







FPGA Adaptive Neural Network Quantization for Adversarial Image Attack Defense

Yufeng Lu, Xiaokang Shi , Jianan Jiang , Hanhui Deng , Yanwen Wang , *Member, IEEE*,
Jiwu Lu , *Member, IEEE*, and Di Wu , *Member, IEEE*

Abstract—Quantized neural networks (QNNs) have become a standard operation for efficiently deploying deep learning models on hardware platforms in real application scenarios. An empirical study on German traffic sign recognition benchmark (GTSRB) dataset shows that under the three white-box adversarial attacks of fast gradient sign method, random + fast gradient sign method and basic iterative method, the accuracy of the full quantization model was only 55%, much lower than that of the full precision model (73%). This indicates the adversarial robustness of the full quantization model is much worse than that of the full precision model. To improve the adversarial robustness of the full quantization model, we have designed an adversarial attack defense platform based on field-programmable gate array (FPGA) to jointly optimize the efficiency and robustness of QNNs. Various hardware-friendly techniques such as adversarial training and feature squeezing were studied and transferred to the FPGA platform based on the designed accelerator of QNN. Experiments on the GTSRB dataset show that the adversarial training embedded on FPGA can increase the model's average accuracy by 2.5% on clean data, 15% under white-box attacks, and 4% under black-box attacks, respectively, demonstrating our methodology can improve the robustness of the full quantization model under different adversarial attacks.

Manuscript received 14 February 2024; revised 28 May 2024 and 6 July 2024; accepted 28 July 2024. Date of publication 22 August 2024; date of current version 5 December 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 61972145, Grant 61932010, and Grant 62102139, in part by the Natural Science Foundation of Hunan Province of China under Grant 2022JJ30168 and Grant 2023JJ20015, and in part by the Fundamental Research Funds for the Central Universities. Paper no. TII-24-0708. (Yufeng Lu and Xiaokang Shi contributed equally to this work.) (Corresponding authors: Di Wu; Yanwen Wang; Jiwu Lu.)

Yufeng Lu and Jiwu Lu are with the National Engineering Research Center for Robot Visual Perception and Control Technology, and the College of Electrical and Information Engineering, Hunan University, Changsha, Hunan 410082, China (e-mail: baymaxlyf@hnu.edu.cn; jiwulu@hnu.edu.cn).

Xiaokang Shi and Yanwen Wang are with the College of Electrical and Information Engineering, Hunan University, Changsha 410082, China, and also with Shenzhen Research Institute, Hunan University, Shenzhen 518000, China (e-mail: shixiaokang2022@hnu.edu.cn; wangyw@hnu.edu.cn).

Jianan Jiang, Hanhui Deng, and Di Wu are with the National Engineering Research Center for Robot Visual Perception and Control Technology, and the School of Robotics, Hunan University, Changsha, Hunan 410082, China (e-mail: jiangjn22@hnu.edu.cn; denghanhui@hnu.edu.cn; dwu@hnu.edu.cn).

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/TII.2024.3438284>.

Digital Object Identifier 10.1109/TII.2024.3438284

Index Terms—Adversarial attack, field-programmable gate array (FPGA), quantized neural networks (QNNs).

I. INTRODUCTION

IN RECENT years, deep neural networks (DNNs) have been widely used in safe-critical environments, such as, automatic driving, face recognition, monitoring systems, and so on. As DNNs have demonstrated an impressive performance in these scenarios, their security has attracted much attention as well. Szegedy et al. [18] first found that DNNs with good performance were also vulnerable to adversarial attacks. These attacks use subtle perturbation on the input images, which may cause the deep learning model to give incorrect results. Such small perturbations are hard to be detected by the human eye, but they can fool the neural network easily [7]. Images with small perturbations used in adversarial attacks are often called adversarial examples. Adversarial examples have been proven to be applicable to various scenarios. For example, Morgulis et al. [14] presented a method to generate adversarial traffic signs that can fool a wide range of classifiers of DNNs, both open-source and production-grade in the real world. Liu et al. [23] developed a T-shirt that could deceive the recognition system so that pedestrians wearing the T-shirt would not be identified. Moreover, adversarial attacks are transferable, which means that they can even influence other models with different structures. Such imperceptible attacks with excellent transferability unveil the blind spots in neural networks and raise issues on reliability and security [29].

Meanwhile, with the advent of high-performance hardware, DNNs with deeper levels, larger model sizes, and more complex building blocks are proposed for more complex tasks. As DNNs' layers deepen, the parameters of models and the multiply-accumulation operations in the calculation process increase dramatically, leading to a high requirement for computation, memory, and power. This makes the deployment of DNNs extremely challenging, especially on low-power mobile devices. As one of the most popular model compression techniques, weight and activation quantization have been extensively studied to reduce the model size and the computational cost significantly. BinaryConnect [4] as the first binary CNN algorithm introduced the gradient shear, which can achieve better accuracy on the CIFAR-10 dataset. After that, MobileNet [9] introduced depthwise separable convolution to construct models with fewer parameters and reduced latency, aligning well with the design

specifications of mobile and embedded applications.

However, when deploying DNNs on the edge devices, billions or trillions of operations per second are required for real-time image classification. Although the latest graphics processing units (GPUs) can achieve this level of performance, their excessive size, price, and power/energy consumption are unsuitable for specific applications in embedded systems. While micro-controllers offer significant price advantages, their performance often poses challenges for running DNNs efficiently, often resulting in inefficient execution or outright inability to support such networks. Moreover, the absence of hardware acceleration support further exacerbates the difficulty in meeting real-time requirements, and they also significant challenges persist in terms of flexibility and scalability.

Therefore, future embedded DNNs need an energy-efficient yet still powerful computing platform [22]. Field-programmable gate arrays (FPGAs) present a promising solution, since multi-functional integrated circuits provide hundreds of thousands of programmable logic blocks and a configurable interconnect that enables custom accelerator architectures to be built in hardware. Moreover, FPGAs consume less power than GPUs. Although a single processing unit in a GPU may cost less energy than an FPGA, the total power that a GPU consumes may significantly higher than an FPGA since a GPU normally involves thousands of processing units when executing the same computation workload. Therefore, FPGAs have been widely used to implement the DNN hardware accelerator in recent years [26].

In this article, we have designed an FPGA-based defense platform for the efficiency and robustness of deploying DNNs, which can handle a variety of adversarial attacks. In general, the key contributions are as follows.

- 1) *Novel Quantization-Aware Training Method*: Our research bridges a gap of existing shortcomings in adversarial robustness for fully quantized FPGA-deployed neural networks. We propose a new quantization-aware training approach specifically tailored for fully quantized models deployed on FPGA platforms, effectively enhancing the model's robustness against adversarial attacks.
- 2) *Deployment of Quantitative Models on FPGA*: We have demonstrated how to effectively integrate fully quantized models into FPGA hardware and have designed an FPGA-based DNN accelerator. This not only improves the model's operational efficiency but also reduces energy consumption, showing the potential of quantized models for hardware acceleration.
- 3) *Evaluation of Model's Adversarial Robustness*: Through extensive experiments, we evaluate the performance of the model under various types of adversarial attacks. The results indicate that our approach significantly enhances the accuracy of the model under these attacks, confirming the effectiveness of our training strategy and hardware design.

II. BACKGROUND AND RELATED WORKS

In this section, we will introduce the background of our work. Adversarial attacks, such as, blurred or distorted traffic signs

in autonomous driving that occur most commonly in practice, significantly weaken the recognition ability of models. Therefore, researchers proposed corresponding defense methods to improve the robustness of deep learning models by minimizing the impact of such adversarial attacks. These approaches can be implemented on the energy-efficient and flexible FPGA platform to demonstrate their practicability. Therefore, the background of our work focuses on these three aspects as follows.

