PAPER Special Section on Reconfigurable Systems

# GUINNESS: A GUI Based Binarized Deep Neural Network Framework for Software Programmers

**Hiroki NAKAHARA**[†a)], **Haruyoshi YONEKAWA**[†b)], **Tomoya FUJII**[†c)], **Masayuki SHIMODA**[†d)], ***and*** **Shimpei SATO**[†e)], ***Members***

**SUMMARY** The GUINNESS (GUI based binarized neural network synthesizer) is an open-source tool flow for a binarized deep neural network toward FPGA implementation based on the GUI including both the training on the GPU and inference on the FPGA. Since all the operation is done on the GUI, the software designer is not necessary to write any scripts to design the neural network structure, training behavior, only specify the values for hyperparameters. After finishing the training, it automatically generates C++ codes to synthesis the bit-stream using the Xilinx SDSoC system design tool flow. Thus, our tool flow is suitable for the software programmers who are not familiar with the FPGA design. In our tool flow, we modify the training algorithms both the training and the inference for a binarized CNN hardware. Since the hardware has a limited number of bit precision, it lacks minimal bias in training. Also, for the inference on the hardware, the conventional batch normalization technique requires additional hardware. Our modifications solve these problems. We implemented the VGG-11 benchmark CNN on the Digilent Inc. Zedboard. Compared with the conventional binarized implementations on an FPGA, the classification accuracy was almost the same, the performance per power efficiency is 5.1 times better, as for the performance per area efficiency, it is 8.0 times better, and as for the performance per memory, it is 8.2 times better. We compare the proposed FPGA design with the CPU and the GPU designs. Compared with the ARM Cortex-A57, it was 1776.3 times faster, it dissipated 3.0 times lower power, and its performance per power efficiency was 5706.3 times better. Also, compared with the Maxwell GPU, it was 11.5 times faster, it dissipated 7.3 times lower power, and its performance per power efficiency was 83.0 times better. The disadvantage of our FPGA based design requires additional time to synthesize the FPGA executable codes. From the experiment, it consumed more three hours, and the total FPGA design took 75 hours. Since the training of the CNN is dominant, it is considerable.

***key words:*** *machine learning, deep learning, pruning, FPGA*

## 1. Introduction

### 1.1 Convolutional deep Neural Network (CNN)

Convolutional deep Neural Networks (CNN) are the current state-of-the-art for many embedded computer vision tasks, such as a hand-written character recognition [13], a face detector [27], a scene determination [24], and an object recognition [7]. They outperform conventional methods for accuracy at the 2012 ILSVRC ImageNet recognition challenge. New networks and their implementations are reported each week. Since applications require more accuracy, modern CNNs contain millions of floating-point parameters and need billions of floating-point operations to recognize a single image. Furthermore, recent CNNs tends to be large by AI researchers. Consequently, the training and inference of the CNNs are almost exclusively done on large clusters of GPUs. However, the GPU platform consumes much power than the CPU and the FPGA. It cannot be used in the embedded system, which requires low power consumption. Also, since the modern CNN needs many operations per an image, the embedded CPU is too slow.

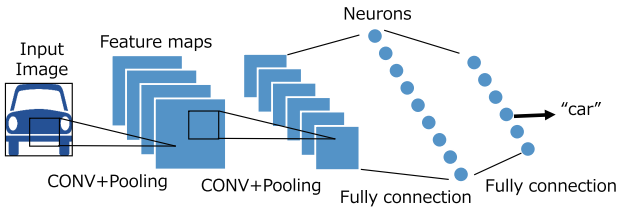


**Fig. 1** Typical convolutional deep neural network (CNN).

### 1.2 Binarized CNN for an FPGA

Recent work by Microsoft showed that the FPGA based deep learning leads us the performance per power efficiency. Furthermore, the low-precision CNNs have been demonstrated [5], [26], [38] which uses only one- or two- bit quantization strategy to reduce the hardware size with considerable accuracy. It is suitable for the FPGA implementations. Since the FPGA consists of a fine-grain element, which is a single-bit LUT (Look-Up Table), it can realize a custom circuit for such a low-precision circuit. On the other hand, the GPU consists of a coarse-grain element, which is configured as 8, 16, 32, and 64-bit arithmetic element. Although it can emulate a low-precision CNN, it cannot efficiently realize it. Recently, a flexible heterogeneous streaming binarized architecture [31], and a variable-width line buffer implementation [36] have been reported. The evaluation results [22] by Intel indicated that the binarized CNN could deliver orders of magnitude improvements in performance and performance/watt over well-optimized software on CPU and GPU. Although FPGA is less efficient than ASIC, the

Manuscript received July 27, 2018.
Manuscript revised December 3, 2018.
Manuscript publicized February 27, 2019.

