Article

An Approach of Binary Neural Network Energy-Efficient Implementation

Jiabao Gao , Qingliang Liu and Jinmei Lai *

State Key of ASIC and System, School of Microelectronics, Fudan University, Shanghai 201203, China; jbgao17@fudan.edu.cn (J.G.); 17112020007@fudan.edu.cn (Q.L.)

* Correspondence: jmlai@fudan.edu.cn

Abstract: Binarized neural networks (BNNs), which have 1-bit weights and activations, are well suited for FPGA accelerators as their dominant computations are bitwise arithmetic, and the reduction in memory requirements means that all the network parameters can be stored in internal memory. However, the energy efficiency of these accelerators is still restricted by the abundant redundancies in BNNs. This hinders their deployment for applications in smart sensors and tiny devices because these scenarios have tight constraints with respect to energy consumption. To overcome this problem, we propose an approach to implement BNN inference while offering excellent energy efficiency for the accelerators by means of pruning the massive redundant operations while maintaining the original accuracy of the networks. Firstly, inspired by the observation that the convolution processes of two related kernels contain many repeated computations, we first build one formula to clarify the reusing relationships between their convolutional outputs and remove the unnecessary operations. Furthermore, by generalizing this reusing relationship to one tile of kernels in one neuron, we adopt an inclusion pruning strategy to further skip the superfluous evaluations of the neurons whose real output values can be determined early. Finally, we evaluate our system on the Zynq 7000 XC7Z100 FPGA platform. Our design can prune 51 percent of the operations without any accuracy loss. Meanwhile, the energy efficiency of our system is as high as 6.55×10^5 Img/kJ, which is $118\times$ better than the best accelerator based on an NVIDIA Tesla-V100 GPU and $3.6\times$ higher than the state-of-the-art FPGA implementations for BNNs.

Keywords: binary neural network (BNN); kernel inclusion similarity; inclusion pruning strategy; FPGA

Citation: Gao, J.; Liu, Q.; Lai, J. An Approach of Binary Neural Network Energy-Efficient Implementation. *Electronics* **2021**, *10*, 1830. <https://doi.org/10.3390/electronics10151830>

Academic Editors: Joo-Young Kim and Valeri Mladenov

Received: 30 May 2021

Accepted: 26 July 2021

Published: 30 July 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Neural networks running on general-purpose CPUs or GPUs are a common solution for various computer vision applications such as image classification [1], radar signal processing [2], and face recognition [3]. However, these solutions tend to be power-hungry because CNNs are computationally intensive, with over billions of operations for the inference of one input [4]. Accordingly, the implementation of CNNs in mobile applications is usually quite challenging [5].

Binarized neural networks (BNNs) can realize efficient inference by optimizing the precision of weights and activations into a single bit [6–8]. Meanwhile, BNNs can directly replace the multiply–accumulate operations by simple XNOR and popcount operations [9], which are well suited to being executed on FPGAs [10]. Nevertheless, BNN inference for the practical classification tasks still involves abundant computations, and thus an efficient hardware accelerator is necessitated. The energy efficiency of the accelerator is extremely important because BNN inference is usually considered for applications in mobile platforms and intelligent devices, which have strict constraints as regards energy consumption. However, the abundant redundancies existing in BNN inference still severely limit the overall energy efficiency of these accelerators [11].

Previous designs which accelerate BNN inference by means of pruning the redundant operations usually obtain savings of around 30 percent of the redundancies without any accuracy loss. Moreover, in order to achieve higher pruning rates, the procedure of retraining the networks is vital. However, they suffer from a serious accuracy drop ranging from 1 to 3 percent. Furthermore, their accelerators also require the consumption of very large amounts of logic resources, which, in particular, results in massive energy consumption, thereby inducing substantial overheads to the overall energy efficiency of the accelerators.

In this paper, we propose an approach to accelerate BNN inference in an energy-efficient way, to effectively address the issues presented above. Firstly, we observe the processes of calculating the outputs of different related kernels. Based on the observation that they contain extensive unnecessary operations, which can be pruned by reusing the calculated outputs of other related kernels, we construct a formula to establish the tight reusing relationships between them. Then, we generalize this reusing relationship to one tile of kernels in one neuron. Benefiting from this generalization, we adopt an inclusion pruning strategy to build the architecture, which brings an opportunity to further prune the evaluations of the redundant neurons whose real output values can be determined early.

To conclude, our key contributions are as follows:

- We propose an approach to accelerate BNN inference in an energy-efficient way. The excessive redundant operations in the binarized convolution processes of multi related kernels are safely skipped by adopting the kernel inclusion similarity scheme.
- In addition, an inclusion pruning strategy is exploited to further save the superfluous evaluations of neurons whose real output values can be determined early, resulting in pruning the whole operations of these neurons without any accuracy loss.
- To the best of our knowledge, our design can prune up to 51 percent of the operations while maintaining the original accuracies, leading to obtaining $118\times$ and $3.6\times$ energy efficiency improvement, respectively, when compared with the prior state-of-the-art works implemented on GPU/FPGA.

