An Optimized Design Technique of Low-bit Neural Network Training for Personalization on IoT Devices

Seungkyu Choi, Jaekang Shin, Yeongjae Choi, and Lee-Sup Kim
School of Electrical Engineering, Korea Advanced Institute of Science and Technology
Daejeon, Republic of Korea
{skchoi, jkshin, yjchoi}@mvlsi.kaist.ac.kr, leesup@kaist.ac.kr

ABSTRACT

Personalization by incremental learning has become essential for IoT devices to enhance the performance of the deep learning models trained with global datasets. To avoid massive transmission traffic in the network, exploiting on-device learning is necessary. We propose a software/hardware co-design technique that builds an energy-efficient low-bit trainable system: (1) software optimizations by local low-bit quantization and computation freezing to minimize the on-chip storage requirement and computational complexity, (2) hardware design of a bit-flexible multiply-and-accumulate (MAC) array sharing the same resources in inference and training. Our scheme saves 99.2% on on-chip buffer storage and achieves 12.8x higher peak energy efficiency compared to previous trainable accelerators.

1 INTRODUCTION

Artificial intelligence (AI) can be accessed by various applications in real life where a vast network is formed with IoT devices [8]. Deep learning, especially the convolutional neural networks (CNNs), is a dominant algorithm in the AI field for decision making of vision tasks with the highest performance. However, the inference of deep learning models trained with global datasets often shows degraded performance on IoT devices due to the different characteristics of the inputs that the device receives compared to those from a general environment.

In order to enhance the performance, incremental learning [4] of personal data on pre-trained models is introduced [7, 12]. Additional training with the data received from the device can improve the inference performance by tuning the model to fit into its personal environment. We define this action as 'device personalization', which is an essential function for IoT systems. Prior works conduct training on cloud servers by transmitting personal data and receiving updated models consistently.

However, in an IoT network consisting of a large number of smart devices, if all applications access to the server to update their model, the transmission traffic will increase exponentially to send/receive a massive amount of data leading to high energy cost and long latency time. Besides, privacy issues from data communication in personal devices cannot be ignored as well. In consequence, realizing on-device CNN training has become an attracting topic to prevent such problems [3, 9].

[3] optimizes the dataflow of the weight gradient computation in the training process to reduce energy consumption. [9] presents an embedded platform for energy-efficient training where the programmable architecture supports flexible dataflow to operate various computations in training. However, both previous processors support only 16-bit or 32-bit fixed point operations for training. For the deep learning accelerator in IoT devices where the inference driven with low precision is the primary task, area and energy overhead will be significant if the hardware utilizes full precision to support training.

Meanwhile, software simulations on low-bit training by data quantization to lessen the computational cost have also been attempted [2, 13]. However, these attempts dealing with server GPUs do not consider implemention issues in low power systems, i. e. high-cost computations and a considerable amount of on-chip data to hold on. Accordingly, following previous low-bit training method cannot be freely applied to a resource-limited low power system.

In this work, we propose a novel design technique for low-bit training of deep CNNs in IoT devices to apply personalization. We first propose several personalization schemes for smart devices using authorized deep learning models. They can be used as reliable benchmarks of our design to certify accurate incremental learning with low bit-width. Then, since low power systems suffer from limited resources, a new quantization mechanism for training to reduce the required on-chip buffer storage is proposed by cutting the amount of data to hold at the same time in the accelerator. Also, through applying a freezing method to the batch normalization layer, the training system becomes tolerant to the mini-batch size allowing less on-chip buffer storage and reducing computational complexity. Lastly, we implement a hardware design of bit-flexible multiply-and-accumulate (MAC) array structure to run both inference and training, which have different bit precision of the data.

To the best of our knowledge, this paper is the first work to design a low-bit incremental learning system with limited on-chip storage targeting the on-device training. The fundamental objective of our design is to run training in the same computing resources used in inference with minimal overhead. The major contributions of this work are summarized as follows.

- Personalization frameworks for IoT devices trained with authorized ImageNet models

DAC '19, June 2-6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06. . . $15.00
https://doi.org/10.1145/3316781.3317769

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NO. 2017R1A2B2009380), and Samsung Advanced Institute of Technology.