A. Adversarial Examples and Attacks

Methods like the fast gradient sign method (FGSM) [7], random + fast gradient sign method (R+FGSM) [19], and basic iterative method (BIM) [10] have been devised to enhance model robustness by generating adversarial examples. In recent years, research on adversarial attacks has expanded beyond traditional image recognition to encompass diverse application domains such as, physical-world scenarios [20], large language models [28], and multiagent systems [12]. Adversarial attacks can be broadly categorized into white-box attacks [7] and black-box attacks [15]. In white-box attacks, the attackers possess complete information about the target model, enabling them to generate precise adversarial examples. Conversely, black-box attacks only afford attackers access to the inputs and outputs of the target model, without knowledge of its internal structure or parameters. Consequently, attackers typically resort to designing and training their own models to generate adversarial examples.

B. Defense Methods

The current defense methods either preprocess the adversarial examples to denoise the perturbation or make the model itself robust. For the first solution, feature squeezing [3], [24] was proposed to detect adversarial images by squeezing the input image. Since a large color bit width is often not necessary to interpret the image (for example, people have no problem recognizing most black-and-white images), it assumes that reducing the color bit depth of the input image can reduce adversarial opportunity. For the second solution, adversarial training [7], [10], [16], [19], [21] is considered to be the most effective method to mitigate the damage of adversarial attacks and improve the model robustness. By adding the adversarial examples with true labels into the training set, the trained model will correctly predict the label of future adversarial examples. By training on benign samples augmented with adversarial examples, adversarial training increases the model's robustness against adversarial examples.

C. FPGA-Based DNN Accelerators

Parallel computing and pipeline scheduling are the most commonly used hardware acceleration methods for designing DNN accelerators based on FPGA [6], [11], [27]. Memory system optimization is also the key to accelerator design. Du et al. [6] optimized the energy efficiency of the FPGA accelerator by avoiding unnecessary data movement. Chen et al. [2] proposed a spatial data flow architecture, which reduces expensive data movement by reusing data locally. Computational complexity

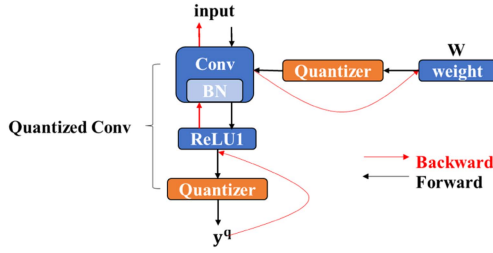


Fig. 1. Forward and backward computation graph for quantization aware training with STE assumption.

optimization is another effective method to improve the acceleration performance. Lei et al. [11] accelerated DNN by performing data quantization. Alessandro et al. [1] used the sparsity of neuron activation in DNN to speed up the calculation speed and reduce memory occupation. Lu et al. [13] improved the performance by compressing synaptic weights in large-scale sparse neural networks. Considering the practical application of FPGA accelerator for image classification, specific designs on network architecture, network parameters, and operation mode are still needed to present hardware-friendly performance.

III. WEIGHT AND ACTIVATION QUANTIZATION

In this section, we introduce the necessity of quantization for our traffic sign recognition algorithm designed for autonomous driving. Autonomous vehicles entail swift and precise recognition of traffic signs, which in turn requires a robust deep learning model. However, the limited computational resources of vehicular platforms, e.g., FPGAs, hinder the real-time inference of full-precision models. To address this issue, we perform quantization strategies on both the weights and activation parameters of our model prior to deployment. This process significantly reduces the model's computational resources, making it suitable for resource-constrained scenarios.

As illustrated in Fig. 1, a quantized network is composed of several quantized convolution blocks, each containing a serial of *conv* + *BN* + quantizer + *ReLU1* + quantizer operators. The design of the quantized network structure aims to optimize both the execution efficiency and accuracy of the model on hardware platforms. Specifically, the convolutional layer serves as the foundation of the neural network, responsible for extracting features from the input data; the batch normalization layer standardizes these features, helping to improve the model's generalization capabilities. The first quantizer converts continuous feature values into fewer discrete levels, significantly reducing the model's storage and computational demands. The *ReLU1* activation function introduces nonlinearity, assisting the network in learning complex data patterns. Finally, the second quantizer further processes the outputs post-*ReLU* activation to ensure that the features retain useful information in their quantized state. Overall, this structured design not only facilitates efficient operation on hardware platforms with limited resources but also maintains sensitivity and classification accuracy of the model towards the input data. During quantization aware training, batch normalization folding is used to simulate the reasoning behavior

closely. As the quantization operator has 0 gradient almost everywhere, we followed common practice to use the operation of straight-through estimator (STE) for gradient computation [8].

An STE used extensively in this work is quantize_k , which quantizes a real number input $r_i \in [0, 1]$ to a k -bits number output $r_o \in [0, 1]$. This STE is defined as (1) and (2).

$$\text{Forward} : r_o = \frac{1}{2^k - 1} \text{round}((2^k - 1)r_i). \quad (1)$$

$$\text{Backward} : \frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o}. \quad (2)$$

It is evident that the output of quantize_k is a real number represented by k -bits. Also, r_o is a k -bits fixed-point integer, so the dot product of two sequences of such real numbers can be efficiently calculated. We use k -bits representation of the weights with $k > 1$ and the STE function f_w^k to weights are applied as (3) and (4).

$$\text{Forward} : r_o = f_w^k(r_i) = \text{quantize}_k \left(\frac{\tanh(r_i)}{\max(|\tanh(r_i)|)} \right). \quad (3)$$

$$\text{Backward} : \frac{\partial c}{\partial r_i} = \frac{\partial r_o}{\partial r_i} \frac{\partial c}{\partial r_o}. \quad (4)$$

By construction, $\frac{\tanh(r_i)}{\max(|\tanh(r_i)|)}$ is a number in $[-1, 1]$, where the maximum in $\max(|\tanh(r_i)|)$ is taken over all weights in that layer. quantize_k will finally quantize this number to k -bits fixed-point ranging in $[0, 1]$.

In addition, an STE is applied to the input activations (r) of each weight layer. Here, the output of the previous layer has passed through a bounded activation function (*ReLU1*), which ensures $r \in [0, 1]$. In this paper, quantization of activation (r) to k -bits is simply as (5). A sample training algorithm of the full quantization model is given as Algorithm 1

$$f_a^k = \text{quantize}_k(r). \quad (5)$$

IV. SYSTEM DESIGN AND IMPLEMENTATION

As described in Section II-B, neither adversarial training nor feature squeezing changes the structure of DNN models. To improve the robustness of the embedded DNN model and realize efficient model forward inference, a DNN accelerator based on the original structure of the full quantization model should be designed first, and then, based on it, either an adversarial-trained model is deployed to FPGA or the color bit depth of the image inputted to FPGA is reduced to enhance embedded DNN model's adversarial robustness.

The FPGA-based system architecture we designed aims to optimize the execution efficiency and energy efficiency of DNNs through a customized hardware accelerator. Building on quantization, we implement strategies including data flow optimization, parallel computing, and pipelining to achieve efficient model computations. For instance, by optimizing data access strategies, we reduced the need for data movement, thereby, lowering power consumption; parallel processing significantly

Algorithm 1: Training an L-layer full quantization model with W-bit weights and A-bit activations. Weights and activations are quantized according to (1) and (2) respectively.