2. Preliminary and Related Works

2.1. Preliminary of BNN

BNNs evolve from conventional DNNs through Binarized Weight Networks (BWN). It has been observed that if both the weights and inputs are binarized, even the additions and subtractions can be degraded to logical bit operations. As a result, XNOR-Net, in which the convolutions are estimated by XNOR and bit-counting operations, is proposed. In this paper, we focus on XNOR-Net, and in the following sections, BNN refers to XNOR-Net [9].

In BNNs, the basic structure contains several essential functions in each convolutional and fully connected layer. These functions include XNOR, popcount, batch normalization (BN), and binarization (BIN). Firstly, the original multiply-accumulate in traditional DNNs becomes XNOR and popcount in BNNs. Then, the output of popcount is normalized in BN, which is essential to guarantee high accuracy in BNNs. BN incorporates full-precision float point (FP) operations, i.e., two FP MUL/DIVs and three FP ADD/SUBs, which can be denoted by:

$$y_{i,j} = \left(\frac{x_{i,j} - E[x_{*,j}]}{\sqrt{\text{Var}[x_{*,j}] + \epsilon}} \right) \cdot \gamma_j + \beta_j \quad (1)$$

where γ_j and β_j are learned in training and fixed in inference. Lastly, the normalized outputs from BN $y_{i,j}$ are binarized in BIN by comparing with zero, which is calculated by:

$$x^b = \text{sign}(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ -1, & \text{otherwise} \end{cases} \quad (2)$$

In other words, BIN acts as the non-linear activation function. By performing these steps, the real output of one neuron can be achieved.

2.2. Related Works

Firstly, BNNs still contain many redundant operations, which can be further pruned [12]. Ref. [13] used weight flipping frequency as an indicator of sensitivity to accuracy for pruning BNNs. They demonstrated that the weights with a high weight flipping frequency, when the training process is sufficiently close to convergence, are less sensitive to accuracy. Then, they shrunk the number of channels in each layer by the same percentage of the insensitive weights to reduce the effective size of the BNNs. Finally, they derived a 20–40 percent reduction in binary operations. However, they could also suffer from up to 1 percent of accuracy degradation. In addition, [14] proposed two metrics including cosine distance and Euclidean distance to measure the importance of each filter in BNNs. Then, they leveraged Bayesian optimization to efficiently determine the pruning ratio for each layer. However, their experimental results showed that the accuracies of their selected networks, which were quantized with the XNOR-Net scheme, both degraded by 2.69–4.79% on the CIFAR-10 [15] dataset. Ref. [16] proposed a two-stage pipeline method combining filter pruning and binarization to further compress the network. When compared to the original model, they achieved a FLPO reduction and model size compression at the cost of roughly a 4% accuracy drop. In contrast, our pruning method skips the redundant operations for particular kinds of neurons by reusing the calculated partial results of the kernels in other neurons, without changing the real values of these neurons, thereby inducing no accuracy loss.

Meanwhile, BNNs have already widely been implemented on FPGAs due to their flexibility and direct bit-manipulation capability [17–21]. Ref. [17] presented a framework for building fast and flexible FPGA accelerators using a flexible heterogeneous streaming architecture. Ref. [19] proposed an FPGA-based BNN accelerator that drastically cut down the hardware consumption by using resource-aware model analysis. Ref. [20] employed an HLS design methodology for the productive development of the FPGA-based BNN accelerator. Their HLS implementation leveraged the optimizations including loop ordering, unrolling, and local buffering. Meanwhile, [22] also used an HLS-based platform to model the architecture. However, they actually focused on accelerating the networks with 1-bit weights and 2-bit activation function outputs by developing a streaming architecture. In addition, [21] focused on binarizing the first layer in BNNs and reducing the processing time for the first layer. However, these designs all neglected to optimize the extensive redundancies in BNN inference, resulting in the limited energy efficiency of their accelerators. Moreover, [18] used a new binarization method in which the weights and the activations were constrained to either 0 or 1; as a result, the convolutional operations could be implemented by AND gates and popcount operations, which might be more suitable for hardware implementation. Then, they further proposed an input data reuse algorithm to decrease the data access from off-chip memory. Owing to these optimizations, the frequent accesses rate and the power consumption were both reduced. However, they also ignored the extensive redundancies in BNNs, which could still induce significant overheads to the overall energy efficiency of the accelerator [11]. Moreover, their method also needed to retrain the network, which might affect the network accuracy. Nevertheless, they only evaluated the small dataset MNIST [22]. By contrast, we focus on pruning the abundant redundancies in the network to enhance the energy efficiency of the accelerator. Meanwhile, since the step of retraining the networks is not required, our design can keep the original accuracies of the networks.