- Low-bit training by local maximum quantization and batch normalization freezing to reduce the required on-chip buffer storage and computational complexity
- Implementation of a deep learning accelerator utilizing bit-flexible MAC array structure for energy-efficient inference/training

2 BACKGROUND

2.1 MAC Operations in Training Deep CNNs

A typical way to learn a deep learning model is to back-propagate the loss gradients through the model and update its weights by a gradient descent optimizer. The loss is produced through feedforward propagation and then it is back-propagated to produce each layer's weight gradients. In the whole training process consisting of the forward and backward phase, there are three types of computations, which are feedforward propagation, back-propagation, and weight gradient computation.

The three computations in a single convolutional layer are visualized in Fig. 1. All of them can be regarded as a convolution process containing multiple MAC operations. We can conclude that in learning deep CNNs, each computation is processed by MAC operations with two inputs among three different data types, which are activations, loss gradients, and weights.

2.2 Challenges of Trainable Deep Learning Accelerator for IoT Devices

There are two crucial implementation issues in designing neural network accelerator chips for a low power system. First, limited computational resources and area can hinder a low-cost design of deep learning processors. The bounded area limits the computing MAC array size and the size of on-chip buffer storage. This phenomenon restricts various software optimizations since the chip holds far less amount of data compared to server GPUs.

Second, the major task of the accelerator in IoT devices should be the inference. Therefore, the accelerator must show high performance in its primary duty while facilitating training mode with minimal area overhead. To fulfill this, designing inference/training reconfigurable computing architecture is necessary.

3 PERSONALIZATION FRAMEWORK

This section introduces the proposed personalization scheme of additional training where IoT devices can achieve higher performance to their incoming personal data. To demonstrate the effectiveness, we customize two classification frameworks

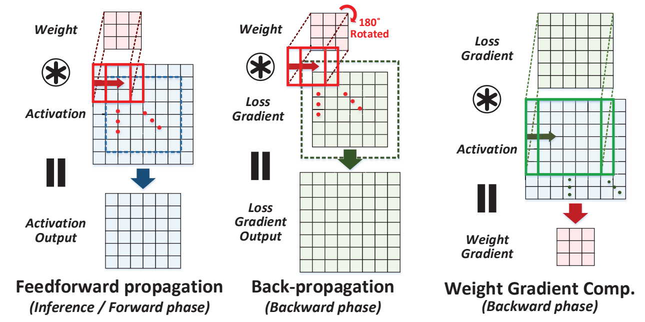


Figure 1: MAC operations in a convolutional layer

using authorized deep learning models for our on-device training benchmarks. To set the frameworks in a condition of running in a low power system, both are simulated with low-bit inference having minimal accuracy drop. Handwriting classification is evaluated in 2-bit/4-bit of weight/activation. ImageNet models are fine-tuned to evaluate in both 4-bit of weight and activation.

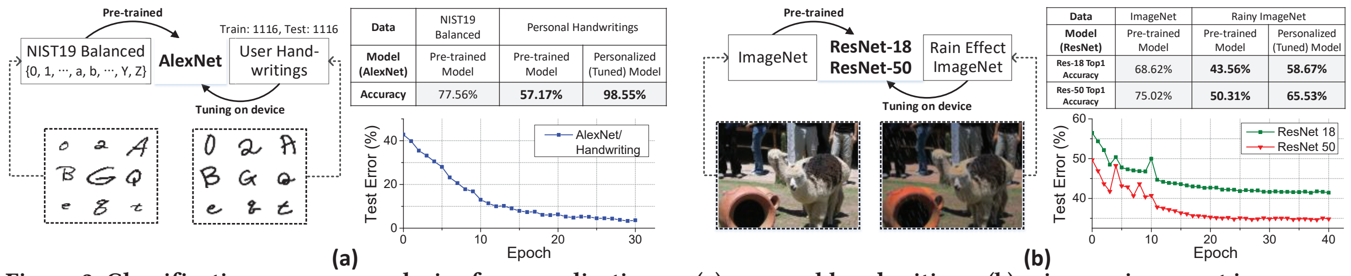
3.1 Handwriting Personalization

Classifying total 62 labels consisting of 10 digit numbers, 26 capital letters and 26 small letters with high accuracy can be a picky task. We organize AlexNet [14] based model classifying 62 classes and trained with NIST19 dataset [11]. For validation, the same amount of data in all 62 classes from the original dataset are extracted to make a balanced NIST19 dataset. The converged model has a classification accuracy of 77.6%.