[†]The authors are with Department of Information and Communications Engineering, Tokyo Institute of Technology, Tokyo, 152–8552 Japan.
a) E-mail: nakahara@ict.e.titech.ac.jp
b) E-mail: yonekawa@reconf.ict.e.titech.ac.jp
c) E-mail: fujii@reconf.ict.e.titech.ac.jp
d) E-mail: shimoda@reconf.ict.e.titech.ac.jp
e) E-mail: satos@eda.ict.e.titech.ac.jp

DOI: 10.1587/transinf.2018RCP0002

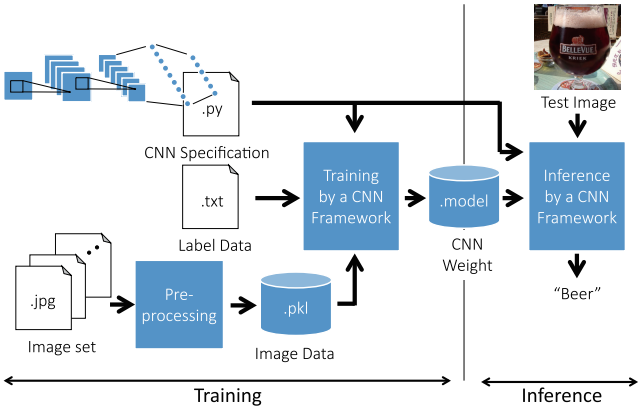


**Fig. 2** Typical CNN design flow using a GPU.

FPGA-ASIC gap may be reduced for designs that heavily utilize hard blocks. Hence, FPGA offers an attractive solution, which deliver superior efficiency improvements over software, without having to lock into a fixed ASIC solution.

The CNN consists of the convolutional layers, pooling layers, and the fully connected layers. The existing works [25], [36] analyzed the profile of the CNN. Previous works indicated that in the convolutional layers are bounded by computations, while the FC layers are bounded by memory access. Although the binarized convolutional layer was accelerated by many of XNOR gates (binarized multipliers), the memory access bottleneck for the FC layers still remain. One challenge to speed up the binarized CNN is eliminate the FC layers. The known method for that is the pruning technique for the FC layer [11]. Even if such procedures, the FC layers still exist. In the paper, we use a binarized average pooling layer instead of internal FC layers.

### 1.3 Known Problem for Software Programers

Figure 2 shows a typical CNN design flow using a GPU. First, the software designers prepare the training data consisting of image set and its labels. Typically, to improve the recognition accuracy, it applies the pre-processing to raw image set. This version supports rotation, flipping, color exchange operations. Also, they write the target CNN structure by Python code. Second, they train the CNN using the deep learning framework on the GPU to accelerate the training time. Then, the CNN weights are generated. Finally, they perform the inference by using the trained weight and input test image.

There are training and inference phases to realize the deep learning on the FPGA In the paper, we assume that the training is done by the GPU, while the FPGA only does the inference. Unfortunately, the software designer must convert the Python code for the CNN into the hardware description language (e.g. Verilog-HDL). Recently, to solve the description gap between the HDL code and the Python one, a high-level synthesis (HLS) has been released. However, even if they can be familiar with the HLS, the refactoring is still necessary to bring out the potential of the FPGA. It means that it requires much design time compared with the GPU based design. To solve this problem, we also propose a GUI based framework which warps the deep learning design tools and the FPGA ones.

There are FPGA based deep learning frameworks from academia [6], [8], [9], [29], [33]. Only our framework supports the binarized CNN to reduce the amount of hardware further. The software designer can realize the deep learning on the FPGA with accessing the API which controls the hardware. Also, in the paper, we modify the training algorithm for the hardware realization.

### 1.4 Contributions of the Paper

1. We developed an open-source GUI based binarized deep learning framework which supports both the training on the GPU and the inference on an FPGA. As far as we know, the tool is the first GUI-based framework of the binarized CNN for an FPGA. Since our GUI requires no knowledge of the hardware design, the software designer can quickly realize the high performance per area CNN on the FPGA. We opened the GUINNESS (GUI based Neural Network Synthesizer) framework for a binary deep neural network toward FPGA implementation as an open source. It is available for every designer at [19].
2. We modified the training algorithm both the training and the inference for the binarized CNN hardware. Since the hardware has a limited number of bit precision, it lacks the minimal bias. Also, for the inference on the hardware, the conventional batch normalization technique requires additional hardware. Our modifications solved these problems.
3. We implemented the VGG-11 benchmark CNN on the Digilent Inc. Zedboard. Compared with the conventional binarized implementations on an FPGA, the classification accuracy was almost the same, the performance per power efficiency is 5.1 times better, as for the performance per area efficiency, it is 8.0 times better, and as for the performance per memory, it is 8.2 times better.
4. We compared the proposed FPGA design with the CPU and the GPU designs. Compared with the ARM Cortex-A57, it was 1776.3 times faster, it dissipated 3.0 times lower power, and its performance per power efficiency was 5706.3 times better. Also, compared with the Maxwell GPU, it was 11.5 times faster, it dissipated 7.3 times lower power, and its performance per power efficiency was 83.0 times better. The disadvantage of our FPGA based design requires additional time to synthesize the FPGA executable codes. From the experiment, it consumed more three hours, and the total FPGA design took 75 hours. Since the training of the CNN is dominant, it is considerable.

This paper is built on past publications [20], [34].