Furthermore, there are many recent works exploring the opportunities to save the redundancies in BNNs to improve the overall energy efficiency of the accelerators. Ref. [23] proposed a neuron pruning technique which could only be applied to the fully connected layers on the BNN, and retrained the network for this adjustment. Accordingly, their optimization scheme was not suitable for saving the redundant operations in the whole network. In [24], the repeated filters, inverse filters, and similar filters were exploited so that the number of operations in the last several convolutional layers of BNNs could be reduced effectively by sharing the results of these filters. However, these reductions did not necessarily lead to a high performance enhancement due to extra kernel local-

ization overhead. More importantly, their schemes could induce a 2.98 percent accuracy degradation. Moreover, [11] proposed an out-of-order architecture to prune the irregular redundant edges. However, their extensive experiment results only show that they could prune 30 percent of the operations on average without any accuracy loss, only resulting in $2.2\times$ inference speed-up. Moreover, their pruning rates could be further improved by 19 percent with regularization at training, whereas their accuracies degraded by 3.3 percent, which is not suitable for being directly applied in the actual scenes [25,26]. Moreover, it is necessary for this accelerator to check whether the current accumulated result is already larger than the threshold of the current neuron once all of the XNOR-popcount operations of each input channel in this neuron have been completed, which requires complex control logic leading to abundant extra energy consumption. Therefore, significant degradations in their energy efficiency are inevitable.

3. Proposed Methods

3.1. Kernel Inclusion Similarity

BNNs can reduce the computation and memory requirements of CNNs by limiting both the activations and weight parameter values to either -1 or 1 . This extreme quantization scheme can reduce the memory requirement for storing the model. Meanwhile, with binarized weights and activations, the dominant computations of a BNN model become binary multiply-accumulate operations, which can be further implemented in a highly hardware-friendly way by simply performing the XNOR and popcount operations.

Hence, the binarized convolution processes of the general kernels and inputs in BNNs can be expressed as:

$$y = \sum_{j=0}^{k-1} \sum_{i=0}^{k-1} (f_{i,j} \odot w_{i,j}) \quad (3)$$

where $f_{i,j}$ represents the input bit located in the i th row and j th column of the current convolutional window, $w_{i,j}$ is the weight located in the i th row and j th column of the related convolutional kernel, k is equal to the size of this kernel, $\odot$ is a bit-wise XNOR operation, and y is the final output of the current kernel convolved with the input bits in the current convolutional window.

Based on these optimized binarized convolution processes, we observe that almost all operations to compute the convolutional results of two related kernels corresponding to the same input are repeated, if these two kernels only have one different weight. We regard this phenomenon as kernel inclusion similarity.

Then, in order to prune these extensive repeated operations to reduce the performance overheads imposed on the energy efficiency of the accelerator for BNN inference, we firstly focus the two specific kinds of kernels satisfying the requirements of the kernel inclusion similarity.

Therefore, we firstly attempt to build an equation to establish a quantitative relationship between their outputs. Then, we can reuse the partially calculated results while utilizing this equation so that the repeated operations during their convolutional processes can be safely skipped—specifically, when one kernel w_b contains b weight values of one, i.e., the values of these b positions are one and the other weight values are zero. Meanwhile, if the weight values of another kernel w_c in the same b positions are one, the value of only one position a in the other $(k * k - b)$ positions is one while the remaining values are all zero. In this circumstance, the kernel inclusion similarity is satisfied.

Moreover, and as depicted in Figure 1, if the weight values in the red circles are different from each other while the remaining values in the kernels w_b and w_c are exactly the same, our proposed kernel inclusion similarity immediately works. In this case, we can directly gain the output y_c of w_c by utilizing the already computed result y_b of w_b . Firstly, the value zero marked by the black circles, representing the corresponding bit of the input, which locates in the position of a , will be added twice by reusing y_b , which is tagged with the blue circles. Then, they will subsequently be subtracted by a constant value one to

obtain y_c . Accordingly, the reusing relationship between the convolutional outputs y_c and y_b can be defined by:

$$y_c - y_b = f_a + f_a - 1 \quad (4)$$

where f_a is the a_{th} bit of the input. Hence, we can directly achieve y_c by reusing y_b instead of performing the normal XNOR-popcount operations. As a result, we can safely skip the redundant operations during the processes of calculating the outputs of related kernels by exploiting our proposed kernel inclusion similarity scheme.

Figure 1. Kernel inclusion similarity scheme.