We gather specific user's personal handwritings to test on the converged model. Testing 1116 data results in an accuracy of 57.2%, showing significant degradation. Then the converged model is trained with 1116 training data as shown in Fig. 2(a). After training some epochs, the accuracy of user's handwritings rises to 98.5%. Self-personalization for handwritings not only recovers its original accuracy but also reduces the classification error rate 29x since users have their distinguishable handwriting shapes compared to typical data.

3.2 Personalization on ImageNet

We construct an ImageNet [10] classification system using well-known ResNet18 and 50 models [15], which are pre-trained with the original ImageNet training dataset. To make the device environment look as if it is in personal circumstances, we set the climate in rainy weather and use 'rainy ImageNet' as the device input. The rainy ImageNet is built by darkening, blurring, and adding randomly generated raindrops to the original ImageNet as shown in the picture of Fig. 2(b).



Data	NIST19 Balanced	Personal Handwritings	
Model (AlexNet)	Pre-trained Model	Pre-trained Model	Personalized (Tuned) Model
Accuracy	77.56%	**57.17%**	**98.55%**

Data	ImageNet	Rainy ImageNet	
Model (ResNet)	Pre-trained Model	Pre-trained Model	Personalized (Tuned) Model
Res-18 Top1 Accuracy	68.62%	**43.56%**	**58.67%**
Res-50 Top1 Accuracy	75.02%	**50.31%**	**65.53%**

Figure 2: Classification accuracy analysis of personalization on (a) personal handwritings, (b) rainy environment images

The top-1 ResNet18 testing accuracy of rainy ImageNet is 43.6%, which shows a considerable accuracy degradation. After training with some rainy training data, the accuracy recovers to 58.7% reducing 1.4x of the classification error rate. The same behavior is shown in ResNet50. The degraded top-1 test accuracy of 50.3% for rainy ImageNet recovers to 65.5% after training.

4 LOW PRECISION ON-DEVICE TRAINING METHODOLOGY

This work aims low-bit training on low power systems. In this section, we present two features of software optimization that intensely reduce the computational complexity of training.

4.1 Local Maximum Quantization (LMQ)

4.1.1 Maximum Quantization. To train with low-bit data, not only activations and weights but also the gradients must be quantized. However, gradient values are unbounded and their absolute values have a big influence on updating model. Thus, the conventional quantization for gradients is made by creating steps inside the +/- range of the absolute maximum value [2, 13].

We define $quantize_n(r)$ as quantizing input $r \in [0, 1]$ to an n-bit level divided by 2^n. Then, the maximum quantization of unbounded input $x \in (-\infty, \infty)$ in equation (2) can be represented in n-bit 2's complement fixed point form by discarding the opposite boundary value which is not selected as the maximum.

$$quantize_n(r) = \frac{1}{2^n} \cdot round(2^n \cdot r) \tag{1}$$

$$\textbf{Max Quantize: } maxq_n(x) = quantize_n\left[\frac{x}{2max(|x|)} + \frac{1}{2}\right] \tag{2}$$

$$\textbf{Recover: } x = 2max(|x|) \cdot \left[maxq_n(x) - \frac{1}{2}\right] \tag{3}$$

Since non-linear operations such as batch normalization are calculated in full precision, recovering step of equation (3) is inevitable. Also, for loss gradients, adding Gaussian noise to the data before quantizing increases the training performance [2].

Our system applies the maximum quantization to weights and activations as well as loss gradients. The common quantization process can share resources, and unbounding all data helps weight gradients to be more accurate for updating their weights.

4.1.2 Local Maximum Quantization (LMQ). The maximum quantization method is powerful to reduce the bit-width, yet it has a fatal flaw that it has to hold all the propagated output from the computing units to find the maximum value and quantize before storing back. In other words, on-chip buffers which are directly connected to computing units must be larger than the size of the largest layer to store all data for common maximum quantization. However, for digital chips in low power systems, the buffer and arithmetic unit arrays cannot blindly grow larger.