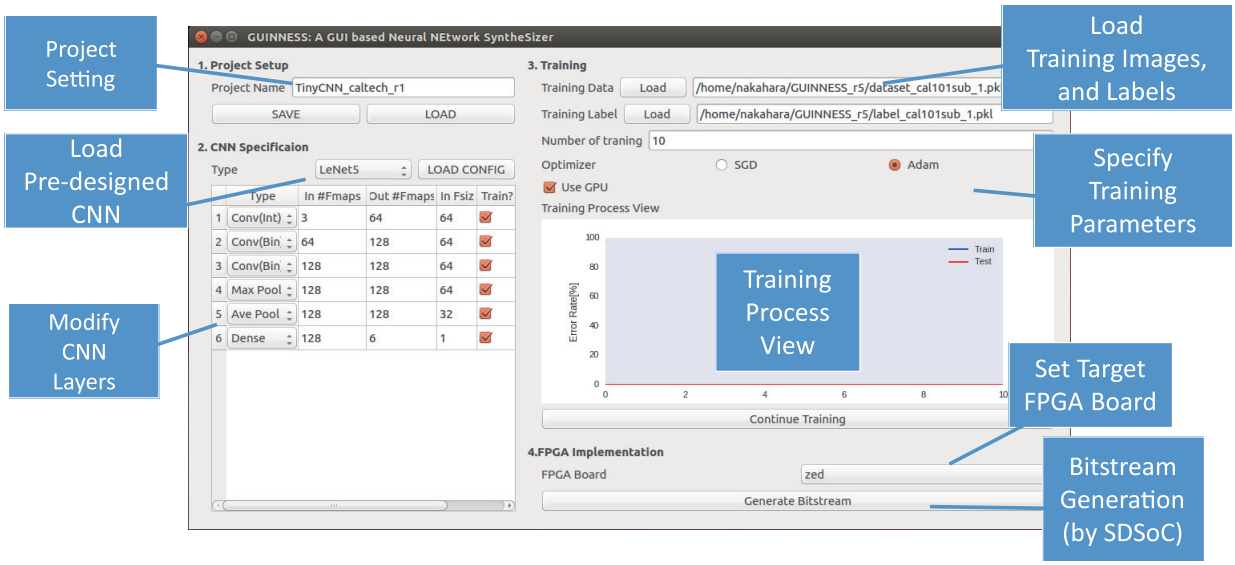


**Fig. 3** The GUINNESS (GUI based binarized deep neural network synthesizer) GUI.

### 1.5 Organization of the Paper

The rest of the paper is organized as follows: Chapter 2 introduces our GUI based design flow; Chapter 3 shows the training algorithms used in the framework; Chapter 4 shows the architecture for the binarized CNN; Chapter 5 shows the experimental results; Chapter 6 concludes the paper.

## 2. Related Work

For FPGA realizations of a CNN, many implementations have been reported including a hand-written character recognition [13], a face detector [27], a scene determination [24], an object recognition [7], and speech recognition [10] are implemented. The recursive residue number system (NRNS) based CNN has been proposed [21]. The restricted Boltzmann machine (RBM) based CNN has been proposed [14] Qiu et al. analyzed the hardware profile for the CNN implementation, and they showed that in the convolutional operation part, the MAC operation was a bottleneck, while in the full connection part, the memory access was bottleneck [25]. To reduce the design time for the CNN, many tools/frameworks from academia/industry have been released [6], [8], [9], [16]–[18], [29], [33], [36]. Also, the OpenCL based CNN on an FPGA is an attractive solution for the software programmer who does not know the hardware [1], [35]. To further reduce the hardware, binary CNNs [20], [22], [31], [34], [36] have been implemented on an FPGA.

## 3. Our GUI Based Design Tool Flow

Figure 3 shows the developed GUI written in PythonQt4, which warps design tools for a deep learning training and an FPGA design. Our GUINNESS framework supports the binarized CNN for the FPGA realization. It consists of a project setting, a CNN specification, a training, and an FPGA synthesis. First, software designers specify the project directory. Second, they specify the CNN parameters including each layer type (convolution, max pooling, average pooling, and dense), the number of feature maps, feature map size, and do/not training. The GUI prepares the pre-designed CNNs such as, the TensorFlow tutorial CNN, the VGG-11, the VGG-16, and LeNet-5, thus, the designers must not specify all the CNN. Third, they specify the training parameters consisting of the training image dataset, the number of training (epochs), the optimization algorithm (SGD, Adam), use/not GPU. Also, the GUI provides the graphical view of the training process. Thus, the users can easily understand the training process. Finally, they set the target FPGA board, since the GUI calls the Xilinx SDSoC which focuses on the support FPGA board. After pushing the "Generate Bitstream" button, the GUI generates the C++ code including pragmas to realize a high performance per area CNN circuit. Then, it synthesizes the CNN circuit. At a time, it converts the pre-trained weight into a text file to be loaded into an on-chip memory of an FPGA.

Figure 4 shows a proposed FPGA design flow, which is performed by the GUINNESS GUI. First, after the end of specifying parameters of the training, the GUI generates the Python scripts to determine the CNN and its training. Then, it calls the Chainer to train the CNN on the GPU, and it produces the binarized CNN weight. Second, after the end of a set of FPGA board and its clock frequency, it converts from the Python scripts into C++ codes consisting of the processing system (PS) code and the programmable logic (PL) one. The PL code is synthesized by the HLS to generate the bitstream to realize the CNN accelerator, while the PS code is compiled by the gcc to control the PL and supply the API for the software designer. Also, after finish the configuration, the binarized CNN is realized, and it loads the con-