Require: a minibatch of inputs and targets (a_0, a^*), previous weights W , learning rate η

Ensure: update weight W^{t+1}

{1 Computing the parameter gradients :}

{1.1 Forward propagation :}

```

1: for k = 1 to L do
2:    $W_k^b \leftarrow f_w^W(W_k)$ 
3:    $\tilde{a}_k \leftarrow forward(a_{k-1}^b, W_k^b)$ 
4:    $a_k \leftarrow ReLU1(\tilde{a}_k)$ 
5:   if k < L then
6:      $a_k^b \leftarrow f_a^A(a_k)$ 
7:   end if
8:   Optionally apply pooling
9: end for
{1.2 Backward propagation :} Compute  $g_{a_L} = \frac{\partial C}{\partial g_{a_L}}$ 
  knowing  $a_L$  and  $a^*$ 
10: for k = L to 1 do
11:   Back-propagate  $g_{a_k}$  through activation function  $ReLU1$ 
12:    $g_{a_{k-1}} \leftarrow backward_{input}(g_{a_k}, W_k^b)$ 
13:    $g_{W_k^b} \leftarrow backward_{weights}(g_{a_k}, a_{k-1}^b)$ 
14:   Back-propagate gradients through the pooling layer if there is one
15: end for
{2 Accumulating the parameters gradients :}
16: for k = 1 to L do
17:    $g_{W_k} = g_{W_k^b} \frac{\partial W_k^b}{\partial W_k}$ 
18:    $W_k^{t+1} \leftarrow Update(W_k, g_{W_k}, \eta)$ 
19: end for

```

increased processing speed. In addition, we implemented advanced batch normalization and activation functions on the FPGA, further enhancing computational speed and data handling capabilities. These design choices were made to minimize latency and energy consumption while ensuring model accuracy, making the system more suitable for real-time and resource-constrained application environments. We illustrate key operations in this section.

A. System Architecture

The defense platform is based on Xilinx's ZYNQ platform, consisting of an ARM processor and an FPGA, to keep a lower power consumption than GPUs. To fully utilize the advantages of ARM and FPGA, it is necessary to specify the tasks of ARM and FPGA before designing the hardware part of the defense platform. The system architecture is shown in Fig. 2.

In the defense platform, FPGA achieves the forward inference acceleration of DNN. By scheduling DDR, DMA, and the DNN IP, the ARM processor completes data transmission, parameter configuration, and the accelerator's startup. The data interaction between ARM and FPGA is implemented through the AXI

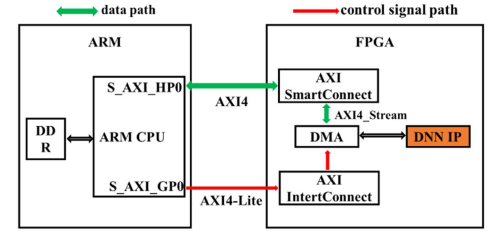


Fig. 2. System architecture of the defense platform.

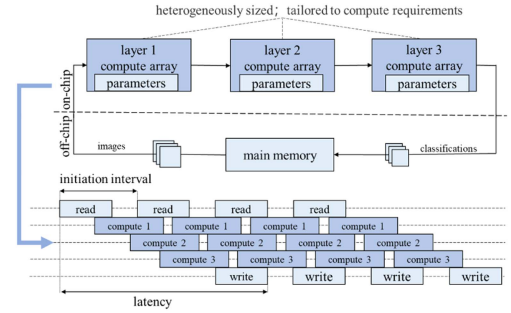


Fig. 3. Heterogeneous flow architecture based on FPGA.

protocol. In addition, the ARM storing input images in sequence in pixels, to reduce the cache space of intermediate results on FPGA.

B. General Scheme of the Hardware Design

The advantage of FPGA lies in its flexible and efficient parallel architecture design capacity, which is highly suitable for processing computation-intensive tasks. For the model to be accelerated, we run the following optimizations.

1) *Algorithm design optimization:* The quantization aware training is adapted to compress the size of DNN, so the storage space required by each parameter is reduced. All parameters can be placed in block RAM (BRAM) within FPGA, avoiding the time and the power consumption cost caused by frequent parameter invocation from off-chip storage resources during the process of DNN forward inference.

2) *Hardware design optimization:* An efficient pipelined structure is constructed based on FPGA. All layers of DNN are placed on the chip, so that all layers can perform parallel computation in the form of a pipeline, increasing data reuse.

C. Hardware Architecture

Based on the proposed design scheme, the heterogeneous streaming architecture using FPGA is presented in Fig. 3, which is customized to fit a given neural network model, instead of using a fixed architecture. Each layer of the neural network has an independent computing engine that communicates with each other through on-chip data streams in FPGA. Each engine starts to compute as soon as the previous engine starts to produce output. In addition, considering the compact size of the full quantization model, all neural network parameters are kept in on-chip memory. The intermediate results are also kept in

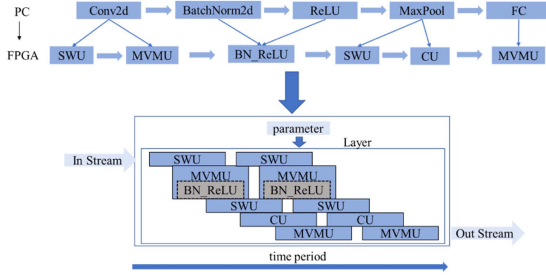


Fig. 4. Hardware architecture of adversarial attack defense platform.

on-chip memory. This avoids most accesses to off-chip memory and minimizes the latency (the time to finish classifying one image) by overlapping computation and communication. It also minimizes the initiation interval since a new image can enter the accelerator as soon as the first compute array is finished with the previous image; energy consumption is reduced as well. In addition, the logical resources needed by each layer are allocated based on the computational cost, so that the time sequence of each layer tends to be consistent and the operating efficiency of the accelerator is maximized.

For adversarial training, the weights of the original model in the on-chip memory of the FPGA are replaced by that of the adversarial-trained model, and then a new bitfile is required but it is an acceptable cost to correctly classify the adversarial examples on the FPGA. For feature squeezing, FPGA needs to intercept the corresponding bit when reading the input image data to reduce the input image's color bit depth, and then, a new bitfile is also required.

Considering the logic resources of FPGA in ZYNQ, a hardware acceleration scheme is designed, and then, the DNN model is deployed to FPGA, as shown in Fig. 4. In order to build pipeline architecture, the convolution layer, the batch normalization layer, the activation layer, the max-pooling layer and the full connection layer adopted by PC are transformed into a new hierarchical structure, which includes sliding window unit (SWU), matrix vector multiply unit (MVMU), batch normalization and activation combination unit (BN_ReLU), and comparison unit (CU). Each layer has its own cache space to store parameters. The convolution layer is divided into two parts (SWU and MVMU). The batch normalization layer and the activation layer are combined into a unit (BN_ReLU) to reduce the computation cost and the on-chip cache space usage. The max-pooling layer is also split into two parts (SWU and CU). The full connection layer is essentially an MVMU.

D. Sliding Window Unit

Convolutions can be transformed to matrix-matrix multiplications, which is the approach followed in this work. For the convolutional layer, the weights from the convolution filters are packed into a filter matrix, while a sliding window is moved across input images to form an image matrix. Subsequently, these matrices are multiplied to generate the output images. For

Algorithm 2: SWU.

Require: the number of images Num, the IFM's height H , the IFM's width W , the sliding window's size K

- 1: assign $line_buffer[K \cdot W] = 0$
- 2: **for** loop_num = 1 to Num $\cdot H$ **do**
- 3: **if** $h == H$ **then**
- 4: $h = 0$
- 5: $line_buffer[K \cdot W] = 0$
- 6: **end if**
- 7: $h++$
- 8: **for** $w = 1$ to W **do**
- 9: read 1-line pixels from the last layer
- 10: use $line_buffer$ to store pixels
- 11: a new-line pixels will overwrites the previous one in $line_buffer$ if $h > 3$
- 12: **end for**
- 13: **if** there are k -line pixels in $line_buffer$ **then**
- 14: output 3-line pixels from array $line_buffer$ in the order of the image matrix or the max-pooling operation
- 15: **end if**
- 16: **end for** = 0

the max-pooling layer, a sliding window is also needed to output data within the window for comparison.