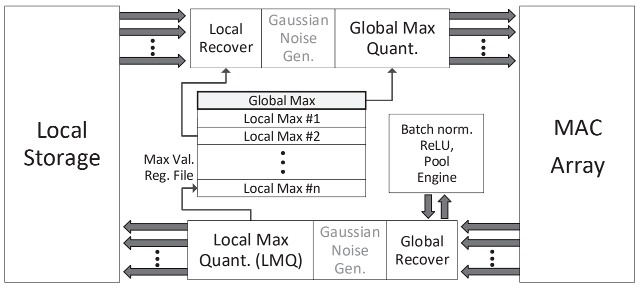


Figure 3: Block diagram of LMQ architecture

Algorithm 1 Pseudo-code for LMQ in the backward phase

```
 1: Quantization level selection: $W, A, G$
    // Exception in the 1st and last layer as usual.
    // Forward phase has the same LMQ process of backward.
 2: $weight_W$ ← Quantize($weight$, $global\_max$)
 3: for $l$ = $layer\_num$ to 1 do  //Loss back-propagation
    //$channel\_in$ is the output channel since it back-propagates.
 4:   for $C_{out}$ = 1 to $channel\_in$ do
 5:     for $C_{in}$ = 1 to $channel\_out$ do
 6:       $loss\_gradient$ ← Recover($loss\_gradient_G$, $local\_max_{Cin}$)
 7:       $loss\_gradient_G$ ← Quantize($loss\_gradient$, $global\_max_l$)
 8:       Backward_convolution($loss\_gradient_G$, $weight_W$)
 9:     end
10:     $loss\_gradient$ ← Recover($loss\_gradient_G$, $global\_max_l$)
11:     Backward_batchnorm, Backward_pool, Backward_relu
12:     $local\_max_{Cout}$ ← Find_max($loss\_gradient$)
13:     $loss\_gradient_G$ ← Quantize($loss\_gradient$, $local\_max_{Cout}$)
14:   end
15:   $global\_max_{l-1}$ ← Find_max($local\_max_{C1}$, $local\_max_{C2}$, ⋯)
16: end
17: for $l$ = $layer\_num$ to 1 do  //Weight update
18:   Weight_gradient_computation($activation_A$, $loss\_gradient_G$)
19:   $weight_l$, $w\_grad_l$ ← Recover($weight_W$), Recover($w\_grad_{A,G}$)
20:   Weight_update($weight_l$, $w\_grad_l$)
21: end
```

To reduce the required on-chip buffer storage for low-bit computing, we reconstruct the quantization in a dual stage. As illustrated in Fig. 3, the system locally quantize a group of final output data from the MAC array with their local maximum value. The local maximum values are stored in a register file for the next layer's data recovery. When all groups of a layer are finally computed, the system finds the global maximum value among the local values. Then, at the computation step of the next layer, the encoded data recover their values with the local maximum to be globally quantized before being streamed to the MAC array.

The reason why LMQ additionally quantizes input with the global maximum is that the next output is accumulated through the whole input data. More specifically, if we only quantize the data with local maximums, different groups of input which are quantized with different local maximums are accumulated together to make outputs. To obtain the proper output, every partial sum accumulated by a single group must be recovered into full precision before accumulating partial sums produced by other groups. Therefore, synchronizing the input with the same maximum value is essential to avoid recovery in the middle of accumulations.

We further design the LMQ system by dividing groups with the channel dimension, which means the data of different output channel is quantized by different local maximums. Algorithm 1 states the detailed LMQ process in the backward training phase.

4.2 Freezing Batch Normalization

An operation showing the most substantial difference between inference and training is the batch normalization (BN). In the

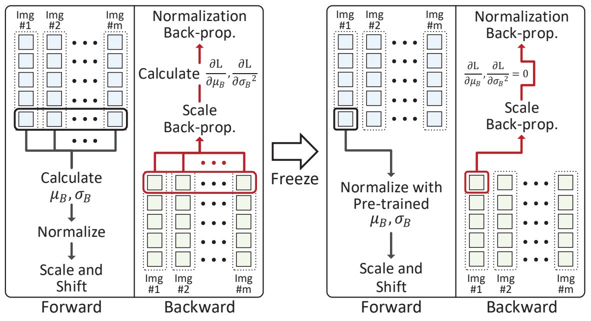

Img #1
Img #2
Img #m
Normalization Back-prop.
Calculate $\frac{\partial L}{\partial \mu_B}, \frac{\partial L}{\partial \sigma_B^2}$
Scale Back-prop.
Calculate μ_B, σ_B
Normalize
Scale and Shift
Forward
Backward
Freeze
Normalize with Pre-trained μ_B, σ_B
$\frac{\partial L}{\partial \mu_B}, \frac{\partial L}{\partial \sigma_B^2} = 0$