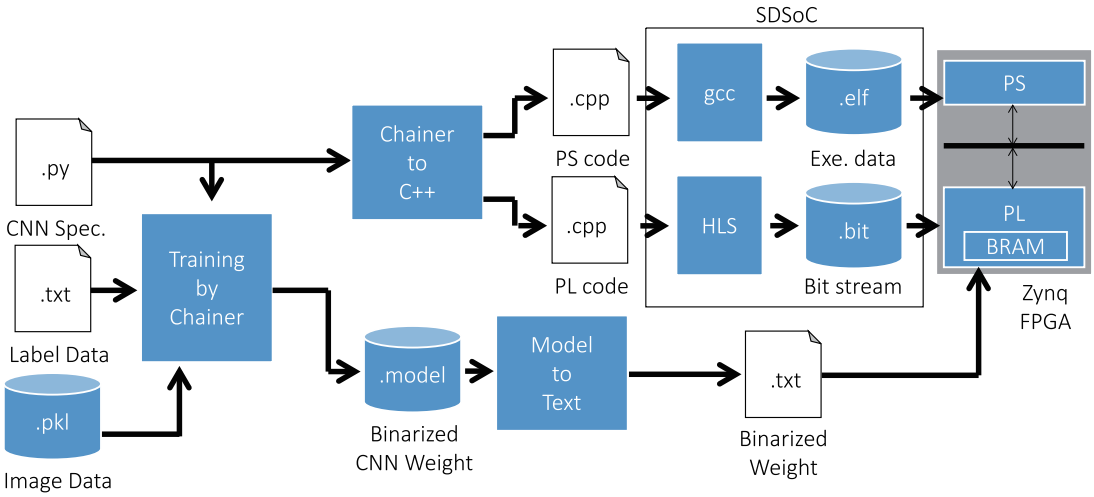


**Fig. 4** Proposed FPGA design flow.

verted weight file to perform the inference. In this version, we use the Xilinx SDSoC to generate these executable codes and the operating system. Thus, the software designer can realize the binarized CNN on the FPGA without no scripts, and can easily access its hardware through the API on the PS.

The disadvantage of the proposed FPGA tool flow is to consume additional design time to synthesize the FPGA executable code compared with the GPU based realization. However, compared with the CNN training time, the FPGA design time is shorter. Thus, its disadvantage can be ignored.

## 4. Training Algorithms Used in the GUINNESS

For both the inference and training, the CNN may take minimum value of the bias. Thus, it is not suitable for the low precision hardware realization. In the worst case, since it takes difference activation, the recognition result tends to be changed. To solve the problem, we modify the training algorithm. First, we explain the batch normalization, which is an essential technique for the binarized CNN.

### 4.1 Batch Normalization

Typically, to accelerate training time and convergence, a set of training data (mini-batch) is back-propagated and it updates weights at a time. It is called by mini-batch training. In this case, since the impact on the difference in the distribution of data for each batch (internal covariate shift), the convergence of the training tends to be slow, and the trainer must carefully determine the initial value of the parameters. These problems are solved by **batch normalization (BN)** [28] which corrects the difference in the distribution by shift and scaling operations.

At the training, the BN finds parameters $\gamma$ and $\beta$ to regularize the variance to 1 and the average to 0 for each mini-batch. The BN algorithm for training is shown as follows:

**Algorithm 4.1:** Input: Mini-batch $B = \{X_1, X_2, \ldots, X_m\}$ Output: Parameters $\gamma$ and $\beta$.

1. Obtain average for each mini-batch: $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} X_i$.
2. Obtain variance for each mini-batch: $\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu_B)^2$.
3. Perform normalization: $\hat{X}_i \leftarrow \frac{X_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$.
4. Obtain $\gamma$ and $\beta$, that regularizes the variance to 1 and the average to 0 for $B_i \leftarrow \gamma\hat{X}_i + \beta$.
5. Terminate.

Above Algorithm performs the normalization for each mini-batch. A hyperparameter $\epsilon$ is set for a coefficient stabilization, which is used to adjust the training time. Since both $\gamma$ and $\beta$ have been already trained during classification, the CNN with BN reads them from the off-chip memory. Thus, the operation for the CNN with BN is introduced as follows:

$$
\begin{aligned}
Y &= \sum_{i=0}^{n} w_i x_i, \\
B &= \gamma Y + \beta, \\
z &= f_{sgn}(B),
\end{aligned}
$$

where $w_i$, $x_i$, $z$ are binary variables, while $Y$, $B$, $\gamma$, and $\beta$ are integer ones. Above expression shows that the CNN with BN requires additional cost for the multiplier and the adder for area, while the memory access for $\gamma$ and $\beta$.

### 4.2 Bias Elimination for CNN Training

As shown in Algorithm 4.1, the BN normalizes an internal variables $Y$. Let $Y'$ be the output of the BN operation. Then, we have

$$
Y' = \gamma\frac{Y - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta
$$

$$= \frac{\gamma}{\sqrt{\sigma_B^2+\epsilon}}\left(Y - \left(\mu_B - \frac{\sqrt{\sigma_B^2+\epsilon}}{\gamma}\beta\right)\right).$$

From above expression, the signed activation function becomes

$$f'_{sgn}(Y) = \begin{cases} 1 & \left(if\ Y < -\mu_B + \frac{\sqrt{\sigma_B^2+\epsilon}}{\gamma}\beta\right) \\ -1 & (otherwise). \end{cases}$$

That is, the value of the active function is determined by the value of the above equation. In this case, since $x_0 = 1$, $Y$ can be equivalent to the following expression:

$$\begin{aligned} Y &= \sum_{i=0}^{n} w_i x_i - \mu_B + \frac{\sqrt{\sigma_B^2+\epsilon}}{\gamma}\beta \\ &= \sum_{i=1}^{n} w_i x_i + \left(w_0 - \mu_B + \frac{\sqrt{\sigma_B^2+\epsilon}}{\gamma}\beta\right). \end{aligned} \quad (1)$$

Thus, for the CNN training, we assume that the original $\mu$ is $w_0 - \mu_B$, then, the bias $w_0$ is merged into $\mu$. The GUINNESS generates the Python script which does not include bias parameters. As a result, we can avoid the bias lost problem even if we use a low precision bias.