3.2. Reference Consistency Kernel Inclusion Similarity

After successfully building the kernel inclusion similarity scheme, we expand it into the hardware-friendly reference consistency kernel inclusion similarity in order to be suitable for being deployed on the hardware. Accordingly, we can simply rearrange the formula of the kernel inclusion similarity scheme for the clarity of the reusing hierarchical relationships:

$$y_c = y_b + f_a + f_a - 1 \quad (5)$$

Based on this equation, we simply select the result y_b of kernel w_b as our basic unit. Specifically, the number of weights whose values are equal to one in kernel w_b is less than w_c . Then, we can utilize this basis to update the result y_c of kernel w_c instead of calculating it by conducting the normal XNOR-popcount operations. As stated in [24], most useful features in images are local, which means that the filter size is preferred to be small in many applications. Among many choices of filter sizes, the filters whose sizes are equal to three in convolution layers are very popular and work well. Thus, if we assume the kernel size $k = 3$, which can be applicable and promising in practice, we can further classify the kernels into nine categories for the sake of better distinguishing the types of basic units. Each category i individually represents that the number of weights whose values are equal to one ranges from zero to nine. Moreover, we utilize C_9^i to equivalently denote the category i . Benefiting from this category strategy, our focus naturally switches from the total number of the kernels in the current layer to the number of categories of these kernels. Furthermore, for the sake of maximizing the reusing opportunities provided by the kernel inclusion similarity scheme while inducing only a little extra storage cost, we only choose three categories as our references, namely $i = 2$, $i = 4$, and $i = 6$. Utilizing these three basic categories as our references, we can make the most of the kernel inclusion similarity scheme in that the other reusing categories including $i = 3$ and $i = 5$ and $i = 7$ can cover the majority of the whole kernels in one neuron. Therefore, the kernels belonging

to the basic categories will firstly be selected as the initial ones whose convolutional results should be computed by performing the XNOR-popcount operations. After obtaining the results of basic kernels and then when encountering the kernels belonging to the reusing categories, their convolutional results will be efficiently calculated by adopting the new optimized reference consistency kernel inclusion similarity scheme and eliminating the unnecessary operations.

Moreover, if the kernel size is larger than three, our proposed method can still work well. If the kernel size is equal to five, which is used as an example for illustration, we can classify these kernels into twenty-five categories. Equally, we only choose three categories as our references, namely $i = 10$, $i = 12$, and $i = 14$. In this case, the other reusing categories simply include $i = 11$, $i = 13$, and $i = 15$. These kernels can also cover the majority of the whole kernels in one neuron due to the fact that the weights in one neural network follow Gaussian distribution. As a result, our proposed method can be effectively applied to the larger kernels whose sizes are equal to five because the numbers of categories of basic categories and reusing categories for these larger kernels both remain the same as the smaller kernels.

3.3. Inclusion Pruning Strategy

Due to the fact that one neuron consists of multiple kernels, the scope of our proposed reference consistency kernel inclusion similarity (RCKIS) scheme can be optimally extended from a single kernel to one neuron to further enhance the pruning opportunities. Therefore, there can exist multiple kernels all conforming to the RCKIS scheme located in different output channels of several neurons while targeting the same input bits. We can pack these kernels together into one tile. Moreover, the kernels in these qualified tiles of different neurons all locate in exactly the same positions. In this circumstance, we can utilize the numerical relationships between the results of these tiles to bridge corresponding neurons. Meanwhile, it has been proven that the batch normalization function and binarization function in BNNs can be replaced by threshold-based comparisons to improve the originally designed structure [17]. Enlightened by this optimization, we observe that the final outputs of specific neurons can be determined early such that the evaluations of these neurons, namely the operations of the whole general kernels in these neurons, can be safely eliminated without affecting the accuracy of the network at all. Thus, we formally introduce the inclusion pruning strategy to save the superfluous evaluations of these neurons in BNN inference without any accuracy degradations.

Hence, when dealing with the first neuron n_f , the kernels in this neuron satisfying the RCKIS scheme will be packed into one tile, which will be regarded as the baseline tile. Consequently, the result r_c of this baseline tile can be firstly calculated and subsequently stored in the on-chip memory:

$$r_c = \sum_{i=0}^{v-1} y_{b_i} \quad (6)$$

where v is the number of kernels in this tile, and y_{b_i} are the corresponding convolutional results of these kernels in this tile. Meanwhile, because the kernels in one network all remain constant during inference, the mapping relationships between the kernels in the qualified tiles of these neurons can be grasped early, as well as the capacities of these tiles v . Therefore, after dealing with the first neuron, the already stored result r_c can be exploited to facilitate the process of achieving the final real output of the next related neuron n_n . For this neuron, the kernels located in exactly the same positions as the kernels in the first neuron will be packed into one tile, named the reusing tile. Then, the result r'_c of this reusing tile in this neuron n_n can be computed by:

$$r'_c = r_c + \sum_{z=0}^{v-1} f_z + \sum_{z=0}^{v-1} f_z - v \quad (7)$$

where f_z are the input bits corresponding to the weights whose characteristics meet the RCKIS scheme. When the reusing relationships between the results of the baseline tile and reusing tile have been achieved, we can further obtain opportunities to prune the superfluous evaluations of these neurons because their real output values can be determined early in some cases. Accordingly, for the first neuron n_f , if the result of its baseline tile is already larger than or is equal to its threshold t , then the final real output of this neuron n_{f_o} can be instantly set to one so that the evaluation for this neuron, including the operations of the whole remaining general kernels contained in this neuron, can be directly skipped:

$$n_{f_o} = 1, \text{ if } r_c \geq t \quad (8)$$

where n_{f_o} is the final real output of the current neuron, and t is the threshold of this neuron. Equally, this strategy can also be applied to the neuron n_n . If the calculated result of the reusing tile in n_n already exceeds the threshold t' of this neuron, then the real output of n_n can also be determined early and is equal to one, yielding the saving of the evaluation for this neuron certainly including the operations of the whole remaining general kernels in this neuron:

$$n_{n_o} = 1, \text{ if } r'_c \geq t' \quad (9)$$

where n_{n_o} is the real output of this neuron n_n . Then, inspired by the fact that the input bits in BNNs are all larger than or equal to zero, the result r'_c will always greater than or at least be equal to one particular optimized result r_o . More importantly, r_o can immediately be obtained as soon as the result r_c of the neuron n_f is computed because the capacities of the current baseline tile and reusing tile v can be determined before the inference process of one BNN network. Therefore, this optimized result r_o can be calculated by:

$$r_o = r_c - v \quad (10)$$

Furthermore, the numerical relationship between r'_c and r_o can motivate us to prune the whole evaluation for the neuron n_n if the final real output of this neuron can be determined early in one specific case. Hence, the real output of this neuron can be directly set to one if r_o already exceeds the threshold of this neuron t' such that the entire evaluation of this neuron n_n , including the operations of its whole general kernels, can be skipped in advance:

$$n_{n_o} = 1, \text{ if } r_o \geq t' \quad (11)$$

4. Design Architecture

Figure 2 depicts the architecture of the processing elements array. Noticeably, we use the ellipses for simplification. There are PIC processing elements in the PE array; as a result, PIC input channels in one neuron can be dealt with at the same time. Moreover, one processing element is targeted for different input channels of T neurons in parallel. Considering the i -th input channels in different neurons as an example for illustration, if two kernels in them conform to the requirements of the kernel inclusion similarity scheme, the kernel chosen as the basic unit and corresponding input bits will firstly be sent to the Shared PE, as demonstrated in Figure 3. In the Shared PE, the convolutional results of the input bits and the weights are computed by performing the XNOR-popcount operations. Then, for another kernel whose convolutional result can be calculated by reusing the result of the basic kernel, the Shared PE will transfer the result of the basic kernel to the KIS PE for further computations. The architecture of KIS PE is illustrated in Figure 4. Based on the kernel inclusion similarity scheme, the input bit related to the only different weight among these two kernels will be correctly identified. Next, only three addition operations are conducted between this input bit and the result of the basic kernel transmitted from the Shared PE to directly achieve the final convolutional result of this kernel. Moreover, there also can exist the neurons whose kernels are all entirely independent of other neurons. In this circumstance, the IND PEs are responsible for calculating their convolutional results by means of implementing the normal XNOR-popcount operations. Therefore, enormous

redundant operations can be pruned by exploiting a pluralistic allocation strategy for different kinds of neurons.

Figure 2. The architecture of the processing elements array.

Figure 3. The Shared PE architecture.

Figure 4. The architecture of KIS PE.

The convolutional results of *PIC* input channels in one neuron will be subsequently sent to the Reusing Processing Module as described in Figure 5. These results are reasonably allocated to three paths. Firstly, for the kernels in one neuron, which are in the range of one baseline tile selected based on the inclusion pruning strategy, this neuron will be distinguished as the first neuron. Then, the Counter Unit will always count the processed cycles, and the results of these kernels will be delivered into the Reuse Engine in time when reaching their cycles. Then, the results of the remaining kernels in this neuron, which are entirely independent of other neurons, will be conveyed into the adder tree for direct accumulations. Thus, for this first neuron, all of the results stored in the Reuse Engine will be accumulated together and then added to the output derived from the adder tree to obtain the final result of this neuron. Meanwhile, for the other neurons n_{other} whose kernels can partially meet the reference consistency kernel inclusion similarity scheme with the ones of the first neuron, the kernels in the baseline tile of the first neuron will

be selected respectively for each neuron in n_{other} . Then, the results of these kernels are accumulated together and subtracted by the number of chosen kernels. After carrying out these operations, if the obtained result is already larger than or equal to the threshold of this neuron, the real output of this neuron can be directly set to one so that the evaluation of this neuron can be entirely skipped. Lastly, for the neurons whose kernels are solely independent of the other neurons, the results of their own channels will all be delivered to the adder trees specifically designed for processing these neurons.