Figure 4: Freezing batch normalization

inference mode, the batch normalization becomes a simple affine transformation (aX + b) because the pre-determined moving mean (μ), moving variance (σ^2) and additional parameters (γ, β) can be calculated together in advance as shown in equation (4).

$$y_i = \gamma \hat{x}_i + \beta = \frac{\gamma}{\sqrt{\sigma^2+\varepsilon}} x_i + \left(\beta - \frac{\gamma \cdot \mu}{\sqrt{\sigma^2+\varepsilon}}\right) = \gamma' x_i + \beta' \quad (4)$$

From equation (4), the batch normalization inference mode needs only one multiply and one add operation per each data. However, for training, every batch normalization at the forward phase must calculate the mean and variance along the mini-batch before applying normalization and affine transformation. In the backward training phase, the loss gradients over mean and variance are also calculated for back-propagation [6]. As a result, the number of operations increases excessively and complex calculations such as root and division are performed. Moreover, the biggest computational overhead comes from requiring large on-chip buffer storage because the system cannot compute in a batch-wise manner and must hold all the data of mini-batches.

To reduce the holding data, we freeze the batch normalization to only update parameters of affine transformation (γ, β) and fix the moving mean and variance value of the layer. The pre-trained model has achieved each layer's moving mean and variance over multiple iterations until the convergence. Hence, we skip the mean/variance calculation in the forward training phase and use the pre-calculated moving mean/variance instead. Consequently, in our training mode, the forward phase has the same computation behavior with the inference and the backward phase can be simplified as following equations.

$$backprop_{freeze-bn}: \ \frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} \cdot \gamma \cdot \frac{1}{\sqrt{\sigma^2+\varepsilon}} \quad (5)$$

$$\frac{\partial E}{\partial \gamma} = \Sigma_{i=1}^{m} \frac{\partial E}{\partial y} \cdot \gamma \cdot \hat{x}_i, \quad \frac{\partial E}{\partial \beta} = \Sigma_{i=1}^{m} \frac{\partial E}{\partial y} \quad (6)$$

The loss gradients back-propagates through equation (5), and transformation parameters (γ, β) are updated by the gradients

Table 1: Computational differences by freezing BN

	Inference		Forward		Backward	
	before	freeze	before	freeze	before	freeze
DIV/ROOT operations	X	X	O	X	O	X
# of images to hold	1	1	*m*	1	*m*	1
Operation count	*2rcom*	*2rcom*	*6rcom*	*2rcom*	*16rcom*	*4rcom*

r: row, *c*: column, *o*: channel size, and *m*: mini-batch size

calculated in equation (6). The frozen batch normalization is visualized in Fig. 4. As shown in the equation (5) and in Fig. 4, only a simple multiplication enables back-propagation. Table 1 compares the computational differences in batch normalization of before and after freezing. The operation count reduces 72.7% in the training process, and the division or root computing units are unnecessary. Furthermore, by handling images of a batch in a sequential way, batch-tolerant computing is available in training, which drastically reduces the number of data to be held.

5 HARDWARE DESIGN

We design a bit-flexible MAC array structure, which can maximize energy efficiency on inference while sharing the same architecture with training.

5.1 Input Precision in Personalization Schemes

For handwriting personalization which is trained by modified AlexNet, weight (W) with 2-bit, activation (A) with 4-bit, and loss gradient (G) with 4-bit can tune the model with no accuracy degradation. For rainy ImageNet classification with ResNet18 and 50, W with 4-bit, A with 4-bit, and G with 8-bit was enough for accurate training. The simulation results of accuracy analysis in low-bit training is described in section 6.

5.2 Bit-flexible MAC: Selective Spatial Fusion

To maintain high energy efficiency in inference mode, lower bit-width data must be computed in the same MAC unit with higher bit-width data. A previous study introduces bit fusion architecture, which can compute MAC operations with dynamic bit level [5]. By fusing multiplier bricks that multiplies with the lowest bit-width, MAC units can be utilized in wider bit-widths.

[5] offers two choices to fuse the bricks. Fusing in a spatial manner connects multiplier bricks to accumulate immediately to make a single output. It has an advantage in area and power of a single MAC unit but requires wide bit-width of SRAM access, which will result in lack of SRAM bandwidth and high power for the whole system. The other is temporal fusion where each brick accumulates in a few successive iterations and produce separate output. It can prevent the wide SRAM accesses, but the MAC unit consumes too much power and takes up a large area.