### 4.3 Batch Normalization Free CNN Inference

Although the BN is an essential technique for the binarized CNN, it requires an additional area for the multiplier and the adder, and the memory access for parameters. In this paper, we introduce **a batch normalization free binarized CNN** which has an equivalence to the BN operation. Since it does not requires additional area, the hardware becomes more straightforward than the one including BN operation.

From Expr. (1), we have the following:

$$\begin{aligned} Y &= \sum_{i=1}^{n} w_i x_i + \left(w_0 - \mu_B + \frac{\sqrt{\sigma_B^2+\epsilon}}{\gamma}\beta\right) \\ &= \sum_{i=1}^{n} w_i x_i + W' \\ &\simeq \sum_{i=1}^{n} w_i x_i + \lceil W' \rceil \end{aligned} \quad (2)$$

The binarized CNN with the BN is converted into the EXNOR-MAC with an integer bias $\lceil W' \rceil$. Since the activation function adopted in our framework is a sign function, its calculation result is always binary value (positive or negative one). Since the input $x_i$ and the weight $w_i$ are binary, the result of the convolution is integer precision. Since it is sufficient to decide positive or negative one, even if $W'$ rounded up to the integer $\lceil W' \rceil$ is used, the same calculation result is accomplished. Therefore, it is possible to replace the BN with integer precision bias. In this paper, we call this the BN free binarized CNN. This expression approximates the BN operation by a shifting one. In the circuit, since the BN operation is replaced into the integer bias, the hardware becomes simpler, and the memory access is reduced. Therefore, the BN free binarized CNN is faster and smaller than conventional ones.

## 5. Binarized CNN Architecture for the FPGA

The proposed GUINNESS framework supports the binarized CNN, which is suitable for the FPGA realization. We tuned the binarized CNN by removing internal full connection layers without affecting the overall accuracy. Then, they are replaced by an implementation-friendly binarized average pooling layer. Figure 5 shows our binarized CNN.

### 5.1 Binarized Average Pooling

In the CNN, almost parameters are focused on the FC layers. To remove them, we replace the internal FC layers into an average pooling one. Previous works (Network-in-network [15] and GoogLeNet [30]) directly connected the output of the convolutional layer to the softmax layer. It implies that the inference can be correctly performed without FC layers. Figure 6 shows that the binarized average pooling circuit. Since the binarized average pooling is equivalent to the 1's counter and the threshold detector, which can be realized by just outputting the MSB of the 1's counter. Thus, the binarized average pooling circuit can be implemented by a simple circuit.

### 5.2 Binarized CNN Using an Average Pooling

For each $L \times L$ feature map of the last layer of the convolutional ones, it can be replaced into a binarized averaging pooling with a $L \times L$ kernel. We attach a FC layer to the output of the 1 dimensional neurons generated by the average pooling layer. This is equivalent to eliminate the internal FC

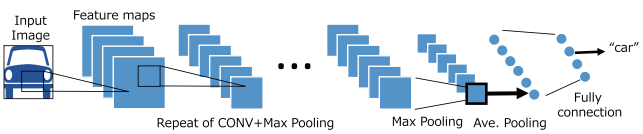


**Fig. 5** Proposed binarized CNN using an average pooling layer.

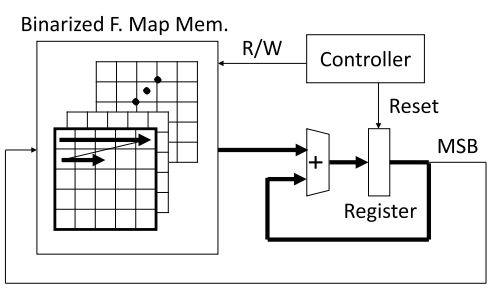


**Fig. 6** Binarized average pooling circuit.

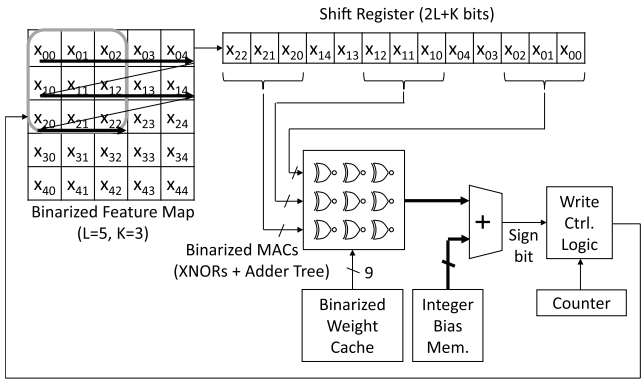


**Fig. 7** Streaming binarized 2D convolutional circuit.

layers and replace with the average pooling one. The reason to remain the last FC layer is to map the output neurons for classification.

### 5.3 Shared XNOR-MAC Circuit with Streaming Operation

Although we used the binarized MAC operation instead of the floating-point one, it consumes much hardware to realize the entirely parallel XNOR-MAC operation. Since the typical CNN has different number of feature maps in the layer, a heterogeneous streaming architecture requires many LUTs for a large size of XNOR operations.