Figure 5. The architecture of Reusing Processing Module.

Figure 6 shows the block diagram of the proposed BNN accelerator. When receiving the signal from the Global Controller to start up the acceleration process, the input bits and weights located in corresponding positions in the IFM buffer and Weight buffer will be conveyed into the PE array. If two kernels of different neurons satisfy the kernel inclusion similarity scheme, the selected basic kernel will be dealt with in the Shared PE by implementing the XNOR-popcount operations. This calculated result will be delivered to the KIS PE to be further reused by the other kernel to save the redundant operations in its convolution process. Therefore, it only takes three addition operations to gain the result of this kernel. Meanwhile, if the kernels in the neurons are completely independent of the other neurons, the results of these kernels are calculated by performing the normal XNOR-popcount operations. After fetching the convolutional result of each single kernel, they will further be handled in the Reusing Processing Module. If the reference consistency inclusion similarity scheme is established in the tiles of multi-neurons, the convolutional results of eligible kernels in the first neuron will be individually allocated to the corresponding Reuse Engine for each neuron. Moreover, the results of these kernels will be then accumulated together in their own Reuse Engine to obtain the final one, which will further be compared with the related threshold in the Comparison Module. If this final result is already larger than or equal to the threshold of this neuron, the real output of this neuron can be directly set to one. In this case, the Global Controller will receive indication signals and immediately control the PE arrays to skip the whole evaluation of this neuron. Finally, the real outputs of the neurons will be reordered in the Reorder Module in order to constitute the normal input feature maps for the next layer.

Figure 6. The overall architecture of the proposed efficient accelerator.

5. Experimental Results

Our proposed accelerator is implemented by the Xilinx Zynq 7000 XC7Z100 FPGA (Xilinx, San Jose, CA, USA), which is used as a showcase in this paper. We select the LeNet-5 [27] and VGG-like [28] network models, which are trained on the MNIST [22] and CIFAR-10 [15] datasets, respectively, to evaluate our design. The resource utilization and power consumption are reported in Vivado Design Suite after implementations.

5.1. Operation Count Reduction

Our approach can save up to 51 percent of the operations while maintaining the original accuracies. In contrast, former works suffer from different degrees of degradation in the accuracies because they ordinarily adopt various training techniques to change the weights themselves in the networks for their pruning schemes, such that the neurons' values start to flip during these training processes, thereby incurring errors. However, our approach incurs no accuracy loss as the final real values of the neurons are not affected at all during our acceleration processes.

5.2. Comparisons to Other Designs

Table 1 presents the implementation results of our accelerator for LeNet-5 and comparisons with another state-of-the-art work. As shown in Table 1, the performance of our accelerator is 6921.97 GOPS at 450 MHz clock frequency, achieving $2.05\times$ enhancement against the latest work accelerating the same network [18]. Moreover, our design is able to obtain an energy efficiency of 4019.73 GOPS/W. In addition, due to the pruning of 57 percent of the operations, our energy efficiency is $2.47\times$ better than [18], which neglects to optimize the redundancies in the network.

Table 1. Comparisons with other designs on MNIST dataset.

	[18]	This Work
Platform	Virtex-7 (XC7VX690T)	Zynq-7000 (XC7Z100)
Dataset	MNIST	MNIST
Network	LeNet	LeNet
Frequency (MHz)	500	450
Throughput (GOPS)	3378	6921.97
Power (W)	2.08	1.72
Efficiency (GOPS/W)	1624	4019.73
Accuracy	98.2%	98.4%

Furthermore, as shown in Table 2, our design also exhibits excellent energy efficiency for a VGG-like network compared with other works accelerating the same network, including the state-of-the-art work [11], which implements lossless accelerator and lossy accelerator to, respectively, accelerate the same network, VGG-like. Our approach can skip 51 percent of the operations while maintaining the original accuracy, leading to the achievement of up to 6.55×10^5 Img/kJ energy efficiency, which is $3.59\times$ higher than the lossless accelerator [11]. Furthermore, when compared to the lossy accelerator [11], the energy efficiency of our design is still $2.47\times$ better than the lossless accelerator [11]. However, the accuracy is degraded severely. Moreover, as demonstrated in Table 2, our accelerator also shows excellent performance in terms of the throughput when compared to the prior designs. Therefore, our approach is more successful.

Table 2. Comparisons with other designs on CIFAR-10 dataset.