We architect our 2/4/8 bit-flexible MAC unit by selectively accommodating spatial fusion to obtain the advantage of spatial fusion while also reducing the SRAM bit-width. In detail, we stack the multiplier bricks in a spatial manner but isolate the accumulation of the partial sum if the data is separated. Separate accumulation gives a diverse choice of input data enabling to share the same data through different multiplier bricks, which results in SRAM bit-width reduction. Separately storing partial sum requires a longer bit-width of the accumulate register than the original spatial fusion but shorter than the temporal fusion. The comparison between spatial, temporal and our selective spatial fusion is described in Table 2 with area synthesis and power analysis included. Our design displays a larger area and higher power compared to spatial fusion due to the separate accumulation. Nevertheless, it reduces SRAM bandwidth into

Table 2: MAC Comparison with various bit fusions

	Temporal Fusion	Spatial Fusion	Selective Spatial Fusion
Area (μm^2)	1260	688.6	1041
Power (μW)	138	64.1	90.7
Register-level reuse	X	O	O
Required SRAM width	12	24	12

half which will drastically reduce the consuming power of SRAM buffer and save high energy in perspective of an overall system.

Fig. 5 (a) to (c) illustrates three different modes that our bit-flexible MAC utilizes. The lowest level mode utilizes 2-bit/4-bit MAC operations. Four data of different output channel are multiplied with the same row input data and are accumulated separately to produce four different 6-bit partial sums. For the second mode of utilizing 4-bit/4-bit operations, two different 4-bit data are divided into four 2-bit data and multiplied with the same data. Two 2-bit/4-bit multiplication outputs are shifted and added to produce final 4-bit/4-bit output and then accumulated to the register. Two partial sums of different output channel are accumulated separately. For the last mode of 8-bit/4-bit, all 2-bit inputs are the connected version of a single input, so all outputs of the multiplier bricks are shifted and added to produce a single output, which will be accumulated to the partial sum.

2D MAC array structure is implemented using the proposed bit-flexible MAC units depicted in Fig. 5 (d). Since our design is only based on the spatial fusion where the data is not streamed in multiple cycles such as the temporal fusion, register-level input reuse is possible by shifting the data to the near MAC units. Register-level reuse can significantly reduce the SRAM data accesses approving more energy-efficient dataflow.

To utilize the MAC array both in inference and training, all three types of input, weights, activations, and gradients can be streamed in both row and column directions. Accordingly, we exploit two directional data shifting to facilitate register-level reuse in both row and column directions. In this way, register reuse is supported in any combination of inputs to compute all three operations mentioned in Fig. 1.

6 EVALUATION

6.1 Experimental Setup

The software simulation is performed on Tensorflow [1]. We analyze the classification results of two personalization schemes introduced in section 3 applying our optimizations of low-bit on-device training. A Verilog RTL model of the proposed accelerator is designed and synthesized by Synopsys Design Compiler under Samsung CMOS 65nm library. Estimation of power and energy consumption is obtained by using Synopsys PrimeTime-PX.

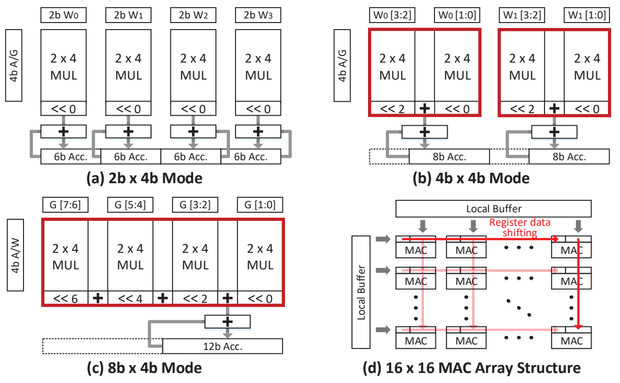


Figure 5: Bit-flexible MAC structure

6.2 Results

6.2.1 Low-bit Training Results of Personalization. Training with low-bit data in LMQ is evaluated. The bit precision for inference is selected beforehand in section 3, where weights/ activations are computed in 2-bit/4-bit for handwriting classification and 4-bit/4-bit for ImageNet classification. Thus, we reduce the bit-width of gradients until the accuracy degradation occurs in the final tuned model.