In the paper, to realize the high-performance with less hardware, we proposed a shared XNOR-MAC circuit supporting a streaming operation as shown in Fig. 7. To reduce the memory access, we use the shift register to make a streaming data flow from the memory for the feature map. Also, it share the different size of XNOR-MAC circuit into a single bitwise XNOR circuit followed by adder-trees, bias adder, and a write controller. The circuit reads the corresponding inputs from the shift register, then it applies to the bitwise binarized MAC operation. Next, it adds the precomputed bias, which is obtained by both the pre-trained bias and the batch normalization value. Since the kernel crosses the boundary of the feature map, we attach the write control logic to the output of the circuit.

## 6. Experimental Results

### 6.1 FPGA Implementation

We implemented the binarized CNN for the VGG-11 on the Xilinx Inc. Zedboard, which has the Xilinx Zynq FPGA (XC7Z020, 53,200 LUTs, 106,400 FFs, 280 18Kb BRAMs, 220 DSP48Es). We used the Xilinx Inc. SDSoC 2016.4 to generate the bitstream with timing constraint 143.78 MHz. Our implementation used 18,325 LUTs, 19,913 FFs, 32 18Kb BRAMs, and a DSP48E. Also, it satisfied the necessary timing constraint. As for the number of operations for the implemented CNN, since it performed $3 \times 3$ MAC operations for 256 feature maps at 143 MHz, it was 329.47 GOPS (Giga Operations Per Seconds). As for the memory bandwidth, for 256 feature maps, since it reads $3 \times 3$ binarized weights, $3 \times 3$ binarized inputs, and send a binarized output at a time, it was 86.9 GB per second. We measured the total board power consumption, which was 2.3 Watt. Since the implemented CNN operated 329.47 GOPS with 11,986 Slices, the area efficiency was 274.8 (GOPS/Slice$\times 10^{-4}$). Also, the performance per power efficiency was 143.25 (GOPS/W).

**Table 1** Comparison with other low bit precision CNN realizations on the FPGA.

| Implementation | Zhao et al. [36] | FINN [31] | Ours |
|---|---|---|---|
| FPGA Board (FPGA) | Zedboard (XC7Z020) | PYNQ board (XC7Z020) | Zedboard (XC7Z020) |
| Clock (MHz) | 143 | 166 | 143 |
| # LUTs | 46900 | 42823 | 14509 |
| # 18Kb BRAMs | 94 | 270 | 32 |
| # DSP Blocks | 3 | 32 | 1 |
| Test Error | 12.27% | 19.9% | 18.2% |
| Time [msec] | 5.94 | 2.24 | 2.37 |
| (FPS) | (168) | (445) | (420) |
| Power [W] | 4.7 | 2.5 | 2.3 |
| FPS/Watt | 35.7 | 178.0 | **182.6** |
| FPS/LUT | $35.8\times10^{-4}$ | $103.9\times10^{-4}$ | **289.4**$\times10^{-4}$ |
| FPS/BRAM | 1.8 | 1.6 | **13.1** |

### 6.2 Compared with Conventional Binarized CNN Implementations

Table 1 compares binarized CNN implementations on the same FPGA. From Table 1, compared with Zhao's implementation, the classification accuracy was almost the same, as for the performance per power efficiency (FPS/Watt), it is 5.1 times better, as for the performance per area efficiency (FPS/LUT), it is 8.0 times better, and as for the performance per memory (FPS/BRAM), it is 7.3 times better. Compared with the FINN, the classification accuracy was almost the same, the performance per power efficiency is almost the same, as for the performance per area efficiency, it is 2.8 times better, and as for the performance per memory, it is 8.2 times better.

### 6.3 Comparison with other Embedded Platforms

We compared our binarized CNN with other embedded platforms. We used the NVIDIA Jetson TX1 board which has both the embedded CPU (ARM Cortex-A57) and the embedded GPU (Maxwell GPU). Following the benchmarking [23], the CPU and GPU run the VGG11 using Caffe [2] version 0.14. Also, we measured the total power consumption. Note that, in the experiment, to measure the latency, we set the number of batch size to one.

Table 2 compares our FPGA implementation with other platforms. Compared with the ARM Cortex-A57, it was 1776.3 times faster, it dissipated 3.0 times lower power, and its performance per power efficiency was 5706.3 times better. Also, compared with the Maxwell GPU, it was 11.5

**Table 2** Comparison with embedded platforms with respect to the VGG11 forwarding (Batch size is 1).

| Platform | Embedded CPU | Embedded GPU | FPGA |
|---|---|---|---|
| Device | ARM Cortex-A57 | Maxwell GPU | Zynq7020 |
| Clock Freq. | 1.9 GHz | 998 MHz | 143.78 MHz |
| Memory | 16GB eMMC Flash | 4GB LPDDR4 | 4.9 Mb BRAM |
| Time [msec] | 4210.0 | 27.23 | **2.37** |
| (FPS) [$s^{-1}$] | (0.23) | (36.7) | **(421.9)** |
| Power [W] | 7 | 17 | 2.3 |
| Efficiency [FPS/W] | 0.032 | 2.2 | **182.6** |
| Design Time [Hours] | 72 | 72 | 75 |

times faster, it dissipated 7.3 times lower power, and its performance per power efficiency was 83.0 times better. Thus, the binarized CNN on the FPGA is suitable for the embedded system.

Also, we compared the design time the FPGA implementation with other ones. For the training, three days were necessary to archive more than 80% classification accuracy using the GPU, so both the CPU and the GPU consumed 72 hours to implement. Additionally, the FPGA design required additional design time to synthesis generated C++ codes. Since it consumed three hours totally, the FPGA design took 75 hours. Although the FPGA design requires more design times, it is considerable.

## 7. Conclusion

In the paper, we showed the GUINNESS framework for a binarized deep neural network toward FPGA implementation. In this framework, the training is done on the GPU, while the inference is realized on the FPGA. Since all the operation is done on the GUI, the software designer must not write any scripts/codes to design the neural network structure, training behavior, only specify the values for hyperparameters. Thus, our tool flow is suitable for the software programmers who are not familiar with the FPGA design. In our tool flow, we modified the training algorithm both the training and the inference for the binarized CNN hardware. For the training, the minimal bias is merged into another parameter for the batch normalization. For the inference, the additional operations for the batch normalization is concentrated into the bias addition. In the experiment, we implemented the VGG-11 benchmark CNN on the Digilent Inc. Zedboard. Compared with the conventional binarized implementations on an FPGA, the classification accuracy was almost the same, the performance per power efficiency is 5.1 times better, as for the performance per area efficiency, it is 8.0 times better, and as for the performance per memory, it is 8.2 times better. We compared the proposed FPGA design with the CPU and the GPU designs. Compared with the ARM Cortex-A57, it was 1776.3 times faster, it dissipated 3.0 times lower power, and its performance per power efficiency was 5706.3 times better. Also, compared with the Maxwell GPU, it was 11.5 times faster, it dissipated 7.3 times lower power, and its performance per power efficiency was 83.0 times better. The disadvantage of our FPGA based design required additional time to synthesize the FPGA executable codes, which was more three hours. However, since the training of the CNN is dominant, it is considerable.

The open-source GUINNESS framework [19] is available for every designer.

## Acknowledgements

This research is supported in part by the Grants in Aid for Scientific Research from JSPS, and the New Energy and Industrial Technology Development Organization (NEDO). In addition, thanks to the Xilinx Inc. University Program (XUP), and the support by the NVIDIA Corporation. Reviewer's comments are improved the paper.

### References

[1] U. Aydonat, S. O'Connell, D. Capalija, A.C. Ling, and G.R. Chiu, "An OpenCL deep learning accelerator on Arria10," FPGA, pp.55–64, 2017.
[2] Caffe, Deep learning framework, http://caffe.berkeleyvision.org/
[3] Chainer, A powerful, flexible, and intuitive framework of neural networks, http://chainer.org/
[4] The CIFAR-10 data set, http://www.cs.toronto.edu/~kriz/cifar.html
[5] M. Courbariaux, I. Hubara, D. Soudry, R.E. Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," Computer Research Repository (CoRR), March 2016, http://arxiv.org/pdf/1602.02830v3.pdf
[6] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi, "Caffeinated FPGAs: FPGA Framework For Convolutional Neural Networks," ICCAD, pp.1–4, 2016.
[7] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," Int'l Symp. on Circuits and Systems (ISCAS), pp.257–260, 2010.
[8] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates," FCCM, pp.152–159, 2017.
[9] K. Guo, L. Sui, J. Qiu, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-Eye: A Complete Design Flow for Mapping CNN onto Customized Hardware," ISVLSI, pp.24–29, 2016.
[10] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang, H. Yang, and W.J. Dally, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," FPGA, pp.75–84, 2017.
[11] S. Han, H. Mao, and W.J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," ICLR, 2016.
[12] M. Kim and P. Smaragdis, "Bitwise neural networks," CoRR, abs/1601.06071, 2016.
[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol.86, no.11, pp.2278–2324, 1998.
[14] B. Li, M.H. Najafi, and D.J. Lilja, "Using stochastic computing to reduce the hardware requirements for a restricted boltzmann machine classifier," FPGA, pp.36–41, 2016.
[15] M. Lin, Q. Chen, and S. Yan, "Network in Network," https://arxiv.org/abs/1312.4400
[16] Z. Liu, Y. Dou, J. Jiang, and J. Xu, "Automatic code generation of convolutional neural networks in FPGA implementation," FPT, pp.61–68, 2016.

[17] Y. Ma, Y. Cao, S.B.K. Vrudhula, and J.-S. Seo, "An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks," FPL, pp.1–8, 2017.
[18] Y. Ma, N. Suda, Y. Cao, J.-S. Seo, and S.B.K. Vrudhula, "Scalable and modularized RTL compilation of Convolutional Neural Networks onto FPGA," FPL, pp.1–8, 2016.
[19] H. Nakahara, H. Yonekawa, T. Fujii, M. Shimoda, and S. Sato, "GUINNESS: AGUI based neural network synthesizer for an FPGA," https://github.com/HirokiNakahara/GUINNESS/
[20] H. Nakahara, T. Fujii, and S. Sato, "A fully connected layer elimination for a binary convolutional neural network on an FPGA," FPL pp.1–4, 2017.
[21] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," FPL, pp.1–6, 2015.
[22] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC," FPT, pp.77–84, 2016.
[23] https://github.com/charlyng/Embedded-Deep-Learning/tree/master/Benchmark-Performance
[24] M. Peemen, A.A.A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," Int'l Conf. on Computer Design (ICCD), pp.13–19, 2013.
[25] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural network," ISFPGA, pp.26–35, 2016.
[26] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," https://arxiv.org/pdf/1603.05279.pdf
[27] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H.P. Graf, "A massively parallel coprocessor for convolutional neural networks," Int'l Conf. on Application-specific Systems, Architectures and Processors (ASAP), pp.53–60, 2009.
[28] I. Sergey and S. Christian, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.
[29] H. Sharma, J. Park, E. Amaro, B. Thwaites, P. Kotha, A. Gupta, J.K. Kim, A. Mishra, and H. Esmaeilzadeh, "DNNWEAVER: From High-Level Deep Network Models to FPGA Acceleration," MICRO, pp.1–6, 2016.
[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," Computer Vision and Pattern Recognition (CVPR), pp.1–9, 2015.
[31] Y. Umuroglu, N.J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," ISFPGA, pp.65–74, 2017.
[32] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," ICLR, 2015.
[33] S.I. Venieris and C.-S. Bouganis, "fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs," FCCM, pp.40–47, 2016.
[34] H. Yonekawa and H. Nakahara, "On-Chip Memory Based Binarized Convolutional Deep Neural Network Applying Batch Normalization Free Technique on an FPGA," IPDPS Workshops, pp.98–105, 2017.
[35] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," FPGA, pp.25–34, 2017.
[36] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," ISFPGA, pp.15–24, 2017.
[37] W. Zhao, H. Fu, W. Luk, T. Yu, S. Wang, B. Feng, Y. Ma, and G. Yang, "F-CNN: An FPGA-based framework for training Convolutional Neural Networks," ASAP, pp.107–114, 2016.
[38] 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," http://arxiv.org/pdf/1606.06160v2.pdf