The convolutional layer's SWU generates the image matrix from incoming feature maps, and an MVMU that actually computes the matrix-matrix product using a different column vector from the image matrix and generates one pixel of the output image each time. In order to better cater for the SIMD parallelism of the MVMU and to minimize cache requirements of intermediate result, the feature maps are interleaved such that each pixel contains all the input feature map (IFM) channel data for that position, as illustrated in Fig. 5(a).

The SWU requires k line-buffers to store pixels from the IFM, where k is the height of the sliding window. Since computing an output feature map (OFM) pixel needs all IFM pixels at a certain sliding window location, those IFM pixels can be processed in any order due to the commutative property of addition operation. Note that interleaving the filter matrix is done offline, and interleaving the input image can be done on-the-fly in the ARM. Storing pixels in this way allows us to implement MVMU and CU by using a single wide on-chip memory (OCM) instead of multiple narrow OCMs, and also allows the output of MVMU and CU to be passed directly to the next layer without any transposition. As illustrated in Fig. 5(b), the incoming IFM data of the convolutional layer is simply stored at the sequential addresses in a buffer, then, the memory locations corresponding to each sliding window are read out to produce the image matrix. A similar operation is performed for the incoming IFM data of the max-pooling layer's SWU in Fig. 5(c). The Algorithm 2 shows the whole process of SWU.

The specific calculation process of SWU is as follows. Initially, a linear buffer (line buffer) is allocated for each image, sized at the window size K multiplied by the image width W .

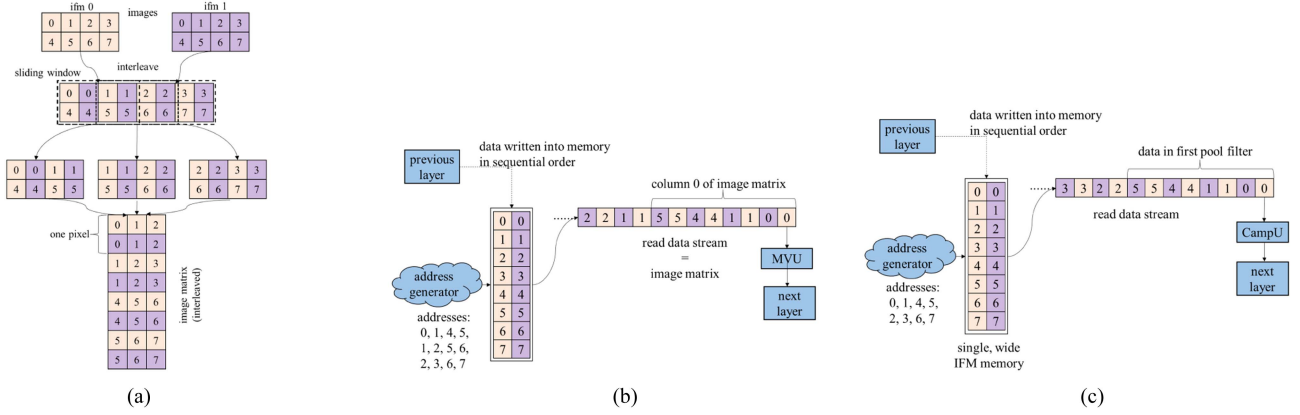


Fig. 5. Sliding window unit. (a) Interleaved channels for IFM data. (b) SWU in convolutional layer. (c) SWU in max-pooling layer.

This buffer temporarily stores the pixel rows that will participate in the computation. At the start of the algorithm, all values in the buffer are initialized to zero. Subsequently, each row of pixels in the image is iterated over. If the current row number reaches the image height H , the row counter h is reset, and the buffer is cleared to prepare for a new image. For each row in the image, a row of pixels is read from the previous layer and stored in the buffer. If the buffer already contains more than three rows of pixels, the new row will overwrite the oldest row. Once the buffer accumulates enough rows, they are output in image matrix order for the next processing step or max-pooling. This sliding window approach allows the FPGA to process the input image incrementally without the need to load the entire image into memory, significantly enhancing the speed and efficiency of data processing. By this method, the convolution operations for each window can be executed in parallel, greatly accelerating the generation of feature maps.

E. Matrix Vectors Multiply Unit

Most calculations in DNN can be represented by MVMUs, so MVMU is the computational core of the DNN accelerator. The calculation of the fully connected layer is essentially a multiplication of the matrix vector. That of the convolution layer also includes the matrix vector multiplication, so MVMU realizes the full connection layer as an independent component and also serves as a part of the convolution layer.

The structure of MVMU is shown in Fig. 6, which consists of an input buffer, an array of processing elements (PEs), and an output buffer. Each PE has multiple single instruction multiple data (SIMD) lanes. The number of PEs (M) in MVMU and SIMD lanes (N) in each PE can be configured to control the throughput. The weight matrix to be used is kept in OCM and distributed between PEs, and the input images stream through the MVMU as each one is multiplied by the matrix. Each PE receives exactly the same control signal and input vector data, but the input vector data in each PE needs to be multiplied and accumulated with different lines of the weight matrix. It can be seen that a PE corresponds to a channel of the OFM of the convolution layer, and the PE's SIMD lanes correspond to channels of the IFM

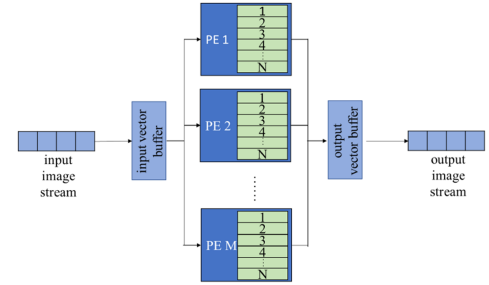


Fig. 6. Overview of the MVMU.

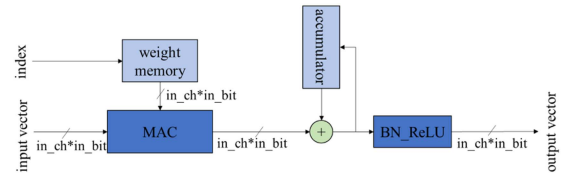


Fig. 7. MVMU PE datapath.

of the convolution layer. Once MVMU receives the pixel of the IFM, it immediately starts to calculate output vectors in parallel. The whole process of MVMU is shown in Algorithm 3.

The datapath of a PE in MVMU is shown in Fig. 7. First, each channel's data, which correlates to the pixel of IFM, and its corresponding weight are parallelly multiplied. Then, all N channels' results are added together. Finally, this summation is added to the accumulator register. The above steps are repeated for $k \times k$ times, where k is the size of the sliding window. Then, the results in the accumulator register will perform BN_ReLU operation to achieve an output vector data.

F. CU

The max-pooling layer of the DNN model is transformed into SWU and CU on FPGA. SWU outputs each pixel of the IFM in the sliding window. The maximum pixel in the sliding window will be found by the CU through bit stitching. Fig. 8 shows the datapath of the CU. First, each channel's data of an input pixel in

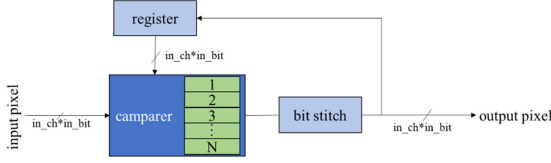


Fig. 8. CU datapath.

Algorithm 3: MVMU.

Require: the number of images Num, the OFM's height H, OFMs' width W, sliding window's size K, the number of PEs M, SIMD lanes N, the bit depth of weights W_BIT, the bit depth of a channel data of IFMs IN_BIT

```

1: for loop_num = 1 to Num · H · W · K · K do
2:   read a pixel
3:   for m = 1 to M do
4:     for n = 1 to N do
5:       result +=
         w((n + 1) · W_BIT - 1, n · W_BIT) ·
         pixel((n + 1) · IN_BIT - 1, n · IN_BIT)
6:     end for
7:   end for
8:   output result
9: end for = 0

```

Algorithm 4: CU.