As shown in Fig. 6(a), for handwriting classification, training with gradients in 4-bit converges to 97.9% having negligible accuracy degradation over gradients in full precision. Furthermore, applying frozen batch normalization layers also converges and follows the same training behavior. For ResNet training, 8-bit for gradients is the shortest bit-width to converge with no accuracy degradation over full precision. Consequently, weight/ activation/gradient bit-width combination of 2/4/4 and 4/4/8 can run both inference and training for device personalization.

6.2.2 Comparison of the On-chip Storage Requirement. The proposed method of LMQ and batch normalization freezing has the common advantage to hold less amount of data on the buffer directed to the computing resources. The LMQ enables channel level quantization which only needs the buffer size of a single channel to hold when producing partial sums of the output, and frozen batch normalization allows batch-wise computation to avoid holding data from multiple images while training.

Fig. 7 depicts the minimum size of the on-chip storage to secure the propagated data i. e. activations or gradients, to fully run deep learning models. With previous maximum quantization [2], the computing system must hold whole data of a single layer to find the maximum value and re-quantize the output with the value to store back in the memory. By this principle, even more data than the full precision (FP) representation must be secured

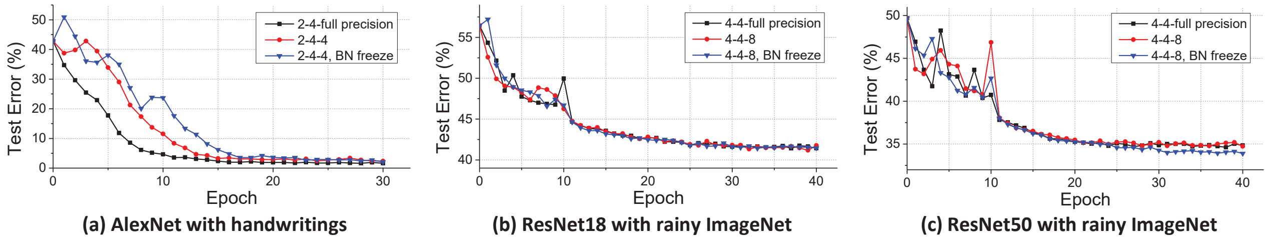


Figure 6: Low-bit training accuracy analysis for personalization schemes

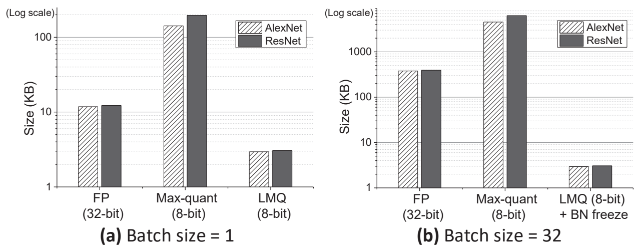


Figure 7: On-chip buffer storage required for computing

in the on-chip buffer, which becomes inefficient on a resource-limited system. However, LMQ locally quantizes the output data in channel level, so the buffer only needs to secure the largest image size among layers.

Through freezing batch normalization (BN) layers, our system becomes tolerant to the training mini-batch size. Fig. 7(b) depicts the required on-chip storage with a mini-batch size of 32. Our system maintains the size near 3KB while others scale up by the mini-batch size to hold all the data for batch normalization. Therefore, applying both LMQ and batch-tolerant computing reduces the required on-chip storage for propagated data by 99.2% compared to full precision computing.

6.2.3 Energy Breakdown and Throughput. With the bit-flexible MAC structure, we can utilize all the MAC operations conducted in both low-bit inference and training with different precision level. We evaluate three different types of MAC operations of ResNet18 on our design to analyze the energy breakdown which is validated in Fig. 8(a). Forward and backward propagation shows similar behavior since they only differ from the order of the layer. On the other hand, lack of reuse in weight gradient computation results in high SRAM and low DRAM energy consumption portion compared to the others.

The peak throughput of running in inference and training mode is given in Fig. 8(b). The bit-flexible MAC units in lower bit mode utilize separate accumulation of multiple output channels allowing no latency overhead over general low-bit MAC units. Therefore, our accelerator design can preserve peak throughput on inference and also achieve high throughput on training computed with the average of all MAC operations.