## Appendix A: Expansion of the GUINNESS

The current version of the GUINNESS (last updated 2nd, June, 2018) can be expanded to a novel CNN and its hardware realization. The GUINNESS consists of mainly the CNN training part and its inference hardware one. As for the training, the user can modify the "function_binary_conv2d.py" script to define a new precision or/and convolution operation, and the "LoadConfig" function in "guinness.py" to define a new pre-defined CNN. Also, to revise a CNN inference hardware, the user can improve the "gen_cpp_code_v3.py" script for the FPGA implementation.

## Appendix B: Tutorial Document of the GUINNESS

The document for the user is uploated to the followings:

"The GUINNESS introduction and BCNN implementation on an FPGA,"
https://www.dropbox.com/s/oe6gptgyi4y92el/guinness_tutorial1_v2.pdf

## Appendix C: Requirements for the Current Version

The following software and libraries are required to execute the GUINNESS.

1. Ubuntu 16.04 LTS (14.04 LTS is also supported)
2. Python 3.5.1 (Note that, my recommendation is to install by Anaconda 4.1.0 (64bit)+Pyenv
3. Python libraries: PyQt4, matplotlib, OpenCV3, numpy, and scipy
4. CUDA 8.0 (with GPU), CuDNN 6.0. This is opptional, however, we strongly recommend to use the GPU within a practical training time.
5. Chainer 1.24.0 (more than version 2.0 is not supported) + CuPy 2.0
6. Xilinx Inc. SDSoC 2017.4, and 2018.2

Additionally, for the FPGA boards, the following ones are supported.

1. Xilinx Inc. ZC702, ZC706, ZCU102, and ZCU104
2. Digilent Inc. Zedboard, and Zybo

**Hiroki Nakahara** received the B.E., M.E., and Ph.D. degrees in computer science from Kyushu Institute of Technology, Fukuoka, Japan, in 2003, 2005, and 2007, respectively. He has held research/faculty positions at Kyushu Institute of Technology, Iizuka, Japan, Kagoshima University, Kagoshima, Japan, and Ehime University, Ehime, Japan. Now, he is an associate professor at Tokyo Institute of Technology, Japan. He was the Workshop Chairman for the International Workshop on Post-Binary ULSI Systems (ULSIWS) in 2014, 2015, 2016 and 2017, respectively. He searved the Program Chairman for the International Symposium on 8th Highly-Efficient Accelerators and Reconfigurable Technologies (HEART) in 2017. He received the 8th IEEE/ACM MEMOCODE Design Contest 1st Place Award in 2010, the SASIMI Outstanding Paper Award in 2010, IPSJ Yamashita SIG Research Award in 2011, the 11st FIT Funai Best Paper Award in 2012, the 7th IEEE MCSoC-13 Best Paper Award in 2013, and the ISMVL2013 Kenneth C. Smith Early Career Award in 2014, respectively. His research interests include logic synthesis, reconfigurable architecture, digital signal processing, embedded systems, and machine learning. He is a member of the IEEE, the ACM, and the IEICE.

**Haruyoshi Yonekawa** received the B.S. and M.S. degrees from Tokyo Institute of Technology (Tokyo Tech), in 2016, and 2018. His current research interests include reconfigurable architecture and deep learning. He received the 24th Reconfigurable Architectures Workshop Best Demo Award in 2017.

**Tomoya Fujii** received the B.S. and M.S. degrees from Tokyo Institute of Technology (Tokyo Tech), in 2016, and 2018. His current research interests include reconfigurable architecture and deep learning.

**Masayuki Shimoda** received the B.E. degree in engineering all from Tokyo Institute of Technology, Tokyo, Japan, in 2018. He is currently a Master Student with the Department of Information and Communications Engineering of Tokyo Institute of Technology. His current research interests include Deep Neural Network and FPGA. He is a member of IEEE.

**Shimpei Sato** received the B.S., M.S., and Ph.D. degrees in engineering from Tokyo Institute of Technology (Tokyo Tech), in 2007, 2009, and 2014. He is currently an Assistant Professor with the Department of Information and Communications Engineering of Tokyo Tech. From 2014 to 2016, he worked in High performance computing area as a post doctoral researcher, where he investigated an application performance analysis/tuning method. His current research interests include approximate computing realization by architecture design and circuit design and their applications.