Require: the number of the IFM's channel IN_CH, the bit depth of a channel data of IFMs IN_BIT, the sliding window's size K

```

1: read a pixel
2: for c = 1 to IN_CH do
3:   temp =
     pixel((c + 1) · IN_BIT - 1, c · IN_BIT)
4:   result((c + 1) · IN_BIT - 1, c · IN_BIT) =
     (temp > result((c + 1) · IN_BIT - 1, c ·
       IN_BIT)) ? temp :
     result((c + 1) · IN_BIT - 1, c · IN_BIT)
5: end for
6: k_cnt ++
7: if k_cnt == K · K then
8:   output result
9: end if = 0

```

the sliding window are compared with the corresponding data of the register in parallel. Then, each channel's bigger data forms a new pixel by bit stitching. Finally, the pixel is assigned to the register. The above steps are repeated for $k \times k$ times. Then, the maximum pixel in the sliding window will be outputted to the next layer. Algorithm 4 shows the whole process of CU.

V. EXPERIMENTS AND RESULTS

The experiments are divided into two parts. The first part of the experiments is completed on PC to select the appropriate quantization scheme and the best defense method. First, we

compare the accuracy of the full quantization model and the full precision model on the clean images. Then, three white-box attacks, FGSM, R+RGSM, and BIM, are used to compare the robustness of the full quantization model and the full precision model. Finally, the effects of adversarial training and the feature squeezing on the adversarial robustness of the full quantization model are studied. Based on the quantization scheme and the defense method in the first part, the second part is an adversarial attack defense platform on FPGA.

1) *Hardware Platform:* A GPU, RTX4000, is used to train the full precision model and the full quantization model, and test their adversarial robustness. Then, we design an adversarial attack defense platform based on Xilinx's Zynq UltraScale+ MP-SoC ZU3EG SBVA484 to study the robustness of the embedded neural network.

2) *Dataset:* The experiment utilized the German traffic sign recognition benchmark (GTSRB) dataset [17], a standard dataset for evaluating traffic sign recognition algorithms, comprising diverse images of traffic signs found on German roadways. The GTSRB dataset encompasses 51 840 landmark images distributed across 42 categories. In addition, it encompasses images captured under various challenging conditions such as, lighting variations, partial occlusion, rotation, and diverse weather conditions. During the data preprocessing stage, we resize the image to a 160×160 dimension and then, perform standard ImageNet normalization following a random horizontal flip operation.

3) *Network:* Considering the memory and computing resources of the FPGA platform, the neural network model's structure used in this work includes five convolution blocks and one full connection layer. Each convolution block is a stack of a convolution layer, a ReLU1 activation layer, a batch-normalization layer, and a max-pooling layer. The model has a very regular structure, consisting exclusively of 3×3 convolution and 2×2 max-pooling layers. The spatial dimensions are steadily reduced from 160×160 pixels to 5×5 pixels, while the number of channels is simultaneously increased from 3 to 400.

4) *Adversarial Examples:* FGSM, R+FGSM, and BIM are used to generate the adversarial examples. Referring to their experimental settings and for more obvious adversarial attack variance, we selected three of the adversarial attack perturbation values, $\epsilon = 0.01, 0.02$, and 0.03 , respectively. Besides, for the R+FGSM attack, we set $\epsilon_1 = \epsilon/2$. For the BIM attack, we set $\alpha = 1/255$ and the number of iterations to $\lfloor \min(\epsilon + 4, 1.25\epsilon) \rfloor$.

A. Choice of Quantization Schemes

1) *Accuracy Comparison Between Full Quantization Model and Full Precision Model:* The quantization of the model is divided into *weight quantization* and *activation quantization*. The weights of the DNN model were fixed to 4-b [5] to simplify the experiment and the full quantization model ran under different *activation* quantization schemes.

As shown in Fig. 9, we changed the *activation* bits of full quantization models. When the bit depth of model *activation* is set to 1, the accuracy of the full quantization model is approximately 7% lower than that of the full precision model. Moreover,

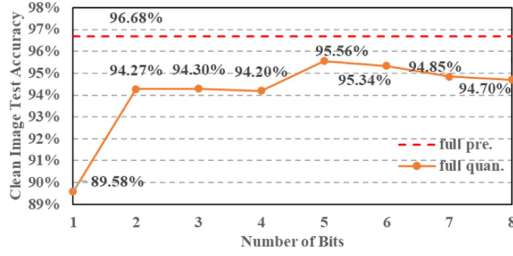


Fig. 9. Accuracy comparison on clean images between different full quantization models and the full precision model.

when the bit depth of the *activation* is increased beyond 1, the accuracy of the full quantization model on clean images only decreases by less than 2% compared to the full precision model. Specifically, a maximum accuracy of 96.56% is achieved when the *weight*'s bit depth is set to 4 and the *activation*'s bit depth is set to 5, respectively. This accuracy is only 1.12% lower than that of the full precision model, which has a bit depth of 32 for both the *weight* and *activation*. These results demonstrate that our methodology can significantly reduce computational and storage requirements while maintaining a high level of accuracy, with a marginal reduction of less than 2%.

2) *Adversarial Robustness Comparison Between Full Quantization Model and Full Precision Model*: FGSM, R+FGSM, and BIM were used to generate adversarial examples based on the GTSRB test set to test the differences in the adversarial robustness of the full quantization model and the full precision model under white-box attacks with different intensities.

As ϵ rises from 0.01 to 0.03. For the FGSM attack shown in Fig. 10(a), accuracy decreases are notable across the board: the full precision model falls from approximately 72% to 45%, the full quantization model with 1-b activation drops from around 82% to 62%, and the 8-b activation model from nearly 48% to 36%. In the case of the R+FGSM attack, depicted in Fig. 10(b), the full precision model's accuracy decreases from 90% to 73%, the 1-b activation model from 85% to 67%, and the 8-b activation model from 74% to 41%. Lastly, Fig. 10(c) demonstrates the effects of the BIM attack, where the full precision model's accuracy sharply reduces from about 82% to 42%, the 1-b activation model from nearly 83% to 64%, and the 8-b activation model plummets from roughly 58% to 22%. These results highlight the varying levels of vulnerability of each model configuration under different attack intensities.

As shown in Fig. 10, for all three types of adversarial examples, the robustness of both the full quantization model and the full precision model decreases as attack intensity increases, demonstrating a negative correlation. The white-box BIM attacks with $\epsilon = 0.03$ have the most significant impact on model accuracy. As the activation bit depth increases, the robustness of the full quantization model decreases. This is because, at lower bit depths, the model's representation is coarser and less sensitive to subtle perturbations, providing a certain level of robustness. When the activation bit depth is 1 b, the full quantization model exhibits greater robustness to white-box BIM and white-box FGSM attacks compared to the full precision model.

TABLE I
ADVERSARIAL TRAINING EXPERIMENTS TESTED WITH $\epsilon = 0.01/0.02/0.03$ ON PC

Defense Method	Clean	FGSM	R+FGSM	BIM	FGSM (B)	R+FGSM (B)	BIM (B)
Full Quan.(w4a2)	94.3	77/68/63	86/79/65	75/55/36	91/87/80	93/91/88	92/88/81
Adv. FGSM	95.4	88/82/75	92/90/87	88/82/73	93/91/87	94/94/93	94/92/89
Adv. R+FGSM	93.9	88/83/78	91/88/82	87/75/55	93/91/89	93/93/92	93/92/91
Adv. BIM	91.2	86/82/76	89/87/84	86/79/70	90/89/87	91/91/90	91/90/89

(B) indicates black-box attack. Bold numbers are the highest accuracy at each column.

This increased robustness is due to the input compression into a finite range, which diminishes the effect of subtle perturbations. However, under white-box R+FGSM attacks, the accuracy of the full quantization model is consistently lower than that of the full precision model, as it cannot fully exploit the advantages of the randomly initialized defense strategy.