6.3 Hardware Spec. and Energy Efficiency

Low-bit training can share the same computing engine with low-bit inference by implementing bit-flexible MAC structure. Therefore, it is self-evident that our system can save tens of times more energy compared to previous deep learning systems only supporting full precision computing in training.

Table 3 compares hardware specifications between trainable hardware accelerators. The proposed accelerator is designed composing of total 120KB of SRAM buffers and an array of 16 by 16 bit-flexible MAC units. It consumes 42.2mW to 45.0mW at 200MHz achieving up to 9.1 TOPS/W. Our low-bit trainable accelerator achieves 12.8 times higher peak energy efficiency compared to the previous trainable ASIC which utilizes 32-bit fixed point MAC operations. The first proposed low-bit deep learning training design is a promising solution for low power IoT devices which suffer from limited computing resources.

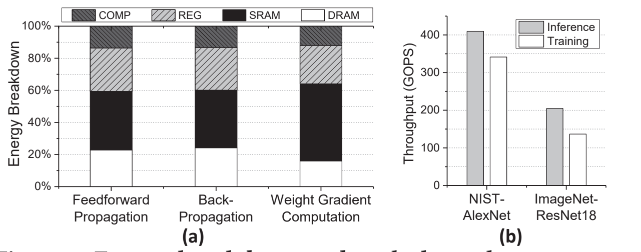


Figure 8: Energy breakdown and peak throughput

Table 3: Comparison with previous hardware

	Jetson TX2	**DT [9]**	**TW [3]**	**Proposed**
Frequency (MHz)	1300	2500	100	200
Bit Precision	16/32 float	16/32 fixed	16 fixed	2/4/8 fixed
Area (mm^2)	/	1.17 (15nm)	/ (65nm)	1.23 (65nm)
Peak Throughput (GMAC/s)	332.8	945	16.2	204.8
Power (W)	7.50	2.65	0.053	0.045
Energy Efficiency (TOPS/W)	0.8875	0.7132	0.617	9.102

7 CONCLUSION

In this work, we proposed a novel design of deep learning personalization on IoT devices, which enables energy-efficient low-bit training sharing same computing engine with inference. Our design consists of both software and hardware optimization to apply low-bit training in low power systems. Local maximum quantization (LMQ) and batch-tolerant computing allowed the training system to minimize the on-chip storage required for the propagated data. We designed an accelerator with a novel bit-flexible MAC array structure to maximize energy efficiency on low-bit inference while supporting training operations. The proposed low-bit training enables the unconstrained design of low power trainable processor and achieves 12.5x higher energy efficiency compared to the previous training processors.

REFERENCES

[1] M. Abadi et al. Tensorflow: A system for large-scale machine learning. OSDI, 2016, pp. 265-283.
[2] S. Zhou et al. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv:1606.06160, 2016.
[3] S. Choi et al. TrainWare: A memory optimized weight update architecture for on-device convolutional neural network training. ISLPED, 2018, pp.19:1-19:6.
[4] Y. Shen et al. Incremental learning vector quantization for character recognition with local style consistency. Springer International Publishing, 2016.
[5] H. Sharma et al. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. ISCA, 2018.
[6] S. Ioffe et al. Batch normalization: Accelerating deep network training by reducing internal covariate shift. ICML, 2015, pp.448-456.
[7] M. Song et al. In-Situ AI: Towards autonomous and incremental deep learning for IoT systems. HPCA, 2018, pp. 92-103.
[8] Smart city artificial intelligence applications and trends. https://www.techemergence.com/smart-city-artificial-intelligence-applications-trends/.
[9] D. Kim et al. DeepTrain: A programmable embedded platform for training deep neural networks. IEEE TCAD, 2018, vol. 37, no. 11, pp. 2360-2370.
[10] O. Russakovsky et al. ImageNet large scale visual recognition challenge. IJCV, 2015.
[11] P. Grother. NIST special database 19 handprinted forms and characters database. NIST, 2016.
[12] B. Harris et al. Architectures and algorithms for user customization of CNNs. ASP-DAC, 2018, pp. 540-547.
[13] S. Wu et al. Training and inference with integers in deep neural networks. ICLR, 2018.
[14] A. Krizhevsky et al. Imagenet classification with deep convolutional neural networks. NIPS, 2012, pp. 1097-1105.
[15] K. He et al. Deep residual learning for image recognition. arXiv:1512.03385, 2015.