saves data and data count as *Run* and *Length*, respectively. In a previous work [11], the RLE have been used to reduce external memory access. They have supposed feature maps are sparse due to activation function of rectified linear unit (ReLU). However, because the partial sums are not rectified using the ReLU, it is not that sparse. Consecutive dense partial sums are less likely to have the same value, and this trait is critical for RLE. If the expected value of $Length$ is lower than a certain value, it is hart to compress the partial sum with RLE. To address this problem, we apply RLE at the bit-level.

To increase the efficiency of the RLE, we exploit the characteristics of the $psum$. Figure 6 shows the distribution of the $psum$ and absolute value of the $psum$ in the second layer of a trained *AlexNet*. The magnitudes of most of the output feature maps are small. When the magnitude of a value is very small, the bits near the MSB are likely to have the same bit value as *sign*. As shown in Figure 7, the probability of '1' near MSB is close to 50%, because the probability that the partial sum is positive is near 50%.

If the sparsity of consecutive bits is near 50%, it is mostly hard to compress using RLE. To reduce the probability that bits near MSBs will be '1', we take the absolute value of partial sums. The density of absolute values of output feature maps of MSBs is almost zero. Therefore, if we encode the absolute value of MSBs using RLE, we can reduce the memory overhead that is required to store of partial sums of which bit width is extended.

Figure 8 shows the proposed *channel loop tiling-error* recovery circuit. To recover the accuracy, we extract $psum$ to eight bits and an extended bit in the fractional parts. To preserve sign of $psum$, seven LSBs and the sign of the rounded $psum$ are concatenated. Then the rounded $psum$ are stored in an output buffer in the same way that a general hardware accelerator saves an output feature map. Decoding the compressed $psum$ proceeds in the opposite way to compression.

B. Evaluation of the proposed method

For the estimation of area and power overhead of the proposed methodology, we design a CNN accelerator in RTL. The accelerator consists of a controller and six PEs which have eight Multiply-and-accumulators, respectively. The designed RTLs are synthesized to a TSMC 65GP cell library at $1GHz$ clock frequency using *Synopsys Design Compiler* [18]. The supply voltage is set to $0.9V$. The total power consumption is the sum of dynamic power and leakage power.

TABLE III: Comparison of memory overhead due to bit extension. The memory overhead is defined by *(additional memory size) / (original memory size)* $\times$ 100%.

Compression with 8-bit run

Extension	*AlexNet*	*DarkNet19*	*ResNet50*	*Extraction*
(IP+1, FP)	0.0986	0.0988	0.1053	0.0986
(IP, FP+1)	0.0991	0.0998	0.1106	0.0993
(IP, FP+2)	0.1989	0.2063	0.2317	0.2035

Compression with 16-bit run

Extension	*AlexNet*	*DarkNet19*	*ResNet50*	*Extraction*
(IP+1, FP)	0.0011	0.0016	0.0151	0.0010
(IP, FP+1)	0.0022	0.0038	0.0262	0.0027
(IP, FP+2)	0.0058	0.0218	0.0756	0.0155

Compression with 32-bit run

Extension	*AlexNet*	*DarkNet19*	*ResNet50*	*Extraction*
(IP+1, FP)	0.0008	0.0017	0.0287	0.0006
(IP, FP+1)	0.0029	0.0062	0.0509	0.0039
(IP, FP+2)	0.0088	0.0407	0.1484	0.0283

To minimize the memory overhead due to the bit extension, we have encoded partial sum with three different $Length$s (8-bit, 16-bit and 32-bit). Without compression, 12.5% of memory overhead per every one bit extension occurs. Table III shows the average memory overhead of different CNNs. The proposed method shows superior compression ratio. The average memory overheads of compressed MSBs with 8-bit, 16-bit and 32-bit RLE are 0.136%, 0.012%, and 0.022%, respectively. Among three $Length$s, 16-bit compression shows the best compression ratio. In most cases, the utilization of the on-chip memory is not 100%, so we can store 0.012% of additional data without increasing the on-chip memory size.

TABLE IV: Area and power overhead of run-length encoder and decoder of different length (eight, 16 and 32-bit).

Component	Circuit Area [μm^2]	Total Power [mW]
8-bit Run-length Encoder	471	0.26
16-bit Run-length Encoder	713	0.38
32-bit Run-length Encoder	1178	0.63
8-bit Run-length Decoder	471	0.26
16-bit Run-length Decoder	665	0.48
32-bit Run-length Decoder	1178	0.63
CNN Accelerator	145,731	43.1

We have also investigated circuit overhead of the 16-bit run-length encoder and decoder circuit (Table IV). The area and power of the encoder and the decoder are very small compared to that of the others. The proposed 16-bit encoder and decoder circuit only consumes 2.0% of additional power with 0.95% of circuit area overhead.

VI. Conclusion

When pre-trained CNNs are implemented on an accelerator, the accuracy of CNNs can be reduced. This paper has introduced the *channel loop tiling-error*, which is a reason for the accuracy reduction during implementation of CNN on an acceletor. We have partitioned this error into exceeding error and rounding error, then analyzed separately how these errors affect the accuracy of four state-of-the-art CNNs (*AlexNet, DarkNet19, ResNet50, Extraction*). Channel loop tiling caused 15.3% of Top-5 accuracy loss on average. We also proposed a solution to recover accuracy loss caused by channel loop tiling. By compressing the extended bits, we minimize memory and circuit overhead. On average, 12.3% of Top-5 accuracy can be recovered at the cost of only 0.012% of additional memory and 1% of circuit area overhead. Our ongoing work seeks to optimize the loop tiling with the consideration of the *channel loop tiling-error* and the performance.

Acknowledgement

This work was supported by the Samsung Research Funding and Incubation Center of Samsung Electronics under Project Number SRFC-TB1703-07.

References

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Proc. NIPS*, 2012, pp.1097-1105.

[2] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", *Proc. CVPR*, 2015, pp.3431-3440.

[3] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", *Proc. CVPR*, 2016, pp.170-171.

[4] A. Canziani, A. Paszke and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications", *arXiv preprint arXiv:1605.07678*, 2016.

[5] C. Zhu, S. Han, H. Mao and W. J. Dally, "Trained Ternary Quantization", *arXiv preprint arXiv:1612.01064*, 2017.

[6] M. Rastegari, V. Ordonez, J. Redmon and A. Farhadi, "Xnor-net: Imagenet Classification using Binary Convolutional Neural Networks", *Proc. ECCV*, 2017, pp. 525-542.

[7] A. Zhou, A. Yao, Y. Guo, L. Xu and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights", *arXiv preprint arXiv:1702.03044*, 2017.

[8] D. Lin, S. Talathi and V. Annapureddy, "Fixed Point Quantization of Deep Convolutional Networks", *Proc. ICML*, 2016, pp.2849-2858.

[9] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients", *arXiv preprint arXiv:1606.06160*, 2016.

[10] G. Philipp, "Ristretto: Hardware-oriented approximation of convolutional neural networks", *arXiv preprint arXiv:1605.06402*, 2016.

[11] Y.-H. Chen, T. Krishna, J. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks", *IEEE JSSC*, 52(1) (2017), pp.127-138.

[12] Y. Shen, M. Ferdman and P. Milder, "Maximizing CNN Accelerator Efficiency Through Resource Partitioning", *Proc. ISCA*, 2017, pp.535-547.

[13] A. Rahman, S. Oh, J. Lee and K. Choi, "Design Space Exploration of FPGA Accelerators for Convolutional Neural Networks", *Proc. DATE*, 2017, pp.1147-1152.

[14] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, "Optimizing Fpga-based Accelerator Design for Deep Convolutional Neural Networks", *Proc. ISFPGA*, 2015, pp.161-170.

[15] M. Motamedi, G. Philipp, and G. Soheil. "PLACID: a platform for FPGA-based accelerator creation for DCNNs" *ACM Transactions on Multimedia Computing, Communications, and Applications*, 13(4) (2017) 62.

[16] J. Redmon, "Darknet: Open Source Neural Networks in C", http://pjreddie.com/darknet/.

[17] O. Russakovsky, J. Deng, H. Su, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *IJCV*, 115(3) (2015), pp.211-252.

[18] *Synopsys Design Compiler User's Manual*, http://www.synopsys.com.