3) *Transferability of Adversarial Examples*: Results of transferability for GTSRB are presented in Fig. 11 when the source network (i.e., the models where adversarial examples generated) is a full precision model or full quantization models with different quantization. The adversarial examples are transferred to full quantization models (target models) with different bit depths of activation.

We found that transferability results are quite poor for the FGSM, R+FGSM, and BIM attacks (values correspond to adversarial accuracy). The transferability of BIM adversarial examples is similar to that of R+FGSM adversarial examples. For the FGSM attack, relatively strong adversarial examples are built on the source models and are more likely to transfer. Compared with other full quantization models, the accuracy of the model with 1-b activation is less than 70% on the FGSM adversarial examples generated by source models, and less than 80% on the R+FGSM adversarial examples and the BIM adversarial examples. It indicates the transferability of adversarial examples is higher in the fully quantized model with 1-b activation, indicating lower security.

In summary, although 5-b activation showed the strongest ability in clean data (Fig. 9), it performed poorly when facing adversarial samples (Fig. 10). In the experiment of adversarial samples, although the full quantization model with 1-b activation showed relatively high robustness under three attacks, its accuracy on clean data was only 89.58%. Finally, combined with the transferability analysis (Fig. 11) and the trade-off between model accuracy and robustness, we chose the full quantization model with 2-b activation for the following experiments.

B. Choice of Defense Methods

1) *Effect of Adversarial Training on the Adversarial Robustness*: We utilize the GTSRB as the initial clean sample and use FGSM, R+FGSM, and BIM to generate different adversarial examples, then, add them to the training set to retrain the full quantization model. Finally, we separately trained a substituted VGG-16 model on the same dataset to generate black-box adversarial examples for black-box attack experiments.

Table I presents the results of the adversarial training methods. Adversarial R+FGSM training and adversarial BIM training reduces the accuracy of the full-quantification model by 0.4% and 3.1% on the clean data, respectively, but adversarial FGSM

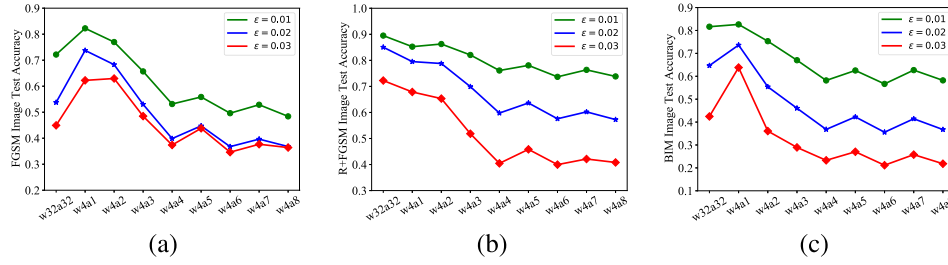


Fig. 10. Accuracy of the full precision model and the full quantization model on adversarial examples. $w_i a_j$ designates a model with an i -bit weight quantization and a j -bit activation quantization. ϵ denotes attack intensity. (a) FGSM. (b) R+FGSM. (c) BIM.

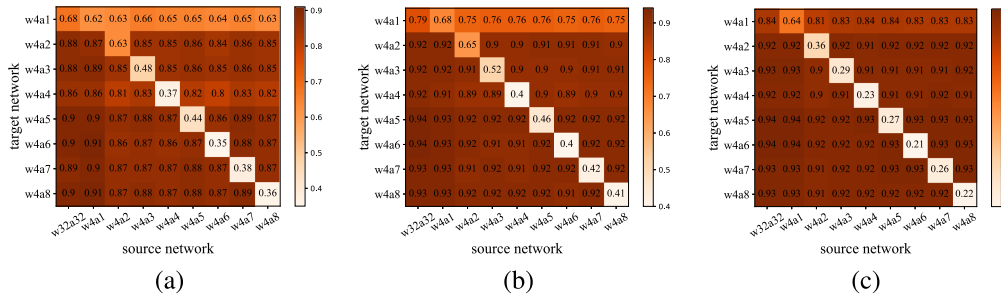


Fig. 11. Adversarial transferability results for the CIFAR10 data set. Columns are relative to source networks and rows to target networks. Values correspond to adversarial accuracy. The lower the value, the more transferability occurs. (a) FGSM. (b) R+FGSM. (c) BIM.

training increases the accuracy of the model by 1.1% on the clean data. All three adversarial training methods can improve the adversarial robustness of the full quantization model against the attacks in the experiment. For different white-box attacks and black-box attacks, under different attack intensities, the method based on FGSM adversarial training has an average improvement of 12.33% on white-box FGSM adversarial examples, 13.00% on white-box FGSM adversarial examples, 25.00% on white-box BIM adversarial examples, 4.33% on black-box FGSM adversarial examples, 3% on black-box R+FGSM adversarial examples, and 4.67% on black-box BIM adversarial examples. Overall, the best way to defend against white-box and black-box adversarial attacks is adversarial FGSM training.

2) Effect of Feature Squeezing on the Adversarial Robustness: We implemented the bit depth reduction operation in Python with the NumPy library. The input and output were in the same numerical scale $[0, 1]$ so that we did not need to change anything of the target models. For reducing to i -bit depth ($1 \leq i \leq 7$), we first multiplied the input value with $2^i - 1$ (minus 1 due to the zero value) then, round to integers. Next we scaled the integers back to $[0, 1]$, divided by $2^i - 1$. The information capacity of the representation was reduced from 8-b to i -bit with the integer-rounding operation. The images' color bit depth was reduced by 3-b, 4-b, and 5-b, respectively. Then, the accuracy of the full quantization model on the adversarial examples was tested.

The experimental results are shown in Table II. For clean images, as the color bit depth increases from 3 to 5 b, the accuracy of the full quantization model decreases less significantly: 13.2% for 3-b, 3.7% for 4-b, and 0.7% for 5-b. Under adversarial

TABLE II
FEATURE SQUEEZING EXPERIMENTS TESTED WITH $\epsilon = 0.01/0.02/0.03$ ON PC

Defense Method	Clean	FGSM	R+FGSM	BIM	FGSM (B)	R+FGSM (B)	BIM (B)
Full Quan.(w4a2)	94.3	77/68/63	86/79/65	75/55/36	91/87/80	93/91/88	92/88/81
Fea. Squ.(3-b)	81.1	77/73/67	79/77/71	77/73/68	79/76/71	81/79/76	79/76/73
Fea. Squ.(4-b)	90.6	82/75/67	86/81/69	82/73/62	87/83/77	89/87/83	88/85/79
Fea. Squ.(5-b)	93.6	80/72/65	78/78/64	79/62/43	90/86/79	92/90/87	90/87/81

(B) indicates black-box attack. Bold numbers are the highest accuracy at each column.

attacks, feature squeezing enhances accuracy against white-box attacks, with the most substantial gains at 3-b (up to 32% improvement) but decreases accuracy against black-box attacks. Considering a 4-b depth shows a consistent improvement in model performance under various adversarial conditions without as drastic a drop in clean data accuracy as the 3-b depth, it may offer a better balance. The 4-b depth also provides sufficient robustness while maintaining higher operational flexibility and generalization, making it a preferable choice for both defense effectiveness and higher clean data accuracy.

C. Robustness of Embedded DNN

In order to analyze the adversarial robustness of the FPGA-based defense platform, 50 images were randomly selected from each type of image in the test set of GTSRB, and there were a total of 2100 images for 42 types in the test set of GTSRB to be tested.