	[20]	[19]	[21]	[23]	[11]	This Work
Platform	Zynq-7000 (XC7Z020)	Stratix-V 5SGSD8	Zynq-7000 (XC7Z020)	Zynq-7000 (XC7Z020)	Zynq-7000 (XC7Z045)	Zynq-7000 (XC7Z100)
Dataset	CIFAR-10	CIFAR-10	CIFAR-10	CIFAR-10	CIFAR-10	CIFAR-10
Network	VGG-like	VGG-like	VGG-like	VGG-like	VGG-like	VGG-like
Frequency (MHz)	143	150	-	143	200	200
Throughput (GOPS)	207.8	9396.41	722	502	2019.70	2953.56
Power (W)	4.7	26.2	3.3	2.2	-	-
Energy efficiency (Img/kJ)	3.58×10^4	-	1.58×10^5	1.85×10^5	1.82×10^5 (lossless)	2.65×10^5 (lossy)
Accuracy	88.5%	86.3%	88.6%	81.8%	88.5%	85.2%

5.3. Cross-Platform Evaluation

As shown in Table 3, the energy efficiency of our accelerator is compared with various systems using CPU and GPU to accelerate the same network, VGG-like. The energy efficiency of our design can reach up to 6.55×10^5 Img/kJ as it eliminates 51 percent of the operations, which is $8.41 \times 10^4\times$ and $118.17\times$ higher than CPU Xeon E5-2640 [19] and GPU V100 [29]. Therefore, our proposed approach demonstrates superior energy efficiency when accelerating the BNN network. On the other hand, the throughput of our accelerator is far superior to other platforms including various kinds of CPU and GPU.

Table 3. Cross-platform evaluation.

	[19]	[19]	[29]	This Work
Platform	CPU	GPU	GPU	Zynq-7000 (XC7Z100)
	Xeon E5-2640	Tesla K40	V100	
Dataset	CIFAR-10	CIFAR-10	CIFAR-10	CIFAR-10
Network	VGG-like	VGG-like	VGG-like	VGG-like
Frequency (MHz)	2500	745	1370	200
Throughput (GOPS)	181.29	1853.87	1237.42	9685.04
Power (W)	95	235	-	14.89
Energy efficiency (Img/kJ)	7.79	5830	5543	6.55×10^5
Accuracy	86.3%	86.3%	89.9%	88.7%

6. Conclusions

In this paper, we propose an approach to accelerate BNN inference in an energy-efficient way. The proposed scheme is capable of reusing the convolutional results of multiple related kernels in BNNs to save the abundant redundancies in them, rather than retraining the network, which was the method exploited by prior works. The reusing relationships between the outputs of these kernels are successfully formed by a newly developed equation. In addition, the inclusion pruning strategy is employed to directly skip the evaluations of redundant neurons whose real output values can be determined

early. Therefore, our approach can effectively alleviate the inefficiencies in the former designs in terms of energy efficiency and the loss of accuracy. Our approach can save 51 percent of the operations without any accuracy loss. Meanwhile, the energy efficiency of our design can achieve values of up to 6.55×10^5 Img/kJ, which is $118\times$ better than the best system design on an NVIDIA Tesla-V100 GPU and $3.6\times$ better than the state-of-the-art design based on FPGA. Moreover, although FPGA is used as a showcase in this paper, our approach can be applied in any mobile devices, thereby making it an excellent solution to accelerate BNNs in an energy-efficient way regardless of the platform.

Author Contributions: Methodology, Q.L.; Supervision, J.L.; Writing—review and editing, J.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China under Grant No. U20A20202.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Chen, Q.; Zhang, W.; Yu, J.; Fan, J. Embedding Complementary Deep Networks for Image Classification. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 9230–9239.
- Le Kernec, J.; Fioranelli, F.; Ding, C.; Zhao, H.; Sun, L.; Hong, H.; Lorandel, J.; Romain, O. Radar Signal Processing for Sensing in Assisted Living: The Challenges Associated with Real-Time Implementation of Emerging Algorithms. *IEEE Signal Process. Mag.* **2019**, *36*, 29–41. [\[CrossRef\]](#)
- Chen, K.; Wu, Y.; Qin, H.; Liang, D.; Liu, X.; Yan, J. R³ Adversarial Network for Cross Model Face Recognition. In Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019; pp. 9860–9868.
- Lahoud, F.; Süssstrunk, S. Zero-Learning Fast Medical Image Fusion. In Proceedings of the 2019 22th International Conference on Information Fusion (FUSION), Ottawa, ON, Canada, 2–5 July 2019; pp. 1–8.
- Wu, D.; Cao, W.; Wang, L. SpWMM: A High-Performance Sparse-Winograd Matrix-Matrix Multiplication Accelerator for CNNs. In Proceedings of the 2019 International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 9–13 December 2019; pp. 255–258.
- Bethge, J.; Bartz, C.; Yang, H.; Chen, Y.; Meinel, C. MeliusNet: Can binary neural networks achieve mobilenet-level accuracy? *arXiv* **2020**, arXiv:2001.05936.
- Shimoda, M.; Sato, S.; Nakahara, H. All binarized convolutional neural network and its implementation on an FPGA. In Proceedings of the 2017 International Conference on Field Programmable Technology (ICFPT), Melbourne, VIC, Australia, 11–13 December 2017; pp. 291–294.
- Li, A.; Su, S.M. Accelerating Binarized Neural Networks via Bit-Tensor-Cores in Turing GPUs. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 1. [\[CrossRef\]](#)
- Rastegari, M.; Ordonez, V.; Redmon, J.; Farhadi, A. XNOR-net: ImageNet classification using binary convolutional neural networks. In *European Conference on Computer Vision*; Springer: Cham, Switzerland, 2016; pp. 525–542.
- Wang, E.; Davis, J.J.; Cheung, P.Y.K.; Constantinides, G.A. LUTNet: Learning FPGA Configurations for Highly Efficient Neural Network Inference. *IEEE Trans. Comput.* **2020**, *69*, 1795–1808. [\[CrossRef\]](#)
- Geng, T.; Li, A.; Wang, T.; Wu, C.; Li, Y.; Shi, R.; Wu, W.; Herbordt, M. O3BNN-R: An Out-of-Order Architecture for High-Performance and Regularized BNN Inference. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 199–213. [\[CrossRef\]](#)
- Kim, H.; Sim, J.; Choi, Y.; Kim, L.-S. NAND-Net: Minimizing Computational Complexity of In-Memory Processing for Binary Neural Networks. In Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, DC, USA, 16–20 February 2019; pp. 661–673. [\[CrossRef\]](#)
- Li, Y.; Ren, F. BNN Pruning: Pruning Binary Neural Network Guided by Weight Flipping Frequency. In Proceedings of the 2020 21st International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 25–26 March 2020; pp. 306–311.
- Guerra, L.; Zhuang, B.; Reid, I.; Drummond, T. Automatic pruning for quantized neural networks. *arXiv* **2020**, arXiv:2002.00523.
- Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. Master's Thesis, University of Toronto, Toronto, ON, Canada, 2009. Available online: <https://www.cs.toronto.edu/kriz/cifar.html> (accessed on 30 April 2021).
- Wang, J.; Jin, X.; Wu, W. TB-DNN: A Thin Binarized Deep Neural Network with High Accuracy. In Proceedings of the 2020 22nd International Conference on Advanced Communication Technology (ICACT), Phoenix Park, Korea, 16–19 February 2020; pp. 419–424.
- Umuroglu, Y.; Fraser, N.J.; Gambardella, G.; Blott, M.; Leong, P.; Jahre, M.; Vissers, K. FINN: A framework for fast, scalable binarized neural network inference. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 65–74.

18. Xian, Z.; Li, H.; Li, Y. Weight Isolation-Based Binarized Neural Networks Accelerator. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 12–14 October 2020; pp. 1–4.
19. Liang, S.; Yin, S.; Liu, L.; Luk, W.; Wei, S. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* **2018**, *275*, 1072–1086. [[CrossRef](#)]
20. Zhao, R.; Song, W.; Zhang, W.; Xing, T.; Lin, J.-H.; Srivastava, M.; Gupta, R.; Zhang, Z. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 15–24.
21. Guo, P.; Ma, H.; Chen, R.; Li, P.; Xie, S.; Wang, D. FBNA: A Fully Binarized Neural Network Accelerator. In Proceedings of the 2018 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, 27–31 August 2018; pp. 51–513.
22. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [[CrossRef](#)]
23. Baskin, C.; Liss, N.; Zheltonozhskii, E.; Bronstein, A.M.; Mendelson, A. Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform. In Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Vancouver, BC, Canada, 21–25 May 2018; pp. 162–169.
24. Fujii, T.; Sato, S.; Nakahara, H. A threshold neuron pruning for a binarized deep neural network on an FPGA. *IEICE Trans. Inf. Syst.* **2018**, *101*, 376–386. [[CrossRef](#)]
25. Chang, Y.-C.; Lin, C.-C.; Lin, Y.-T.; Chen, Y.-C.; Wang, C.-Y. A Convolutional Result Sharing Approach for Binarized Neural Network Inference. In Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 780–785.
26. Nurvitadhi, E.; Sheffield, D.; Sim, J.; Mishra, A.; Venkatesh, G.; Marr, D. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, China, 7–9 December 2016; pp. 77–84.
27. Yonekawa, H.; Nakahara, H. On-Chip Memory Based Binarized Convolutional Deep Neural Network Applying Batch Normalization Free Technique on an FPGA. In Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, USA, 29 May–2 June 2017; pp. 98–105.
28. Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv* **2016**, arXiv:1602.02830.
29. Li, A.; Geng, T.; Wang, T.; Herbordt, M.; Song, S.L.; Barker, K. BSTC: A novel binarized-soft-tensor-core design for accelerating bit-based approximated neural nets. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, CO, USA, 17–22 November 2019; pp. 1–30.