Based on the designed DNN accelerator, we first deployed the full quantization model with 4-b weights and 2-b activation to FPGA to compare the adversarial robustness of the model PC and that of the model on FPGA. Then, the first SWU

TABLE III

COMPARISON OF ADVERSARIAL ROBUSTNESS OF THE FULL QUANTIZATION MODEL BETWEEN FPGA AND PC

Platform	Clean	FGSM	R+FGSM	BIM	FGSM(B)	R+FGSM(B)	BIM(B)
PC	90.9	74/60/53	87/76/60	81/60/37	88/83/76	91/88/85	88/85/77
FPGA	90.0	74/62/54	87/76/60	81/60/41	87/83/75	89/88/85	87/84/77

TABLE IV

FPGA EXPERIMENTS TESTED WITH $\epsilon = 0.01/0.02/0.03$

Defense Method	Clean	FGSM	R+FGSM	BIM	FGSM (B)	R+FGSM (B)	BIM (B)
Fea.Squ.(4-b)	84.2	77/68/58	82/75/63	80/72/61	80/76/70	82/81/77	80/77/72
Adv. FGSM	92.5	83/75/67	90/86/82	88/79/70	90/87/84	91/90/88	90/87/84

(B) indicates black-box attack. Bold numbers are the highest accuracy at each column.

TABLE V

PERFORMANCE COMPARISON OF FULLY QUANTIZATION MODELS ON PC AND FPGA WITH EXTERNAL METHOD

Platform	Adversarial FGSM	Patch attack
PC	90.9	91.2
FPGA	92.5	88.2

of the accelerator was set to read the low 4-b data of each channel of the input image to analyze the impact of the feature squeezing on the robustness of the model. Finally, the weights in the on-chip memory of FPGA were replaced with the weights of the adversarial-trained model to analyze the impact of the adversarial training on the robustness of the model. The results are shown in Tables III and IV, respectively. First, the accuracy of the full quantization model on clean data on FPGA is 0.9% smaller than that on PC. The adversarial examples also affect the accuracy of the platform. Then, although the image color bit depth reduction on FPGA improves the robustness of the platform against the three white-box attacks, it reduces the accuracy of the platform on clean data and the robustness against the three black-box attacks. Finally, adversarial FGSM training improves the accuracy on clean data from 90.0% to 92.5%; it also improves the robustness of the platform against all six attacks.

D. Comparison With External Studies

We compare our work with the studies proposed in reference [25], as their research is highly relevant and offers a benchmark for comparison. Reference [25] introduced a patch-based adversarial training method and applies this technique to train three different models, which demonstrate robust performance on the GTSRB open dataset. For our experiments, we utilized the same test dataset which comprises 2100 images for 42 types extracted from the GTSRB datasets. To compare with reference [25], we separately train two models using our proposed method and the method in reference [25] on the same GTSRB dataset. Then, we implement the two well-trained models on both the same PC and the same FPGA platform, respectively. We guarantee that the trained models on both platforms achieve their best performance. The results of this comparison are presented in Table V.

In Table V, although the model trained with the external method slightly outperforms that trained with our method on the

PC platform, the performance when deployed on the FPGA platform drops by 3%. In contrast, the performance of our method improves by 1.6% when deployed on the FPGA platform. This is because our training method is specifically designed for FPGA platforms and the comparison result demonstrates its effectiveness.

E. Performance of Embedded Computing

1) *Image Processing and Throughput*: We compared the throughput of the designed defense platform with that of Intel's CPU (i9-10850K) and Nvidia's GPU (RTX4000). The run-time for 2100 images is measured from the moment when the FPGA accelerator is started, to the moment when the calculation of the last layer for the last image is finished. The number of images processed per second by different hardware platforms are presented in Fig. 12(a). The full quantization model running on different platforms has the same network topology and weight parameters. FPGA processes 152 more images per second than CPU and 75 less than GPU. The full quantization model with 4-b weights and 2-b activation requires 192.14 million operations (MOPS) to classify one image. According to the experimental results in Fig. 12(a), we can get the throughput of CPU, GPU, and FPGA as shown in Fig. 12(b). The throughput of FPGA reaches 55.37, about 3.15 times that of CPU and 0.79 times that of GPU.

2) *Power Consumption and Energy Efficiency*: In Fig. 12(c), we can see that the power consumption of CPU and GPU exceeds 95 W and 35 W, respectively, while that of FPGA is only about 2.8 W. In terms of energy efficiency, the FPGA-based defense platform still provides a significant advantage, as shown in Fig. 12(d). The energy efficiency of FPGA is 20.03 GOP/(s · W), which is much higher than that of the GPU [1.83 GOP/(s · W)]. Based on the above experimental results, although GPU has higher throughput than FPGA, its power consumption and power efficiency are inferior. Therefore, FPGAs are more suitable for real-time edge applications.

3) *Resource Utilization*: Fig. 12(e) shows the resource utilization of the designed defense platform employed on the Zynq UltraScale+ MPSoC ZU3EG FPGA, which includes 71.6 k LUTs, 216 BRAMs (36 KB), and 360 digital signal processor (DSP) slices. It can be seen that the utilization rate of LUT, BRAM, and DSP of FPGA reached 45%, 34%, and 100%, respectively. Despite the high resource utilization and the resulting long paths in the interconnect, the defense platform can still be synthesized for an adequate clock frequency of $f_{\max} = 200$ MHz. This is possible because the architecture fully distributes the computation as well as all the required data onto the different computational units. The independent computational units have no dependencies, allowing for mostly local routing and minimal global interconnections, which can be efficiently pipelined.

VI. DISCUSSION

1) *Impact and Implication*: Our well-designed defense platform can be implemented in most practical scenarios, which however, may still face practical issues when being deployed in

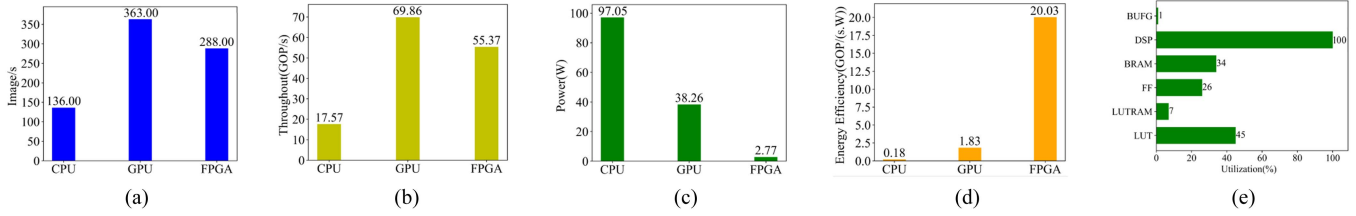


Fig. 12. Comparison of different accelerators on CPU, GPU, and FPGA. (a) Images processing. (b) Throughput. (c) Power consumption. (d) Energy efficiency. (e) Resource utilization.

extreme conditions. i) Bottlenecks in resource and hardware: FPGAs may face performance bottlenecks with complex models, which require further minimizing memory and computational overhead while maintaining algorithm performance. ii) Latency and real-time processing: Low latency and stringent real-time requirements under extreme conditions are difficult for existing hardware. iii) Power consumption: Power consumption needs to be further reduced for battery-powered devices, demanding extremely high energy efficiency. Future work may focus on addressing these challenges, ensuring broader practicability and efficiency, and making the platform viable for harsh conditions.

2) *Tradeoff and Applications*: We find that while quantized models untrained against adversarial attacks perform well on clean datasets, they are at the cost of robustness when subjected to attacked datasets. Therefore, the design and deployment of practical models should be in the context of specific application scenarios, incorporating appropriate adversarial training strategies and quantization schemes to achieve an optimal balance between accuracy and robustness.

3) *Applications of FPGA-based defence platform*: Defense platforms based on FPGAs are pioneers in delay control and real-time performance, which are well-suited for applications demanding high real-time and system security, such as autonomous driving, and aerospace control systems. These scenarios require not only a rapid response to changes in the external environment but also the capability to withstand potential malicious attacks to ensure continuous and stable system operation. Therefore, the characteristics of the FPGA platform make it an ideal choice for implementing efficient and secure computing in these fields.

VII. CONCLUSION

This work focused on the security of the quantized neural networks, which were widely deployed in CPU/GPU/FPGA. It also paved a possible direction to bridge the following two critical areas in deep learning: 1) efficiency and 2) robustness. Although the full quantization model had poor adversarial robustness, adversarial training improved the model's average accuracy from 90.0% to 92.5% on clean data, from 66% to 81% under white-box attacks, and from 84% to 88% under black-box attacks, without increasing any computing and storage cost of FPGA. In the meantime, the designed defense platform could accelerate DNN's forward inference and fully use the logic resources of FPGA with a high power efficiency. In the future, we will explore methods that can further improve the performance

of adversarial training at the edge, such as, adding random noises during training.

REFERENCES

- [1] A. Aïmeur et al., "NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 3, pp. 644–656, Mar. 2019.
- [2] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.
- [3] S. Cohen, O. Arbili, Y. Mirsky, and L. Rokach, "TTTS: Tree test time simulation for enhancing decision tree robustness against adversarial examples," in *Proc. AAAI Conf. Artif. Intell.*, 2024, vol. 38, pp. 20993–21000.
- [4] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2015, vol. 28, pp. 3123–3131.
- [5] T. Dettmers and L. Zettlemoyer, "The case for 4-bit precision: K-bit inference scaling laws," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 7750–7774.
- [6] L. Du et al., "A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things," *IEEE Trans. Circuits Syst. I. Reg. Papers*, vol. 65, no. 1, pp. 198–208, Jan. 2018.
- [7] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014, *arXiv:1412.6572*.
- [8] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning," *Coursera, Video Lectures*, vol. 264, no. 1, pp. 2146–2153, 2012.
- [9] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
- [10] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," 2016, *arXiv:1611.01236*.
- [11] P. Lei, J. Liang, Z. Guan, J. Wang, and T. Zheng, "Acceleration of FPGA based convolutional neural network for human activity classification using millimeter-wave radar," *IEEE Access*, vol. 7, no. 99, pp. 88917–88926, 2019.
- [12] G. Liu and L. Lai, "Efficient adversarial attacks on online multi-agent reinforcement learning," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2024, vol. 36, pp. 24401–24433.
- [13] Y. Lu, C. Wang, L. Gong, and X. Zhou, "SparseNN: A performance-efficient accelerator for large-scale sparse neural networks," *Int. J. Parallel Program.*, vol. 46, no. 11, pp. 1–12, 2018.
- [14] N. Morgulis, A. Kreines, S. Mendelowitz, and Y. Weisglass, "Fooling a real car with adversarial traffic signs," 2019, *arXiv:1907.00374*.
- [15] X. Shao and Y. Shi, "Neural adaptive control for MEMS gyroscope with full-state constraints and quantized input," *IEEE Trans. Ind. Informat.*, vol. 16, no. 10, pp. 6444–6454, Oct. 2020.
- [16] N. D. Singh, F. Croce, and M. Hein, "Revisiting adversarial training for imagenet: Architectures, training and generalization across threat models," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, 2024, vol. 36, pp. 13931–13955.
- [17] J. Stalldkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Netw.*, vol. 32, pp. 323–332, 2012.
- [18] C. Szegedy et al., "Intriguing properties of neural networks," 2013, *arXiv:1312.6199*.
- [19] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," 2017, *arXiv:1705.07204*.
- [20] D. Wang, W. Yao, T. Jiang, C. Li, and X. Chen, "RFLA: A stealthy reflected light adversarial attack in the physical world," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2023, pp. 4455–4465.

- [21] Z. Wang, T. Pang, C. Du, M. Lin, W. Liu, and S. Yan, "Better diffusion models further improve adversarial training," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 36246–36263.
- [22] W. Wu, P. Yang, W. Zhang, C. Zhou, and X. Shen, "Accuracy-guaranteed collaborative DNN inference in industrial IoT via deep reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 4988–4998, Jul. 2021.
- [23] K. Xu et al., "Adversarial T-shirt! evading person detectors in a physical world," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 665–681.
- [24] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," 2017, *arXiv:1704.01155*.
- [25] B. Ye, H. Yin, J. Yan, and W. Ge, "Patch-based attack on traffic sign recognition," in *2021 IEEE Int. Intell. Transp. Syst. Conf.*, 2021, pp. 164–171.
- [26] X. Zhang, G. Xiao, M. Duan, Y. Chen, and K. Li, "PH-CF: A phased hybrid algorithm for accelerating subgraph matching based on CPU-FPGA heterogeneous platform," *IEEE Trans. Ind. Informat.*, vol. 19, no. 7, pp. 8362–8373, Jul. 2023.
- [27] Z. Zhang, M. A. P. Mahmud, and A. Z. Kouzani, "FitNN: A low-resource FPGA-based CNN accelerator for drones," *IEEE Internet Things J.*, vol. 9, no. 21, pp. 21357–21369, Nov. 2022.
- [28] S. Zhu et al., "Autodan: Automatic and interpretable adversarial attacks on large language models," 2023, *arXiv:2310.15140*.
- [29] Y. Zhuo, Z. Yin, and Z. Ge, "Attack and defense: Adversarial security of data-driven FDC systems," *IEEE Trans. Ind. Informat.*, vol. 19, no. 1, pp. 5–19, Jan. 2023.



Hanhui Deng received the B.E. degree in microelectronics from Northwestern Polytechnical University, Xi'an, China, in 2017, and the M.S. degree in computer science from Newcastle University, Newcastle upon Tyne, U.K., in 2019. He is currently working toward the Ph.D. degree in design engineering with Hunan University, Changsha, China.

His research interests include machine learning and cyber–physical–social systems.



Yanwen Wang (Member, IEEE) received the M.S. degree in electrical engineering from the Missouri University of Science and Technology, Missouri, MO, USA, in 2013, and the Ph.D. degree in computer science from Hong Kong Polytechnic University, Hong Kong, Hong Kong, in 2019.

He is currently an Associate Professor with the College of Electrical and Information Engineering, Hunan University, Changsha, China. His research interests include mobile computing, signal processing, and data analysis.



Yufeng Lu received the B.S. degree in electronic information engineering from the College of Electrical and Information Engineering, Anhui University of Finance and Economics, China, in 2019 and the M.S. degree in electronic and communication engineering from Hunan University, China, in 2022.

His research interests include deep learning and FPGA.



Xiaokang Shi received the B.S. degree in electronic information engineering from Huaqiao University, Xiamen, China, in 2022. He is currently working toward the M.S. degree in electronic information with the College of Electrical and Information Engineering, Hunan University, Changsha, China.

His research interests include wireless sensing and embedded systems.



Jianan Jiang received the B.E. degree in mechanical engineering from Donghua University, Shanghai, China, in 2022. He is currently working toward the M.E. degree in design science with Hunan University, Changsha, China.

His research interests include deep learning and its application to computer vision.



Jiwu Lu (Member, IEEE) received the B.S. degree from Zhejiang University, Hangzhou, China, the M.S. degree from Siegen University, Siegen, Germany, and the Ph.D. degree in electronic engineering from Twente University, Enschede, The Netherlands, in 2011.

He is currently a Professor with Hunan University, Changsha, China. He worked with the National Institute of Standards and Technology in the US. His research interests include interdisciplinary research, including energy harvesting and edge computing in IoT, and power semiconductor devices in microelectronics.



Di Wu (Member, IEEE) received the Ph.D. degree in computer science from the University of California, Irvine, CA, USA, in 2013.

He is currently a Professor with Hunan University, Changsha, China, and an Adjunct Researcher with the University of California. His research interests include future networking, intelligent analytics, and smart architecture.

Dr. Wu has actively served on many conference committees. He is currently an Associate Editor for IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS.